The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

**How to Check the Ordering Part Number**

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

**For More Information**

Please contact your local sales office for additional information about Cypress products and solutions.

**About Cypress**

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

F$^2$MC-16FX 16-Bit Microcontroller

# MB96300 Series Hardware Manual

Doc. No. 002-19737 Rev. **

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

## Copyrights

© Cypress Semiconductor Corporation, 2006-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.

# Preface

The information for microcontroller supports is shown in the following website.

Be sure to refer to the "Check Sheet" for the latest cautions on development.

See "Check Sheet" at the following support page "Check Sheet" lists the minimal requirement items to be checked to prevent problems beforehand in system development..

URL: http://www.cypress.com/support/

## 1. Objectives and intended reader

Thank you very much for your continued patronage of Cypress products.

The MB96300 Super series has been developed as a general-purpose version of the $F^2$MC-16FX series, which is an original 16-bit single-chip microcontroller compatible with the Application Specific IC (ASIC).

This manual explains the functions and operation of the MB96300 Super series for designers who actually use the MB96300 Super series to design products. Please read this manual first.

## 2. Structure of this manual

1. Overview

***The MB96300 Super series is a family member of the $F^2$MC-16FX micro controllers. It consist of many different series of microcontrollers, which are targetting different applications. Programming between all members of the Super series is common.***

2. CPU

***This chapter explains the CPU.***

3. Interrupts

***This chapter explains the interrupt functions and operations.***

4. DMA

***This chapter explains the DMA functions and operations.***

5. Delayed Interrupt

***This chapter explains the functions and operations of the delayed interrupt.***

6. Clocks

**This chapter describes the clocks used by F2MC-16FX family micro controllers.**

7. Clock Modulator

***This chapter provides an overview of the Clock Modulator and its features. It describes the register structure and operation of the Clock Modulator.***

8. Resets and Startup

***This chapter describes the resets and the startup of for the F2MC-16FX family microcontrollers.***

9. Standby Mode and Voltage Regulator Control Circuit

***This chapter explains the functions and operations of the standby mode control circuit and the control of the internal voltage regulator. The regulator control can be used to optimize power consumption, especially in standby modes.***

10. Source Clock Timers

***This chapter explains the functions and operations of the three source clock timers (RC clock timer, Main clock timer and Sub clock timer).***

11. Watchdog Timer and Watchdog Reset

***This chapter explains the functions and operations of the Watchdog timer and reset.***

12. External Bus Interface

***This chapter explains the functions and operations of the external bus interface.***

13. I/O Ports

***This chapter explains the functions and operations of the I/O ports.***

14. 16-Bit I/O Timer

***This chapter explains the functions and operations of the 16-bit I/O Timer.***

15. 16-Bit Reload Timer (With Event Count Function)

***This chapter explains the functions and operations of the 16-bit Reload Timer (with the Event Count Function).***

16. Programmable Pulse Generator

***This chapter explains the functions and operations of the Programmable Pulse Generator.***

17. External Interrupts

***This chapter explains the functions and operations of the External Interrupts.***

18. A/D Converter

***This chapter explains the function and operation of the A/D converter.***

19. Alarm Comparator

***This chapter explains the functions and operations of the Alarm Comparator.***

20. USART

***This chapter explains the functions and operation of the LIN USART.***

21. 400 kHz I2C Interface

**This section describes the functions and operation of the fast I2C interface.**

22. CAN Controller

***This chapter explains the functions and operations of the CAN controller.***

23. Clock Output Function

***This chapter describes the functions and operations of the clock output function.***

24. Real Time Clock

***This chapter explains the functions and operations of the Real Time Clock.***

25. Clock Calibration Unit

***This chapter explains the functions and operation of the Clock Calibration Unit***

26. LCD Controller/Driver

***This chapter describes the functions and operations of the LCD Controller/Driver.***

27. Stepper Motor Controller

***This chapter explains the functions and operations of the stepper motor controller.***

28. Sound Generator

***This chapter explains the functions and operations of the sound generator.***

29. USB Function

***This chapter explains the functions and operation of the USB Function.***

30. USB Mini-Host

***This chapter describes the functions and operations of USB Mini-host.***

31. Memory Patch Function

***This chapter explains the memory patch function and how the data patch or a debug function can be realised with that.***

32. ROM/RAM Mirroring Module

**This chapter explains the ROM mirroring module.**

33. Flash Memory

***This chapter explains the functions and operation of the flash memory.***

34. Mask-ROM Memory

***This chapter explains the functions and operation of the Mask-ROM memory.***

35. ROM/Flash Security

***This chapter explains the functions and operation of the ROM/Flash security.***

36. Examples of Serial Programming Connection

***This chapter describes how to connect the MCU for in-circuit serial programming of the Flash memory.***

# Contents

**Preface** 3

**1. Overview** 23

1.1 Features....23
1.1.1 Features....23
1.2 Super Series Lineup....24
1.2.1 Super Series Lineup....24
1.3 Block Diagram of MB96V300B....24
1.3.1 Block Diagram of MB96V300B....25
1.4 General Note on Using This Document....26
1.4.1 Device Dependent Resources....26
1.4.2 Described Resources....26
1.5 16-bit I/O-Timer Configuration....26
1.5.1 Connection between Input Capture Units and Free Running Timers....26
1.5.2 Connection between Output Compare Units and Free Running Timers....27
1.5.3 Relation between Timer Clear Function and OCU, Compare Clear and Free Running Timer....27
1.6 Input Capture Unit Source Select for LIN-USART....27
1.6.1 Overview of Connection between LIN-USART and ICU....27
1.6.2 Input Capture Unit Source Select....28
1.7 Peripheral Resource Pin Relocation....33
1.7.1 Overview of the Peripheral Resource Relocation Register....33
1.7.2 Peripheral Resource Relocation Register 0 (PRRR0)....34
1.7.3 Peripheral Resource Relocation Register 1 (PRRR1)....35
1.7.4 Peripheral Resource Relocation Register 2 (PRRR2)....36
1.7.5 Peripheral Resource Relocation Register 3 (PRRR3)....37
1.7.6 Peripheral Resource Relocation Register 4 (PRRR4)....38
1.7.7 Peripheral Resource Relocation Register 5 (PRRR5)....39
1.7.8 Peripheral Resource Relocation Register 6 (PRRR6)....40
1.7.9 Peripheral Resource Relocation Register 7 (PRRR7)....41
1.7.10 Peripheral Resource Relocation Register 8 (PRRR8)....42
1.7.11 Peripheral Resource Relocation Register 9 (PRRR9)....43
1.7.12 Peripheral Resource Relocation Register 10 (PRRR10)....44
1.7.13 Peripheral Resource Relocation Register 11 (PRRR11)....45
1.7.14 Peripheral Resource Relocation Register 12 (PRRR12)....46
1.7.15 Peripheral Resource Relocation Register 13 (PRRR13)....47

**2. CPU** 49

2.1 Outline of the CPU....49
2.1.1 Outline of the CPU....49
2.2 Hardware Structure....50
2.2.1 Hardware Structure of the CPU....50

2.2.2 Hardware structure of the 16FX Core ..... 51
2.3 Memory Space ..... 53
2.3.1 Memory Areas ..... 53
2.3.1.1 Memory areas in the F2MC-16FX system ..... 53
2.3.1.2 I/O area ..... 54
2.3.1.3 Extended I/O area ..... 55
2.3.1.4 RAM area ..... 55
2.3.1.5 ROM area ..... 56
2.3.1.6 Vector table area ..... 56
2.3.2 Linear Addressing Method ..... 56
2.3.2.1 24-bit operand specification ..... 56
2.3.3 Bank Addressing Method ..... 57
2.3.3.1 Bank addressing method ..... 57
2.3.3.2 Initialization of bank registers ..... 58
2.3.4 Multi-byte Data in Memory Space ..... 59
2.3.4.1 Multi-byte data allocation in memory space ..... 59
2.3.4.2 Accessing multi-byte data ..... 59
2.4 Special Registers ..... 60
2.4.1 Accumulator (A) ..... 61
2.4.1.1 Accumulator (A) ..... 61
2.4.2 User Stack Pointer (USP) and System Stack Pointer (SSP) ..... 62
2.4.2.1 User stack pointer (USP) and system stack pointer (SSP) ..... 62
2.4.3 Processor Status (PS) ..... 63
2.4.3.1 Processor status (PS) ..... 63
2.4.3.2 Condition code register (CCR) ..... 63
2.4.3.3 Register bank pointer (RP) ..... 65
2.4.3.4 Interrupt level mask register (ILM) ..... 65
2.4.4 Program Counter (PC) ..... 66
2.4.4.1 Program counter (PC) ..... 66
2.4.5 Direct Page Register (DPR) ..... 67
2.4.5.1 Direct page register (DPR) ..... 67
2.4.6 Bank register (PCB, DTB, ADB, USB, SSB) ..... 67
2.4.6.1 Bank Register ..... 67
2.5 General-Purpose Registers ..... 68
2.5.1 Register Bank ..... 68
2.5.1.1 Register bank ..... 68
2.5.2 Addressing General-Purpose Registers ..... 69
2.5.2.1 Addressing General-purpose registers ..... 69
2.6 Prefix Codes ..... 70
2.6.1 Bank Selection Prefix ..... 70
2.6.1.1 Bank select prefix ..... 70
2.6.2 Common Register Bank Prefix (CMR) ..... 72
2.6.2.1 Common register bank prefix (CMR) ..... 72
2.6.3 Flag Change Inhibit Prefix (NCC) ..... 72
2.6.3.1 Flag change inhibit prefix code (NCC) ..... 72
2.6.4 Prefix Code Restrictions ..... 73
2.6.4.1 Interrupt rejecting instructions ..... 73
2.6.4.2 Restrictions on interrupt rejecting instructions and prefix codes ..... 74
2.6.4.3 Multiple prefix codes in succession ..... 74

**3. Interrupts 75**

3.1 Outline of Interrupts ..... 75
3.1.1 Hardware interrupts ..... 75

3.1.2 Software interrupts ..... 76
3.1.3 Exceptions ..... 76
3.1.4 Direct memory access (DMA) ..... 76
3.2 Interrupt Vector ..... 77
3.2.1 Interrupt vector ..... 77
3.3 Interrupt Control Registers (ICR) ..... 79
3.3.1 Interrupt control register (ICR) ..... 79
3.4 Non Maskable Interrupt (NMI) ..... 81
3.4.1 NMI control status register (NMI) ..... 81
3.5 Interrupt Flow ..... 83
3.5.1 Interrupt flow ..... 83
3.6 Hardware Interrupts ..... 85
3.6.1 Hardware interrupts ..... 85
3.6.2 Structure of the hardware interrupt system ..... 85
3.6.3 Hardware interrupt operation ..... 85
3.6.4 Hardware interrupt processing time ..... 87
3.7 Software Interrupts ..... 87
3.7.1 Software interrupts ..... 87
3.7.2 Structure of the software interrupt system ..... 87
3.7.3 Software interrupt operation ..... 88
3.8 Multiple interrupts ..... 89
3.8.1 Multiple hardware interrupts ..... 89
3.8.2 Multiple software interrupts ..... 89
3.8.3 Interrupt acceptance priority ..... 90
3.9 Exceptions ..... 91
3.9.1 Software exceptions (op-code) ..... 91
3.9.1.1 Execution of an undefined instruction ..... 91
3.9.1.2 INT9 ..... 92
3.9.1.3 INTE (System reserved, only available with DSU) ..... 92
3.9.2 Hardware exceptions (non maskable interrupts) ..... 92
3.9.2.1 HW-INT9 ..... 93
3.9.2.2 NMI ..... 93
3.9.2.3 Tool break (VENMI, system reserved, only available with DSU) ..... 94
3.9.2.4 Instruction break (VEIB, system reserved, only available with DSU) ..... 94

**4. DMA 95**

4.1 Overview ..... 95
4.1.1 DMA compared to interrupt handling: ..... 95
4.1.2 DMA functions ..... 96
4.1.3 Structure ..... 97
4.2 DMA Registers ..... 98
4.2.1 DMA Interrupt Request Select Register (DISEL) ..... 99
4.2.1.1 DMA Interrupt Request Select Register (DISEL) ..... 99
4.2.2 DMA Status Register (DSR) ..... 99
4.2.2.1 DMA Status Register (DSR) ..... 99
4.2.3 DMA Stop Status Register (DSSR) ..... 100
4.2.3.1 DMA Stop Status Register (DSSR) ..... 100
4.2.4 DMA Enable Register (DER) ..... 101
4.2.4.1 DMA Enable Register (DER) ..... 101
4.3 DMA Descriptor ..... 102
4.3.1 DMA IO Address Pointer Bank Select (IOABK) ..... 103
4.3.1.1 DMA IO Address Pointer Bank Select Register (IOABK) ..... 104
4.3.2 Data Count Register (DCT) ..... 104

4.3.2.1 Data Count Register (DCT) ..... 104
4.3.2.2 Transfer type and decrement of DCT ..... 105
4.3.3 I/O Register Address Pointer (IOA) ..... 105
4.3.3.1 I/O Register Address Pointer (IOA) ..... 105
4.3.4 DMA Control Register (DMACS) ..... 105
4.3.4.1 DMA Control Register (DMACS) ..... 105
4.3.5 Buffer Address Pointer (BAP) ..... 107
4.3.5.1 Buffer Address Pointer (BAP) ..... 107
4.4 DMA Controller Operation ..... 108
4.5 Examples of DMA transfers ..... 110

**5. Delayed Interrupt 115**

5.1 Outline of Delayed Interrupt Module ..... 115
5.2 Delayed Interrupt Register ..... 116
5.3 Delayed Interrupt Operation ..... 117

**6. Clocks 119**

6.1 Clocks ..... 119
6.2 Clock Control Registers ..... 122
6.2.1 Clock Selection Register (CKSR) ..... 124
6.2.2 Clock Monitor Register (CKMR) ..... 127
6.2.3 Clock Stabilization Select Register (CKSSR) ..... 129
6.2.4 Clock Frequency Control Register (CKFCR) ..... 132
6.2.5 PLL and clock frequency Control Register (PLLCR) ..... 134
6.2.6 Clock Input and LVD Control Register (CILCR) ..... 137
6.3 Clock Modes ..... 139
6.3.1 Definition of clock modes ..... 139
6.3.2 Clock source switching ..... 139
6.3.3 Activating and Disabling source clocks ..... 140
6.4 Configuration of the PLL ..... 141
6.4.1 Components of the PLL clock multiplier circuit ..... 141
6.4.2 Setting the VCO clock Multiplier Select bits VMS[2:0] ..... 142
6.5 Oscillation Stabilization Wait Time ..... 143
6.5.1 Oscillation Stabilization Wait Interval ..... 143
6.6 Connection of an Oscillator or an External Clock to the Microcontroller ..... 144

**7. Clock Modulator 147**

7.1 Overview ..... 147
7.2 Register Description ..... 148
7.2.1 Clock Modulator Control Register (CMCR) ..... 148
7.2.2 Clock Modulation Parameter Register (CMPR) ..... 151
7.3 Application Note ..... 157

**8. Resets and Startup 159**

8.1 Resets ..... 159
8.1.1 Causes of a reset ..... 159
8.2 Reset, System clock and Stabilization Wait Times ..... 161
8.2.1 Reset causes and stabilization wait times ..... 161
8.2.2 Stabilization wait time in case of an External reset ..... 161
8.2.3 Stabilization of the Main oscillator ..... 162
8.2.4 Stabilization of the Sub oscillator ..... 162
8.3 Startup after Power and External reset ..... 162

8.3.1 Initialization after startup ....162
8.3.2 Reset extension ....162
8.3.3 Source clock timers and clock ready monitor bits ....164
8.3.4 Start of program execution after Power or External reset ....164
8.3.5 Transition to Main clock mode ....164
8.3.6 Transition to PLL clock mode ....165
8.3.7 Transition to Sub clock mode ....165
8.4 Boot ROM program execution and Operation mode and ROM Configuration Block ....165
8.4.1 External bus ....165
8.4.2 Mode pins ....165
8.4.3 Operation modes ....166
8.4.4 ROM Configuration Block (RCB) ....167
8.4.5 Function description of RCB Markers ....168
8.4.5.1 Security and Protection Configuration Block (SPCB) ....168
8.4.5.2 Boot Sequence Configuration Block (BSCB) ....168
8.4.5.3 BDM Configuration Block (BDCB) ....170
8.4.6 Flowchart Internal Vector Mode ....171
8.4.7 Flowchart in External Vector Mode ....172
8.5 Reset Control Registers ....173
8.5.1 Reset Configuration Register (RCR) ....173
8.5.2 Reset Cause and Clock Status Register (RCCSR/RCCSRC) ....176
8.5.2.1 Configuration of the Reset Cause and Clock Status Register (RCCSR/RCCSRC) ....176
8.5.2.2 Notes about reset cause bits ....179
8.5.2.3 Notes about clock missing flags ....179
8.5.3 Clock Input and LVD Control Register (CILCR) ....179
8.6 Operation of the Clock stop detection function and reset ....182
8.6.1 Function of the clock stop detection circuit ....182
8.6.2 Configuration of the clock stop detection circuit ....182
8.6.3 Clock missing flags MCMF and SCMF ....183
8.6.4 Clock stop detection reset ....183
8.6.5 Clock stop detection reset and Standby modes ....184
8.6.6 Clock stop detection reset and Watchdog timer ....184
8.7 Operation of the low voltage reset function ....184
8.7.1 Function of the low voltage reset ....184
8.7.2 Configuration of the low voltage detection and reset circuit ....184

**9. Standby Mode and Voltage Regulator Control Circuit 187**

9.1 Overview of the CPU Operating Modes ....187
9.1.1 CPU operating modes and current consumption ....187
9.1.2 Transitions between different operating modes ....188
9.1.3 Run Mode ....189
9.1.4 Sleep Mode ....190
9.1.5 Timer Mode ....191
9.1.6 Stop Mode ....191
9.2 Standby Mode Control Register (SMCR) ....192
9.3 Voltage Regulator Control Register (VRCR) ....194
9.4 Standby Modes ....198
9.4.1 Sleep mode (RC Sleep, Main Sleep, PLL Sleep, Sub Sleep mode) ....198
9.4.1.1 Functions in Sleep Mode ....199
9.4.1.2 Switching to Sleep Mode ....199
9.4.1.3 Release of Sleep Mode ....199
9.4.2 Timer Mode (RC Timer, Main Timer, PLL Timer, Sub Timer mode) ....201

9.4.2.1 Functions in Timer Mode ..... 201
9.4.2.2 Switching to Timer Mode ..... 201
9.4.2.3 Release of Timer Mode ..... 202
9.4.3 Stop Mode ..... 203
9.4.3.1 Functions in Stop Mode ..... 203
9.4.3.2 Switching to Stop Mode ..... 203
9.4.3.3 Release of Stop Mode ..... 204
9.5 Mode Change Table and operation status ..... 205
9.6 Usage Notes on Standby Mode ..... 206
9.7 Voltage Regulator Operation ..... 207
9.7.1 Changing the Voltage Regulator operation mode ..... 208
9.7.1.1 Overview ..... 208
9.7.1.2 Setting Voltage regulator to Low Power mode by user program ..... 208
9.7.1.3 Permitted configurations for using the Low Power mode of the voltage regulator ..... 209
9.7.2 Setting the Voltage Regulator output voltage ..... 210
9.7.2.1 Voltage setting in High Power mode ..... 210
9.7.2.2 Voltage setting in Low Power modes ..... 210

**10. Source Clock Timers 211**

10.1 Overview ..... 211
10.2 RC Clock Timer ..... 211
10.2.1 RC Clock Timer Control Register (RCTCR) ..... 212
10.2.2 Operations of RC Clock Timer ..... 215
10.3 Main Clock Timer ..... 215
10.3.1 Main Clock Timer Control Register (MCTCR) ..... 216
10.3.2 Operations of Main Clock Timer ..... 219
10.4 Sub Clock Timer ..... 219
10.4.1 Sub Clock Timer Control Register (SCTCR) ..... 220
10.4.2 Operations of Sub Clock Timer ..... 222

**11. Watchdog Timer and Watchdog Reset 225**

11.1 Outline of Watchdog Timer and Reset ..... 225
11.2 Watchdog Timer Control Registers ..... 226
11.2.1 Watchdog Timer Configuration register (WDTC) ..... 227
11.2.2 Watchdog Timer Clear Pattern register (WDTCP) ..... 230
11.3 Watchdog Timer Operation ..... 231
11.3.1 Activation and Deactivation of Watchdog Timer ..... 231
11.3.2 Clearing the Watchdog counter ..... 231
11.3.3 Stopping the Watchdog counter ..... 231
11.3.4 Selecting the clock source for the Watchdog Counter ..... 231
11.3.5 Standby modes ..... 232
11.3.6 Disabling the source oscillator ..... 232
11.3.7 Setting the Watchdog Timer interval ..... 232
11.3.8 Selecting the operation mode of the Watchdog Timer ..... 232
11.3.9 Watchdog Timer reset causes ..... 233

**12. External Bus Interface 235**

12.1 Outline of External Bus ..... 235
12.1.1 Features of the External Bus Interface ..... 235
12.1.2 Terminology ..... 235
12.1.3 External bus areas ..... 236

12.1.4 Memory Space in Each Bus Mode ..... 237
12.2 External bus configuration registers ..... 238
12.2.1 External Bus Mode registers (EBM) ..... 242
12.2.2 External Bus Clock and Function register (EBCF) ..... 243
12.2.3 External Bus Address output Enable registers (EBAE[2:0]) ..... 247
12.2.4 External Bus Control Signal register (EBCS) ..... 248
12.2.5 External Area Configuration registers (EACH/EACL[5:0]) ..... 250
12.2.6 External Area Select register (EAS[5:2]) ..... 256
12.3 External Memory Access Control Signal Operation ..... 259
12.3.1 Multiplexed external bus mode ..... 259
12.3.2 Non-multiplexed external bus mode ..... 267
12.3.3 Ready Function ..... 274
12.3.3.1 Ready Function ..... 274
12.3.4 Hold Function ..... 277
12.3.4.1 Hold Function ..... 277
12.3.4.2 Hold request during Standby-Mode ..... 277
12.3.4.3 Watchdog timer ..... 277
12.3.4.4 Timing diagram ..... 278
12.4 Notes on using the external bus ..... 279
12.5 External Boot Vector fetch ..... 282
12.6 Pin status in different MCU states ..... 284
12.6.1 Status of external bus pins ..... 284

**13. I/O Ports 287**

13.1 I/O Ports ..... 287
13.2 I/O Port Registers ..... 288
13.2.1 Port Data Register (PDRnn) ..... 289
13.2.1.1 Port Data Register ..... 290
13.2.1.2 Reading the Port Data Register ..... 290
13.2.2 External Pin State Register (EPSRnn) ..... 291
13.2.2.1 External Pin State Register ..... 291
13.2.2.2 Access to the external pin state data register ..... 292
13.2.3 Data Direction Register (DDRnn) ..... 292
13.2.3.1 Data direction register ..... 292
13.2.3.2 Reading the Data Direction Register ..... 292
13.2.4 Port Input Enable Register (PIERnn) ..... 292
13.2.4.1 Port Input Enable Register (PIER) ..... 293
13.2.5 Port Input Level Register (PILRnn) and Extended Port Input Level Register (EPILRnn) ..... 293
13.2.5.1 Port Input Level Register (PILRnn) and Extended Port Input Level Register(EPILRnn) ..... 294
13.2.6 Port Output Drive Register (PODRnn) ..... 295
13.2.6.1 Port Output Drive Register (PODRnn) ..... 295
13.2.7 Port High Drive Register (PHDR) ..... 296
13.2.7.1 Port High Drive Register ..... 296
13.2.8 Pull-Up Control Register (PUCR) ..... 297
13.2.8.1 Pull-Up Control Register (PUCR) ..... 297
13.2.8.2 Block Diagram of Pull-Up Control Register (PUCR) ..... 297
13.3 Register usage ..... 298

**14. 16-Bit I/O Timer 299**

14.1 Outline of 16-bit I/O Timer ..... 299

14.2 16-Bit I/O Timer Registers ..... 301
14.3 16-bit Free-Running Timer ..... 302
14.3.1 Data Register (TCDTn) ..... 303
14.3.2 Control Status Register (TCCSLn) ..... 304
14.3.3 16-bit Free-Running Timer Operation ..... 307
14.3.3.1 16-bit Free-Running Timer operation ..... 307
14.3.3.2 Clearing the counter by an overflow ..... 307
14.3.3.3 Clearing the counter at Compare Clear Register value match ..... 307
14.3.3.4 16-bit Free-Running Timer timing ..... 308
14.4 Output Compare Unit ..... 308
14.4.1 Output Compare Register (OCCP(2n) / OCCP(2n+1)) ..... 309
14.4.2 Control Status Registers of Output Compare (OCS(2n) / OCS(2n+1)) ..... 310
14.4.3 16-bit Output Compare Operation ..... 314
14.4.3.1 Sample output waveform when CMOD[1:0] = "00B" ..... 314
14.4.3.2 Sample output waveform with two compare registers when CMOD[1:0] = "01B" ..... 314
14.4.3.3 Sample output waveform when CMOD[1:0] = "10B" ..... 315
14.4.3.4 Sample output waveform when CMOD[1:0] = "11B" ..... 316
14.4.3.5 Output compare timing ..... 317
14.5 Input Capture Unit ..... 318
14.5.1 Input Capture Unit Register Details ..... 319
14.5.2 16-bit Input Capture Operation ..... 323
14.5.2.1 Example of input capture fetch timing ..... 323
14.5.2.2 Input Capture input timing ..... 324

**15. 16-Bit Reload Timer (With Event Count Function) 325**

15.1 Outline of 16-Bit Reload Timer (with Event Count Function) ..... 325
15.2 16-Bit Reload Timer (with Event Count Function) ..... 327
15.2.1 Timer Control Status Register (TMCSRn) ..... 328
15.2.1.1 Register layout of Timer Control Status Register (TMCSRn) ..... 328
15.2.1.2 Register contents of Timer Control Status Register (TMCSRn) ..... 328
15.2.2 Register Layout of 16-bit Timer Register (TMRn)/16-bit Reload Register (TMRLRn) ..... 330
15.3 Internal Clock and External Event Counter Operations of 16-bit Reload Timer ..... 331
15.4 Underflow Operation of 16-bit Reload Timer ..... 333
15.5 Output Pin Functions of 16-bit Reload Timer ..... 333
15.6 Counter Operation State ..... 335
15.7 Cascading of 16-bit Reload Timers ..... 336

**16. Programmable Pulse Generator 339**

16.1 Outline of Programmable Pulse Generator ..... 339
16.2 Registers ..... 342
16.2.1 Register list ..... 342
16.2.2 PPG Control Status register (PCNn) ..... 343
16.2.3 General Control Register 1 (GCN1g) ..... 345
16.2.4 General Control Register 2 (GCN2g) ..... 346
16.2.5 PPG Cycle Setting Register (PCSRn) ..... 347
16.2.6 PPG Duty Setting Register (PDUTn) ..... 348
16.2.7 PPG Timer Register (PTMRn) ..... 348
16.3 Operation of Programmable Pulse Generator ..... 349
16.3.1 PWM Operation ..... 349
16.3.2 One-Shot Operation ..... 350

16.3.3 Restart Operation....351
16.4 Cautions....352

**17. External Interrupts 353**

17.1 Outline of External Interrupts....353
17.2 External Interrupt Registers....355
17.3 Notes on using the External Interrupt functions....356

**18. A/D Converter 359**

18.1 Outline of A/D Converter....359
18.2 Registers for A/D Converter....360
18.2.1 Control status register (ADCS)....362
18.2.1.1 Control Status Register (ADCS)....362
18.2.1.2 Control Status Register (ADCSH)....362
18.2.1.3 Control Status Register (ADCSL)....364
18.2.2 Data Register (ADCR)....365
18.2.2.1 Data register (ADCR)....365
18.2.3 Setting Register (ADSR)....365
18.2.3.1 Setting Register (ADSR)....366
18.2.4 Extended Configuration Register (ADECR)....368
18.2.4.1 ADC Extended Configuration Register (ADECR)....369
18.2.5 Analog Input Enable Register (ADERx)....372
18.2.5.1 Analog input enable register (ADERx)....372
18.3 Operation of A/D Converter....372
18.3.1 Single mode 1 / 2....372
18.3.2 Continuous mode....373
18.3.3 Stop mode....373
18.4 Conversion using DMA....373
18.4.1 Conversion using DMA....374
18.5 Conversion Data Protection Function....375
18.5.1 Explanation of A/D Conversion Data Protection Function....375
18.5.2 Data protection function for the case A/D conversion result is read by CPU....375
18.5.3 Data protection function for the case A/D conversion result is transferred by DMA....376
18.5.4 Example of flow of Conversion Data Protection Function (when DMA is used)....377
18.5.5 Cautions....377

**19. Alarm Comparator 379**

19.1 Outline of Alarm Comparator....379
19.2 Alarm Comparator Registers....380
19.2.1 Alarm Comparator Control/Status register (ACSRn)....381
19.2.2 Alarm Comparator Extended Control/Status Register (AECSRn)....382
19.3 Alarm Comparator operating modes....383

**20. USART 385**

20.1 Overview of USART....385
20.2 Configuration of USART....387
20.2.1 Block diagram of USART....387
20.3 USART Pins....391
20.4 USART Registers....391
20.4.1 Serial Control Register (SCRn)....391
20.4.2 Serial mode register (SMRn)....394

20.4.3 Serial Status Register (SSRn) ..... 396
20.4.4 Reception and Transmission Data Register (RDRn/TDRn) ..... 398
20.4.4.1 Reception and transmission data registers (RDRn/TDRn) ..... 398
20.4.5 Extended Status/Control Register (ESCRn) ..... 399
20.4.6 Extended Communication Control Register (ECCRn) ..... 401
20.4.7 Baud Rate/Reload Counter Register (BGRn) ..... 403
20.4.8 Extended Serial Interrupt Register (ESIRn) ..... 404
20.5 USART Interrupts ..... 406
20.5.1 LIN-USART interrupts ..... 406
20.5.2 USART DMA functions ..... 407
20.5.3 Receive interrupt Generation and Flag Set Timing ..... 408
20.5.4 Transmission Interrupt Generation and Flag Set Timing ..... 409
20.6 USART Baud Rates ..... 410
20.6.1 USART baud rate selection ..... 410
20.6.2 Setting the Baud Rate ..... 411
20.6.2.1 Calculating the baud rate ..... 411
20.6.2.2 Suggested division ratios for different peripheral clock CLKP1 frequencies and baud rates ..... 412
20.6.2.3 Using external clock ..... 412
20.6.2.4 Counting example ..... 413
20.6.3 Restarting the Reload Counter ..... 413
20.6.3.1 Programmable restart ..... 413
20.7 Operation of USART ..... 415
20.7.1 Operation in Asynchronous Mode (Op. Modes 0 and 1) ..... 416
20.7.1.1 Operation in asynchronous mode ..... 416
20.7.2 Operation in Synchronous Mode (Operation Mode 2) ..... 417
20.7.2.1 Operation in synchronous mode (operation mode 2) ..... 417
20.7.2.2 Clock inversion and start/stop bits in mode 2 ..... 418
20.7.2.3 Clock supply: ..... 418
20.7.2.4 Error detection: ..... 419
20.7.2.5 Communication: ..... 419
20.7.3 Operation with LIN Function (Operation Mode 3) ..... 420
20.7.3.1 Operation in asynchronous LIN mode (operation mode 3) ..... 420
20.7.4 Direct Access to Serial Pins ..... 423
20.7.5 Bidirectional Communication Function (Normal Mode) ..... 424
20.7.5.1 Bidirectional communication function ..... 424
20.7.6 Master-Slave Communication Function (Multiprocessor Mode) ..... 426
20.7.6.1 Master-slave communication function ..... 426
20.7.7 LIN Communication Function ..... 429
20.7.7.1 LIN-master-slave communication function ..... 429
20.7.8 Sample Flowcharts for USART in LIN communication (Operation Mode 3) ..... 430
20.7.8.1 USART as master device ..... 430
20.7.8.2 USART as slave device ..... 431
20.8 Notes on Using USART ..... 432
20.8.1 Notes on using USART ..... 432

**21. 400 kHz I2C Interface 437**

21.1 I2C Interface Overview ..... 437
21.2 I2C Interface Registers ..... 439
21.2.1 Bus Status Register (IBSRn) ..... 440
21.2.2 Bus Control Register (IBCRn) ..... 443
21.2.2.1 Bus control register (IBCRn) ..... 443
21.2.2.2 Bus control register (IBCRn) contents ..... 445

21.2.2.3 SCC, MSS and INT bit competition ....446
21.2.3 Ten Bit Slave Address Register (ITBAn) ....449
21.2.4 Ten Bit Slave Address Mask Register (ITMKn) ....450
21.2.5 Seven Bit Slave Address Register (ISBAn) ....451
21.2.6 Seven Bit Slave Address Mask Register (ISMKn) ....452
21.2.7 Data Register (IDARn) ....452
21.2.8 Clock Control Register (ICCRn) ....453
21.3 I2C Interface Operation ....455
21.4 Programming Flow Charts ....458

**22. CAN Controller 461**

22.1 Overview ....461
22.2 Functional Description ....462
22.3 Register Description ....465
22.4 CAN Device Related Registers ....468
22.4.1 CAN Output Enable Register (COERn) ....468
22.5 CAN Protocol Related Registers ....468
22.5.1 CAN Control Register (CTRLRn) ....468
22.5.2 Status Register (STATRn) ....470
22.5.3 Error Counter (ERRCNTn) ....471
22.5.4 Bit Timing Register (BTRn) ....472
22.5.5 Test Register (TESTRn) ....473
22.5.6 BRP Extension Register (BRPERn) ....474
22.6 Message Interface Register Sets ....474
22.6.1 IFx Command Request Registers (IFxCREQn) ....475
22.6.2 IFx Command Mask Register (IFxCMSKn) ....476
22.6.3 IFx Mask Registers (IFxMSK1n, IFxMSK2n) ....478
22.6.4 IFx Arbitration Registers (IFxARB1n, IFxARB2n) ....479
22.6.5 IFx Message Control Register (IFxMCTRn) ....479
22.6.6 IFx Data A and Data B Registers (IFxDTA1n, IFxDTA2n, IFxDTB1n, IFxDTB2n) ....480
22.7 Message Object in the Message Memory ....480
22.8 Message Handler Registers ....483
22.8.1 Interrupt Register (INTRn) ....483
22.8.2 Transmission Request Registers (TREQR1n, TREQR2n) ....484
22.8.3 New Data Registers (NEWDT1n, NEWDT2n) ....486
22.8.4 Interrupt Pending Registers (INTPND1n, INTPND2n) ....487
22.8.5 Message Valid Registers (MSGVAL1n, MSGVAL2n) ....488
22.9 CAN Application ....489

**23. Clock Output Function 497**

23.1 Overview of the Clock Output Function ....497
23.2 Clock Output Configuration and Activation Registers ....497
23.3 Operation of the Clock Output Function ....502

**24. Real Time Clock 503**

24.1 Outline of Real Time Clock ....503
24.2 Real Time Clock Registers ....505
24.2.1 Timer Control Registers ....506
24.2.1.1 Timer Control Register (WTCR) ....506
24.2.2 Timer Control Extended Register (WTCER) ....509
24.2.3 Clock Select Register (WTCKSR) ....510
24.2.4 Sub-Second Register ....510

24.2.4.1 Sub-Second Register (WTBR) ..... 511
24.2.5 Second/Minute/Hour Registers (WTSR, WTMR, WTHR) ..... 511
24.2.5.1 Second/minute/hour registers ..... 512
24.3 Operation ..... 513
24.4 Cautions ..... 514
24.4.1 Cautions ..... 514

**25. Clock Calibration Unit 517**

25.1 Outline ..... 517
25.2 Register Description ..... 520
25.2.1 Clock Calibration Unit registers ..... 520
25.2.2 Clock Calibration Unit Control Register (CUCR) ..... 520
25.2.3 Clock Calibration Unit Duration Timer Data Register (CUTD) ..... 521
25.2.4 Clock Calibration Unit Calibration Timer Data Register (24 bits) (CUTR) ..... 522
25.3 Application Notes ..... 523
25.3.1 32 kHz ..... 523
25.3.2 100 kHz ..... 524
25.3.3 Accuracy ..... 524
25.3.4 Power Dissipation ..... 524
25.3.5 Measurement Limits ..... 525

**26. LCD Controller/Driver 527**

26.1 Configuration of LCD Controller/Driver ..... 527
26.1.1 LCD Controller/Driver’s divide resistors ..... 530
26.2 LCD Controller Registers ..... 532
26.2.1 LCD Control register (LCR) ..... 532
26.2.2 LCD Common Pin switching Register (LCDCMR) ..... 534
26.2.3 LCD Extended Control Register (LECR) ..... 535
26.3 LCD Enable Registers (LCDER, LCDVER) ..... 536
26.4 LCD Controller/Driver Display RAM ..... 537
26.4.1 Memory area (VRAM) for setting display data ..... 537
26.4.2 Correspondence between VRAM and Common/Segment Pins ..... 538
26.5 Operation of LCD Controller/Driver ..... 538
26.5.1 LCD Controller/Driver (LCDC) Operation ..... 538
26.5.2 Output waveform during LCD Controller/Driver operation (1/2 duty cycle) ..... 539
26.5.3 Output waveform during LCD controller/driver operation (1/3 duty cycle) ..... 541
26.5.4 Output waveform during LCD controller/driver operation (1/4 duty cycle) ..... 543
26.6 Cautions ..... 545

**27. Stepper Motor Controller 547**

27.1 Outline of Stepper Motor Controller ..... 547
27.2 Stepper Motor Controller Registers ..... 548
27.3 PWM Control register (PWCn) ..... 550
27.3.1 PWM Control register (PWCn) ..... 550
27.4 PWM Extended Control register (PWECn) ..... 551
27.4.1 PWM Extended Control register (PWECn) ..... 551
27.5 PWM1 and PWM2 Compare Registers (PWC1n, PWC2n) ..... 551
27.5.1 PWM1 and PWM2 Compare registers (PWC1n, PWC2n) ..... 551
27.6 PWM1 and PWM2 Selection registers (PWS1n, PWS2n) ..... 554
27.6.1 PWM1 and PWM2 Selection registers (PWS1n, PWS2n) ..... 554
27.6.2 PWM1 Selection register (PWS1n) ..... 554
27.6.3 PWM2 Selection register (PWS2n) ..... 554

27.7 Operation of Stepper Motor Controller....555
27.8 Notes on Using Stepper Motor Controller....557

**28. Sound Generator 559**

28.1 Outline of Sound Generator....559
28.2 Sound Generator Registers....561
28.2.1 Sound Generator Control Register (SGCRn)....561
28.2.2 Amplitude Data Register (SGARn)....564
28.2.3 Frequency Data Register (SGFRn)....566
28.2.4 Tone Count Register (SGTRn)....566
28.2.5 Decrement Grade Register (SGDRn)....567

**29. USB Function 569**

29.1 USB Function Overview....569
29.2 USB Function Block Diagram....569
29.3 USB Function Register Description....571
29.3.1 UDC Control Register (UDCCn)....575
29.3.2 EP0 Control Register (EP0Cn)....577
29.3.3 EP1 to EP5 Control Register (EP1Cn to EP5Cn)....578
29.3.4 Time Stamp Register (TMSPn)....582
29.3.5 UDC Status register (UDCSn)....583
29.3.6 UDC Interruption Enable Register (UDCIEn)....585
29.3.7 EP0I Status Register (EP0ISn)....586
29.3.8 EP0O Status Register (EP0OSn)....588
29.3.9 EP1 to EP5 Status Register (EP1Sn to EP5Sn)....590
29.3.10EP0 to EP5 Data Register (EP0DTn to EP5DTn)....594
29.4 Description of the USB function operation....595
29.4.1 USB Function operation....595
29.4.2 Detecting Connection and Disconnection....597
29.4.2.1 USB connection example....597
29.4.3 Each Register Operation with Command Response....599
29.4.3.1 Read Command USB function operation....599
29.4.3.2 Write Command USB function operation....600
29.4.4 Suspend Mode Function....601
29.4.4.1 USB function SUSPEND mode....601
29.4.5 Wake-up Function....602
29.4.6 DMA Transfer Function....602
29.4.6.1 Packet Transfer Mode....603
29.4.6.2 Data Number automatic Transfer Mode....604
29.4.7 NULL Transfer Function....606
29.4.7.1 NULL Transfer Mode....606

**30. USB Mini-Host 607**

30.1 USB Mini-host features....607
30.2 Differences in USB Mini-host....608
30.3 USB Mini-host Block Diagram....609
30.4 USB Mini-host Registers....610
30.4.1 Host Control Register (HCNTn)....611
30.4.2 Host Interrupt Register (HIRQn)....614
30.4.3 Host Error Status Register (HERRn)....616
30.4.4 Host State Status Register (HSTATEn)....618
30.4.5 SOF Interruption FRAME Comparison Register (HFCOMPn)....620

30.4.6 Retry Timer Setting Registers (HRTIMERLn, HRTIMERMn, HRTIMERHn) ..... 621
30.4.7 Host Address Register (HADRn) ..... 622
30.4.8 EOF Setting Register (HEOFn) ..... 623
30.4.9 FRAME Setting Register (HFRAMEn) ..... 624
30.4.10 Host Token Endpoint Register (HTOKENn) ..... 625
30.5 USB Mini-host operation ..... 626
30.5.1 Device Connection ..... 626
30.5.2 USB Bus Reset ..... 627
30.5.3 Token Packet ..... 628
30.5.4 Data Packet ..... 629
30.5.4.1 Data Packet ..... 630
30.5.5 Handshake Packet ..... 630
30.5.6 Retry Function ..... 630
30.5.7 SOF Interrupt ..... 631
30.5.8 Error Status ..... 633
30.5.8.1 Error Status ..... 633
30.5.9 Packet End ..... 634
30.5.10 Suspend Resume ..... 635
30.5.11 Device disconnection ..... 637
30.6 Token Flow Chart ..... 638

**31. Memory Patch Function 641**

31.1 Outline of the Memory Patch Function ..... 641
31.2 Registers of the Memory Patch Function ..... 643
31.3 Operation of the Memory Patch Function ..... 652

**32. ROM/RAM Mirroring Module 655**

32.1 Outline of ROM/RAM Mirroring Module ..... 655
32.2 ROM Mirroring Register (ROMM) ..... 656

**33. Flash Memory 659**

33.1 Overview of the Flash Memory ..... 659
33.1.1 Flash memory types ..... 659
33.1.2 Flash memory features ..... 660
33.1.3 Writing to/erasing flash memory ..... 660
33.2 Block Diagram of the Flash Memory and Sector Configuration of the Flash Memory ..... 661
33.3 Flash Memory Modes ..... 664
33.4 Program/Data Flash (Main Flash) Memory Control Registers ..... 664
33.4.1 Memory Control Status Register (MCSRA, MCSRB) ..... 664
33.4.2 Memory Timing Configuration Register (MTCRA, MTCRB) ..... 667
33.4.3 Flash Memory Write Control registers 0-5 (FMWC0-FMWC5) ..... 672
33.5 Data Flash Memory Control Registers ..... 673
33.5.1 Data Flash Control Status register (DFCSA, DFCSB) ..... 673
33.5.2 Data Flash Write command sequencer Control register (DFWCA, DFWCB) ..... 676
33.5.3 Data Flash Write command sequencer Status register (DFWSA, DFWSB) ..... 678
33.6 Starting the Flash Memory Automatic Algorithm ..... 680
33.7 Confirming the Automatic Algorithm Execution State ..... 682
33.7.1 Data Polling Flag (DQ7) ..... 683
33.7.1.1 Data polling flag (DQ7) ..... 683
33.7.2 Toggle Bit Flag (DQ6) ..... 684
33.7.2.1 Toggle bit flag (DQ6) ..... 684
33.7.3 Timing Limit Exceeded Flag (DQ5) ..... 685

33.7.3.1 Timing limit exceeded flag (DQ5) ....685
33.7.4 Sector Erase Timer Flag (DQ3) ....686
33.7.4.1 Sector erase timer Flag (DQ3) ....686
33.7.5 Toggle Bit-2 Flag (DQ2) ....686
33.7.5.1 Toggle bit-2 flag (DQ2) ....686
33.8 Detailed Explanation of Writing to and Erasing Flash Memory ....687
33.8.1 Setting The Read/Reset State ....688
33.8.2 Writing Data by submitting the write command sequence ....688
33.8.2.1 Starting the write automatic algorithm ....688
33.8.2.2 Fast write mode ....689
33.8.2.3 Example for writing to the flash memory ....689
33.8.3 Writing to the Data Flash by using the Write command sequencer ....691
33.8.3.1 Using the Write command sequencer ....691
33.8.3.2 Interrupt functions of the Write command sequencer and DMA access ....691
33.8.4 Erasing All Data (Chip Erase) ....692
33.8.5 Erasing Optional Data (Sector erase) ....692
33.8.5.1 Starting the Sector erase automatic algorithm ....692
33.8.5.2 Example for erasing sectors in the flash memory ....693
33.8.6 Suspending Sector Erase ....695
33.8.6.1 Suspending erasing of flash memory sectors ....695
33.8.7 Restarting Sector Erase ....695
33.8.8 Protecting sectors from being erased/written to ....695
33.8.8.1 Function of the sector write protection ....696
33.8.8.2 Activating the write protection by user program ....696
33.8.8.3 Activating the write protection without user program interaction ....696
33.9 Notes on using Flash Memory ....699
33.9.1 Notes on using flash memory ....699
33.10 Flash memory programming example ....700

**34. Mask-ROM Memory 707**

34.1 Overview ....707
34.2 Mask-ROM Memory Control Registers ....707

**35. ROM/Flash Security 713**

35.1 Overview of the ROM/Flash Security ....713
35.2 Usage of the ROM/Flash Security ....714

**36. Examples of Serial Programming Connection 717**

36.1 Basic Configuration of Serial Programming Connection ....717
36.2 Example of connecting a PC for programming the Flash Microcontroller ....719
36.3 Example of connecting a programming tool for programming the Flash Microcontroller ....720

**37. Appendix 721**

37.1 I/O Map of MB96V300 ....721
37.1.1 I/O maps of MB96V300 ....721
37.2 Instructions ....754
37.2.1 Instruction Types ....754
37.2.2 Addressing ....755
37.2.3 Direct Addressing ....757
37.2.3.1 Direct addressing ....757
37.2.4 Indirect Addressing ....761
37.2.4.1 Indirect addressing ....761

37.2.5 Execution Cycle Count ..... 767
37.2.6 Effective address field ..... 767
37.2.7 How to Read the Instruction List ..... 768
37.2.7.1 Description of instruction presentation items and symbols ..... 769
37.2.8 F$^2$MC-16FX Instruction List ..... 771
37.2.9 Instruction Map ..... 783
37.3 Timing Diagrams in Flash Memory Mode ..... 805
37.4 Revision History ..... 812

**Revision History** **815**

# 1. Overview

The MB96300 Super series is a family member of the F$^2$MC-16FX micro controllers. It consist of many different series of microcontrollers, which are targetting different applications. Programming between all members of the Super series is common.

## 1.1 Features

The feature set of F$^2$MC-16FX family's MB96300 Super series makes it especially well suited for automotive applications.

### 1.1.1 Features

- 16-bit core CPU, up to 56 MHz CPU clock, 17.9 ns instruction cycle time
- 0.18µm CMOS Process Technology
- Optimized instruction set for controller applications (bit, byte, word and long-word data types; 23 different addressing modes; barrel shift; variety of pointers)
- 8-byte instruction execution queue
- Signed multiply (16 bit □ 16 bit) and divide (32 bit/16 bit) instructions available
- Internal voltage regulator supports reduced internal MCU voltage, offering low EMI and low power consumption figures
- Code Security Feature
- Up to 5 FULL-CAN interfaces; conforming to Version 2.0 Part A and Part B, ISO16845 certified
- Powerful interrupt functions (8 progr. priority levels; up to 16 external interrupts)
- Fast Interrupt processing
- Up to 16 channels DMA - Automatic transfer function independent of CPU, can be assigned freely to resources
- Three independent clock timers (23-bit RC clock timer, 23-bit Main clock timer, 17-bit Sub clock timer)
- Watchdog Timer
- Up to 10 channels full duplex USARTs (SCI/LIN)
- Up to 2 channels I2C with 400 kbit/s
- USB function (correspond to USB Full Speed) and USB Mini-Host with up to 6 endpoints
- Up to 40 channels analog inputs for A/D Converter (Resolution 10 bits or 8 bits)
- Up to 6 channels16-bit reload timer
- Up to 12 channels ICU (Input capture unit) 16 bit
- Up to 12 channels OCU (Output compare unit) 16 bit
- Up to 4 channels 16-bit free running timer
- Up to 20 channels □ 16-bit Programmable Pulse Generator
- Up to 6 channels Stepper Motor Controller with integrated high current output drivers
- LCD controller with up to 4 COM □ 72 SEG, internal or external voltage generation
- Memory Patch Function, can also be used to implement embedded debug support
- Low Power Consumption - 13 operating modes (different Run, Sleep, Timer modes, Stop mode)
- 3-16 MHz external clock (3.5-56 MHz when using Fast Clock Input mode), on-chip PLL with programmable multiplication factor 1 ... 16

- 32 kHz Subsystem Clock
- 100kHz/2MHz internal RC clock
- External bus interface with up to 6 Chip select signals, 8bit or 16bit data, 24bit address, multiplexed or non-multiplexed, programmable timing
- Up to 2 channels Alarm comparators
- Programmable input levels (Automotive / CMOS-Schmitt trigger / TTL) for all ports
- Programmable Pull-up resistors for all ports
- Programmable output driving strength for EMI optimization
- Package : 48-pin / 64-pin / 80-pin / 100-pin / 120-pin / 144-pin plastic QFP and LQFP
- Controller Area Network (CAN) - License of Robert Bosch GmbH

## 1.2 Super Series Lineup

Table 1-1 provides an overview of the MB96300 Super series.

### 1.2.1 Super Series Lineup

The F$^2$MC-16FX MB96300 Super series covers the F$^2$MC-16FX series as shown in table 1.2-1.

Table 1-1. MB96300 Super Series Lineup

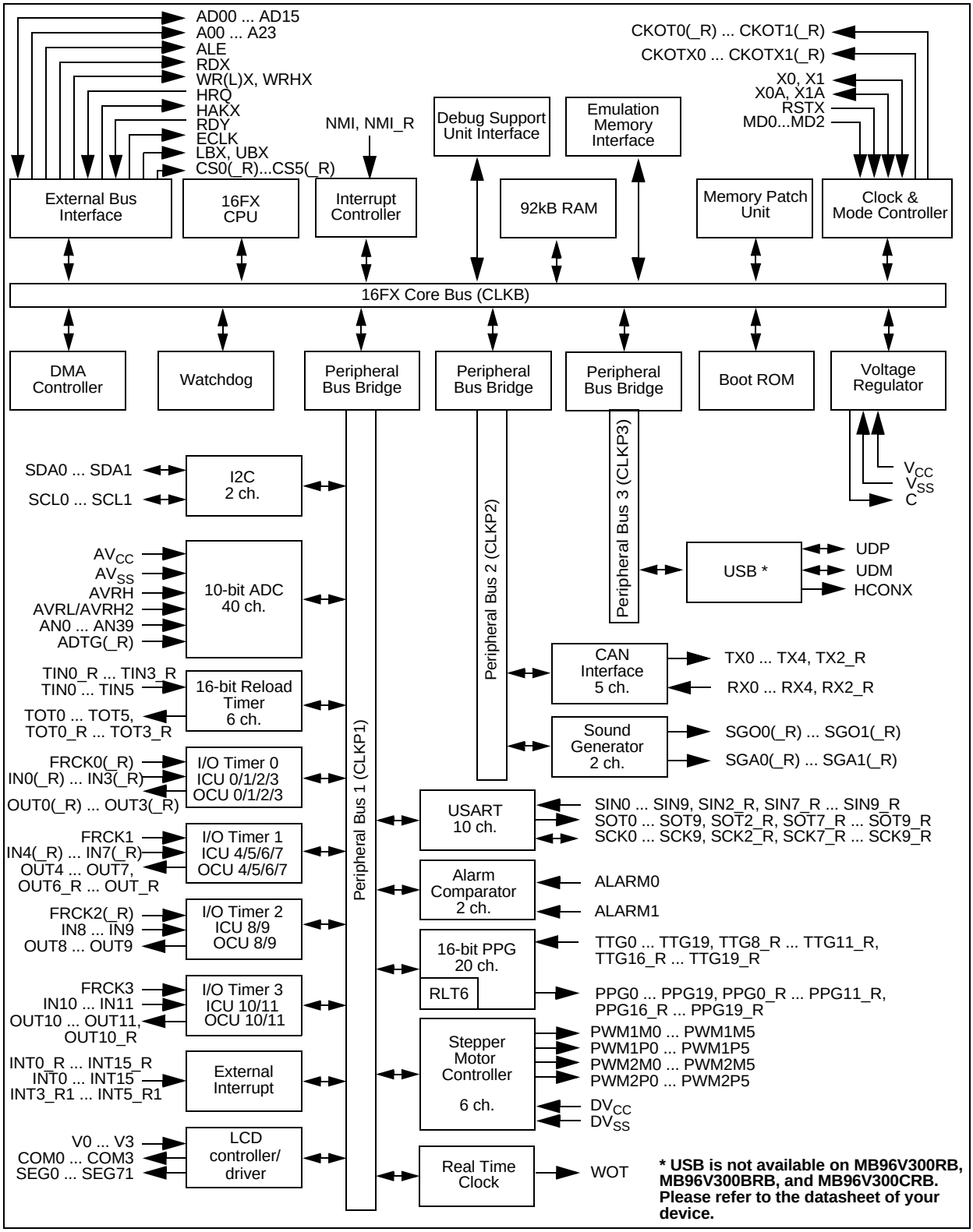
| Series | Flash Products | Mask ROM Products |
|---|---|---|
| MB96310 | MB96F31x | MB9631x |
| MB96320 | MB96F32x | MB9632x |
| MB96330 | MB96F33x | MB9633x |
| MB96340 | MB96F34x | MB9634x |
| MB96350 | MB96F35x | MB9635x |
| MB96370 | MB96F37x | MB9637x |
| MB96380 | MB96F38x | MB9638x |
| MB96390 | MB96F39x | MB9639x |

## 1.3 Block Diagram of MB96V300B

Figure 1-1 shows the block diagram of MB96V300B.

## 1.3.1 Block Diagram of MB96V300B

Figure 1-1. Block Diagram of MB96V300B



## 1.4 General Note on Using This Document

This chapter contains some general notes about this document.

### 1.4.1 Device Dependent Resources

The derivatives of a Microcontroller series may have different sets of resources (e.g. features, number of channels, interconnections). Details about an individual device can be found in the following documents:

- Number of resources and available channels: Datasheet of the series.
- Peculiarities of a resource in a derivative (e.g. interconnections between resources): Chapter "Overview" in the Hardware manual of the series.

### 1.4.2 Described Resources

All chapters about resources of the Microcontroller like USART, Reload timers, General purpose ports describe the function of only one channel of this resource. The behavior of all other resources of the same type is the same. If there is an exception it is described in the Chapter "Overview" or in the corresponding chapter of this Hardware manual.

In the Hardware manual chapters the placeholder ’n’ is used for a resource channel number (e.g. in the register name). This placeholder needs to be replaced with the resource channel number of the resource to be used.

Table 1-2. Example: Registers of Reload Timer 0 and Reload Timer 1

| Register Name as in the Chapter "Reload Timer" | Register Name Define in Header File for Reload Timer 0 | Register Name Define in Header File for Reload Timer 1 |
|---|---|---|
| TMCSRH**n** | TMCSRH**0** | TMCSRH**1** |
| TMCSRL**n** | TMRL**0** | TMRL**1** |
| TMRH**n** | TMRH**0** | TMRH**1** |
| TMRL**n** | TMRL**0** | TMRL**1** |
| TMRLRH**n** | TMRLRH**0** | TMRLRH**1** |
| TMRLRL**n** | TMRLRL**0** | TMRLRL**1** |

## 1.5 16-bit I/O-Timer Configuration

The number of available Free Running timers, Input Capture Units and Output Compare Units differs between different devices. Multiple Input Capture Units and Output Compare Units are connected to one Free Running timer. This chapter describes the relation between these modules.

### 1.5.1 Connection between Input Capture Units and Free Running Timers

Table 1-3. Connection between Input Capture Units and Free Running Timers

| Free Running Timer | Input Capture Unit |
|---|---|
| FRT 0 | ICU 0/1/2/3 |
| FRT 1 | ICU 4/5/6/7 |
| FRT 2 | ICU 8/9 |
| FRT 3 | ICU 10/11 |

Table 1-3 shows the connection between Input Capture Units and Free Running Timers. For the availability of resource channels please refer to the datasheet of the used device.

### 1.5.2 Connection between Output Compare Units and Free Running Timers

Table 1-4. Connection between Output Compare Units and Free Running Timers

| Free Running Timer | Output Compare Unit |
|---|---|
| FRT 0 | OCU 0/1/2/3 |
| FRT 1 | OCU 4/5/6/7 |
| FRT 2 | OCU 8/9 |
| FRT 3 | OCU 10/11 |

Table 1-4 shows the connection between Output Compare Units and Free Running Timers. For the availability of resource channels please refer to the datasheet of the used device.

### 1.5.3 Relation between Timer Clear Function and OCU, Compare Clear and Free Running Timer

Table 1-5. Connection between OCU, Compare clear and Free running timer for timer clear

| Free running timer | Timer is cleared at match with following compare register | Note |
|---|---|---|
| FRT 0 | OCCP0 | OCCP0 as MB90340 series |
| FRT 1 | OCCP4 | OCCP4 as MB90340 series |
| FRT 2 | OCCP8 | |
| FRT 3 | OCCP10 | |

Compare function of OCCPx is enabled by OCS:CST[1:0] bits.

## 1.6 Input Capture Unit Source Select for LIN-USART

The input source for the Input Capture units can be selected between an external pin (INn) and the LIN-USART Sync Field output. This chapter describes how to select the Input capture input and which LIN-USART can be connected to which ICU.

### 1.6.1 Overview of Connection between LIN-USART and ICU

Table 1-6. Connection between LIN-USARTs and ICUs

| LIN-USART | ICU | Free running timer for ICU | ICU source select bit |
|---|---|---|---|
| USART0 | ICU0 | FRT0 | ICE01:ICUS0 |
| USART1 | ICU1 | | ICE01:ICUS1 |
| USART2 | ICU6 | FRT1 | ICE67:ICUS6 |
| USART3 | ICU7 | | ICE67:ICUS7 |
| USART4 | ICU4 | | ICE45:ICUS4 |
| USART5 | ICU5 | | ICE45:ICUS5 |
| USART6 | ICU8 | FRT2 | ICE89:ICUS8 |
| USART7 | ICU9 | | ICE89:ICUS9 |
| USART8 | ICU10 | FRT3 | ICE1011:ICUS10 |
| USART9 | ICU11 | | ICE1011:ICUS11 |

**Note:** For availabilty of the required USART, Input capture unit and Free running timer channels please refer for the datasheet of your device.

## 1.6.2 Input Capture Unit Source Select

The source selection for ICU is done in the Input capture edge registers ICExy.

Figure 1-2. Input Capture Edge Register ICE01

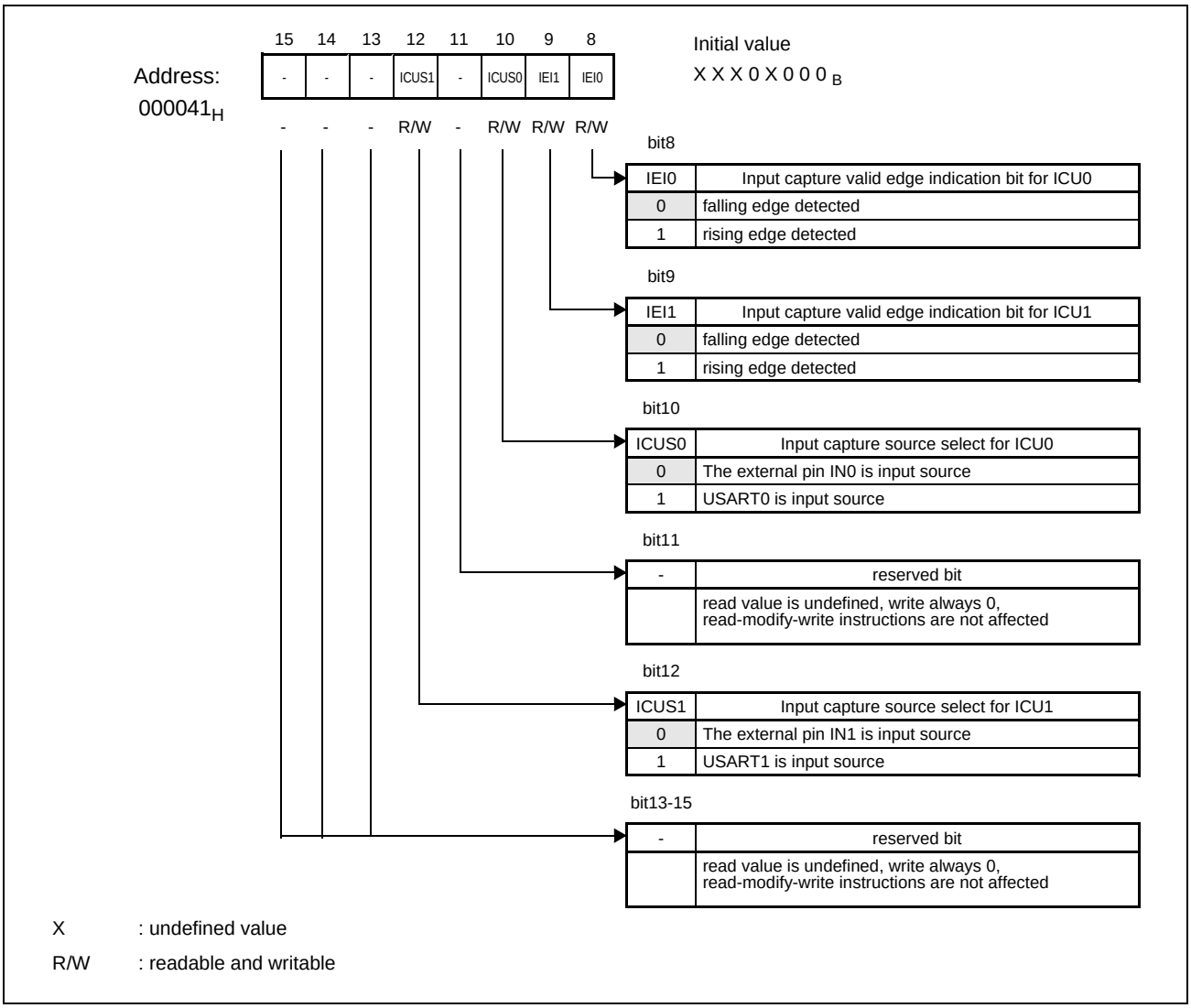


Figure 1-3. Input Capture Edge Register ICE45

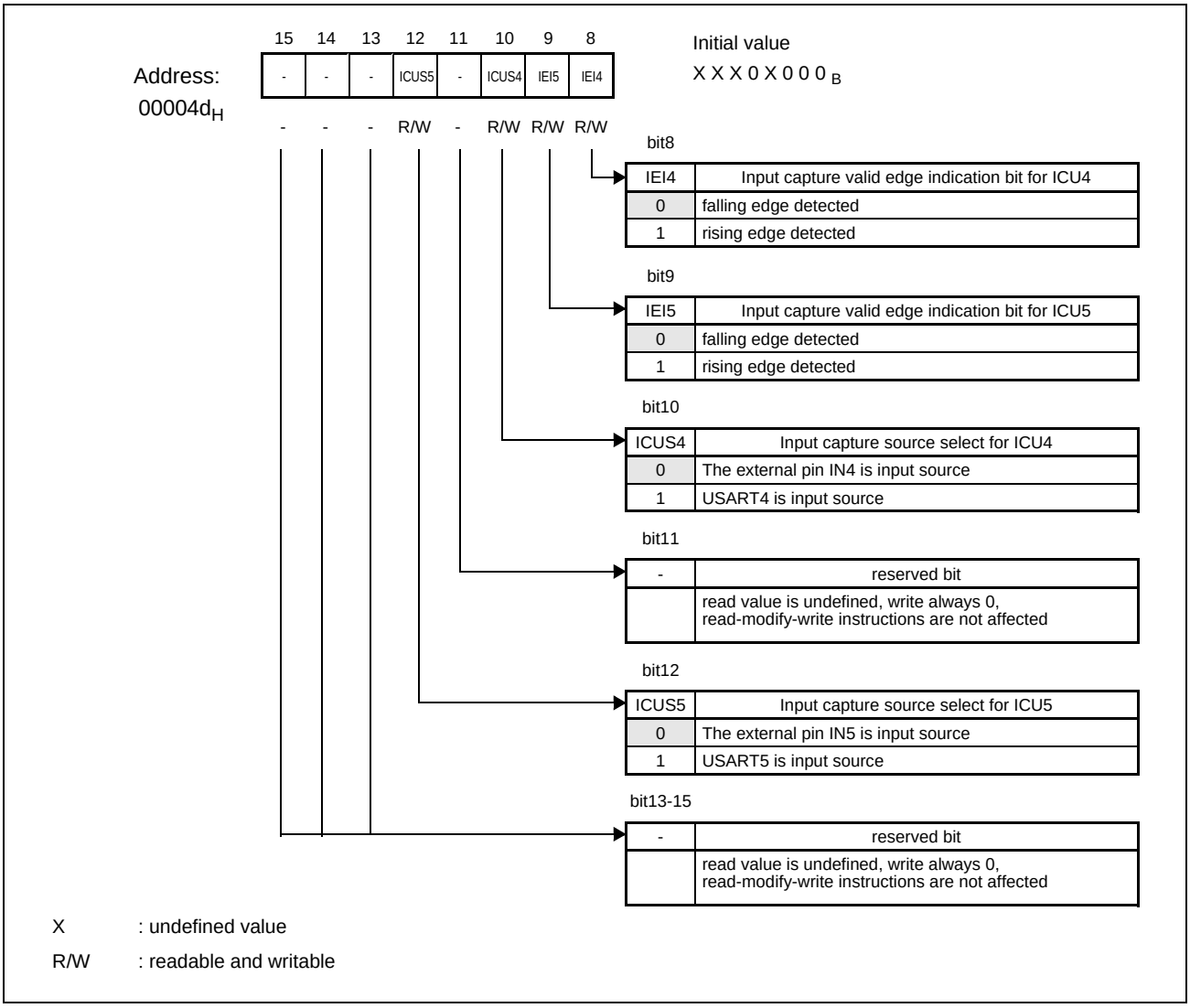


Figure 1-4. Input Capture Edge Register ICE67

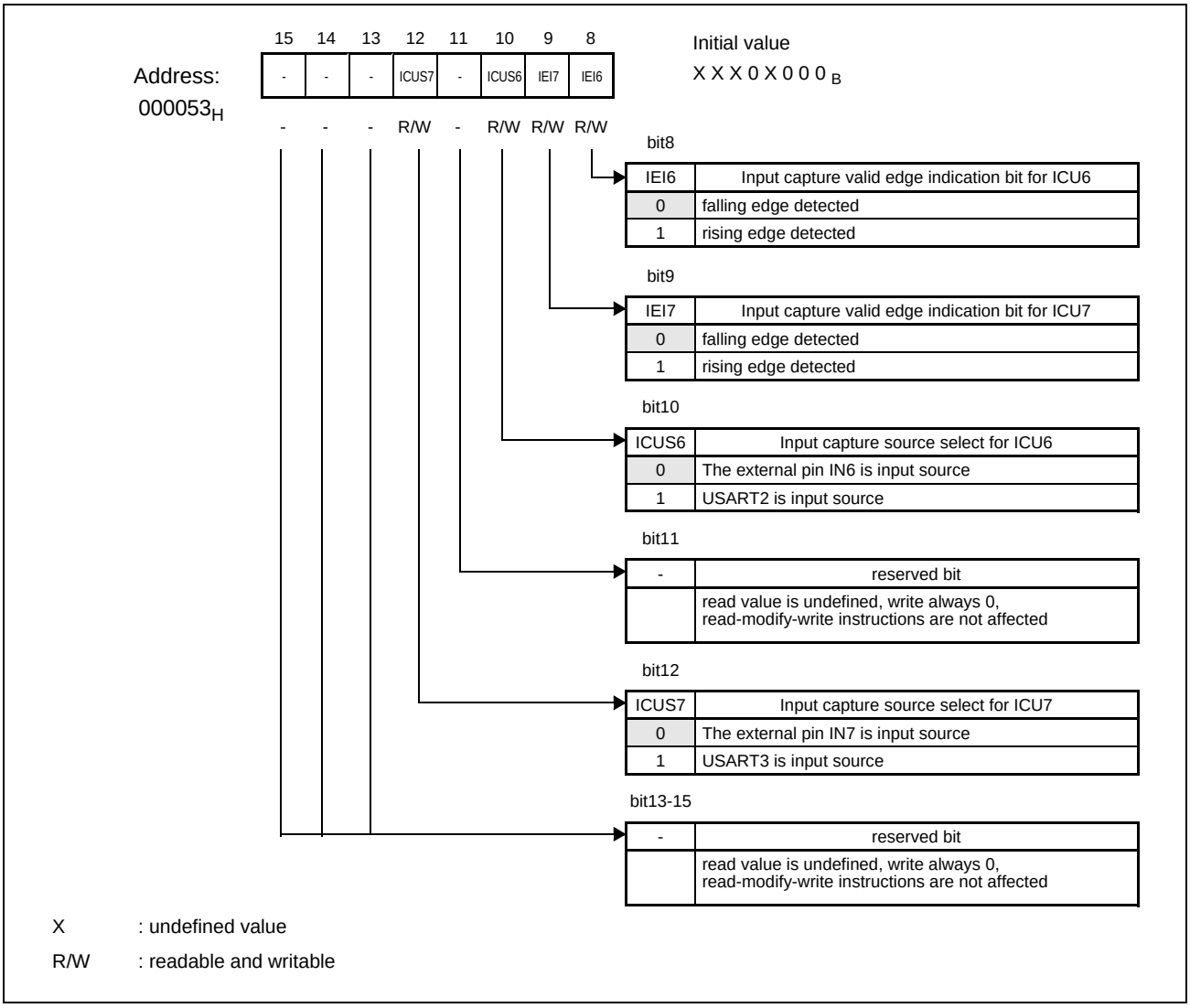


Figure 1-5. Input Capture Edge Register ICE89

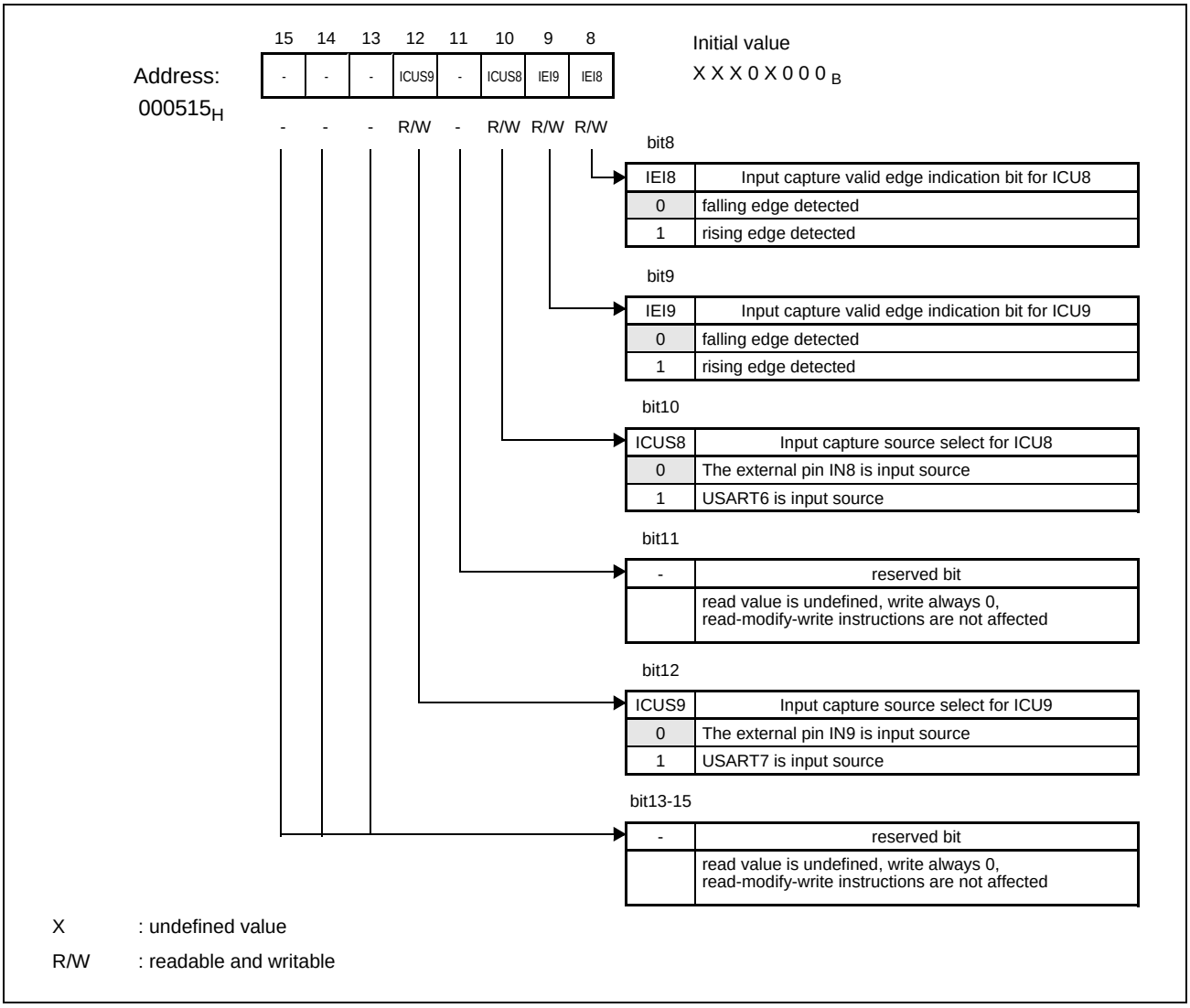
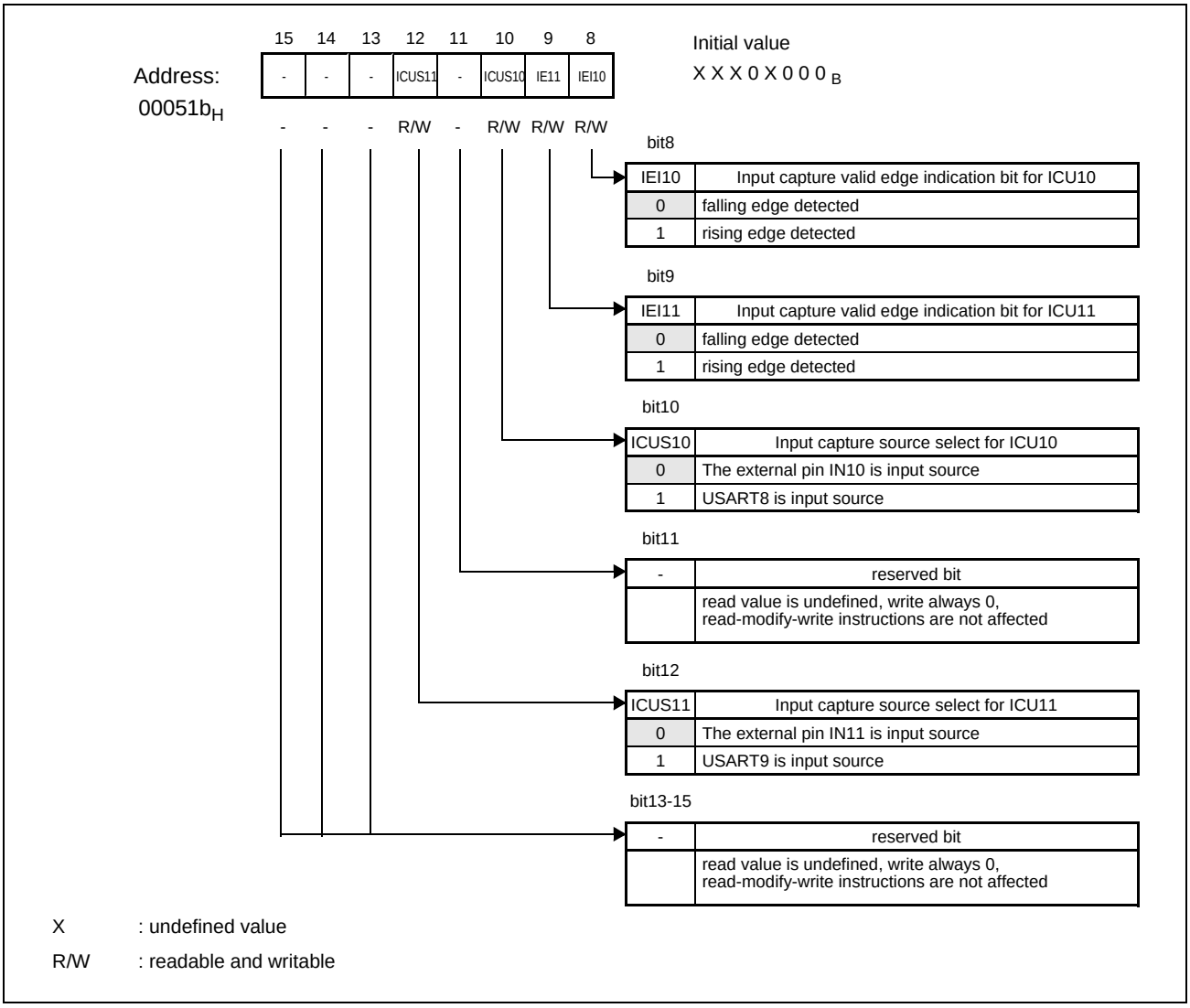


Figure 1-6. Input Capture Edge Register ICE1011



## 1.7 Peripheral Resource Pin Relocation

The input or output pin of some resources can be relocated. The location of these resource pins is defined by the peripheral resource pin relocation registers PRRR0 to PRRR13.

### 1.7.1 Overview of the Peripheral Resource Relocation Register

Table 1-7. Peripheral Resource Pin Relocation Register (PRRR0 to PRRR13)

| Address | Name | BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 | Description |
|---|---|---|---|---|---|---|---|---|---|---|
| 0004d6 | PRRR0 | INT7_R | INT6_R | INT5_R | INT4_R | INT3_R | INT2_R | INT1_R | INT0_R | Peripheral resource relocation register 0 |
| 0004d7 | PRRR1 | INT15_R | INT14_R | INT13_R | INT12_R | INT11_R | INT10_R | INT9_R | INT8_R | Peripheral resource relocation register 1 |
| 0004d8 | PRRR2 | PPG7_R | PPG6_R | PPG5_R | PPG4_R | PPG3_R | PPG2_R | PPG1_R | PPG0_R | Peripheral resource relocation register 2 |
| 0004d9 | PRRR3 | TOT3_R | TIN3_R | TOT2_R | TIN2_R | TOT1_R | TIN1_R | TOT0_R | TIN0_R | Peripheral resource relocation register 3 |
| 0004da | PRRR4 | IN7_R | IN6_R | IN5_R | IN4_R | IN3_R | IN2_R | IN1_R | IN0_R | Peripheral resource relocation register 4 |
| 0004db | PRRR5 | OUT7_R | OUT6_R | - | - | OUT3_R | OUT2_R | OUT1_R | OUT0_R | Peripheral resource relocation register 5 |
| 0004dc | PRRR6 | CKOTX1_R | CKOT1_R | SCK2_R | SOT2_R | SIN2_R | FRCK0_R | SGA0_R | SGO0_R | Peripheral resource relocation register 6 |
| 0004dd | PRRR7 | TX2_R | RX2_R | INT5_R1 | INT4_R1 | INT3_R1 | CS3_R | NMI_R | ADTG_R | Peripheral resource relocation register 7 |
| 0004de | PRRR8 | SOT9_R | SIN9_R | SCK8_R | SOT8_R | SIN8_R | SCK7_R | SOT7_R | SIN7_R | Peripheral resource relocation register 8 |
| 0004df | PRRR9 | - | - | CKOT0_R | OUT10_R | FRCK2_R | SGA1_R | SGO1_R | SCK9_R | Peripheral resource relocation register 9 |
| 000660 | PRRR10 | TTG11_R | TTG10_R | TTG9_R | TTG8_R | PPG11_R | PPG10_R | PPG9_R | PPG8_R | Peripheral resource relocation register 10 |
| 000661 | PRRR11 | TTG19_R | TTG18_R | TTG17_R | TTG16_R | PPG19_R | PPG18_R | PPG17_R | PPG16_R | Peripheral resource relocation register 11 |
| 000662 | PRRR12 | - | - | - | CS5_R | CS4_R | CS2_R | CS1_R | CS0_R | Peripheral resource relocation register 12 |
| 000663 | PRRR13 | - | - | - | - | - | - | - | - | Peripheral resource relocation register 13 |

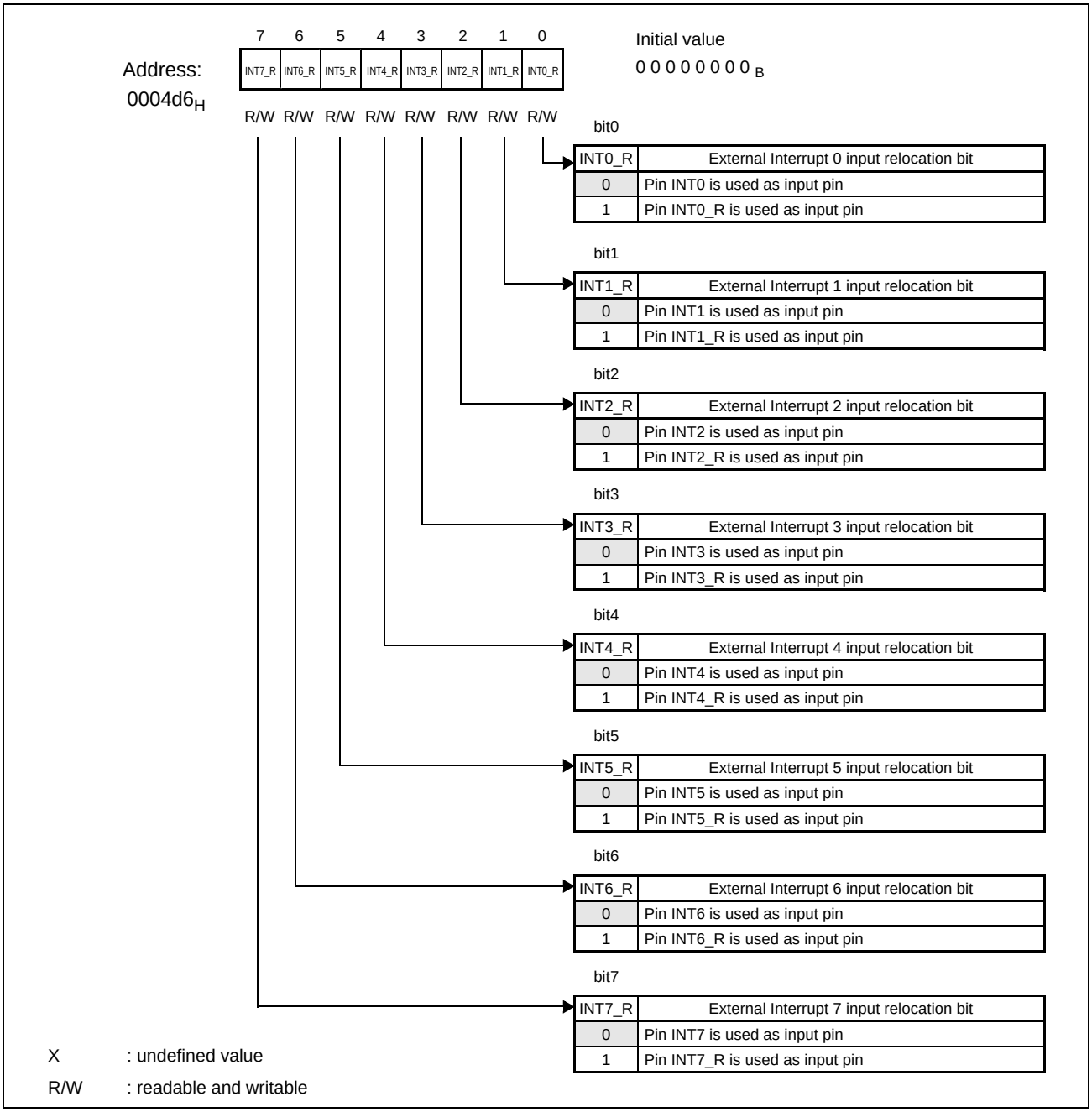
#### Availability of resource relocation pins

If a resource relocation pin is not available on the device, the corresponding bit must be treated as undefined bit. Hence the read value of this bit is undefined,'0' must always be written to this bit and read modify write instructions on the register are not affected by this bit.

See the data sheet for the availability of resource relocation pins.

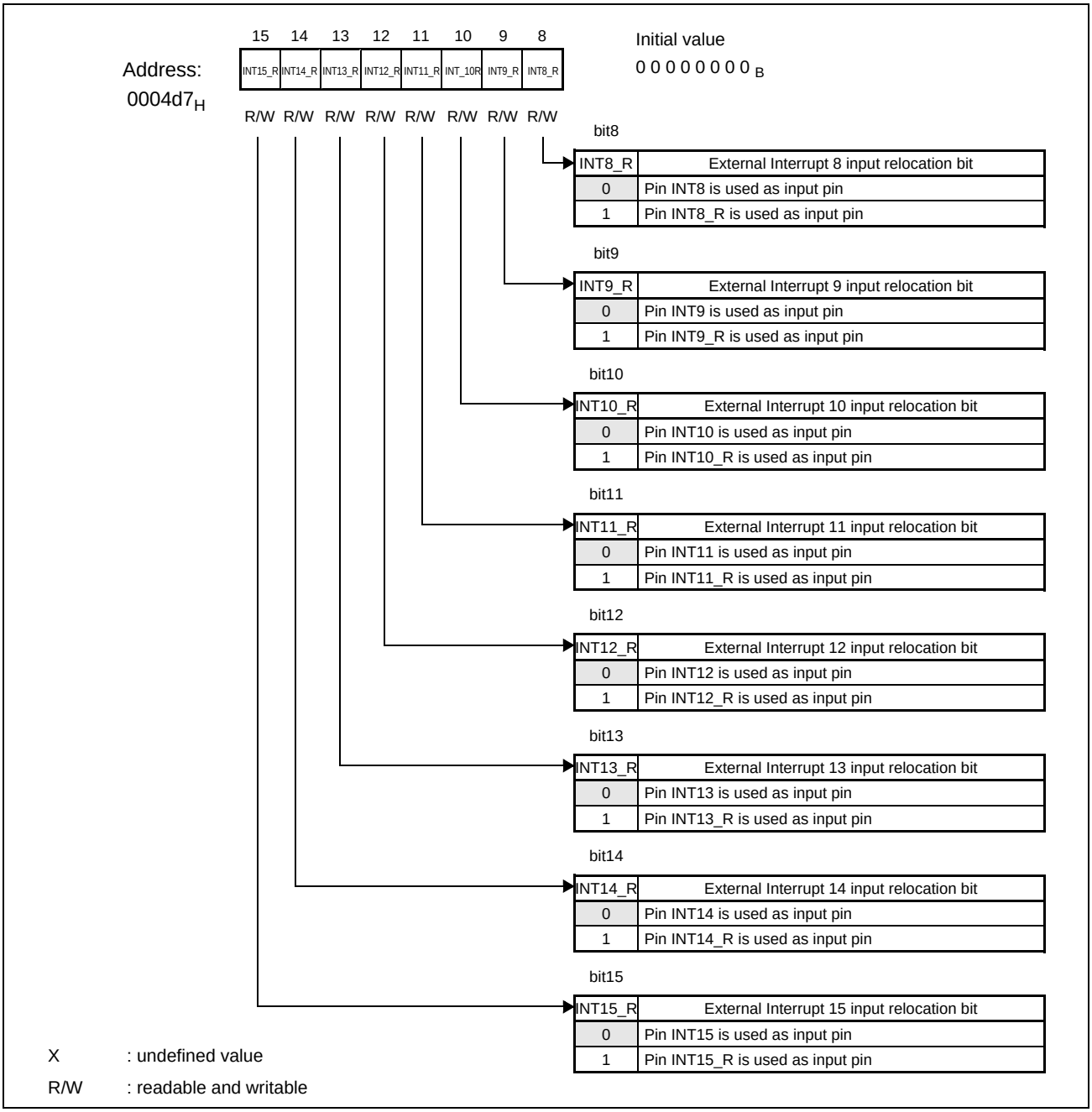
## 1.7.2 Peripheral Resource Relocation Register 0 (PRRR0)

Figure 1-7. Peripheral Resource Relocation Register 0 (PRRR0)



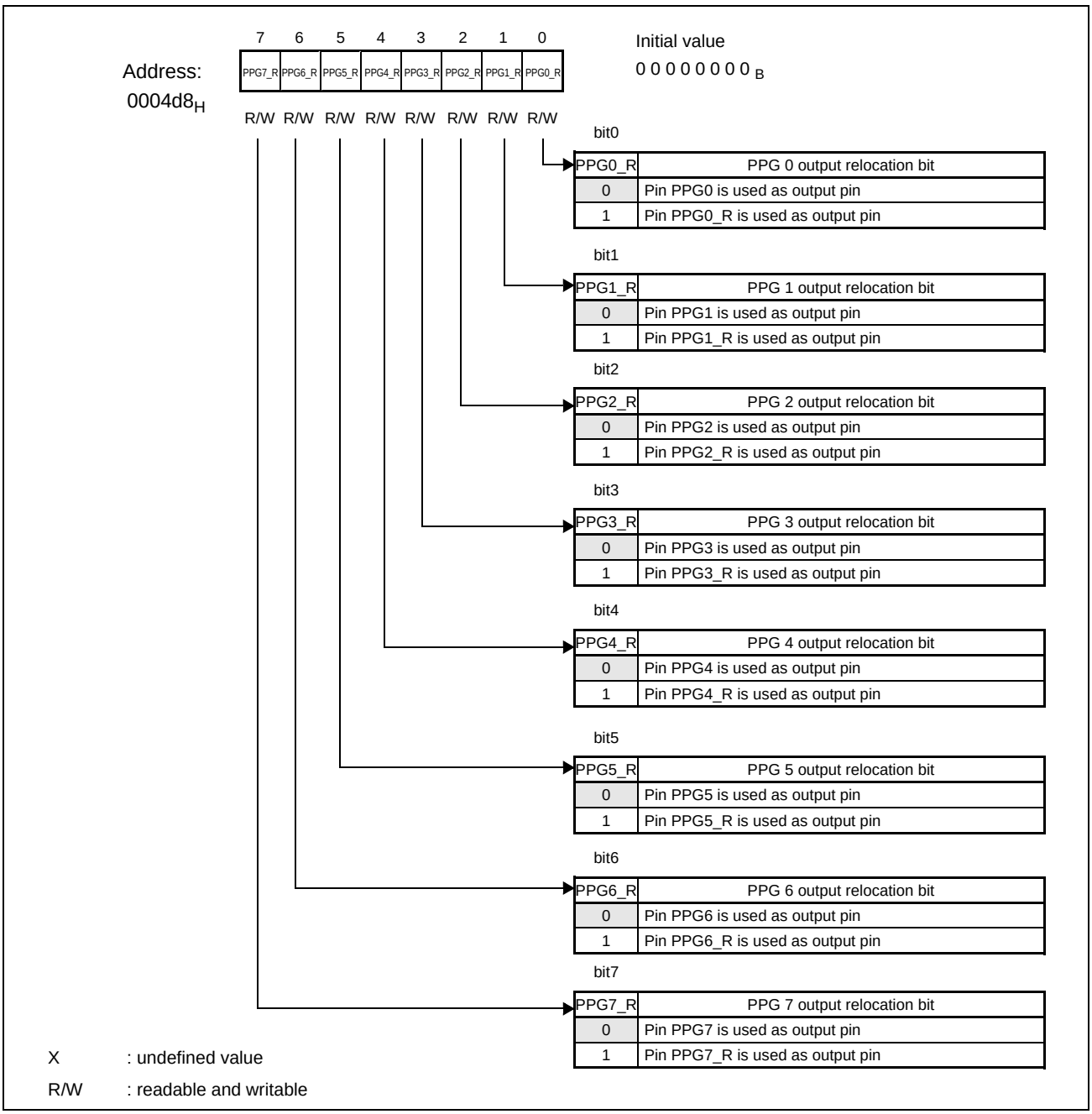
## 1.7.3 Peripheral Resource Relocation Register 1 (PRRR1)

Figure 1-8. Peripheral Resource Relocation Register 1 (PRRR1)



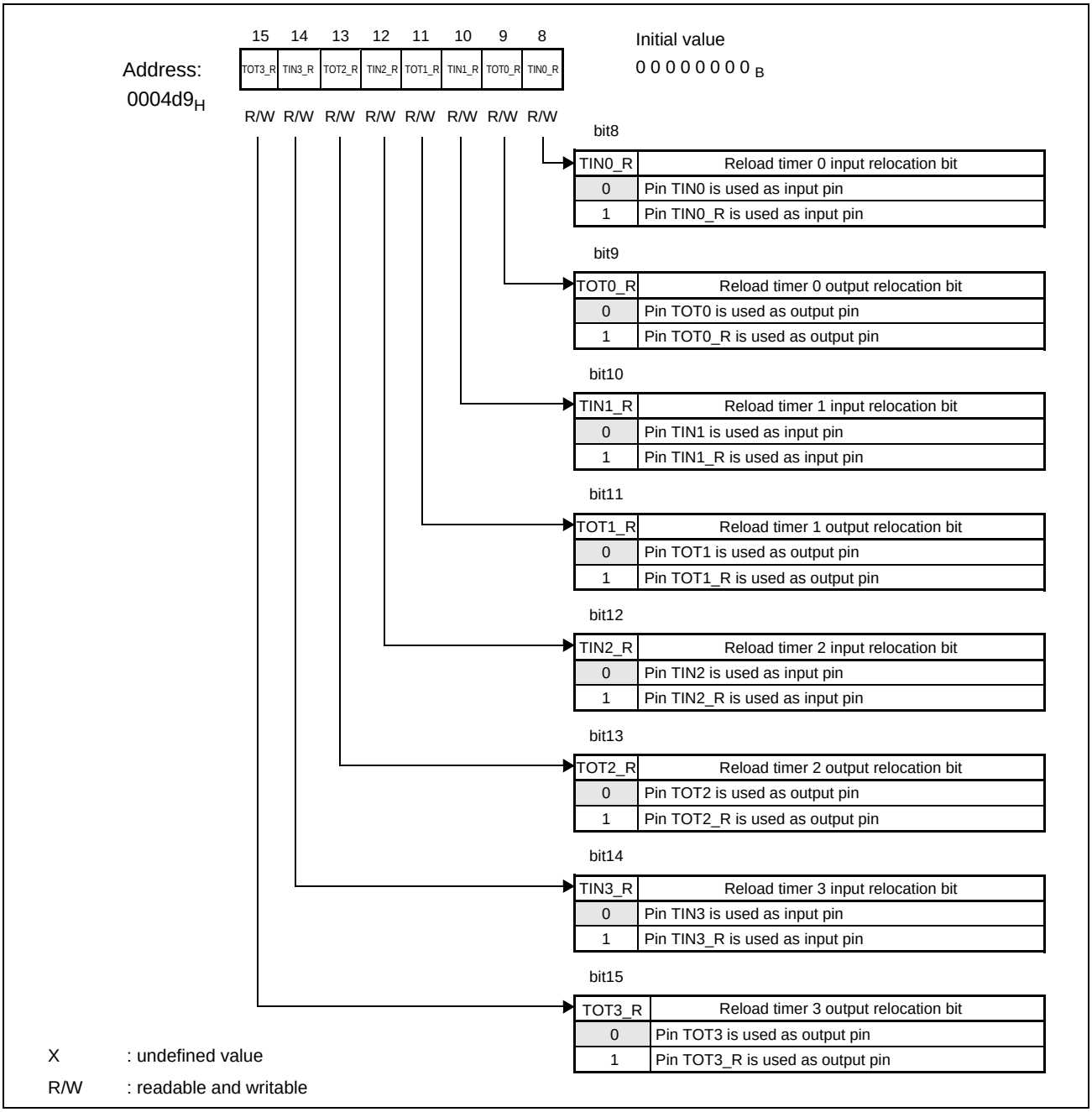
## 1.7.4 Peripheral Resource Relocation Register 2 (PRRR2)

Figure 1-9. Peripheral Resource Relocation Register 2 (PRRR2)



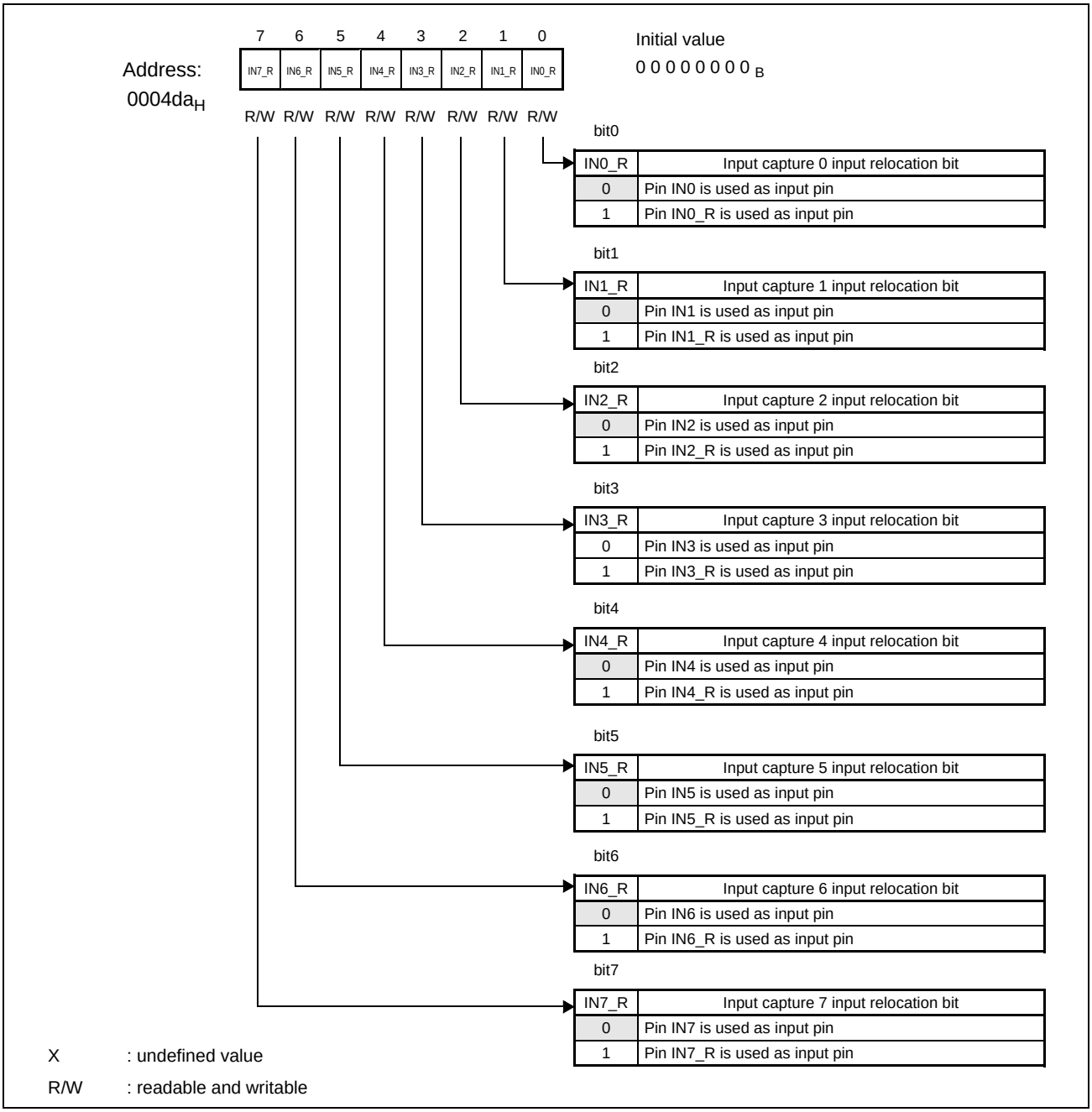
## 1.7.5 Peripheral Resource Relocation Register 3 (PRRR3)

Figure 1-10. Peripheral Resource Relocation Register 3 (PRRR3)



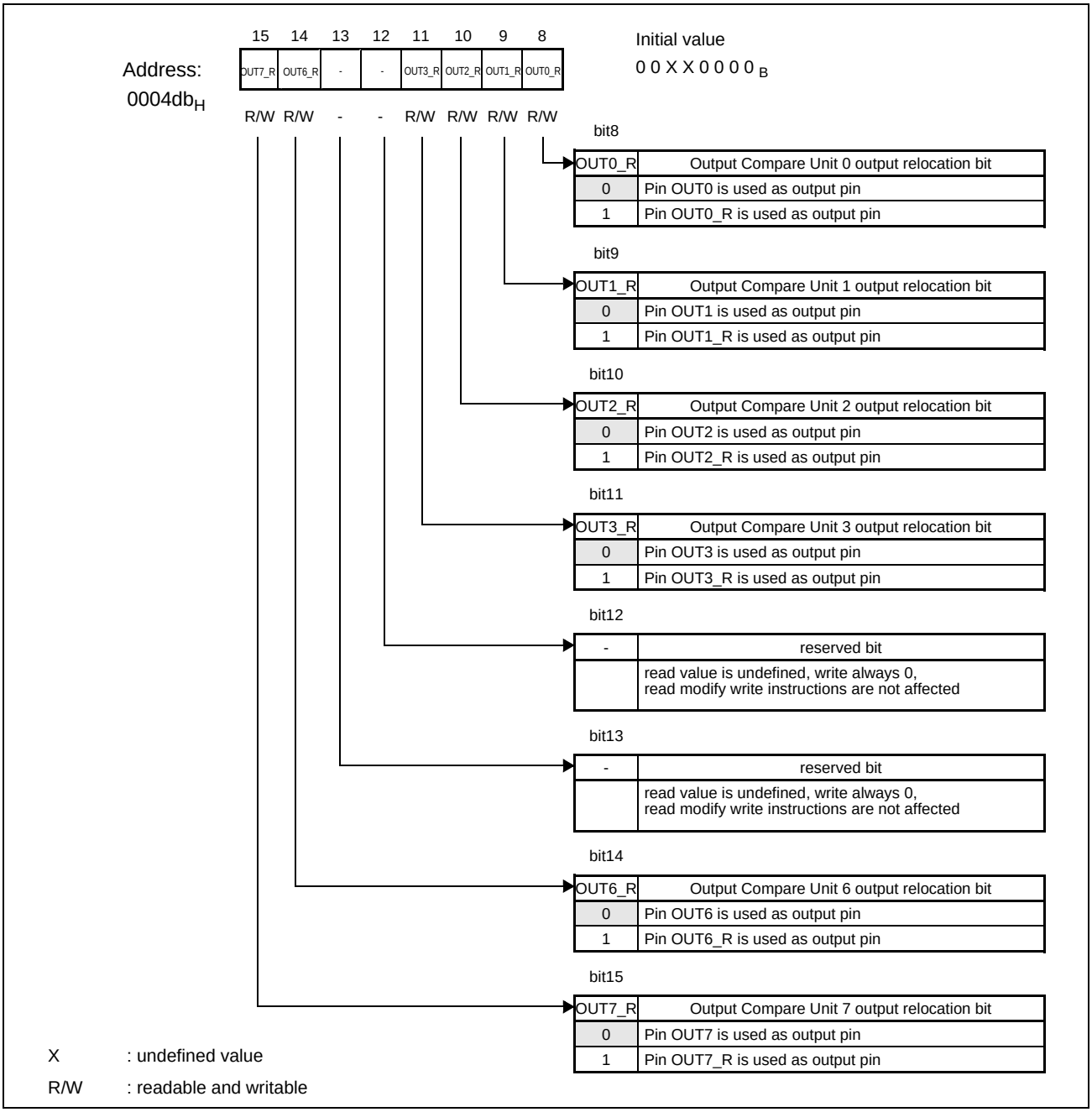
## 1.7.6 Peripheral Resource Relocation Register 4 (PRRR4)

Figure 1-11. Peripheral Resource Relocation Register 4 (PRRR4)



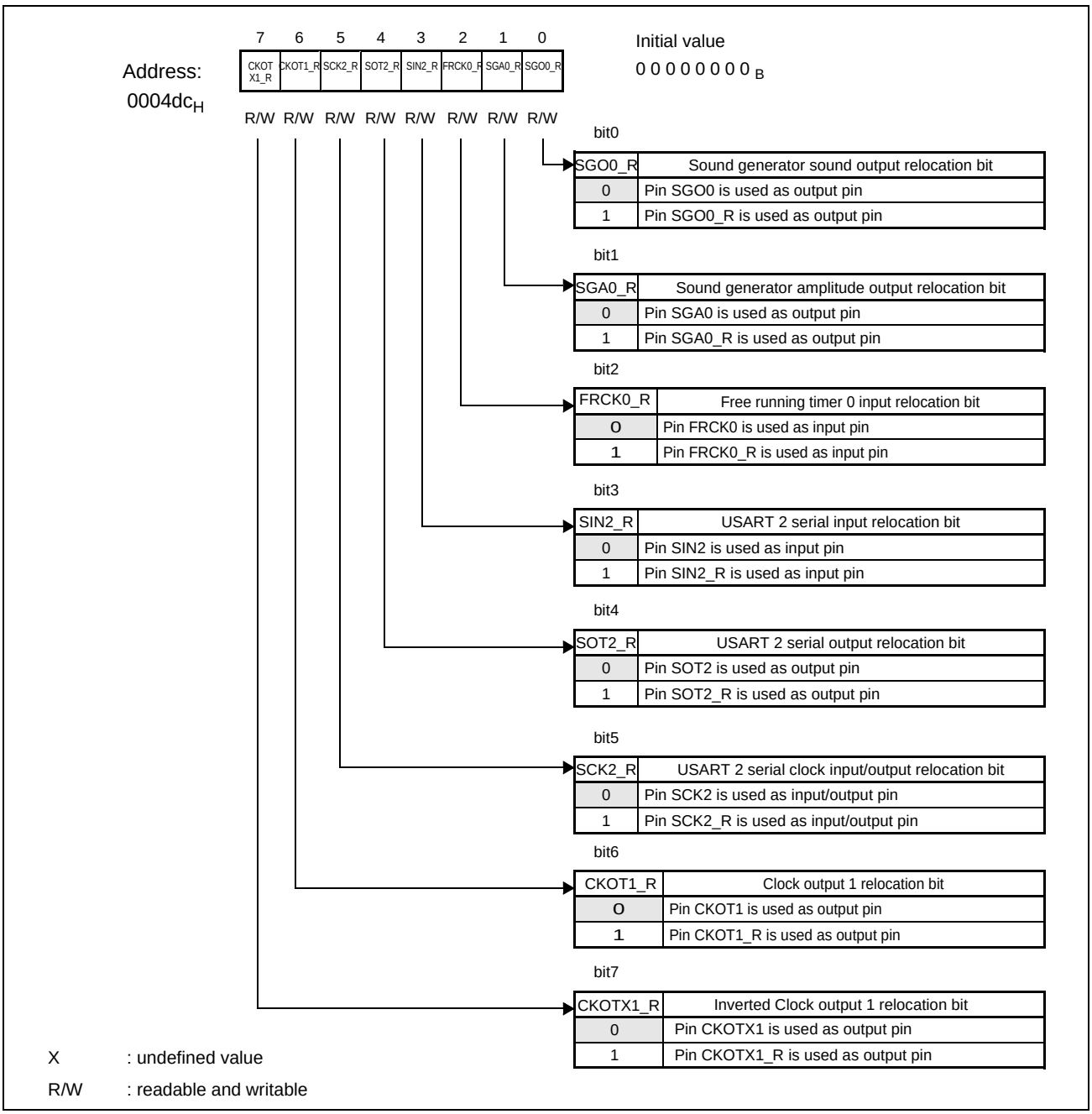
## 1.7.7 Peripheral Resource Relocation Register 5 (PRRR5)

Figure 1-12. Peripheral Resource Relocation Register 5 (PRRR5)



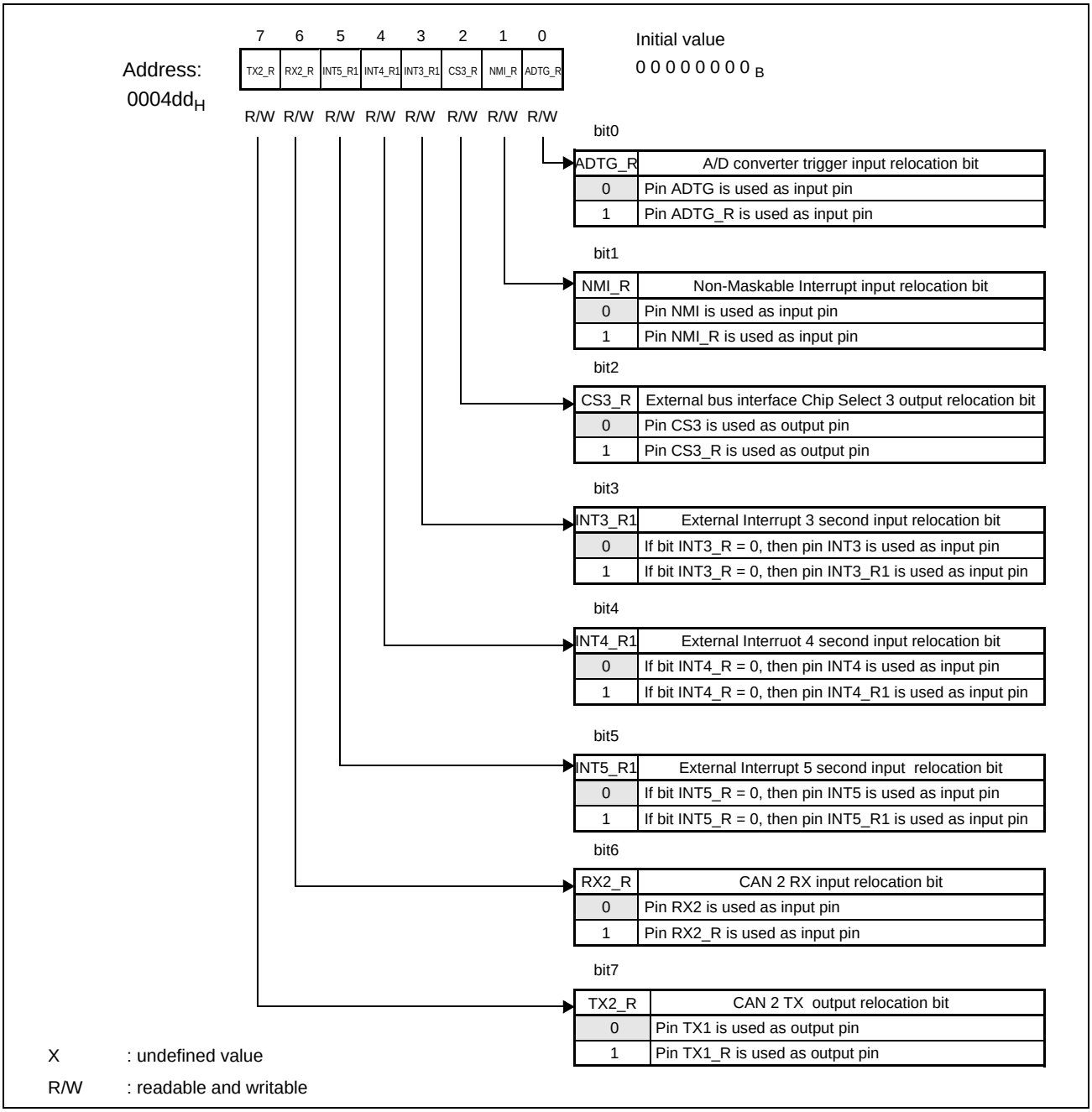
## 1.7.8 Peripheral Resource Relocation Register 6 (PRRR6)

Figure 1-13. Peripheral Resource Relocation Register 6 (PRRR6)



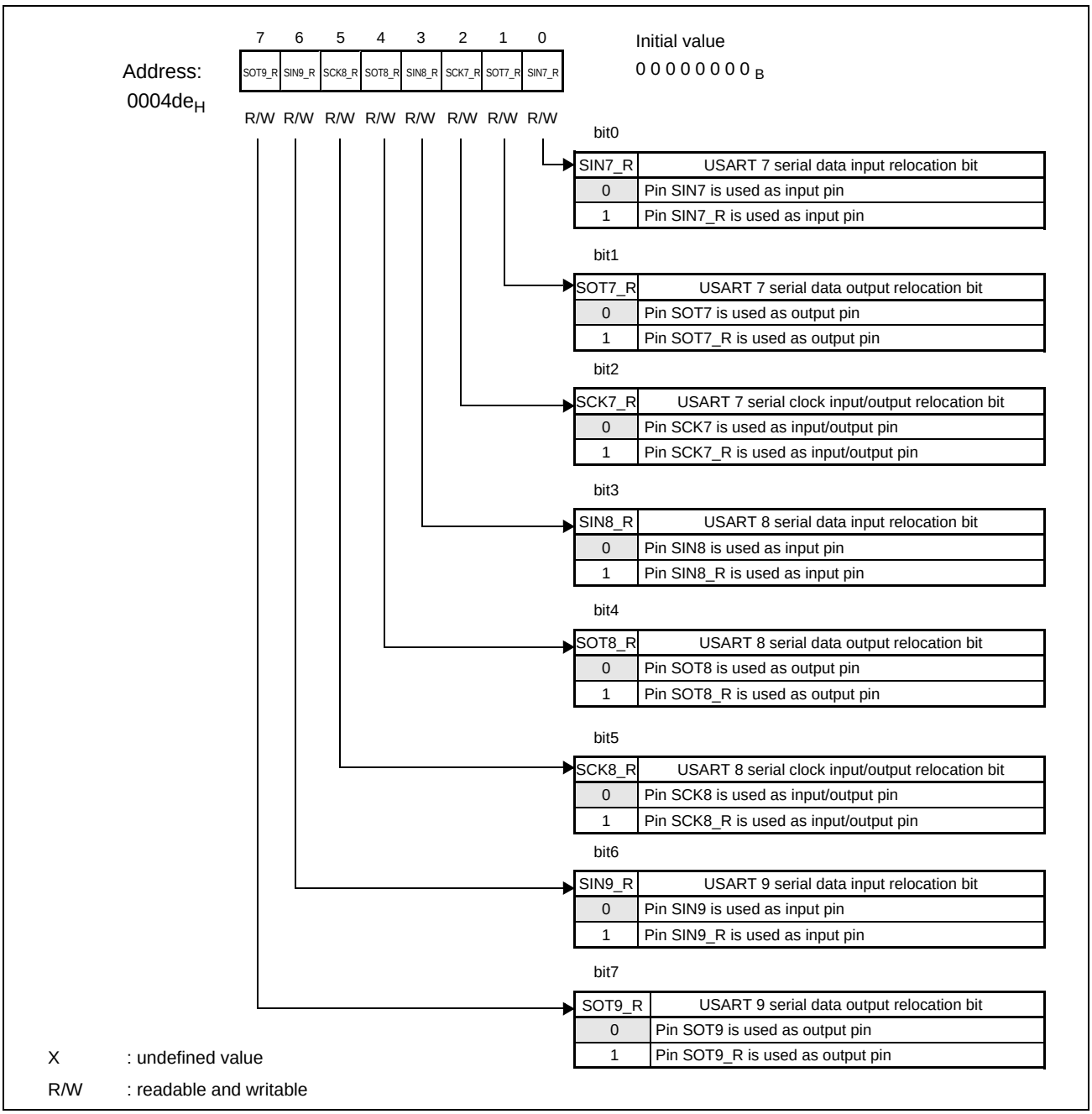
## 1.7.9 Peripheral Resource Relocation Register 7 (PRRR7)

Figure 1-14. Peripheral Resource Relocation Register 7 (PRRR7)



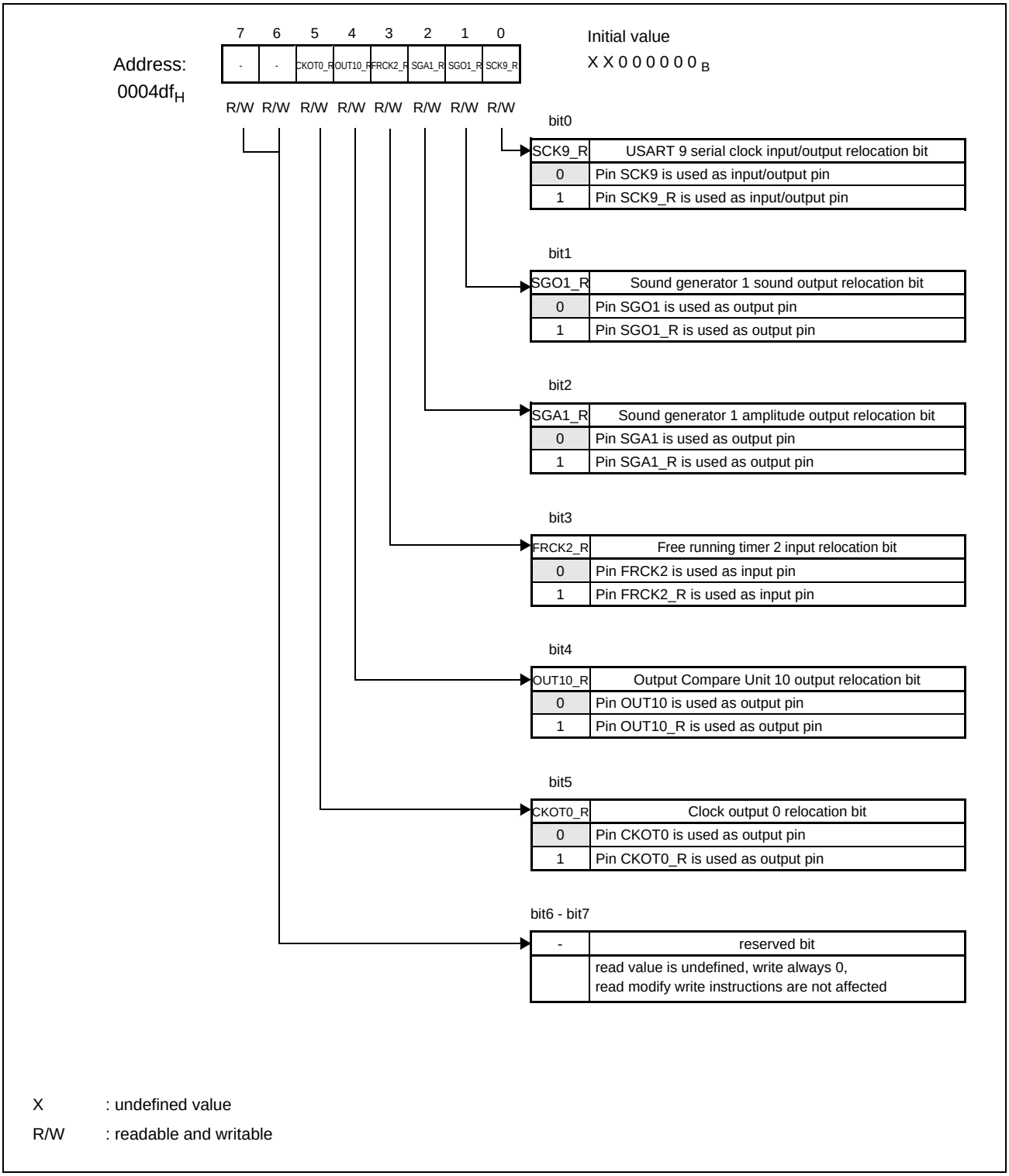
## 1.7.10 Peripheral Resource Relocation Register 8 (PRRR8)

Figure 1-15. Peripheral Resource Relocation Register 8 (PRRR8)



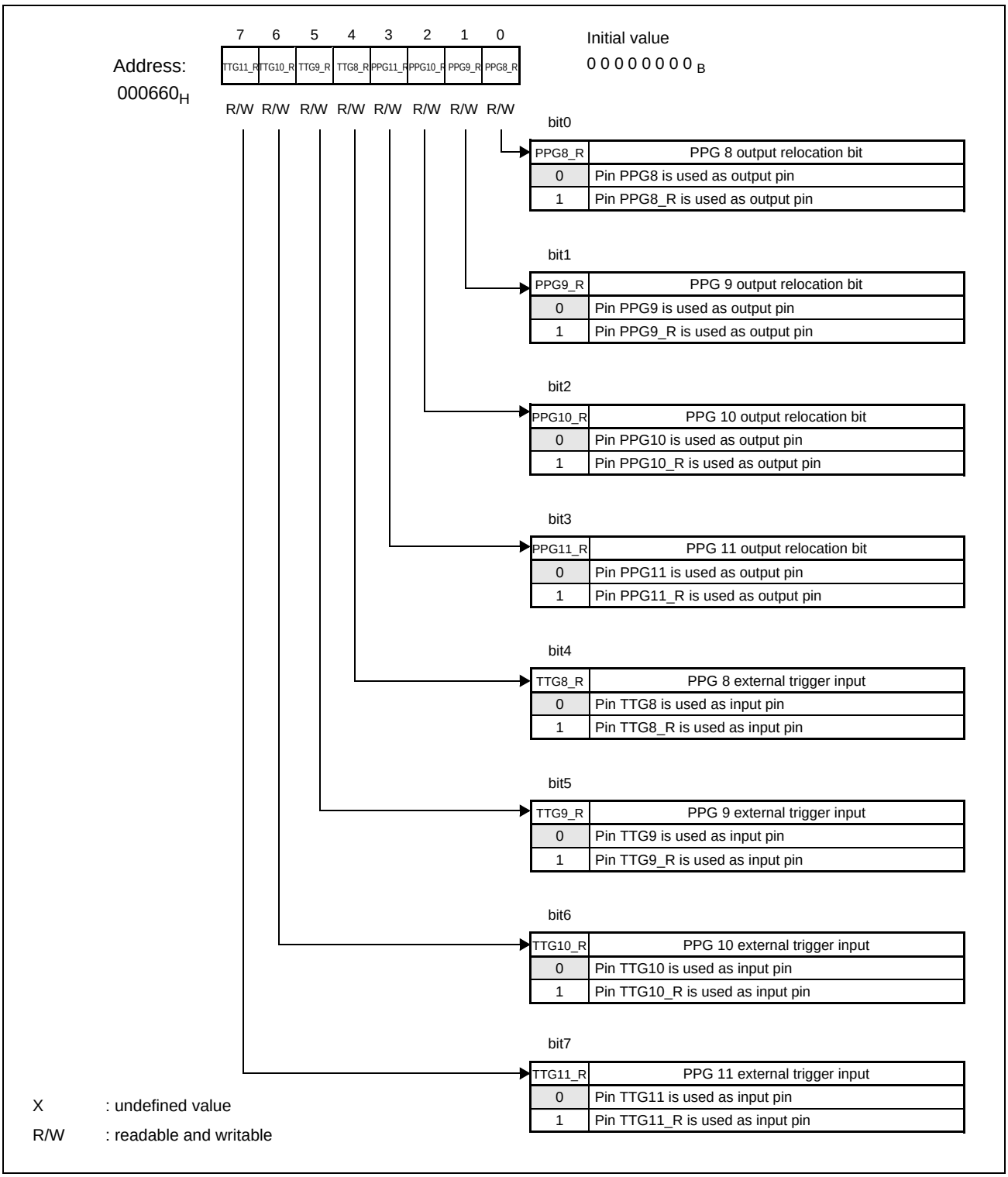
## 1.7.11 Peripheral Resource Relocation Register 9 (PRRR9)

Figure 1-16. Peripheral Resource Relocation Register 9 (PRRR9)



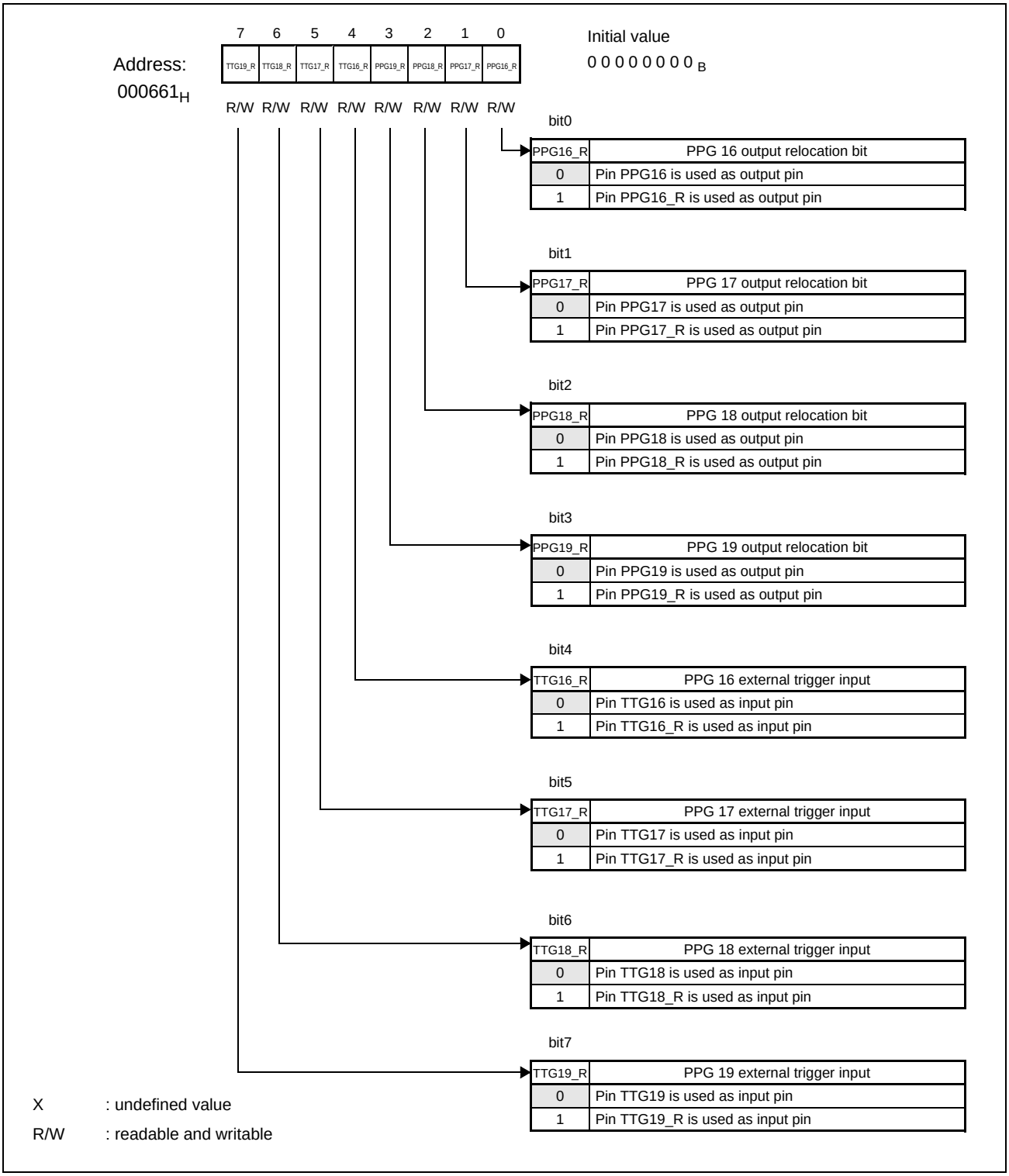
## 1.7.12 Peripheral Resource Relocation Register 10 (PRRR10)

Figure 1-17. Peripheral Resource Relocation Register 10 (PRRR10)



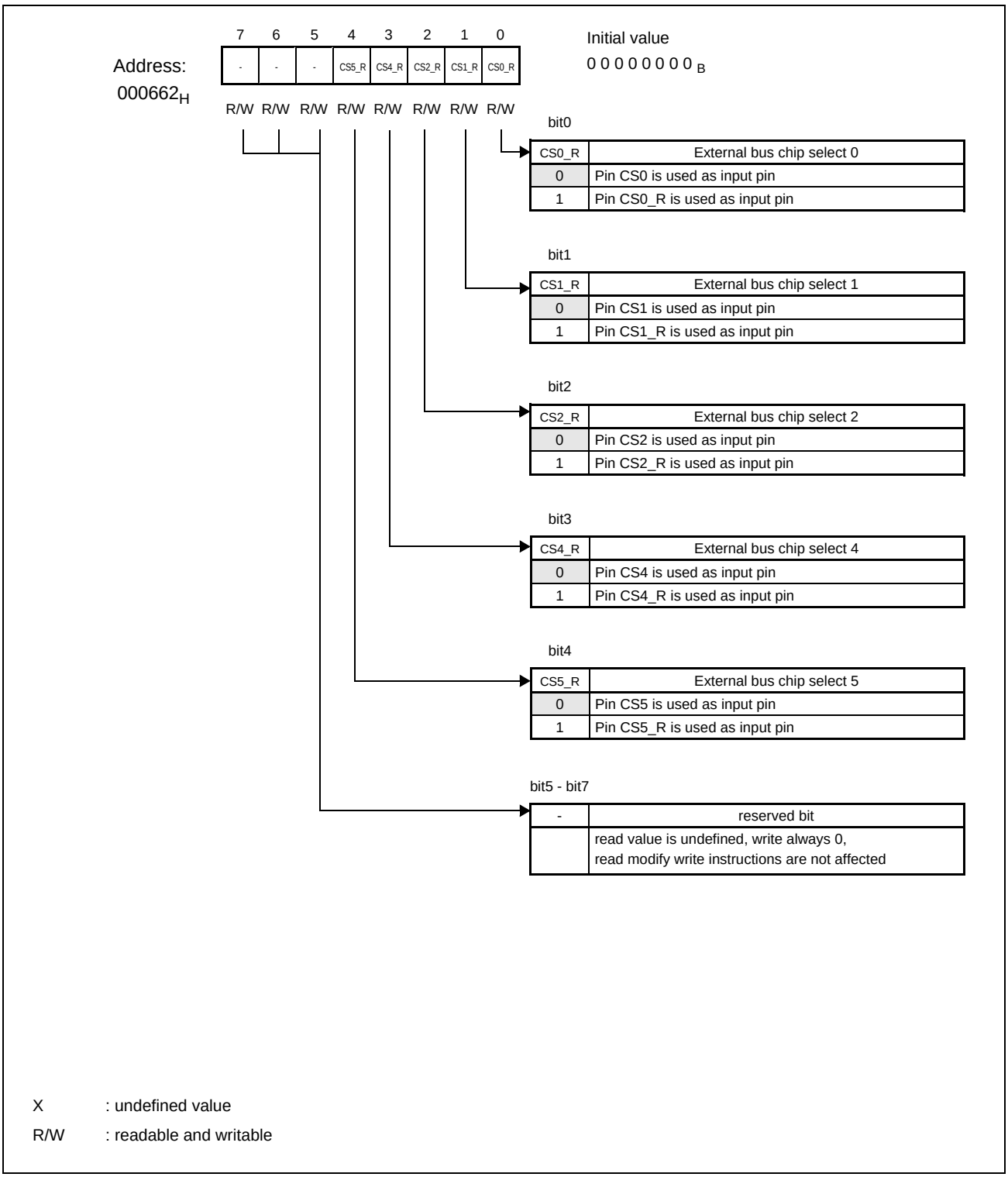
## 1.7.13 Peripheral Resource Relocation Register 11 (PRRR11)

Figure 1-18. Peripheral Resource Relocation Register 11 (PRRR11)



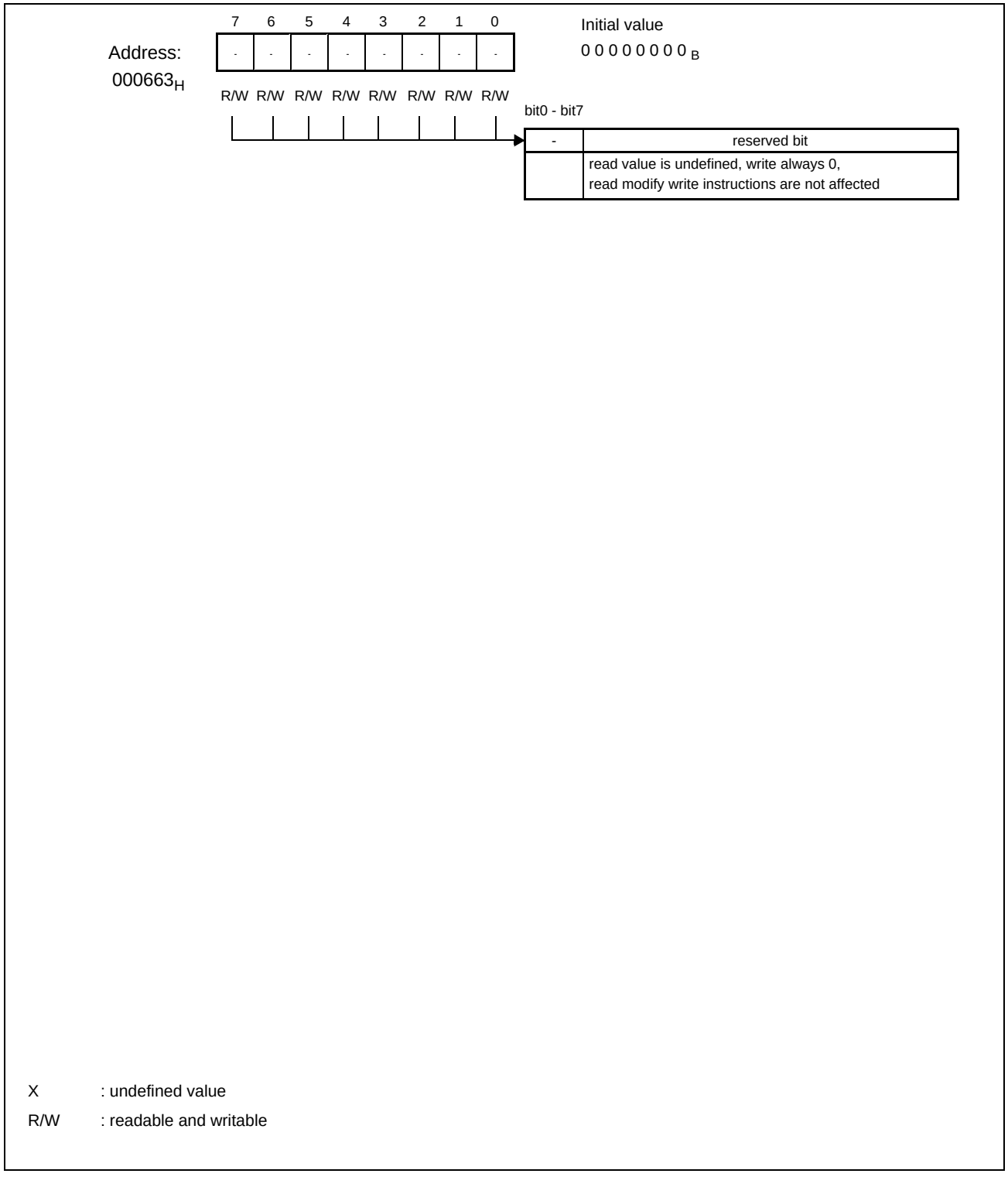
## 1.7.14 Peripheral Resource Relocation Register 12 (PRRR12)

Figure 1-19. Peripheral Resource Relocation Register 12 (PRRR12)



## 1.7.15 Peripheral Resource Relocation Register 13 (PRRR13)

Figure 1-20. Peripheral Resource Relocation Register 13 (PRRR13)



# 2. CPU

This chapter explains the CPU.

## 2.1 Outline of the CPU

The F$^2$MC-16FX CPU core is a 16-bit CPU designed for applications that require high-speed real-time processing, such as home-use or vehicle-mounted electronic appliances. The F$^2$MC-16FX instruction set is designed for controller applications, and is capable of high-speed, highly efficient control processing.

### 2.1.1 Outline of the CPU

In addition to 16-bit data, the F$^2$MC-16FX CPU core can process 32-bit data by using an internal 32-bit accumulator. 32-bit data can be processed with some instructions. Up to 16 MBytes of memory space can be used, which can be accessed by either the linear pointer or bank method. The instruction set is compatible to F$^2$MC-16LX. The instruction set is compatible with high-level languages, has a rich set of addressing modes, multiplication and division instructions, and bit processing. The features of the F$^2$MC-16FX CPU are explained below.

- Fast execution speed
  - Minimum instruction execution time: 16 ns (at 64 MHz; 4-MHz oscillation, 16 times clock multiplication)
  - Basic instructions are executed in one cycle
  - High speed processing using a 5 stage pipeline
  - 8 byte instruction queue
- General purpose registers: 32 banks x 8 words x 16 bits
- Memory space: 16 MBytes, accessed in linear or bank method
- Instruction set optimized for controller applications
  - High code efficiency
  - Rich data types: Bit, byte, word, long word
  - Extended addressing modes: 23 types
  - High-precision operation (32-bit length) based on 32-bit accumulator
  - Signed and unsigned multiplication and division instructions
- Powerful interrupt functions
  - Fast response speed (about 10 clock cycles CLKB)
  - Eight priority levels (programmable)
  - Non maskable interrupt (NMI)
  - DMA transfer can serve interrupt requests (16 channels max.) without involving CPU
- Instruction set compatible with high-level language (C)/multitasking
  - System stack pointer
  - Instruction set symmetry
  - Shift instructions

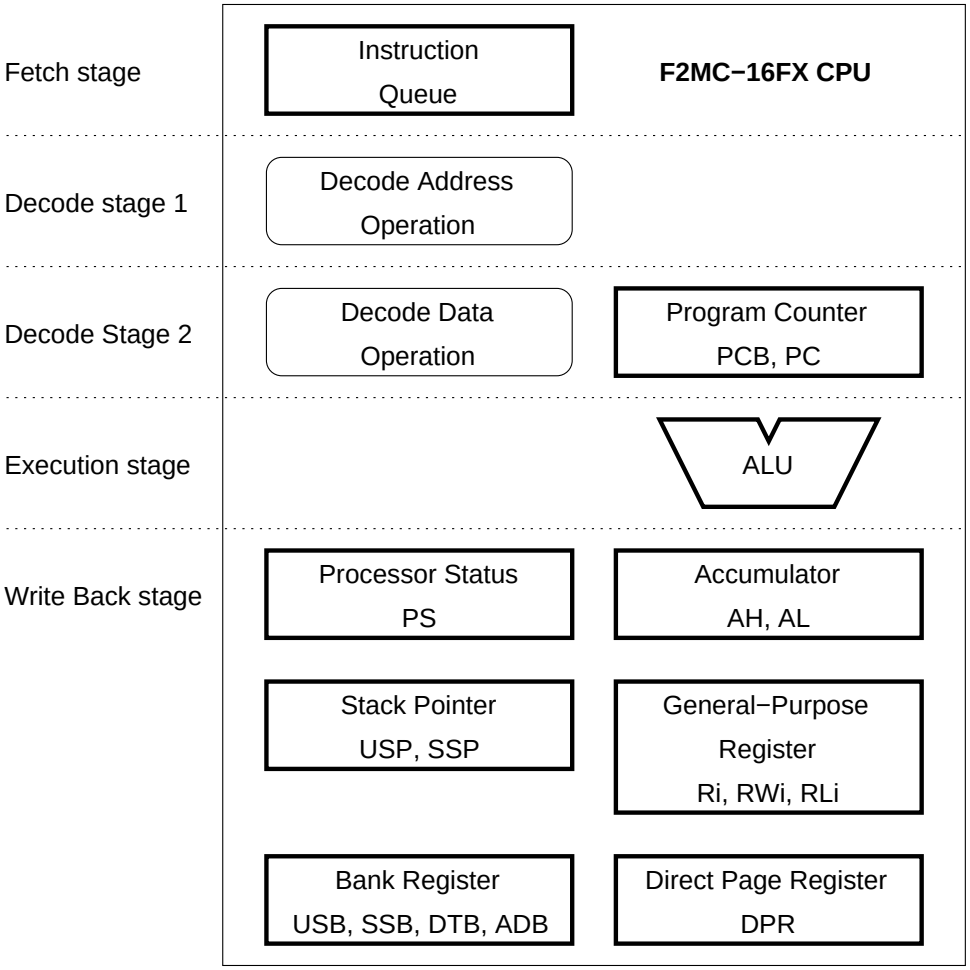
## 2.2 Hardware Structure

**This section explains the hardware structure of the CPU and the 16FX core.**

### 2.2.1 Hardware Structure of the CPU

**CPU Block Diagram**

Figure 2-1. CPU block diagram

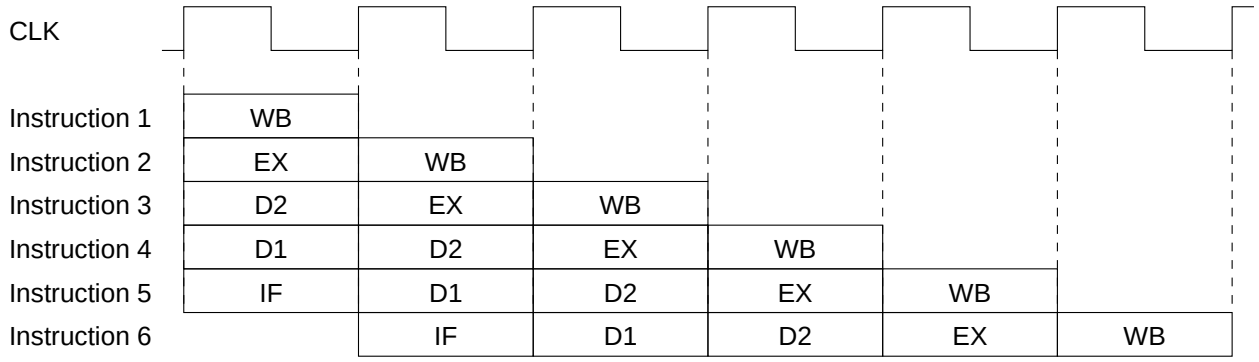



**CPU Pipeline Operation**

To execute most instructions in one clock cycle, the CPU uses a five-stage instruction pipeline. The pipeline consists of the following stages:

- Instruction fetch (IF): Fetches the instruction from instruction queue.
- Instruction decode 1 (D1): Decodes the instruction and controls address operation.
- Instruction decode 2 (D2): Decodes the instruction and selects operands and data operation.
- Execution (EX): Executes the operation.
- Write back (WB): Writes the operation result to a register or memory location.

Figure 2-2. Instruction Pipeline




Instructions are not executed out of order. Therefore, if instruction A enters the pipeline ahead of instruction B, instruction A always reaches write back stage before instruction B.

The standard instruction execution speed is one instruction per cycle. However, transfer instructions that involve memory wait, branch instructions and multi-cycle instructions require more than one cycle to execute. The instruction execution speed also drops if the delivery of instructions during code fetch is slow.

### Instruction Queue

The CPU has an instruction queue of 8 byte.

The instruction queue is filled by the fetch unit. Prefetch is used on consecutive addresses for code fetch. The prefetch mechanism removes drawbacks due to the latency of the pipelined implementation of the CPU and the system bus of the 16FX core.

### Program counter

The program counter bank (PCB, upper 8 bits of the program address) and the program counter (PC, lower 16 bits of the program address) are controlled by the decode stage 1.

The 24-bit address of the concatenation of {PCB, PC} points to the instruction, which is executed next.

### ALU

The ALU is controlled by decode stage 2. The operation mode of the ALU is selected and the operands are loaded. The execution of the operation is performed in the next cycle.

The ALU is used for logical and arithmetical operations. Multiplication and division are included.

### CPU registers and memory access

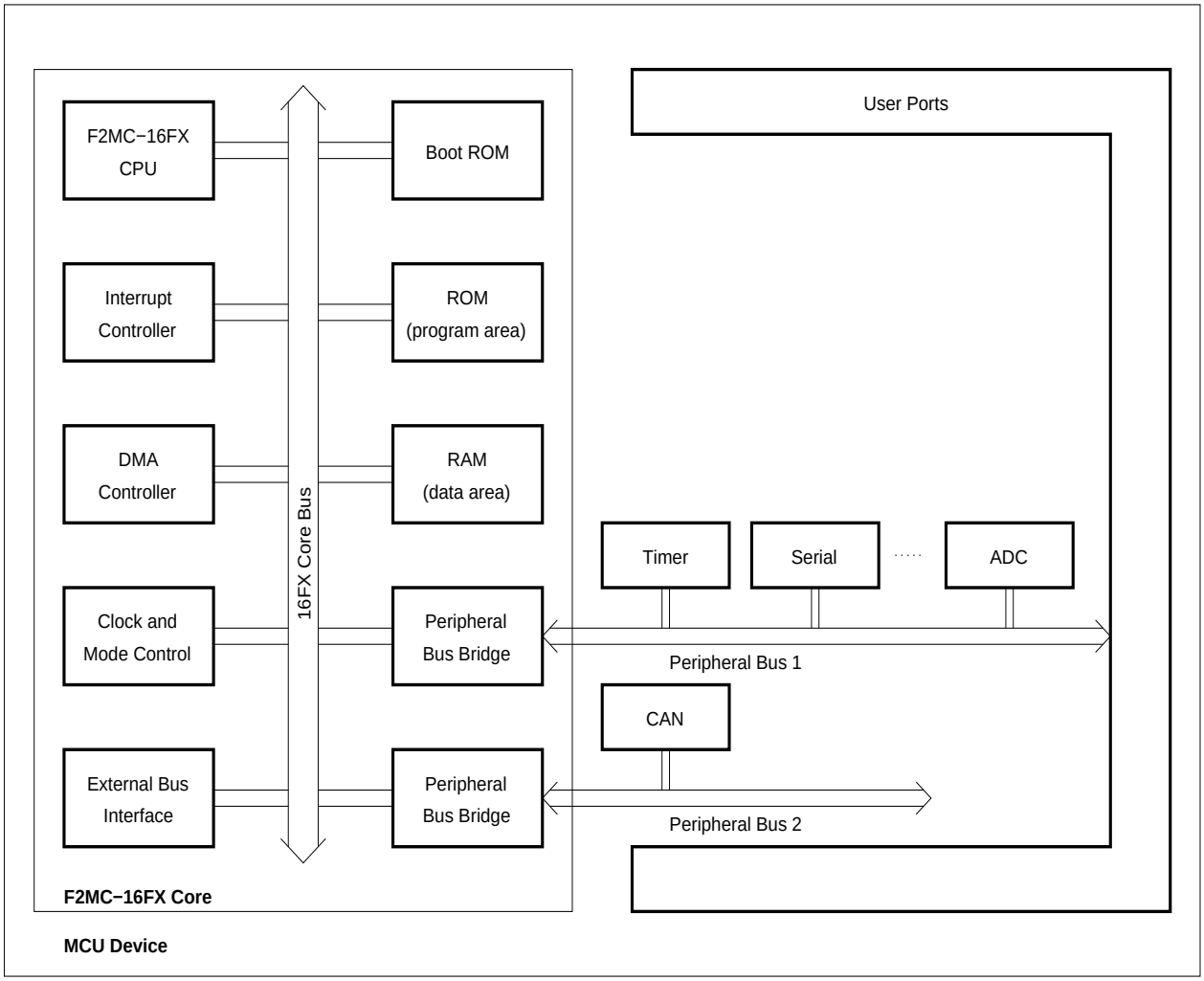
In the write back stage the result of the operation is written to CPU registers and/or to a memory location. All CPU registers except the program counter are assigned to the last pipeline stage.

## 2.2.2 Hardware structure of the 16FX Core

### Block Diagram

A sample configuration and the principle structure of a MCU device based on the 16FX Core is shown in Figure 2-3.

Figure 2-3. MCU Device based on the 16FX Core




### Interrupt Controller

The interrupt controller evaluates the priority of incoming interrupt requests (IRQ) and selects the interrupt number with the highest priority. If accepted, the selected interrupt service is processed by the CPU. Each hardware IRQ has its own interrupt level register to control its priority.

### DMA Controller

The DMA controller can also serve IRQs, but without interrupting the actual program execution of the CPU. This can be used to automate data transfer between peripherals and memory.

Depending on the device, up to 16 DMA channels are usable. Each DMA channel can select an IRQ number to be served.

### Clock and Mode Control

This unit has control over the operation mode and monitors correct operation of the device. It supplies all units with their appropriate clock, depending on the operation mode.

### External Bus Interface

The external bus interface is an optional component. Its availability depends on the configuration of the specific device.

### Boot ROM

After device initialization by reset, the program counter points to the boot ROM. The CPU starts the execution of the boot ROM program. After further device initialization the reset vector is fetched and the boot ROM code branches to user program execution starting at the reset vector.

### Peripheral Bus Bridge

The peripheral bus bridge acts as an interface between the system bus of the 16FX core and the peripheral bus connecting to all other MCU internal peripheral resources.

The peripheral bus bridge synchronizes between core clock and peripheral clock domains.

## 2.3 Memory Space

An F$^2$MC-16FX CPU has a 16-Mbyte memory space.

2.3.1 Memory Areas

2.3.2 Linear Addressing Method

2.3.3 Bank Addressing Method

2.3.4 Multi-byte Data in Memory Space

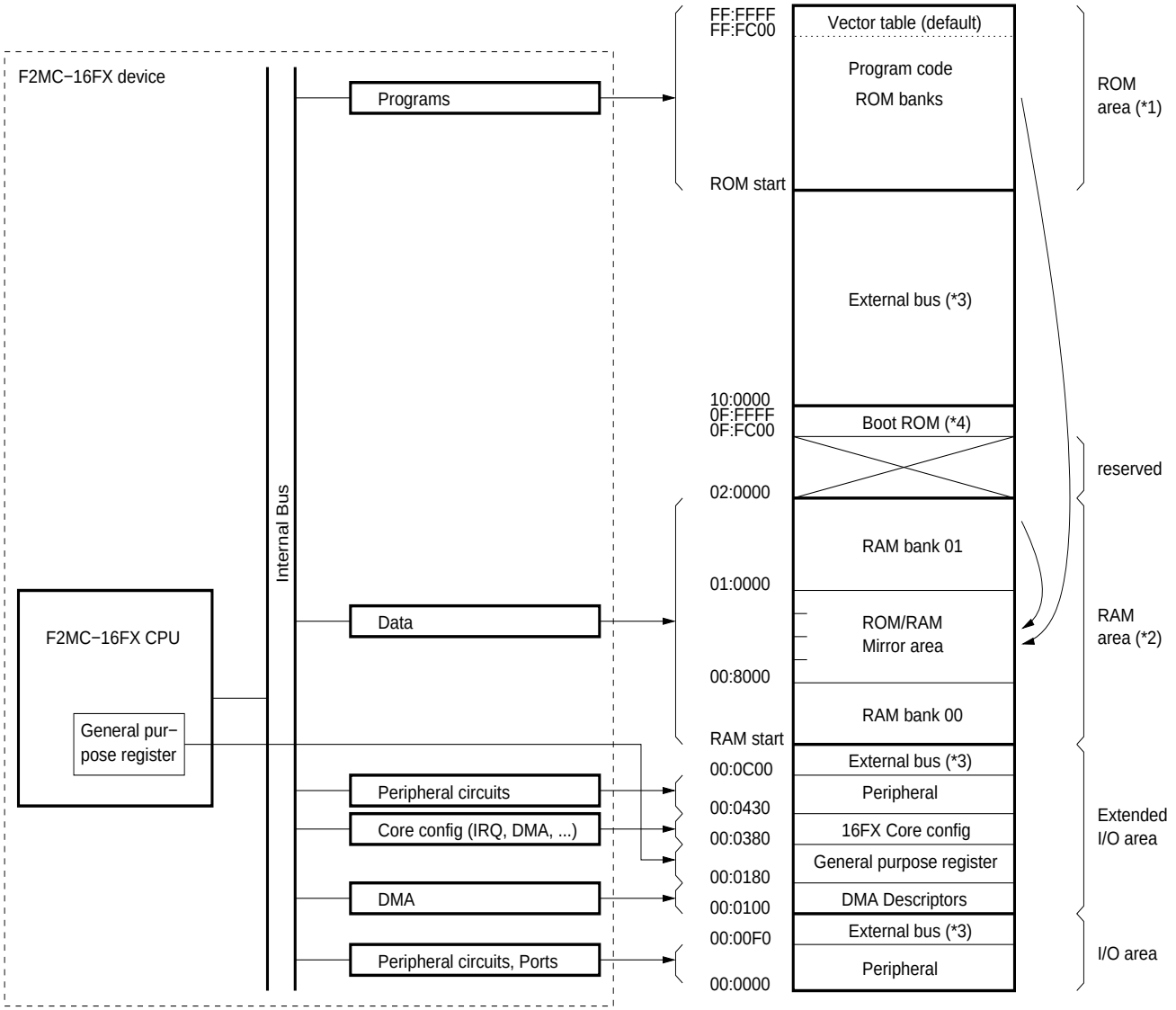
### 2.3.1 Memory Areas

All I/O addresses, programs and data are located in the 16-megabyte memory space of the F$^2$MC-16FX CPU. The CPU is able to access each resource through an address indicated by the 24-bit address bus.

#### *2.3.1.1 Memory areas in the F$^2$MC-16FX system*

Figure 2-4 shows a sample relationship between the F$^2$MC-16FX system and memory map.

Figure 2-4. Relationship between F$^2$MC-16FX system and memory




*1 The size of the internal ROM differs for each model
*2 The size of the internal RAM differs for each model
*3 Because the external bus interface is optional, access is not possible on all devices
*4 The size of the boot ROM differs for each model

### *2.3.1.2 I/O area*

The address range of the I/O area is from 00:0000$_H$ to 00:00FF$_H$. This address range can be used for direct I/O addressing by specifying the 8 bit address as operand together with the instruction. If an instruction is using the direct I/O addressing method, the register is accessed regardless of the values specified by the direct page register (DPR) or the data bank register (DTB).

#### Peripheral function control register I/O area (address: 00:0000$_H$ to 00:00EF$_H$)

These registers control the built-in peripheral functions and ports.

#### External IO area (address: 00:00F0$_H$ to 00:00FF$_H$)

These addresses access the direct IO address range of the external bus area.

The external bus interface is an optional component.

### 2.3.1.3 Extended I/O area

#### DMA descriptor area (address: $00:0100_H$ to $00:017F_H$)

This area retains the transfer modes, I/O addresses, transfer count, and buffer addresses.

This area can be used as ordinary RAM.

#### General-purpose register area (address: $00:0180_H$ to $00:037F_H$)

General-purpose registers (GPRs) used for 8-bit, 16-bit and 32-bit arithmetic operations and transfers are mapped into this area. This area can be used as ordinary RAM.

Though GPRs are memory mapped, direct access by instructions using these registers is greatly improved with respect to 16LX.

#### 16FX Core control register (address: $00:0380_H$ to $00:042F_H$)

This area is for CPU core internal configuration registers.

#### Peripheral function control register extended I/O area (address: $00:0430_H$ to $00:0BFF_H$)

Registers in this area control the built-in peripheral functions.

#### External bus area (address: $00:0C00_H$ to $00:0FFF_H$)

This area accesses the extended IO range of the external bus area in bank 0.

If less than 28 kByte RAM is implemented (this varies with the MCU device), the external bus area may be extended until the RAM start address subtracted by 1.

The external bus interface is an optional component.

### 2.3.1.4 RAM area

#### Data area in bank 00 (address: $00:1000_H$ to $00:7FFF_H$)

At this area static RAM is implemented with a size up to 28 kByte.

The size of internal RAM differs for each device. Thus the RAM start address depends on the implemented memory size.

#### Mirror area in bank 00 (address: $00:8000_H$ to $00:FFFF_H$)

This area is used to access the top-most section of 32 kByte ROM by default. The ROM mirror function is variable in size. There are up to 4 segments of 8 kByte each selectable as ROM mirror. The ROM bank to be accessed via bank 00 can be selected in the ROMM register.

If the ROM mirror function is switched off, RAM on bank 01 can be accessed via bank 00 in addition.

The mirror area in bank 00 can be used in order to support the small model of the C compiler. Since the least significant 16 bits are identical, the corresponding part for data access to the selected ROM or RAM bank can be referenced without using the far specification in the pointer declaration.

#### Data area in bank 1 (address: $01:0000_H$ to $01:FFFF_H$)

At this area static RAM is implemented with a size up to 64 kByte.

The size of internal RAM differs for each device.

#### 2.3.1.5 ROM area

**Program area (address: up to FF:FFFF$_H$)**

ROM is built in as an internal program area.

The size and the ROM start address of internal ROM differs for each device.

By default, addresses FF:FC00$_H$ to FF:FFFF$_H$ are allocated by the vector table.

#### 2.3.1.6 Vector table area

**Vector table area**

This area is used as the vector table for the interrupt vectors and the reset vector. The start address of the corresponding processing routine is set as data in each vector table entry.

The area of the vector table can be changed by the user program in the table base register (TBR). It can be located either in the RAM, ROM or external area. This area is located at 1 kByte starting at the address defined by the TBR. The address of the vector table area is

$VEC_{START}$ = TBR * $100_H$ up to

$VEC_{END}$ = TBR * $100_H$ + $3FF_H$.

The address of the vector table area is FF:FC00$_H$ to FF:FFFF$_H$ by default.

The vector table for vectored call instruction is located in the actual program bank specified by the PCB register. The CALLV instruction uses these 32 byte area, located at the top of the program bank from PCB:FFE0$_H$ to PCB:FFFF$_H$. The TBR has no influence on the location of the vectors used by the CALLV instruction.

Care must be taken, if the interrupt vector area (upper 8 entries of the interrupt vector table entries) is located at the top of the program bank and the CALLV instruction is used. Both vector tables then share the same location.

### 2.3.2 Linear Addressing Method

At linear addressing an entire 24-bit address is specified by an instruction. There are two types of linear addressing:

- 24-bit operand specification: Directly specifies a 24-bit address using operands.
- 32-bit register indirect specification: Indirectly specifies the 24 low-order bits of a 32-bit general-purpose register value as the address.

#### 2.3.2.1 24-bit operand specification

Figure 2-5 shows an example of 24-bit operand specification. Figure 2-6 shows an example of 32-bit register indirect specification.

Figure 2-5. Example of linear method (24-bit register operand specification)

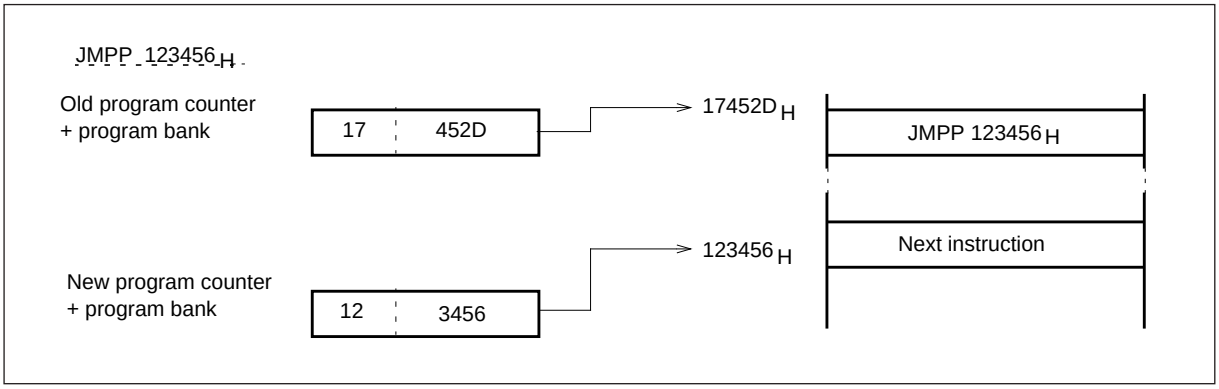
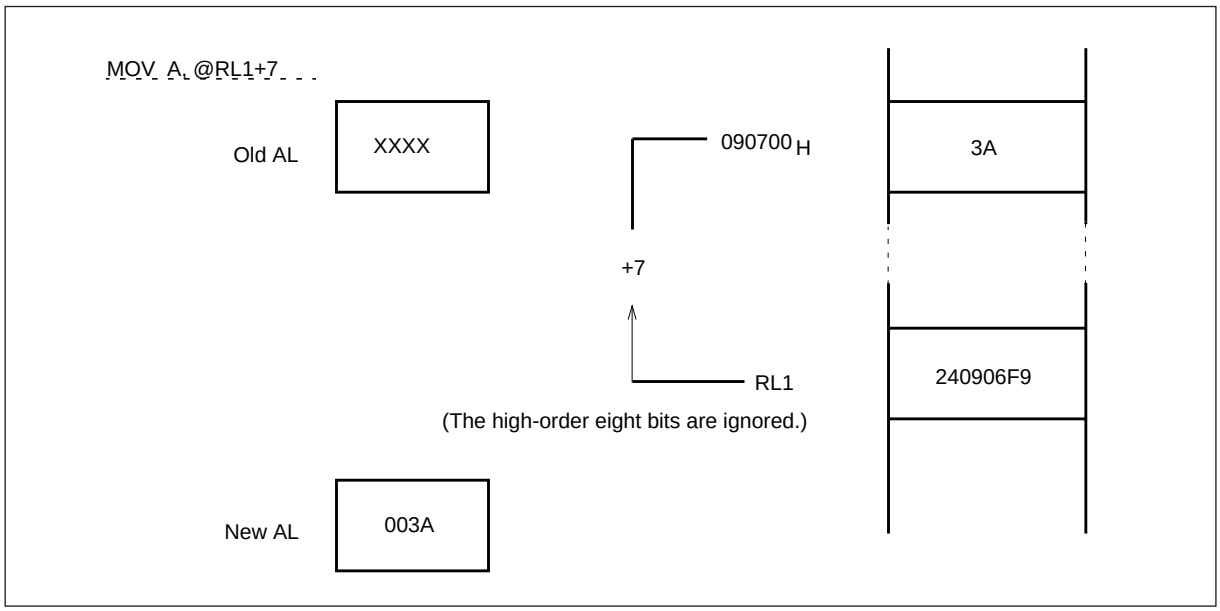


Figure 2-6. Example of linear method (32-bit register indirect specification)




## 2.3.3 Bank Addressing Method

In the bank addressing method the eight high-order bits of an address are specified by an appropriate bank register, and the remaining 16 low-order bits are specified by an instruction.

### *2.3.3.1 Bank addressing method*

In bank addressing, the 16-MByte space is divided into 256 banks (64 kBytes each). The five bank registers described below are used to specify the banks (upper 8 address bits) corresponding to each space:

#### Program bank register (PCB)

The 64-kByte bank specified by the PCB is called program (PC) space. The PC space contains instruction codes, the vector table of the CALLV instruction and immediate value data, for example.

#### Data bank register (DTB)

The 64-Kbyte bank specified by the DTB is called data (DT) space. The DT space contains readable/writable data, and control/data registers for internal and external resources.

#### User stack bank register (USB) and system stack bank register (SSB)

The 64-kByte bank specified by the USB or SSB is called stack (SP) space. The SP space is accessed when a stack access occurs during a push/pop instruction or interrupt register saving. The S flag in the condition code register determines the stack space USB or SSB to be accessed.

#### Additional bank register (ADB)

The 64-kByte bank specified by the ADB is called additional data (AD) space. The AD space, for example, contains data that cannot fit into the DT space.

Table 2-1 lists the default spaces used in each addressing mode, which are pre-determined to improve instruction coding efficiency.

Table 2-1. Default space

| Default space | Addressing mode |
|---|---|
| Program space | PC indirect, program access or branch |
| Data space | Addressing mode using @RW0, @RW1, @RW4, @RW5, @A, addr16 or dir |
| Stack space | Addressing mode using PUSHW, POPW, @RW3 or @RW7 |
| Additional data space | Addressing mode using @RW2 or @RW6 |

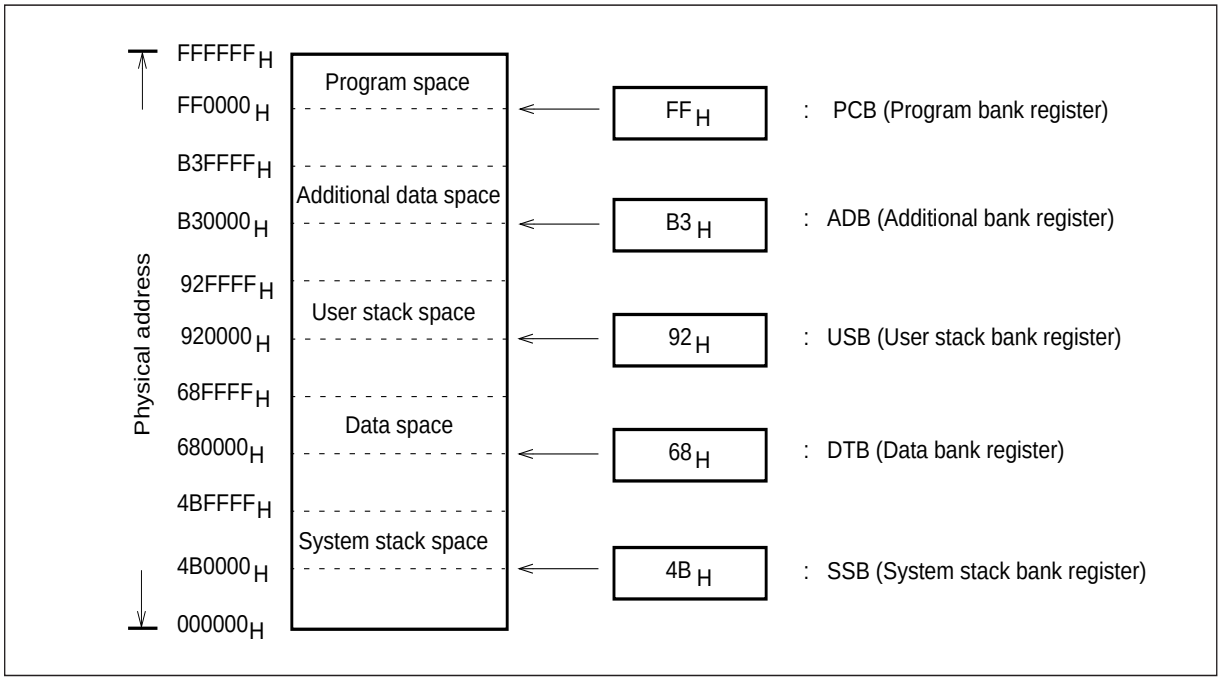
To use a non-default space for an addressing mode, specify a prefix code corresponding to a bank before the instruction. This enables access to the bank space corresponding to the specified prefix code.

Table 2-2. Bank selection prefixes

| Bank selection prefix | Selected Space |
|---|---|
| PCB | Program space |
| DTB | Data space |
| ADB | Additional data space |
| SPB | Either the system or the user stack space is used, according to the stack flag status upon selection. |

Table 2-7 is an example of a memory space divided into register banks and shows the physical address of each space.

Figure 2-7. Bank addressed memory space and values of bank registers




### 2.3.3.2 *Initialization of bank registers*

After reset, the DTB, USB, SSB and ADB are initialized to $00_H$. Because the reset address is fixed to the Boot ROM program start address $0F:FC00_H$, the PCB is initialized to $0F_H$.

In external vector mode a value specified by the reset vector on address $FF:FFDC_H$ is loaded when leaving the Boot ROM code execution. In internal vector mode the PCB is loaded with a fixed value defined by the product.

At starting user program from specified reset vector address, the DT, SP, and AD spaces are allocated in bank $00_H$ ($000000_H$ to $00FFFF_H$), and the PC space is allocated in the bank specified by the reset vector.

Table 2-3. Initialization of bank registers

| Bank register | Initialization by reset | Initialization by Boot ROM | |
|---|---|---|---|
| | | External vector mode | Internal vector mode |
| DTB | $00_H$ | $00_H$ | $00_H$ |
| USB | $00_H$ | $00_H$ | $00_H$ |
| SSB | $00_H$ | $00_H$ | $00_H$ |
| ADB | $00_H$ | $00_H$ | $00_H$ |
| PCB | $0F_H$ | Byte read from $FF:FFDE_H$. | Value defined by the product. |

## 2.3.4 Multi-byte Data in Memory Space

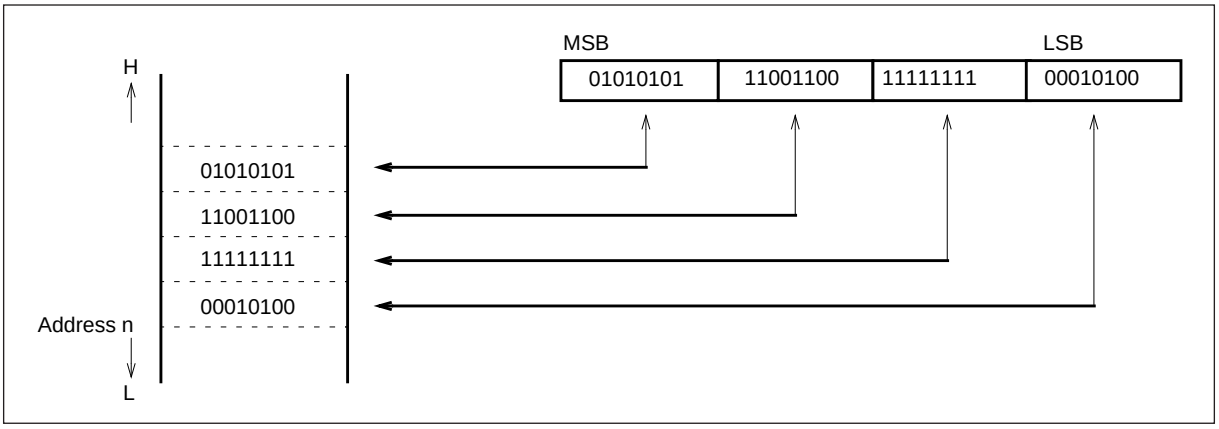
Data is written to memory from the low-order address on. Therefore, for a 32-bit data item, the low-order 16 bits are transferred before the high-order 16 bits.

If a reset signal is input immediately after the low-order bits are written, the high-order bits might not be written.

### *2.3.4.1 Multi-byte data allocation in memory space*

Figure 2-8 is a diagram of multi-byte data configuration in memory. The low-order eight bits of a data item are stored at address n, then next eight bits are stored at address address n+1, etc.

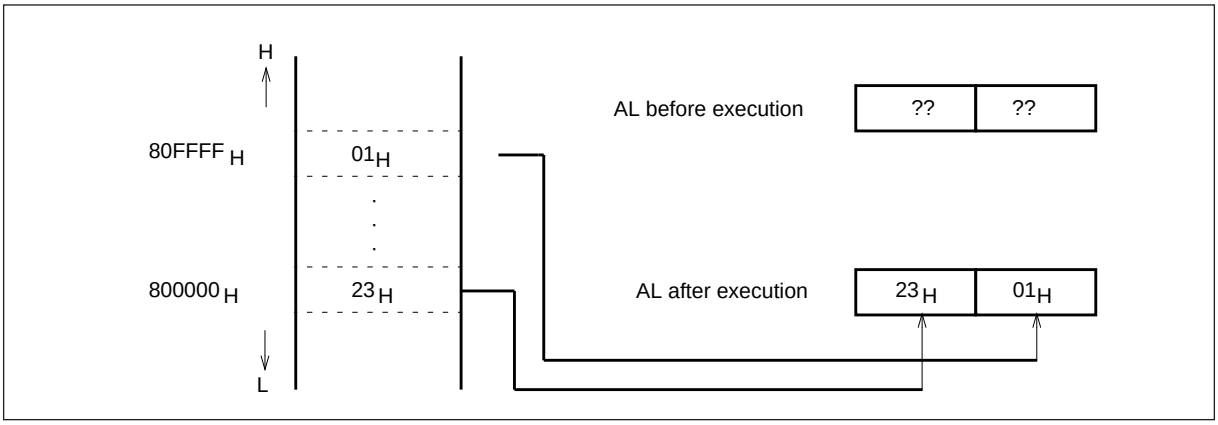
Figure 2-8. Sample allocation of multi-byte data in memory




### *2.3.4.2 Accessing multi-byte data*

Fundamentally, accesses are made within a bank. For an instruction accessing a multi-byte data item, address $FFFF_H$ is followed by address $0000_H$ of the same bank. Figure 2-9 is an example of an instruction accessing multi-byte data.

Figure 2-9. Execution of MOVW A, 080FFFFH




## 2.4 Special Registers

The F2MC-16FX CPU registers are classified into two types: special registers and general-purpose registers.

This section explains the special registers of the F2MC-16FX CPU. The special registers are dedicated internal hardware of the CPU, and they have specific use defined by the CPU architecture. These registers can be accessed without using an address. The register operations are defined by specific instructions.

2.4.1 Accumulator (A)

2.4.2 User Stack Pointer (USP) and System Stack Pointer (SSP)

2.4.3 Processor Status (PS)

2.4.4 Program Counter (PC)

2.4.5 Direct Page Register (DPR)

2.4.6 Bank register (PCB, DTB, ADB, USB, SSB)

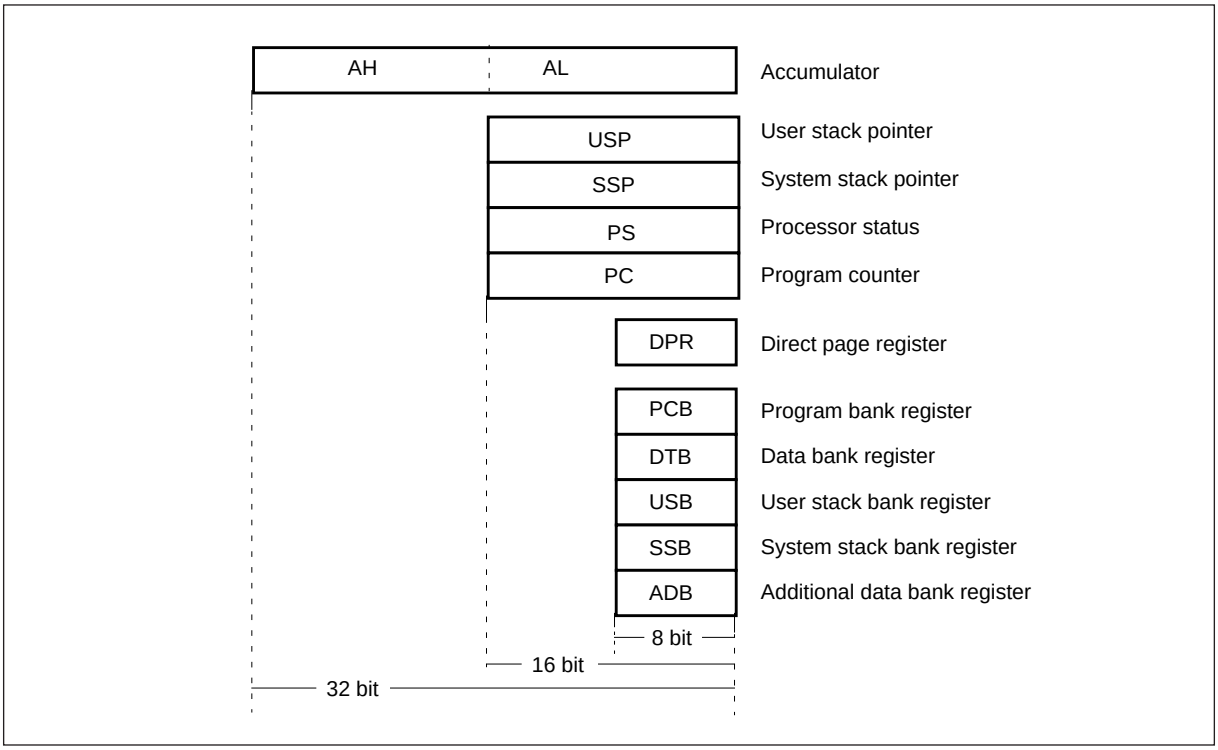
### Special registers

The F2MC-16FX CPU has the following special registers:

- Accumulator (A=AH:AL): Two 16-bit accumulators (can be used as a single 32-bit accumulator)
- User stack pointer (USP): 16-bit user stack pointer
- System stack pointer (SSP): 16-bit system stack pointer
- Processor status (PS): 16-bit register indicating the system status
- Program counter (PC): 16-bit register holding the address of the next instruction to be executed
- Program bank register (PCB): 8-bit register indicating the program bank
- Data bank register (DTB): 8-bit register indicating the data bank
- User stack bank register (USB): 8-bit register indicating the user stack bank
- System stack bank register (SSB): 8-bit register indicating the system stack bank
- Additional bank register (ADB): 8-bit register indicating the additional data bank
- Direct page register (DPR): 8-bit register indicating the page for direct access

Figure 2-10 is a diagram of the special registers.

Figure 2-10. Special registers




## 2.4.1 Accumulator (A)

The accumulator (A) register consists of two 16-bit arithmetic operation registers (AH and AL), and is used as a temporary storage for operation results and transfer data.

### *2.4.1.1 Accumulator (A)*

The A register consists of two 16-bit arithmetic operation registers (AH and AL). The A register is used as a temporary storage for operation results and transfer data. During 32-bit data processing, AH and AL are used together. Only AL is used for word processing in 16-bit data processing mode or for byte processing in 8-bit data processing mode (see Figure 2-11 and Figure 2-12). The data stored in the A register can be operated upon with the data in memory or registers (Ri, Rwi, or Rli). In the same manner as with the $F^2$MC-8L, when a word or shorter data item is transferred to AL, the previous data item in AL is automatically sent to AH (data preservation function). The data preservation function and operation between AL and AH help to improve processing efficiency.

When a byte or shorter data item is transferred to AL, the data is sign-extended or zero-extended and stored as a 16-bit data item in AL. The data in AL can be handled either as word or byte.

When a byte-processing arithmetic operation instruction is executed on AL, the high-order eight bits of AL before the operation are ignored. After the operation the heigh-order eight bits become zero.

The A register is not initialized by a reset. The A register holds an undefined value immediately after a reset.

Figure 2-11. Example of a 32-bit data transfer

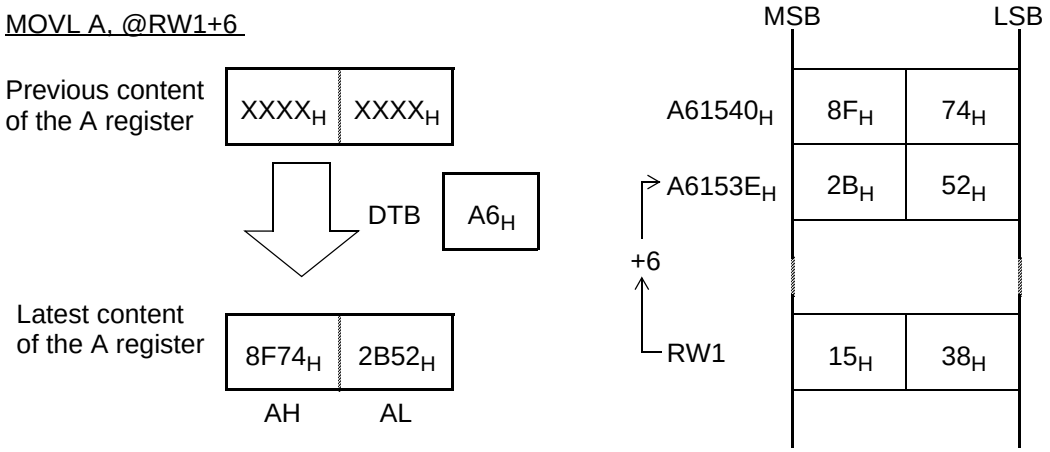
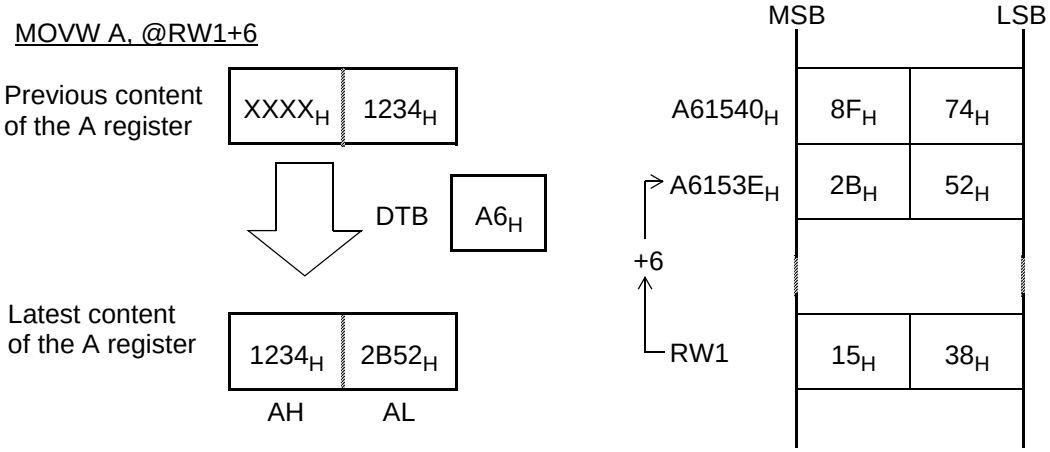



Figure 2-12. Example of AL-AH transfer by means of data preservation




## 2.4.2 User Stack Pointer (USP) and System Stack Pointer (SSP)

USP and SSP are 16-bit registers that indicate the memory addresses for saving and restoring data when a push/pop instruction or subroutine is executed.

### *2.4.2.1 User stack pointer (USP) and system stack pointer (SSP)*

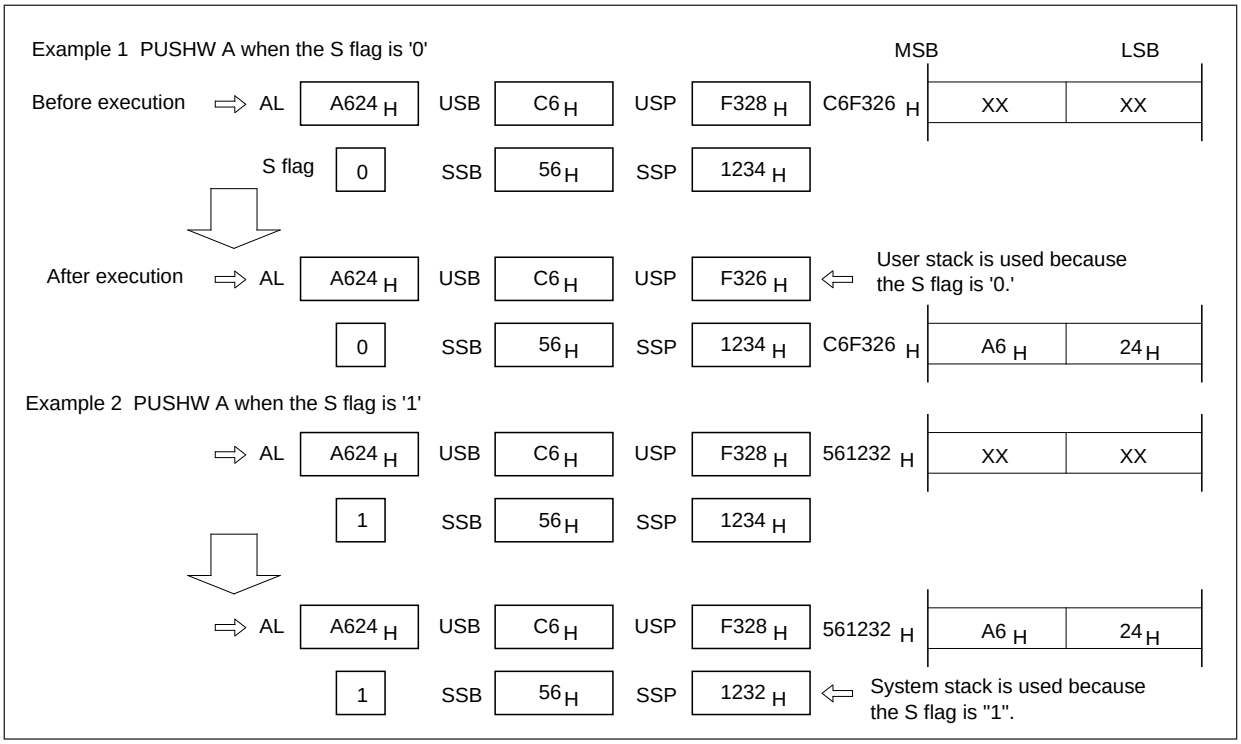
USP and SSP are 16-bit registers that indicate the memory addresses for saving and restoring data in the event of a push/pop instruction or subroutine execution. The USP and SSP registers are used by stack instructions.

The USP register is enabled when the S flag in the processor status register is “0”, and the SSP register is enabled when the S flag is “1” (see Figure 2-13). Since the S flag is set when an interrupt is accepted, register values are always saved in the memory area indicated by SSP during interrupt processing. SSP is used for stack processing in an interrupt routine, while USP is used for stack processing outside an interrupt routine. If the stack space is not divided, use only the SSP.

During stack processing, the high-order eight bits of an address are indicated by SSB (for SSP) or USB (for USP).

USP and SSP are not initialized by a reset. Instead, the values are undefined.

Figure 2-13. Stack manipulation instruction and stack pointer




**Note:** Specify an even-numbered address in the stack pointer whenever possible. An odd value will cause drawback in stack performance.

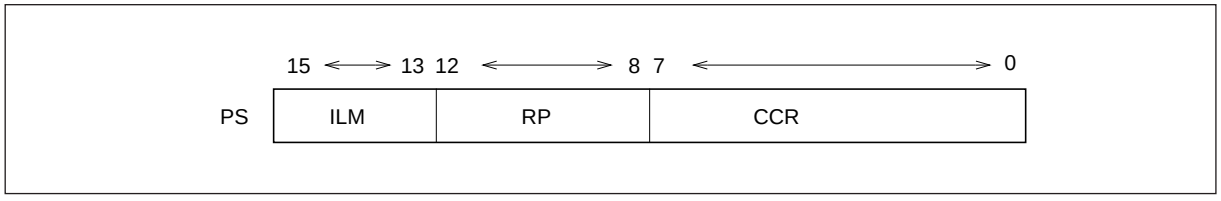
## 2.4.3 Processor Status (PS)

The PS register consists of the bits controlling the CPU Operation and indicating the CPU status.

### *2.4.3.1 Processor status (PS)*

As shown in Figure 2-14, the high-order byte of the PS register consists of a register bank pointer (RP) and an interrupt level mask register (ILM). The RP indicates the start address of a register bank. The low-order byte of the PS register is a condition code register (CCR), containing the flags to be set or reset depending on the results of instruction execution or interrupt occurrences.

Figure 2-14. Processor status (PS) structure




### *2.4.3.2 Condition code register (CCR)*

Figure 2-15 is the diagram of the condition code register configuration.

Figure 2-15. Condition code register (CCR) configuration

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| P | I | S | T | N | Z | V | C | PS: CCR |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | initial value after reset |
| 1 | 0 | 1 | X | X | X | X | X | value after Boot ROM execution |

### P: Priviledged mode flag:

P = 1 indicates user mode, P = 0 indicates privileged mode.

The P flag is cleared by a reset. However, the P flag will be set during execution of the Boot ROM code.

Only NMI, HW-INT9 (EDSU) and DSU interrupts will clear the P flag and disable all other hardware interrupts. If the P flag is cleared, ILM defines system interrupt levels of the privileged mode (P0 to P7). These interrupt levels have higher priority than any ILM setting in user mode (U0 to U7).

The P flag can be set by dedicated instructions (OR CCR #imm, POPW PS) or by restoring the processor status (RETI, JCTX @A). Restoring P=0 is not accepted, if P has been ’1’ before.

### I: Interrupt enable flag:

Interrupts other than software interrupts are enabled when the I flag is 1 and are masked when the I flag is 0. The I flag is cleared by a reset.

### S: Stack flag:

When the S flag is 0, USP is enabled as the stack pointer.

When the S flag is 1, SSP is enabled as the stack pointer.

The S flag is set by an interrupt reception or a reset.

### T: Sticky bit flag:

A value of "1" is set in the T flag when there is at least one "1" in the data shifted out from the carry after execution of a logical right/arithmetic right shift instruction, otherwise, "0" is set in the T flag.

In addition, "0" is set in the T flag when the shift amount is zero.

### N: Negative flag:

The N flag is set when the MSB of the operation result is "1", and is otherwise cleared.

### Z: Zero flag:

The Z flag is set when the operation result is all zeroes, and is otherwise cleared.

### V: Overflow flag:

The V flag is set when an overflow of a signed value occurs as a result of operation execution and is otherwise cleared.

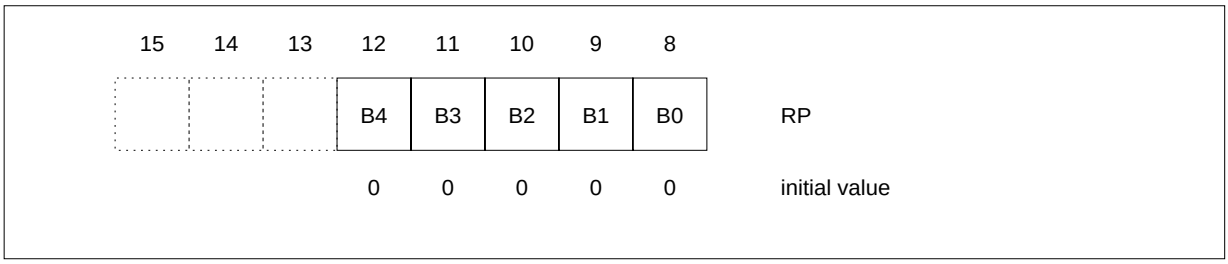
### C: Carry flag:

The C flag is set when a carry-up or carry-down from the MSB or LSB occurs as a result of operation execution, and is otherwise cleared.

### 2.4.3.3 Register bank pointer (RP)

The RP register indicates the relationship between the general-purpose registers of the F$^2$MC-16FX and the internal RAM addresses. Specifically, the RP register indicates the first memory address of the currently used register bank in the following conversion expression: [$00180_H$ + (RP)*$10_H$]. The RP register consists of five bits, and can take a value between $00_H$ and $1F_H$. Register banks can be allocated at addresses from $000180_H$ to $00037F_H$ in memory.

Figure 2-16. Register bank pointer (RP)

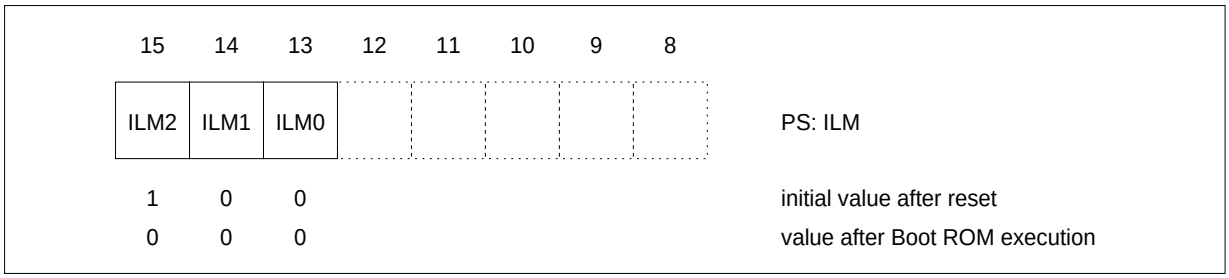



The RP register is initialized to all zeroes by a reset. An instruction may transfer an eight-bit immediate value to the RP register; however, only the low-order five bits of that data are used.

### 2.4.3.4 Interrupt level mask register (ILM)

The ILM register consists of three bits, indicating the CPU interrupt masking level. An interrupt request is accepted only when the priority of the interrupt is higher than that indicated by the ILM register and the P flag. Highest priority interrupt is level P0 and lowest priority is level U7. Therefore, for an interrupt to be accepted, its level value must be smaller than the current ILM value (see Figure 2-17). In addition, the P flag has to be considered. When an interrupt is accepted, the level value of that interrupt is set in the P flag and ILM register. Thus, an interrupt of the same or lower priority cannot be accepted subsequently.

Figure 2-17. Interrupt level register (ILM)




ILM is initialized to $100_B$ by a reset. However, during execution of the Boot ROM program ILM is set to $000_B$.

An instruction may transfer an eight-bit immediate value to the ILM register, but only the low-order three bits of that data are used (MOV ILM #imm, POPW PS, RETI, JCTX @A). If P=1 (in user level), any ILM change is possible. If P=0 (priviledged level), an ILM change is only accepted, if the new value defines a user level U0 to U7 (with P=1) or if the privileged level (P0 to P7) is increased. The lower levels of the privileged mode P0 to P7 can not be reached by execution of an instruction from a higher level. Writing 0 to the P flag and reducing the level with P=0 is only possible by NMI, HW-INT9 or a DSU interrupt.

**Notes:** The P flag can be understood as bit extension of ILM. Then it defines the most significant bit of the the interrupt level mask {P, ILM}.

After initialization with reset the CPU is in level P4. This disables all interrupts, including NMI, except for the DSU. After execution of the Boot ROM program the CPU is in level U0. Peripheral interrupts are disabled.

All privileged mode levels P0 to P7 are locked against entering or decreasing the level by an instruction. The levels P0 to P7 can only be increased. This protects the operation of HW-INT9, NMI and DSU operation. Only DSU can interrupt the NMI or mask its acceptance during a debug session.

The user levels U0 to U7 are backward compatible to $F^2MC$-16LX interrupt levels 0 to 7. The P flag is not writeable in a user level.

Table 2-4. Levels indicated by the P flag and interrupt level mask (ILM) register

| Level | P flag | ILM value | Acceptable interrupt level | | |
|---|---|---|---|---|---|
| P0 | 0 | 0 | none | Interrupts disabled | |
| P1 | 0 | 1 | Level < P1 | Interrupts disabled | |
| P2 | 0 | 2 | Level < P2 | Interrupts disabled | |
| P3 | 0 | 3 | Level < P3 | DSU | |
| P4 | 0 | 4 | Level < P4 | DSU | |
| P5 | 0 | 5 | Level < P5 | NMI, DSU | |
| P6 | 0 | 6 | Level < P6 | NMI, DSU | |
| P7 | 0 | 7 | Level < P7 | HW-INT9, NMI, DSU | |
| U0 | 1 | 0 | Level < U0 | User Interrupts disabled | HW-INT9, NMI, DSU |
| U1 | 1 | 1 | Level < U1 | User level 0 | |
| U2 | 1 | 2 | Level < U2 | User level 0-1 | |
| U3 | 1 | 3 | Level < U3 | User level 0-2 | |
| U4 | 1 | 4 | Level < U4 | User level 0-3 | |
| U5 | 1 | 5 | Level < U5 | User level 0-4 | |
| U6 | 1 | 6 | Level < U6 | User level 0-5 | |
| U7 | 1 | 7 | Level < U7 | User level 0-6 | |

## 2.4.4 Program Counter (PC)

The PC register is a 16-bit counter that indicates the low-order 16 bits of the memory address of an instruction code to be executed by the CPU.

### *2.4.4.1 Program counter (PC)*

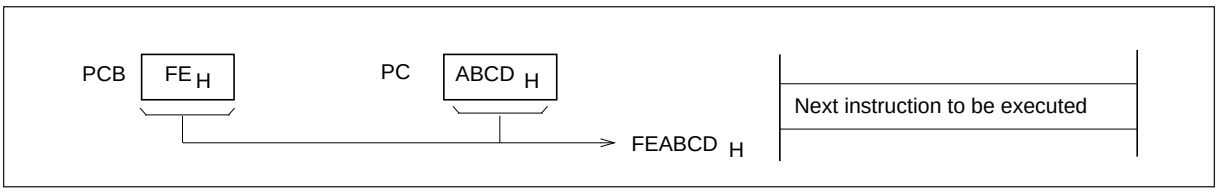
The PC register is a 16-bit counter that indicates the low-order 16 bits of the memory address of an instruction code to be executed by the CPU. The high-order eight bits of the address are indicated by the program bank register (PCB).

The PC register is updated by a branch instruction, subroutine call instruction, interrupt or reset. Within a linear program segment, the PC is incremented by the number of bytes of the last instruction.

The PC register can also be used as a base pointer for operand access.

Figure 2-18 shows the program counter.

Figure 2-18. Program counter




The reset address is fixed to the Boot ROM program start address of 0F:FC00$_H$. At reset, the PC is initialized to FC00$_H$ and the PCB is initialized to 0F$_H$.

In external vector mode a value specified by the reset vector on address FF:FFDC$_H$ is loaded when leaving the Boot ROM code execution. In internal vector mode the PCB and the PC are loaded with fixed values defined by the product.

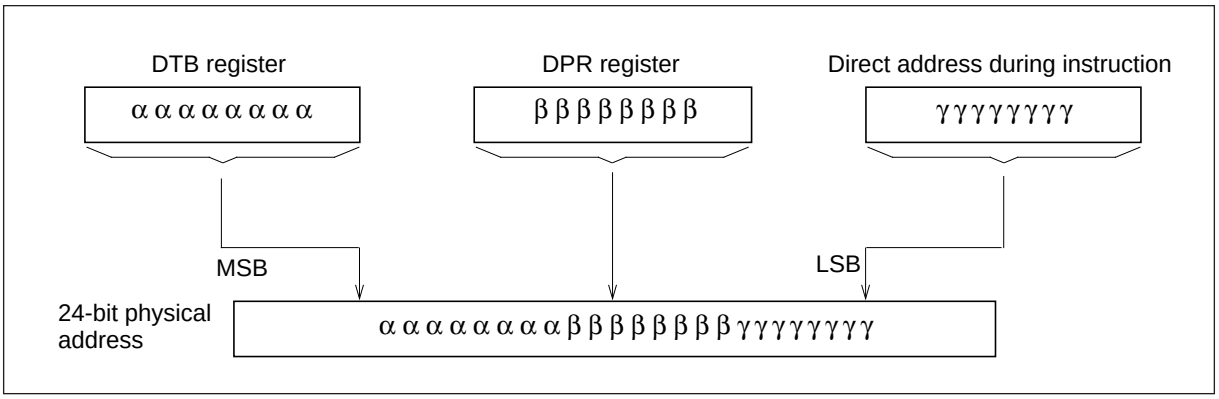
## 2.4.5 Direct Page Register (DPR)

The direct page register (DPR) specifies bits 8 to 15 (addr 8 to addr 15) of the operand address for direct addressing instructions.

### *2.4.5.1 Direct page register (DPR)*

DPR specifies bits 8 to 15 of the instruction operands in direct addressing mode as shown in Figure 2-19.

Figure 2-19. Generating a physical address in direct addressing mode




DPR is eight bits long, and is initialized to $01_H$ by a reset. DPR can be read or written to by an instruction.

## 2.4.6 Bank register (PCB, DTB, ADB, USB, SSB)

Each bank register indicates a memory bank where a program space, data space, user stack space or additional data space is allocated

### *2.4.6.1 Bank Register*

All bank registers are one byte long. Each bank register (PCB, DTB, USP, SSP, ADB) indicates the memory bank where the PC, DT, SP (user), SP (system), or AD space is allocated.

Bank registers other than PCB can be read and written to. PCB can be read but cannot be written to. The PCB register is updated upon the JMPP or CALLP instruction, branching to the entire 16 MByte space, upon the RETP or RETI instruction or upon an interrupt.

For details of the operation of bank registers, see section 2.3.3 Bank Addressing Method.

#### Program counter bank register (PCB)

Initial value: $0F_H$ after reset, and later a value from reset vector at user program start.

#### Data bank register (DTB)

Initial value: $00_H$.

#### User stack bank register (USB)

Initial value: $00_H$.

#### System stack bank register (SSB)

Initial value: $00_H$.

### Additional data bank register (ADB)

Initial value: $00_H$.

## 2.5 General-Purpose Registers

The $F^2$MC-16FX CPU registers are classified into two types: special registers and general-purpose registers.

This section explains the general-purpose registers (GPRs) of the F2MC-16FX CPU.

GPRs can be accessed without addressing, similar to the special registers. The register operations are defined by specific instructions.

In addition, the GPRs are accessible within the CPU address space. They can be used as ordinary memory.

2.5.1 Register Bank

2.5.2 Addressing General-Purpose Registers

### 2.5.1 Register Bank

A register bank consists of eight words. The register bank can be used as general-purpose registers for arithmetic operations.

#### *2.5.1.1 Register bank*

A register bank consists of eight words. The register bank can be used as the following general-purpose registers for arithmetic operations:

- byte registers R0 to R7 (8 bit),
- word registers RW0 to RW7 (16 bit) and
- long word registers RL0 to RL3 (32 bit).

In addition, registers of the register bank can be used as pointers and counters.

Table 2-5 lists the functions of the registers. Table 2-6 indicates the relationship between the registers.

Table 2-5. Register functions

| | |
|---|---|
| R0 to R7 | Used as operands of instructions.<br>Note: R0 is also used as a counter for shift or normalization instructions. |
| RW0 to RW7 | Used as pointers.<br>Used as operands of instructions.<br>Note: RW0 is used as a counter for string instructions. |
| RL0 to RL3 | Used as long pointers.<br>Used as operands of instructions. |

Table 2-6. Relationship between registers

<table>
<tr><td rowspan="4"></td><td>RW0</td><td rowspan="2">RL0</td></tr>
<tr><td>RW1</td></tr>
<tr><td>RW2</td><td rowspan="2">RL1</td></tr>
<tr><td>RW3</td></tr>
<tr><td>R0</td><td rowspan="2">RW4</td><td rowspan="4">RL2</td></tr>
<tr><td>R1</td></tr>
<tr><td>R2</td><td rowspan="2">RW5</td></tr>
<tr><td>R3</td></tr>
<tr><td>R4</td><td rowspan="2">RW6</td><td rowspan="4">RL3</td></tr>
<tr><td>R5</td></tr>
<tr><td>R6</td><td rowspan="2">RW7</td></tr>
<tr><td>R7</td></tr>
</table>

In the same manner as for an ordinary RAM area, the register bank values are not initialized by a reset. The status before a reset is maintained. When the power is turned on, the register bank will have an undefined value.

## 2.5.2 Addressing General-Purpose Registers

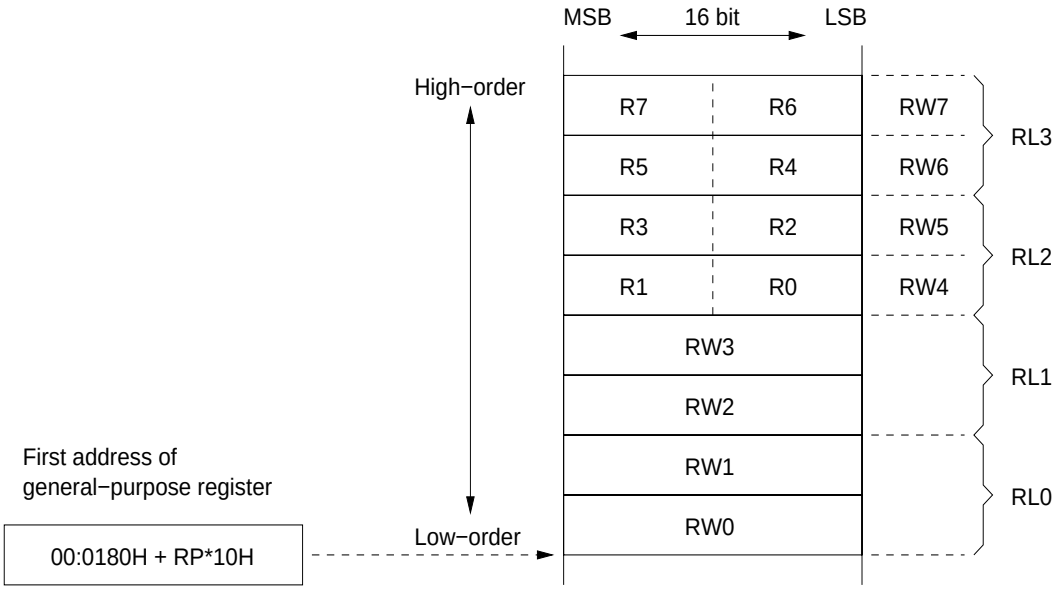
The general-purpose registers of the $F^2$MC-16FX use the register bank pointer (RP) to specify the currently used register bank. The register banks can be addressed between $00:0180_H$ and $00:037F_H$ in the memory space.

### *2.5.2.1 Addressing General-purpose registers*

The $F^2$MC-16FX general-purpose registers are located from addresses $000180_H$ to $00037F_H$ of memory space. The register bank pointer (RP) indicates which of the above addresses are currently being used as a register bank. Each bank has the following three types of registers. These registers are mutually dependent as described in Figure 2-20.

- R0 to R7: 8-bit general-purpose registers
- RW0 to RW7: 16-bit general-purpose registers
- RL0 to RL3: 32-bit general-purpose registers

Figure 2-20. General-purpose registers




The relationship between the high-order and low-order bytes of a byte or word register is expressed as follows:

$RW_{(i+4)} = R_{(i*2+1)}*256+R_{(i*2)}$ [i=0 to 3]

The relationship between the high-order and low-order bytes of RLi and RW can be expressed as follows:

$RL_{(i)} = RW_{(i*2+1)}*65536+RW_{(i*2)}$ [i=0 to 3]

## 2.6 Prefix Codes

Placing a prefix code before an instruction partially changes the operation of that instruction. There are three types of prefix codes:

- Bank selection prefix
- Common register bank prefix
- Flag change inhibit prefix

This section explains each prefix.

2.6.1 Bank Selection Prefix

2.6.2 Common Register Bank Prefix (CMR)

2.6.3 Flag Change Inhibit Prefix (NCC)

2.6.4 Prefix Code Restrictions

### 2.6.1 Bank Selection Prefix

Placing a bank selection prefix before an instruction enables that instruction to access any specified memory space, regardless of the addressing method being used.

#### *2.6.1.1 Bank select prefix*

The memory space used for accessing data is determined for each addressing mode.

When a bank select prefix is placed before an instruction, the memory space used for accessing data by that instruction can be selected regardless of the addressing mode.

Table 2-7 lists the bank select prefixes and the corresponding memory spaces.

Table 2-7. Bank select prefix

| Bank select prefix | Space selected |
|---|---|
| PCB | PC space |
| DTB | Data space |
| ADB | AD space |
| SPB | Either the SSP or USP space is used according to the stack flag value. |

Note, that the effect of the prefixes varies for following instructions:

### Transfer instructions (I/O access)

| | | | |
|---|---|---|---|
| MOV A, io | MOV io, A | MOVW A, io | MOVW io, A |
| MOV io, #imm8 | MOV io, imm16 | MOVX A, io | |

The I/O space is accessed, regardless of any bank selection prefix is specified.

### Bit operation instructions (I/O access)

| | | | |
|---|---|---|---|
| MOVB A, io:bp | MOVB io:bp, A | SETB io:bp | CLRB io:bp |
| BBC io:bp, rel | BBS io:bp, rel | WBTC io:bp | WBTS io:bp |

The I/O space is accessed, regardless of any bank selection prefix is specified.

### Branch instructions

| |
|---|
| RETI |

The system stack bank (SSB) is used, regardless of any bank selection prefix is specified.

### String operation instructions

| | | | |
|---|---|---|---|
| FILS | FILSW | SCEQ | SCWEQ |
| MOVS | MOVSW | | |

The bank register specified by the operand is used, regardless of any bank selection prefix is specified.

### Stack operation instructions

| | | |
|---|---|---|
| PUSHW | POPW | POPW PS |
| LINK | UNLINK | |

The system stack bank (SSB) or user stack bank (USB) is used according to the S flag, regardless of any bank selection prefix is specified.

**Processor status word change / interrupt control instructions**

| | | | |
|---|---|---|---|
| AND CCR, #imm8 | OR CCR, #imm8 | MOV ILM, #imm8 | POPW PS |

The specified bank selection prefix affects the next instruction.

## 2.6.2 Common Register Bank Prefix (CMR)

The common register bank prefix (CMR) can be placed before an instruction that accesses a register bank. Then the target register of that instruction is changed to the common register bank. This is the register bank selected with RP=0, regardless of the current value of the register bank pointer (RP).

### *2.6.2.1 Common register bank prefix (CMR)*

To simplify data exchange between multiple tasks, it is necessary to establish a relative easy method of accessing the same pre-determined register bank, regardless of the RP value. To achieve this, the $F^2$MC-16FX has a so called common register bank that can be used by all tasks. The common bank is located between address $00:0180_H$ and $00:018F_H$. It is selected when the RP value is set to ’0’.

When CMR is placed before an instruction that accesses a register bank, that instruction accesses the common bank, regardless of the current RP value.

Note, that the effect of the prefix varies for the following instructions:

**String operation instructions**

| | | | |
|---|---|---|---|
| FILS | FILSW | SCEQ | SCWEQ |
| MOVS | MOVSW | | |

Placing a CMR prefix before the string instructions listed above is ignored. The counter register RW0 is always referenced by using the actual value of the register bank pointer RP, regardless of a CMR prefix is specified or not.

**Stack operation instructions**

| | |
|---|---|
| LINK | UNLINK |

The system stack bank (SSB) or user stack bank (USB) is used according to the S flag, regardless of any bank selection prefix is specified.

**Processor status word change / interrupt control instructions**

| | | | |
|---|---|---|---|
| AND CCR, #imm8 | OR CCR, #imm8 | MOV ILM, #imm8 | POPW PS |

The specified prefix affects the next instruction.

## 2.6.3 Flag Change Inhibit Prefix (NCC)

Flag changes associated with the execution of an instruction can be inhibited by placing a flag change inhibit prefix code (NCC) before that instruction.

### *2.6.3.1 Flag change inhibit prefix code (NCC)*

To disable flag changes, use the flag change inhibit prefix code (NCC). Placing NCC before an instruction disables flag changes associated with that instruction.

Note, that the effect ot the prefix varies for the following instructions:

### String operation instructions

| | | | |
|---|---|---|---|
| FILS | FILSW | SCEQ | SCWEQ |

Placing a NCC prefix before the string instructions listed above is ignored. Flags in the CCR are changing according to the instruction specifications, regardless of a NCC prefix is specified or not.

### Processor status word change / interrupt control instructions

| | | | |
|---|---|---|---|
| AND CCR, #imm8 | OR CCR, #imm8 | MOV ILM, #imm8 | POPW PS |

The specified prefix affects the next instruction.

### Branch instructions

| | | | |
|---|---|---|---|
| INT #vct8 | INT9 | INT addr16 | INTP addr24 |
| RETI | | | |

CCR changes according to the instruction specifications regardless of a NCC prefix is specified or not.

### Context switch instruction

| |
|---|
| JCTX @A |

CCR changes according to the instruction specifications regardless of the prefix.

## 2.6.4 Prefix Code Restrictions

This section lists the instructions, which reject interrupt requests during execution. If a prefix code is placed before such an instruction, the setting of the prefix code remains effective until the first instruction is executed after this interrupt rejecting instruction.

If conflicting prefix codes are specified consecutively, only the last specified prefix code is valid.

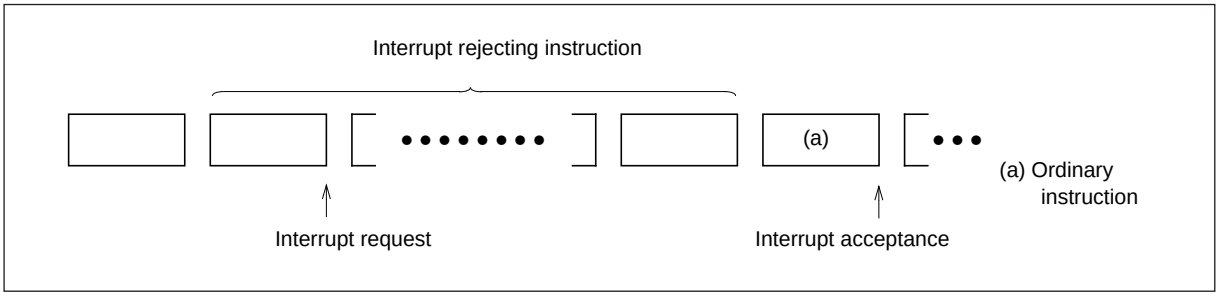
### *2.6.4.1 Interrupt rejecting instructions*

Interrupt requests are not accepted during execution of following instructions and prefix codes:

| | | | |
|---|---|---|---|
| MOV ILM, #imm8 | AND CCR, #imm8 | OR CCR, #imm8 | POPW PS |
| PCB | ADB | DTB | SPB |
| NCC | CMR | | |

If a valid interrupt request occurs during execution of any of the above instructions, the interrupt is postponed until an instruction other than the above is executed. For details, see Figure 2-21.

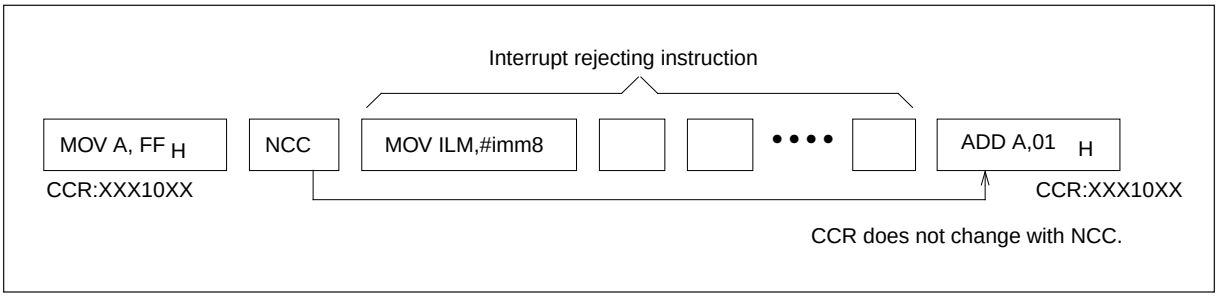
Figure 2-21. Interrupt rejecting instruction




### 2.6.4.2 Restrictions on interrupt rejecting instructions and prefix codes

When a prefix code is placed before an interrupt rejecting instruction, the prefix code affects the first instruction after the code other than the interrupt rejecting instruction. For details, see Figure 2-22.

Figure 2-22. Interrupt rejecting instructions and prefix codes

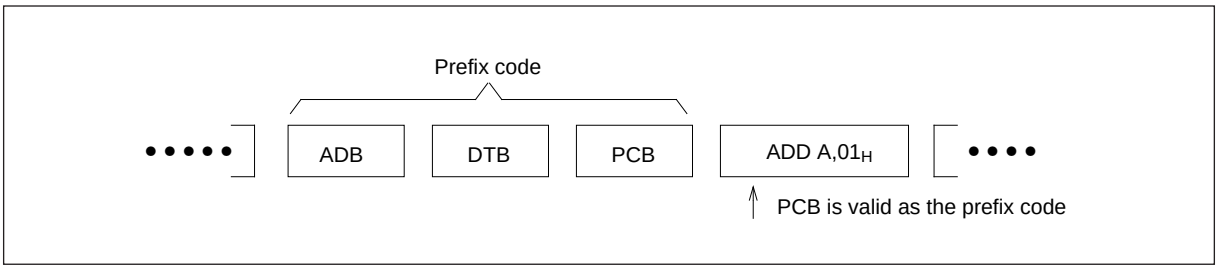



### 2.6.4.3 Multiple prefix codes in succession

When conflicting prefix codes are placed consecutively, the last one is valid.

In the figure below, conflicting prefix codes are PCB, ADB, DTB, and SPB. For details, see Figure 2-23.

Figure 2-23. Consecutive prefix codes




Prefixes CMR and NCC are not conflicting with bank selection prefixes PCB, ADB, DTB and SPB. CMR is not conflicting with NCC.

# 3. Interrupts

This chapter explains the interrupt functions and operations.

3.1 Outline of Interrupts

3.2 Interrupt Vector

3.3 Interrupt Control Registers (ICR)

3.4 Non Maskable Interrupt (NMI)

3.5 Interrupt Flow

3.6 Hardware Interrupts

3.7 Software Interrupts

3.8 Multiple interrupts

3.9 Exceptions

## 3.1 Outline of Interrupts

The F$^2$MC-16FX has interrupt functions that terminate the currently executed program and transfer control to another specified program when a specific event occurs. There are four types of interrupt functions:

- Hardware interrupt: Interrupt processing due to an internal resource event
- Software interrupt: Interrupt processing due to a software event (instruction)
- Exception: Handling of an operation exception
- DMA: Data transfer without CPU interaction due to an internal resource event.

### 3.1.1 Hardware interrupts

A hardware interrupt is activated by an interrupt request from an internal resource. A hardware interrupt request occurs when both the interrupt request flag and the interrupt enable flag in an internal resource are set.

#### Specifying an interrupt level

An interrupt level can be specified for the hardware interrupt. To specify an interrupt level, use the level setting bits (IL0, IL1, and IL2) in the interrupt control register ICR.

For each hardware interrupt its own interrupt level (IL) can be specified. Access to a dedicated IL can be done by setting the index IX. Both IX and IL are accessible through the interrupt control register ICR.

#### Masking a hardware interrupt request

A hardware interrupt request can be masked by using the I flag and the ILM bits (ILM0, ILM1, and ILM2). The interrupt is executed only, when the I flag is set and the value of the interrupt level IL is smaller than the interrupt level mask ILM. In addition the P flag has to be set for hardware interrupt acceptance. P, I and ILM are parts of the processor status word PS of the CPU.

When an unmasked interrupt request occurs, the CPU saves 12 bytes of data that consists of registers PS, PC, PCB, DTB, ADB, DPR, and A in the memory area indicated by the system stack bank and pointer registers (SSB and SSP).

### 3.1.2 Software interrupts

Interrupts requested by executing the INT instruction are software interrupts. An interrupt request by the INT instruction does not have an interrupt request or enable flag. An interrupt request is issued always by executing the INT instruction.

No interrupt level is assigned to the INT instruction. Therefore, ILM is not updated when the INT instruction is used. Instead, the I flag is cleared and the continuing interrupt requests are suspended.

### 3.1.3 Exceptions

Following software exceptions can be processed:

- Undefined instruction
- INT9
- INTE (only available on the EVA device)

Following hardware exceptions can be processed:

- NMI
- HW-INT9 (embedded debug support)
- DSU break factors (only available on the EVA device)

Exception processing is basically the same as interrupt processing. When an exception is detected during instruction execution, exception processing is performed. In general, exception processing occurs as a result of an unexpected operation. Therefore, use exception processing only for debugging programs or for activating recovery software in an emergency case.

### 3.1.4 Direct memory access (DMA)

#### DMA Function

$F^2$MC-16FX offers a DMA function to automatically transfer data between peripheral resources and memory upon an interrupt.

The number of DMA channels is device dependent.

When a DMA data transfer of a specified count is completed, an interrupt processing program is automatically executed on the original IRQ channel. The handling of such an interrupt by DMA completion is same as for standard type of hardware interrupts.

For a detailed decription of DMA, please refer to DMA chapter on page 95.

## 3.2 Interrupt Vector

Hardware and software interrupts use the same vector table. The execution of interrupt service routines can be triggered by asserting the specific IRQ line or by executing the INT instruction and specifying the number of the interrupt vector. Interrupt vectors are allocated between addresses as shown in Table 3-2. The location of the Interrupt vector table can be selected by the Table base register.

### 3.2.1 Interrupt vector

#### Interrupt Vector Table Base Register (TBR)

Figure 3-1. Interrupt vector Table base register (TBR)

| | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address: 0x3A3, 0x3A2 | TB23 | TB22 | TB21 | TB20 | TB19 | TB18 | TB17 | TB16 | TB15 | TB14 | TB13 | TB12 | TB11 | TB10 | – | – | TBRH, TBRL |
| Read/Write: | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R) | (R) | |
| Initial Value: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | |

R: readable
R/W: readable and writable

The Table Base Register allows to relocate the interrupt vector table to any memory location in steps of 1 kBytes.

The value of the TBR defines the most significant 14 bits TB[23:10] of the 24-bit start address of the interrupt vector table. The least significant bits of TB[9:0] are fixed to ’0’.

The table base register TBR is initialized with $FFFC_H$ at reset, which results in an initial table base TB = $FFFC00_H$.

The interrupt vector table has a size of 1 kByte (256 vector entries).

Table 3-1. Examples for TBR

| TBR value | start address of Interrupt vector table (table base) | end of Interrupt vector table | Comment |
|---|---|---|---|
| $FFFC_H$ | FF.FC00H | FF.FFFFH | as F2MC-16LX (default) |
| $FB00_H$ | FB.0000H | FB.03FFH | start of ROM bank FB |
| $00FC_H$ | 00.FC00H | 00.FFFFH | end of bank 00 (can be external memory) |
| $0010_H$ | 00.1000H | 00.13FFH | inside of RAM-area |
| $0000_H$ | 00.0000H | 00.03FFH | do not use, because of IO-area |

#### Interrupt Vector Table

The interrupt vector table referenced during interrupt processing is assigned to addresses 256*TBR to 256*TBR+$3FF_H$ in memory. The reset defaults are from $FFFC00_H$ to $FFFFFF_H$ for the location of the vector table. If the vector table should not be located at top of ROM memory, another TBR value has to be configured.

Hardware interrupts, exceptions and software interrupts share the same vector table. Hence the interrupt service routine can either be called by a hardware interrupt or by the corresponding software interrupt.

The three bytes of each start address of the interrupt service routines have to be written to the appropriate interrupt vectors (VecAddr = 4*(255-INT#) + 256*TBR).

Table 3-2. Interrupt vector table

| Interrupt / Vector number | Vector address | Index of level register in ICR | Hardware IRQ / Interrupt cause |
|---|---|---|---|
| INT 0<br>CALLV 0/1 (*1) | TB+3FC$_H$ | -- | -- |
| INT 1<br>CALLV 2/3 (*1) | TB+3F8$_H$ | -- | -- |
| INT 2<br>CALLV 4/5 (*1) | TB+3F4$_H$ | -- | -- |
| INT 3<br>CALLV 6/7 (*1) | TB+3F0$_H$ | -- | -- |
| INT 4<br>CALLV 8/9 (*1) | TB+3EC$_H$ | -- | -- |
| INT 5<br>CALLV 10/11 (*1) | TB+3E8$_H$ | -- | -- |
| INT 6<br>CALLV 12/13 (*1) | TB+3E4$_H$ | -- | -- |
| INT 7<br>CALLV 14 (*1) | TB+3E0$_H$ | -- | -- |
| INT 8<br>MODE Byte | TB+3DC$_H$ | -- | Reset |
| INT 9 | TB+3D8$_H$ | -- | INT9 instruction |
| INT 10 | TB+3D4$_H$ | -- | Undefined Instruction Exception |
| INT 11 | TB+3D0$_H$ | -- | NMI |
| INT 12 | TB+3CC$_H$ | IL12 | Delayed Interrupt |
| INT 13 | TB+3C8$_H$ | IL13 | RC Timer |
| INT 14 | TB+3C4$_H$ | IL14 | MC Timer |
| INT 15 | TB+3C0$_H$ | IL15 | SC Timer |
| INT 16 | TB+3BC$_H$ | IL16 | <reserved for PLL Unlock> |
| INT 17 | TB+3B8$_H$ | IL17 | Device specific peripheral. |
| INT 18 | TB+3B4$_H$ | IL18 | Device specific peripheral. |
| INT 19 | TB+3B0$_H$ | IL19 | |
| ... | ... | ... | |
| INT 254 | TB+004$_H$ | -- | -- |
| INT 255 | TB+000$_H$ | -- | -- |

(*1) When the program bank register (PCB) is same as TBRH, the CALLV instruction vector area overlaps the vector table of the INT#0 to INT#7 instruction. Ensure that the CALLV instruction does not use the same address as that of the INT#0 to INT#7 instruction, or do not use INT#0 to INT#7.

## 3.3 Interrupt Control Registers (ICR)

For each peripheral resource that has an interrupt function, there is an interrupt control register (ICR). The interrupt control register sets the interrupt level (IL) for the peripheral resource it is assigned to.

### 3.3.1 Interrupt control register (ICR)

Figure 3-2 is a diagram of the bit configuration of the interrupt control register.

Figure 3-2. Interrupt control register (ICR)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|
| Address: 0x3A1 | IX7 | IX6 | IX5 | IX4 | IX3 | IX2 | IX1 | IX0 | ICR: IX |
| Read/Write: | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Initial Value: | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|
| Address: 0x3A0 | − | − | − | − | − | IL2 | IL1 | IL0 | ICR: IL |
| Read/Write: | (−) | (−) | (−) | (−) | (−) | (R/W) | (R/W) | (R/W) | |
| Initial Value: | X | X | X | X | X | 1 | 1 | 1 | |

−: no access
R/W: readable and writable

#### ICR [bits 15 to 8]: IX[7:0] Index of the interrupt level (IL) to be accessed

These bits are readable and writable, and specify the index of the interrupt level of the corresponding internal resource. It selects the number of the interrupt level to be accessed. IL[n] belongs to the peripheral interrupt request number IRQ[n], which both are related to the interrupt INT[n].

The system interrupts INT0 to INT11 have a fixed priority and thus have no interrupt levels. Writing to interrupt levels below the index of 12 has no effect, reading returns an undefined value. The same restriction applies for not available hardware interrupts above a device dependent maximum interrupt number.

Figure 3-3 illustrates the access to level registers by the IX pointer. The dashed line around IX and the selected IL shows the actual contents of the ICR.

Level configuration is written to or read from IL, where IX points to. To write the level configuration to a dedicated IL, specify the according index by writing IX before or simultaneously by word access. To read from a dedicated IL, IX must be written before reading IL.

#### Caution for the use of concurrent tasks:

In the case of concurrent tasks accessing the interrupt level information, be careful at the handling of the indexed access:

- Use word access to write information to ICR:IX and ICR:IL simultaneously.
- At read access, set the index ICR:IX and read the whole ICR register using word access. Check the ICR:IX value to match the intended index to be read for validation of the correct ICR:IL entry.

#### ICR [bits 7 to 3]: unused bits

Read access returns an undefined value.

Write always 0 to these bits.

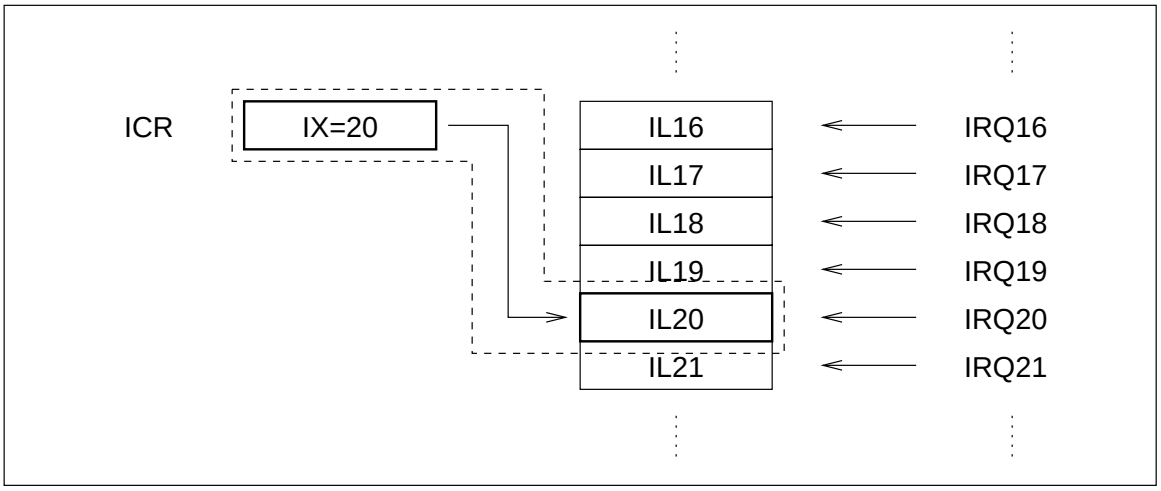
### ICR [bits 2 to 0]: IL[2:0] Interrupt level setting bits

These bits are readable and writable, and specify the interrupt level of the corresponding internal resources. Upon a reset, these bits are initialized to level 7 (no interrupt). Table 3-3 describes the relationship between the interrupt level setting bits and interrupt levels.

Table 3-3. Interrupt level setting bits and interrupt levels

| IL2 | IL1 | IL0 | Level |
|---|---|---|---|
| 0 | 0 | 0 | 0 (Strongest) |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 (Weakest) |
| 1 | 1 | 1 | 7 (No interrupt) |

Figure 3-3. Relationship between index (IX), level (IL) and IRQ number, example for IX = 20



## 3.4 Non Maskable Interrupt (NMI)

The F2MC-16FX CPU has a non maskable interrupt. The feature of the external NMI pin can be enabled, it's level can be defined and a flag to quit the NMI request is provided.

### 3.4.1 NMI control status register (NMI)

Figure 3-2 is a diagram of the bit configuration of the interrupt control register.

Figure 3-4. NMI control status register (NMI)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|
| Address: 0x3A5 | – | – | – | – | FIX9 | LEV | EN | FLAG | NMI |
| Read/Write: | (–) | (–) | (–) | (–) | (R/W) | (R/W) | (R/W1) | (R/W0) | |
| Initial Value: | – | – | – | – | 0 | 1 | 0 | X | |

–: no access
R/W: readable and writable
R/W!: readable, bit can be set only
R/W0: readable, bit can be cleared only

#### NMI [bits 15 to 10]: unused bits

Read access returns an undefined value.

Write always 0 to these bits.

#### NMI [bit 11]: FIX9 - Fix the location of the INT9 vector.

The FIX9 bit changes to an alternative location of the interrupt vector of INT9.

If it is set to '1', the interrupt vector is obtained from address 0F:FFD8H. If it is set to '0' the INT9 vector location is defined by the TBR+3D8H.

At reset the FIX9 bit is initialized to '0'.

The function is used by the firmware executed at device startup. Making the interrupt vector of INT9 independent from the TBR improves the reliability of the embedded debug support unit (EDSU).

#### NMI [bit 10]: LEV - Signal activity level of the NMI pin.

The LEV bit defines the signal activity level of the NMI pin. A value of '1' defines logic high active input, a value of '0' define logic low active input of the NMI pin.

If the EN bit is not set, the LEV bit is readable and writable. If the EN bit is set, the state of LEV is locked. In that case LEV can only be read, writing to LEV with EN='1' has no effect.

At reset the LEV bit is initialized to 1, thus the NMI pin is active high by default.

#### NMI [bit 9]: EN - Non Maskable Interrupt feature enable bit

The EN bit enables the feature to have a dedicated NMI pin. If EN is set to '0' the device has no NMI. The CPU will not react on signal level change at the NMI input. The pin can be used for an other function or general purpose. If EN is set to '1', the NMI pin is enabled. The CPU branches to the NMI exception processing, if an active signal level is detected at the NMI pin (defined by the LEV bit).

If EN is set to ’1’, both the LEV and EN bits are locked for writing. Neither the signal level can be changed nor the NMI can be disabled after the NMI feature was enabled once. Only a device reset can change the EN bit back to ’0’.

At reset the EN bit is initialized to ’0’, thus the NMI feature is not enabled by default.

The LEV and EN bits must not be activated at same time (changed using the same access). EN must be enabled individually at last. Otherwise, an NMI can be caused due to relaxation time of the spike filter.

### NMI [bit 8]: FLAG - Non Maskable Interrupt Flag

The NMI FLAG stores an asynchronous event of the NMI occurrence at the NMI pin.

A spike filter is used to filter out short pulses for spike suppression. The polarity of the pulses depends on the definition in the LEV bit.

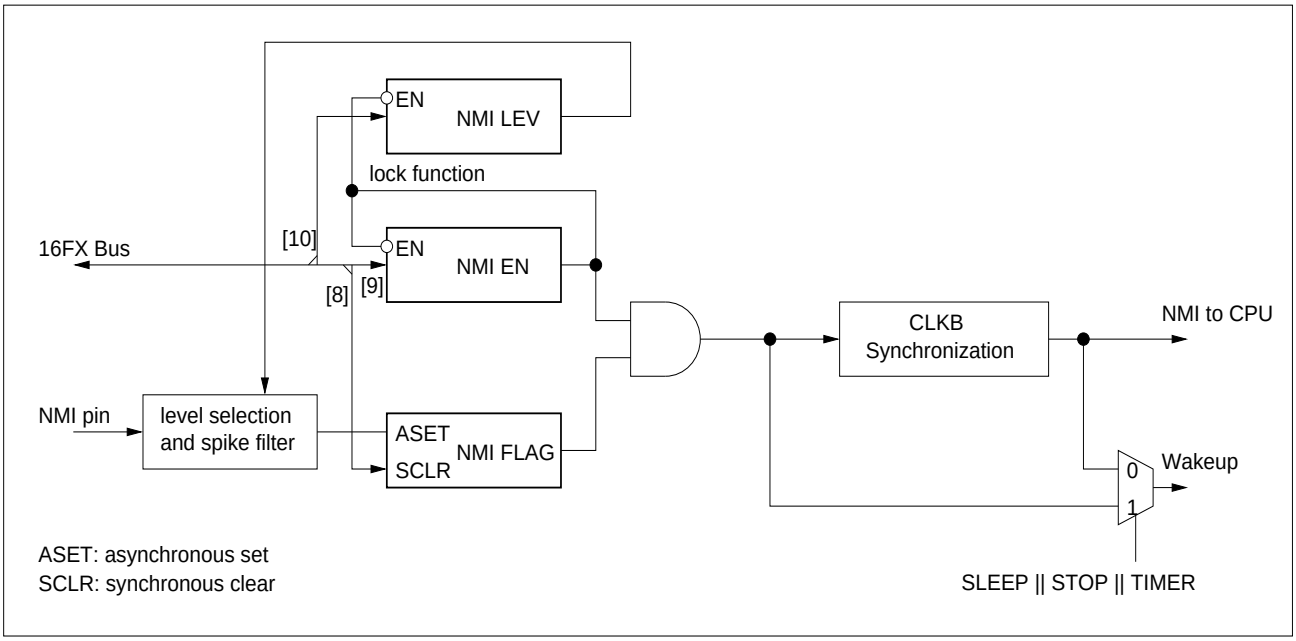
The NMI flag is set by the hardware event (NMI occurrence) and can be cleared by software to quit the interrupt. An interrupt is only caused, if both the EN bit and FLAG are set.

The NMI FLAG can be read and cleared. Writing ’1’ to FLAG is ignored.

The NMI FLAG is undefined after reset. Before enabling the NMI, this flag should be cleared.

For bit manipulation, an RMW-read operation returns always ’1’ for this flag.

Figure 3-5. Operation of the NMI control/status register



## 3.5 Interrupt Flow

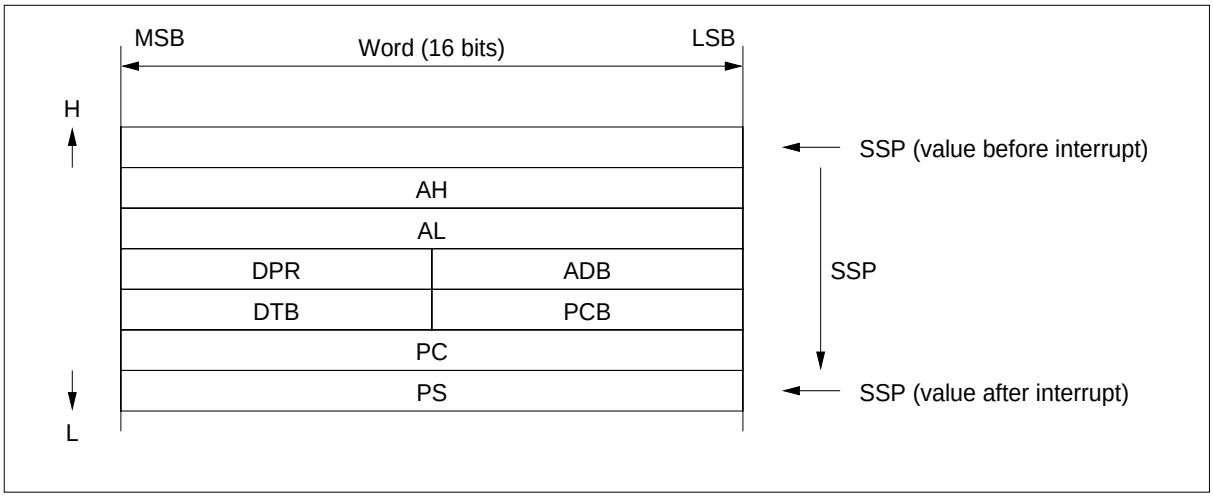
Figure 3-7 shows the interrupt flow.

### 3.5.1 Interrupt flow

The interrupt processing flow is entered at occurrence of hardware interrupts, software interrupts or exceptions. For a detailed interrupt flow chart see Figure 3-7.

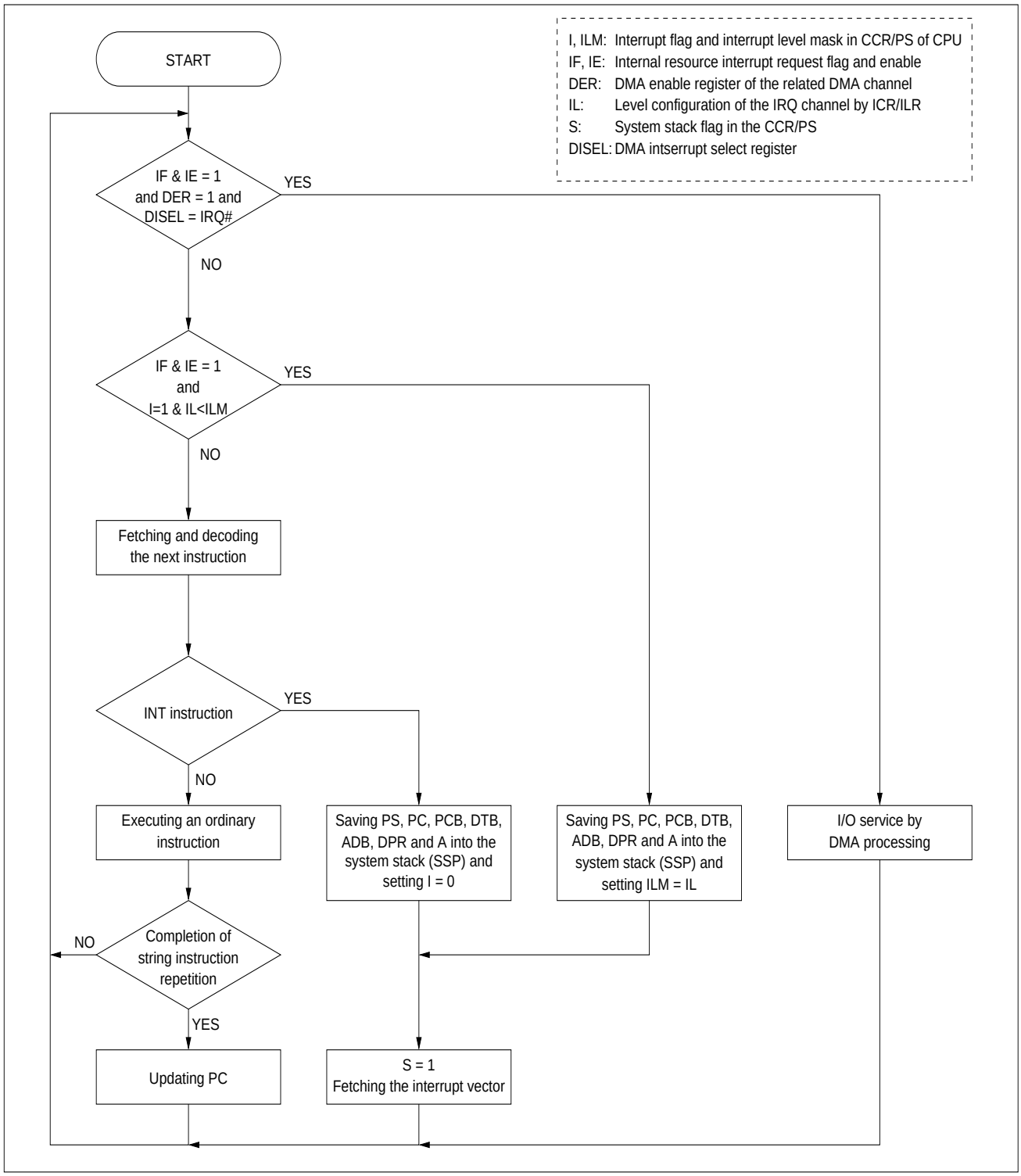
The CPU special registers are saved on the stack before the interrupt is processed (see Figure 3-6).

Figure 3-6. Register saving during interrupt processing




At the end of the interrupt processing, the context of the CPU registers is restored while executing the RETI instruction. The CPU returns to normal program execution.

Figure 3-7. Interrupt flow



## 3.6 Hardware Interrupts

In response to an interrupt request signal from an internal resource, the CPU pauses current program execution and transfers control to the interrupt processing program defined by the user.

### 3.6.1 Hardware interrupts

A hardware interrupt occurs when the relevant conditions are satisfied as a result of two operations:

- comparison between the interrupt request level (IL) and the value in the interrupt level mask register (ILM) of PS in the CPU, and
- hardware reference to the I flag value of PS.

The CPU performs the following processing when a hardware interrupt occurs:

- Saves the values in the PC, PS, AH, AL, PCB, DTB, ADB, and DPR registers of the CPU to the system stack.
- The S flag is set.
- Sets ILM in the PS register. The currently requested interrupt level (IL) is automatically set.
- Fetches the corresponding interrupt vector and branches to the processing indicated by that value.

If the device is in standby mode, a hardware interrupt with IL<7 generates a wake-up event to the clock and mode control unit.

### 3.6.2 Structure of the hardware interrupt system

The interrupt status is indicated by internal resources, the ICR for the interrupt controller, and the PS value of the CPU. To use a hardware interrupt, make the following set-up:

- Interrupt vector (in memory)
  - Consider the TBR value for a non-default location of the vector table.
  - The start address of the interrupt service routine has to be written to the appropriate interrupt vector (VecAddr = 4*(255-INT#) + 256*TBR).
- Peripheral resource
  - Use the Interrupt enable and request bits to control interrupt requests from peripheral resources.
- Interrupt controller
  - Assign interrupt levels (ICR:IL) for each interrupt, which can occur.
  - If interrupts occur simultaneously, a higher priority is defined by lower interrupt levels. IL=7 disables the interrupt.
  - If multiple requests are at the same level, the interrupt controller selects the request with the lowest interrupt number. In the case of same levels configured, the delayed interrupt has the lowest priority, independent from its interrupt number.
  - There is a fixed relationship between the interrupt requests and the ILs. A level can be defined by IL[n] for each hardware interrupt request IRQ[n] (for n >= 12).
- CPU
  - ILM and I in the PS register are used to compare the requested interrupt level (IL) with the current interrupt level mask (ILM) and to identify the interrupt enable status (I). For acceptance of hardware interrupts, the I flag has to be set and ILM has to be larger than IL.
  - During interrupt processing, the CPU saves 12 bytes to the memory area indicated by SSB and SSP. Thus the system stack pointer has to be initialized before using interrupts.
  - The CPU fetches three bytes of the interrupt vector and loads them onto PC and PCB. The interrupt handler routine has to start at this location. As a result, the interrupt processing program defined by the user is executed next. Normal operation is resumed at execution of the RETI instruction.

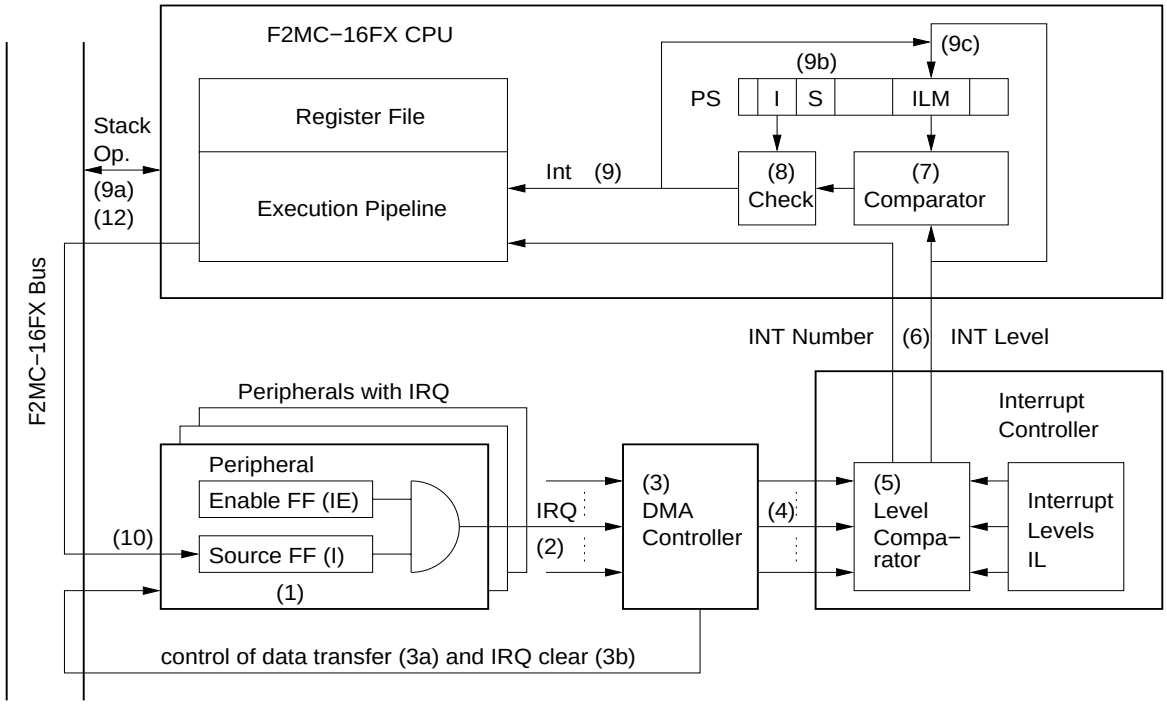
### 3.6.3 Hardware interrupt operation

Interrupt requests (IRQs) from peripheral resources are fed through the DMA controller before connecting to the interrupt controller. The DMA controller decides depending on it's channel configuration (DMA interrupt request select register DISEL and DMA enable DER:ENx bit), if the IRQ is handled by DMA transfer or passed to the interrupt controller. DMA transfers are

accepted regardless of the status of the I flag and the interrupt level. The DMA controller has a fixed priority scheme, channel 0 has highest priority and channel 15 has lowest priority.

Figure 3-8 shows the processing flow from the occurrence of a hardware interrupt to the release of the interrupt request in an interrupt processing program.

Figure 3-8. Occurrence and release of hardware interrupt




1. An interrupt cause occurs in a peripheral.
2. The interrupt enable bit in the peripheral is referenced. If interrupts are enabled, the peripheral issues an interrupt request (IRQ).
3. The DMA controller checks, if the IRQ should be handled by DMA. It evaluates for each channel, if the interrupt number of the asserted IRQ is selected and if DMA is enabled.
   - If the evaluation is true, the transfer is handled by DMA (3a). If the evaluation is false, interrupt processing is done by the interrupt controller, proceeding with step 4.
   - At the end of the DMA transfer, the interrupt bit is cleared in the peripheral (3b).
   - If the final transfer count is reached, the DMA completion interrupt is processed by the interrupt controller.
4. The interrupt controller receives the interrupt request.
5. The interrupt controller determines the priority levels of simultaneously requested interrupts.
6. The interrupt controller transfers the highest priority interrupt level and the corresponding interrupt number to the CPU.
7. The interrupt level requested by the interrupt controller is compared with the ILM value of the processor status register.
8. If the comparison shows that the requested level is lower than the current interrupt processing level (IL<ILM), the I flag value of the same processor status register is checked.
9. If the check in step 8 shows, that the I flag indicates interrupt enable status, interrupt processing is performed as soon as the currently executing instruction is completed.
   - To save the CPU status, special CPU registers are transferred to the system stack (9a).
   - The S bit is set to ’1’ (9b).
   - The requested level is written to the ILM bits (9c).
   - The interrupt vector is fetched.

- Then control is transferred to the interrupt processing routine (branch to the address read as interrupt vector).

10. When the interrupt cause of step 1 is cleared by software in the user interrupt processing routine, the interrupt request is completed.
11. The RETI instruction is used to return from the interrupt processing routine as its last instruction.
12. The CPU status is restored from system stack and normal program execution is resumed.

### 3.6.4 Hardware interrupt processing time

The time required for the CPU to execute the interrupt processing (stack operation, interrupt vector fetch, branch to the interrupt vector) is shown below. The value is valid if stack operation and interrupt vector fetch are executed without any wait cycles.

- Interrupt start: 10 cycles + c
- Interrupt return: 9 cycles + c (RETI instruction)

Table 3-4. Compensation values (c) for interrupt processing cycle count

| Address indicated by the stack pointer | Compensation value |
|---|---|
| Internal area, even-numbered address | 0 |
| Internal area, odd-numbered address | +2 |

In addition wait cycles for bus transfers have to be added, if any (e.g. access to vector table in slower ROM memory or external area).

## 3.7 Software Interrupts

In response to execution of a special instruction, control is transferred from the program currently executed by the CPU to the interrupt processing program defined by the user. This is called the software interrupt function. A software interrupt occurs always when the software interrupt instruction is executed.

### 3.7.1 Software interrupts

A software interrupt request issued by the INT instruction has no interrupt request or enable flag. A software interrupt request is always issued and accepted by executing the INT instruction.

The INT instruction does not have an interrupt level. Therefore, the INT instruction does not update ILM. The INT instruction clears the I flag to suspend subsequent hardware interrupt requests.

The CPU performs the following processing when a software interrupt occurs:

- Saves the values in the PC, PS, AH, AL, PCB, DTB, ADB, and DPR registers of the CPU to the system stack.
- The S flag is set.
- Clears I in the PS register. Hardware interrupts are automatically disabled.
- Fetches the corresponding interrupt vector and branches to the processing indicated by that value.

### 3.7.2 Structure of the software interrupt system

Software interrupts are fully handled within the CPU. To use a software interrupt, make the following set-up:

- Interrupt vector (in memory)
  - Consider the TBR value for a non-default location of the vector table.
  - The start address of the interrupt service routine has to be written to the appropriate interrupt vector (VecAddr = 4*(255-INT#) + 256*TBR).
- CPU
  - During interrupt processing, the CPU saves 12 bytes to the memory area indicated by SSB and SSP. Thus the system stack pointer has to be initialized before using interrupts.

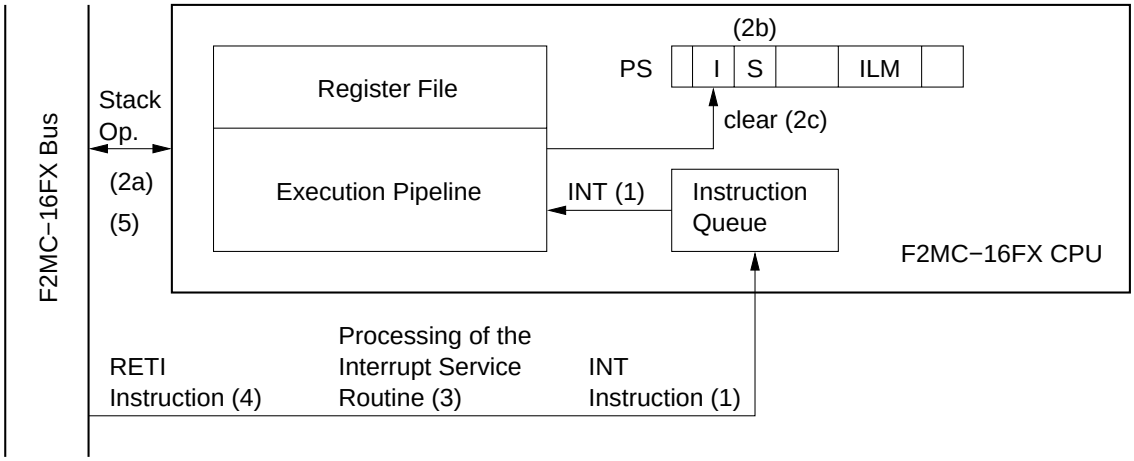
- The CPU fetches three bytes of the interrupt vector and loads them onto PC and PCB. The interrupt handler routine has to start at this location. As a result, the interrupt processing program defined by the user is executed next. Normal operation is resumed at execution of the RETI instruction.

### 3.7.3 Software interrupt operation

When the CPU fetches and executes the software interrupt instruction, the software interrupt processing sequence is activated. The software interrupt processing sequence saves 12 bytes (PS, PC, PCB, DTB, ADB, DPR, and A) to the memory area indicated by SSB and SSP. The sequence then fetches three bytes of interrupt vector and loads them into PC and PCB, resets the I flag, and sets the S flag. Then, the sequence performs branch processing. As a result, the interrupt processing program defined by the user application program is executed next.

Figure 3-9 illustrates the flow from the occurrence of a software interrupt until the return from the interrupt processing program.

Figure 3-9. Occurrence and release of software interrupt




1. The software interrupt instruction is executed.
2. Interrupt processing is performed by the CPU according to the software interrupt instruction.
   - To save the CPU status, special CPU registers are transferred to the system stack (2a).
   - The S bit is set to ’1’ (2b).
   - The I flag is cleared to disable hardware interrupts (2c).
   - The interrupt vector is fetched.
   - Then control is transferred to the interrupt processing routine (branch to the address read as interrupt vector).
3. The Interrupt service routine is processed by the CPU.
4. The interrupt processing is completed with the RETI instruction in the user interrupt processing routine.
5. The CPU restores its context of special registers from system stack.
6. The CPU proceeds program execution with the next instruction after the INT instruction.

## 3.8 Multiple interrupts

The F$^2$MC-16FX CPU supports multiple interrupts (simultaneous occurring interrupts and nested interrupt processing).

### 3.8.1 Multiple hardware interrupts

If an hardware interrupt of a higher priority (lower level value) occurs while another interrupt is being processed, control is transferred to the higher priority interrupt after the currently executing instruction is completed. After processing of the higher priority interrupt is completed, the original interrupt processing is resumed.

An interrupt of the same or lower priority may be generated while another interrupt is being processed. If this happens, the new interrupt request is suspended until the current interrupt processing is completed, unless the ILM value or I flag is changed by an instruction.

A DMA transfer cannot be interrupted and activated from multiple sources. While a DMA transfer is being processed, all other DMA requests are suspended. At simultaneous occurrence of requests for the DMA controller, the lowest channel number is processed first.

For a detailed description of DMA, please refer to DMA chapter on page 95.

### 3.8.2 Multiple software interrupts

Software interrupts can not occur simultaneously. They are entered by executing the INT instruction and are always accepted. However, if an INT instruction is placed within an interrupt service routine, nested execution of software interrupts is possible.

### 3.8.3 Interrupt acceptance priority

Following table lists all interrupts with conditions for their acceptance.

Table 3-5. Control of interrupt acceptance priority

| Event | INT# | Type | Level | Acceptance condition | Action, if accepted | |
|---|---|---|---|---|---|---|
| Hardware event | - | Instruction Break (VEIB)<br>*system reserved* | P2 | Current instruction execution is finished,<br>ILM>2 \|\| P == 1 | Save CPU status to system stack<br>S = 1<br>Branch to interrupt vector | P = 0<br>ILM = 2 |
| | - | Tool Break (VENMI)<br>*system reserved* | P2 | Current instruction execution is finished, string instruction is interrupted,<br>ILM>2 \|\| P == 1 | | P = 0<br>ILM = 2 |
| | 11 | NMI | P4 | Current instruction execution is finished, string instruction is interrupted,<br>ILM>4 \|\| P == 1 | | P = 0<br>ILM = 4 |
| | 9 | Address match detection (HW-INT9) | P6 | Current instruction execution is finished, string instruction is interrupted,<br>ILM>6 \|\| P == 1 | | P = 0<br>ILM = 6 |
| | from 13 on | Peripheral IRQ | IL<br>U0...U7 | Current instruction execution is finished, string instruction is interrupted,<br>ILM > IL<br>P == 1<br>I == 1<br>For multiple requests with same IL, smallest IRQ number is accepted. | | ILM = IL |
| | 12 | Delayed INT | IL<br>U0...U7 | Current instruction execution is finished, string instruction is interrupted,<br>ILM > IL<br>P == 1<br>I == 1<br>No peripheral IRQs pending with same IL. | | ILM = IL |
| Software event (Instruction) | - | Software Instruction Break (INTE)<br>*system reserved* | P2 | always accepted | | P = 0<br>ILM = 2<br>I = 0 |
| | 9 | INT9 | - | | | I = 0 |
| | 10 | Undefined instruction exception | - | | | I = 0 |
| | all | INT instruction | - | | | I = 0 |
| | - | RETI instruction | - | | restore CPU status (including P, I, S, ILM) | |

IL and ILM: Interrupt level and mask
I: Interrupt enable flag (Peripheral type interrupts)
S: System stack flag
P: Privileged mode flag (bit 7 of CCR, PS)

Following table defines the naming of the interrupt levels, its corresponding P flag and ILM values. It also lists the interrupt cause, which can request the interrupt level.

Table 3-6. Interrupt levels

| Name | Category | P flag | ILM value | Priority | Used by |
|---|---|---|---|---|---|
| P0 | Priviledged mode | 0 | 0 | Highest | - |
| P1 | | 0 | 1 | | - |
| P2 | | 0 | 2 | | DSU |
| P3 | | 0 | 3 | | - |
| P4 | | 0 | 4 | | NMI |
| P5 | | 0 | 5 | | - |
| P6 | | 0 | 6 | | HW-INT9 |
| P7 | | 0 | 7 | | - |
| U0 | User mode | 1 | 0 | | Peripherals |
| U1 | | 1 | 1 | | |
| U2 | | 1 | 2 | | |
| U3 | | 1 | 3 | | |
| U4 | | 1 | 4 | | |
| U5 | | 1 | 5 | | |
| U6 | | 1 | 6 | | |
| U7 | | 1 | 7 | Lowest | no request |

## 3.9 Exceptions

The F$^2$MC-16FX performs exception processing at occurrence of various software and hardware events.

### 3.9.1 Software exceptions (op-code)

Software exceptions are always accepted. Same as software interrupts, software exceptions disable any hardware interrupt acceptance. The software exceptions occur at code execution of following specific op-codes:

#### *3.9.1.1 Execution of an undefined instruction*

All codes that are not defined in the instruction map are handled as undefined instructions. When an undefined instruction is executed, processing similar to the INT #10 software interrupt instruction is performed. Specifically, the PC value saved in the stack is the address at which the undefined instruction is stored. Processing can be restored by the RETI instruction, however it is of no use, because the same exception occurs again.

**Operation:**

(SSP)<-(SSP)-2, ((SSP))<-(AH)

(SSP)<-(SSP)-2, ((SSP))<-(AL)

(SSP)<-(SSP)-2, ((SSP))<-(DPR):(ADB)

(SSP)<-(SSP)-2, ((SSP))<-(DTB):(PCB)

(SSP)<-(SSP)-2, ((SSP))<-(PC)

(SSP)<-(SSP)-2, ((SSP))<-(PS)

(S)<-1, (I)<-0

(PCB)<-Vector #10 address (upper byte)

(PC)<-Vector #10 address (lower word)

#### 3.9.1.2 INT9

This instruction branches to the interrupt processing routine indicated by vector #9. Executing the RETI instruction in the interrupt routine restores the processing subsequent to the INT9 instruction.

**Operation:**

(SSP)<-(SSP)-2, ((SSP))<-(AH)

(SSP)<-(SSP)-2, ((SSP))<-(AL)

(SSP)<-(SSP)-2, ((SSP))<-(DPR):(ADB)

(SSP)<-(SSP)-2, ((SSP))<-(DTB):(PCB)

(SSP)<-(SSP)-2, ((SSP))<-(PC)+1

(SSP)<-(SSP)-2, ((SSP))<-(PS)

(S)<-1, (I)<-0

(PCB)<-Vector #9 address (upper byte)

(PC)<-Vector #9 address (lower word)

#### 3.9.1.3 INTE (System reserved, only available with DSU)

INTE is used to insert a software break point for the debug system, using the in circuit emulator (ICE). At insertion of a software instruction break, the first byte of the original instruction is replaced by INTE.

This instruction branches to the interrupt processing routine indicated by a fixed vector defined by the DSU. The PC value saved in the stack is the address at which INTE is stored. Executing the RETI instruction in the interrupt routine restores the processing at this location (INTE can be replaced by the original instruction at removal of the software break point).

The privileged mode flag (P flag) is cleared and the ILM register is set to 2 (enters level P2). This disables all hardware interrupts and exceptions. The P flag and ILM are restored at execution of the RETI instruction.

**Operation:**

(SSP)<-(SSP)-2, ((SSP))<-(AH)

(SSP)<-(SSP)-2, ((SSP))<-(AL)

(SSP)<-(SSP)-2, ((SSP))<-(DPR):(ADB)

(SSP)<-(SSP)-2, ((SSP))<-(DTB):(PCB)

(SSP)<-(SSP)-2, ((SSP))<-(PC)

(SSP)<-(SSP)-2, ((SSP))<-(PS)

(S)<-1, (I)<-0, (P)<-0, (ILM)<-2

(PCB)<-Fixed vector from DSU (upper byte, $00_H$, address is ignored by DSU)

(PC)<-Fixed vector from DSU (lower word, $0400_H$, address is ignored by DSU)

Without the DSU, INTE is handled same as the undefined instruction exception. Interrupt vector #10 is referenced. The P flag is not cleared and ILM is not updated.

### 3.9.2 Hardware exceptions (non maskable interrupts)

Hardware exceptions are external events, which are not maskable by any software instruction. Hardware exceptions with a higher level number, than the actual processed one, are suspended until execution of the RETI instruction restores the previous level. In addition, hardware exceptions disable any hardware interrupt acceptance.

At occurrence of multiple hardware exceptions at the same time, they will be accepted with following priority: VEIB > VENMI > NMI > HW-INT9. If the current interrupt level mask and P flag setting allows it, hardware exceptions are accepted at the end of each instruction execution and during execution of string instructions.

### *3.9.2.1 HW-INT9*

HW-INT9 is used by the address match detection function. With that function embedded debug support (operand address break or data value break) or a simple memory protection can be provided.

The privileged mode flag (P flag) is cleared and ILM is set to 6 (enters level P6). This disables all hardware interrupts from peripherals. The P flag and ILM are restored at the execution of the RETI instruction.

**Operation:**

(SSP)<-(SSP)-2, ((SSP))<-(AH)

(SSP)<-(SSP)-2, ((SSP)<-(AL)

(SSP)<-(SSP)-2, ((SSP))<-(DPR):(ADB)

(SSP)<-(SSP)-2, ((SSP))<-(DTB):(PCB)

(SSP)<-(SSP)-2, ((SSP))<-(PC)

(SSP)<-(SSP)-2, ((SSP))<-(PS)

(S)<-1, (P)<-0, (ILM)<-6

(PCB)<-Vector #9 address (upper byte)

(PC)<-Vector #9 address (lower word)

### *3.9.2.2 NMI*

NMI provides external hardware exception handling.

The privileged mode flag (P flag) is cleared and ILM is set to 4 (enters level P4). This disables all hardware interrupts from peripherals and the HW-INT9. The P flag and ILM are restored at execution of the RETI instruction.

**Operation:**

(SSP)<-(SSP)-2, ((SSP))<-(AH)

(SSP)<-(SSP)-2, ((SSP))<-(AL)

(SSP)<-(SSP)-2, ((SSP))<-(DPR):(ADB)

(SSP)<-(SSP)-2, ((SSP))<-(DTB):(PCB)

(SSP)<-(SSP)-2, ((SSP))<-(PC)

(SSP)<-(SSP)-2, ((SSP))<-(PS)

(S)<-1, (P)<-0, (ILM)<-4

(PCB)<-Vector #11 address (upper byte)

(PC)<-Vector #11 address (lower word)

### 3.9.2.3 Tool break (VENMI, system reserved, only available with DSU)

VENMI is provided for debugging with DSU. It implements various break factors.

The privileged mode flag (P flag) is cleared and ILM is set to 2 (enters level P2). This disables all hardware interrupts and exceptions. The P flag and ILM are restored at execution of the RETI instruction.

**Operation:**

(SSP)<-(SSP)-2, ((SSP))<-(AH)

(SSP)<-(SSP)-2, ((SSP))<-(AL)

(SSP)<-(SSP)-2, ((SSP))<-(DPR):(ADB)

(SSP)<-(SSP)-2, ((SSP))<-(DTB):(PCB)

(SSP)<-(SSP)-2, ((SSP))<-(PC)

(SSP)<-(SSP)-2, ((SSP))<-(PS)

(S)<-1, (P)<-0, (ILM)<-2

(PCB)<-Fixed vector from DSU (upper byte, $00_H$, address is ignored by DSU)

(PC)<-Fixed vector from DSU (lower word, $0400_H$, address is ignored by DSU)

### 3.9.2.4 Instruction break (VEIB, system reserved, only available with DSU)

VEIB is provided for debugging with DSU. It implements the instruction break before execution. Opposed to other hardware exceptions, VEIB is not accepted during execution of a string instruction.

# 4. DMA

This chapter explains the DMA functions and operations.

4.1 Overview

4.2 DMA Registers

4.3 DMA Descriptor

4.4 DMA Controller Operation

4.5 Examples of DMA transfers

## 4.1 Overview

DMA enables automatic data transfer between memory and memory/peripheral resource registers or vice versa without interaction of the CPU. DMA replaces 16LX family's EI$^2$OS while offering the extended functionality at higher performance.

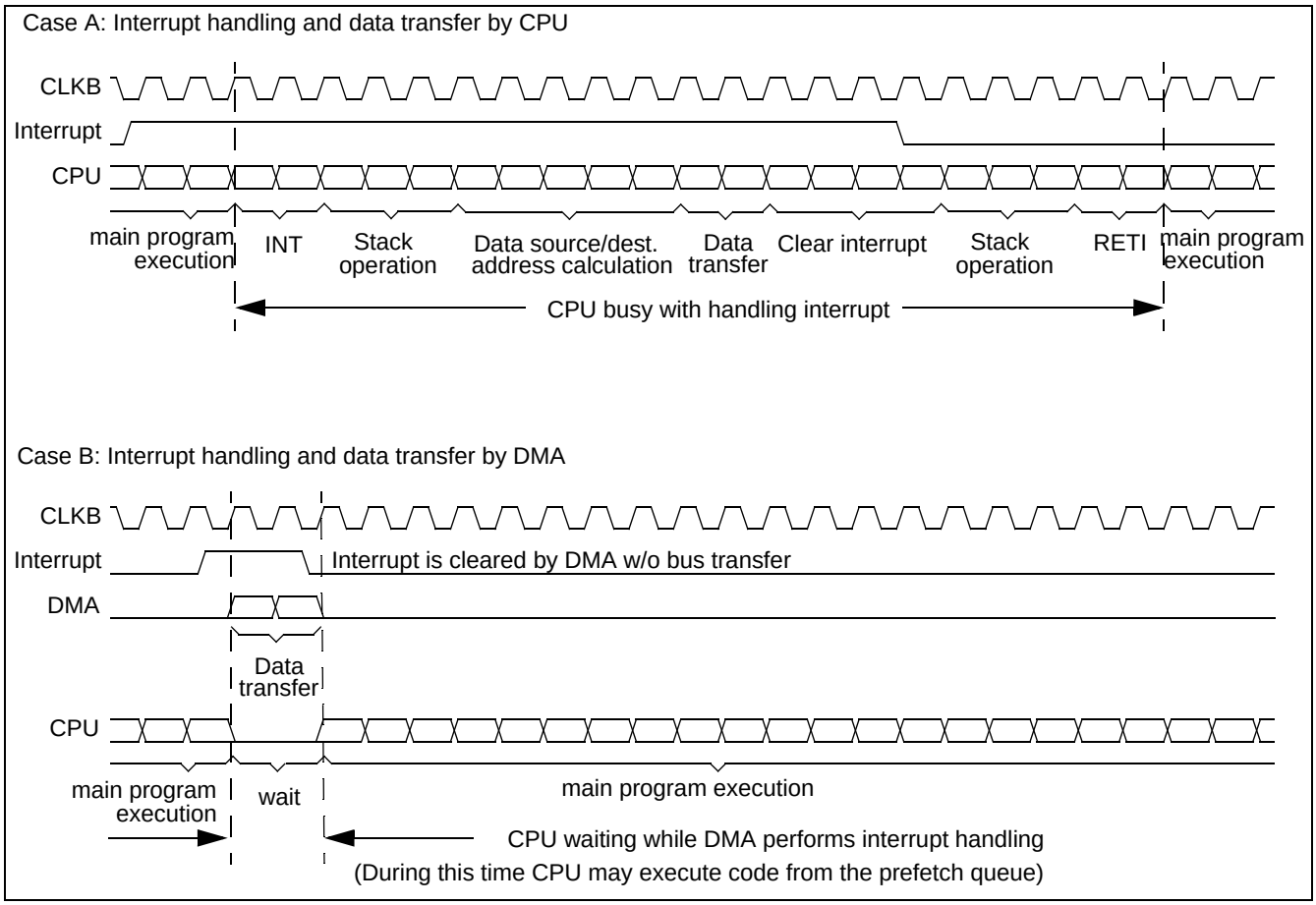
This section describes the DMA functions.

### 4.1.1 DMA compared to interrupt handling:

DMA has the following advantages compared to handling the transfer via interrupt (see Figure 4-1):

- DMA reduces CPU processing load for interrupt handling
- No CPU register is used for the transfer, eliminating the need for register saving, which increases the transfer speed.
- The CPU bus is only used for the transfer operation itself. As consequence, the transfer speed is increased.

Figure 4-1. Comparison of interrupt handling by CPU and by DMA (principle only, number of clock cycles may differ depending on the program code)




## 4.1.2 DMA functions

The DMA controller has the following functions:

- Automatic data transfer between peripheral resources (I/O) and memory
- CPU bus allocation during DMA transfer
- Up to 16 DMA transfer channels with fixed priority scheme (the smaller the DMA channel number, the higher the priority of the request).
- Allow selection of whether or not to increment or decrement the transfer source and destination addresses (the buffer address may either be incremented, decremented or left unchanged; the I/O register address may either be incremented or left unchanged).
- DMA transfer can be started by a selectable peripheral resource interrupt (hardware IRQ).
- The DMA transfer is controlled by:
  - The DMA Enable Register (DER, bits 0...15)
  - The DMA interrupt select registers (DISEL 0 ... 15)
  - The DMA Stop Status Register (DSSR, bits 0...15)
  - The DMA Status Register (DSR, bits 0...15)
  - The DMA descriptors (8 byte each; DCT, IOA, DMACS, BAP)
  - The DMA IOA bank registers (IOABK)[1]

- DMA transfer can be stopped upon an error condition in the peripheral resource.
- At the end of a DMA transfer, processing automatically branches to the according interrupt service routine of the peripheral resource.

### 4.1.3 Structure

If enabled, the DMA controller serves interrupt requests of peripheral resources before they are passed to the interrupt controller. The structure of the interrupt system can be understood in that way, that the DMA controller is placed between the peripheral resources (IRQ sources) and the interrupt controller (see Figure 3-8 on page 86).

I/O service by DMA is handled by the following units:

- Peripheral resources
  - IF, IE: Interrupt flag and enable bits are used to control interrupt requests from resources (IRQ = IF && IE).
- DMA controller
  - DER and DISEL enable the DMA operation for a selected interrupt request. If DMA operation is not enabled, the request is passed to the interrupt controller.
  - Transfer information is stored in the DMA descriptors (data count, I/O address pointer, Buffer Address Pointer and control information) and the IOA bank registers.
  - The DMA controller processes the request (data transfer from/to peripheral) and clears the interrupt flag in the peripheral resource.
  - The DMA controller identifies its status (DSR) and indicates, if a stop condition has occured (DSSR). In case of DMA is stopped or completed, an interrupt is issued to the interrupt controller.
- Interrupt controller
  - The interrupt controller handles the interrupt due to completion or cancellation of DMA.
  - The ICR assigns interrupt levels, which determine the priority levels of simultaneously requested interrupts.
- CPU
  - I and ILM are used to compare the requested and current interrupt levels and to identify the interrupt enable status.
  - If the interrupt level is accepted, the CPU handles the interrupt request.

1. IOA bank registers are not available for all devices. Please refer to the Datasheet.

## 4.2 DMA Registers

The DMA controller has three registers (DER, DSR, DSSR, each with one bit per DMA channel) and one register to select the interrupt (DISEL, one byte for each DMA channel).

The DMA descriptor is used to set-up the DMA transfer. It is explained in section 4.3 DMA Descriptor.

### DMA Register list

Figure 4-2. DMA Register list

| Address: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x381, 0x380 | IS7 | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 | IS7 | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 | DISEL1, DISEL0 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value: | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |

| Address: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x38F, 0x38E | IS7 | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 | IS7 | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 | DISEL15, DISEL14 |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value: | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |

| Address: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x391, 0x390 | DTE15 | DTE14 | DTE13 | DTE12 | DTE11 | DTE10 | DTE9 | DTE8 | DTE7 | DTE6 | DTE5 | DTE4 | DTE3 | DTE2 | DTE1 | DTE0 | DSRH, DSRL |
| | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | R/W0 | |
| Initial Value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Address: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x393, 0x392 | STP15 | STP14 | STP13 | STP12 | STP11 | STP10 | STP9 | STP8 | STP7 | STP6 | STP5 | STP4 | STP3 | STP2 | STP1 | STP0 | DSSRH, DSSRL |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Address: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x395, 0x394 | EN15 | EN14 | EN13 | EN12 | EN11 | EN10 | EN9 | EN8 | EN7 | EN6 | EN5 | EN4 | EN3 | EN2 | EN1 | EN0 | DERH, DERL |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

R: : Readable
R/W: : Readable & writable
R/W0: : Readable & clearable (write 0 accepted, write 1 ignored
-: : No access; read returns undefined value, write has no effect

### Note for channels not available on the device:

At read operation, not available DSR, DSSR and DER bits return undefined value.

At read operation, not available DISEL registers return undefined value.

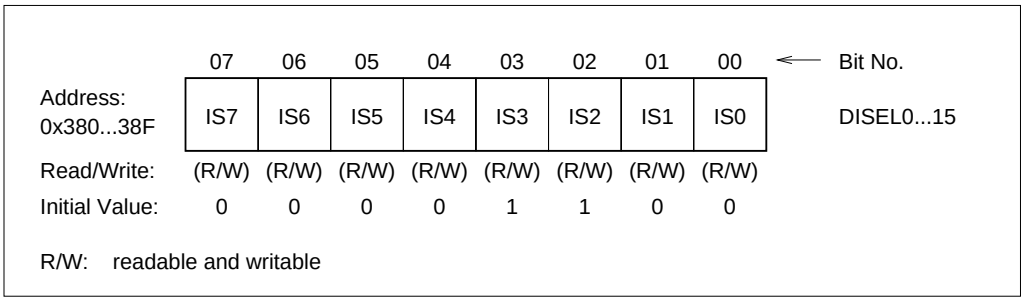
Write ’0’ to not available bits/registers.

## 4.2.1 DMA Interrupt Request Select Register (DISEL)

Each DMA channel has one Interrupt Request Select Register (DISEL0 ... DISEL15). This register defines which IRQ number is used to trigger the DMA transfer on this channel.

### *4.2.1.1 DMA Interrupt Request Select Register (DISEL)*

Figure 4-3. DMA Interrupt Request Select Register (DISEL) configuration




**[bit 7 to 0] IS: Interrupt select**

For each DMA channel a DISELx register exists, which defines the IRQ number to trigger the DMA transfer on this channel. The valid range of the selectable IRQ number is from 0x0C (12) to a device dependent maximum number. Only hardware interrupts can trigger DMA operation.

**Note:**

DISEL can be read and written.

The DISEL register is initialized to 0x0C (12) at reset.

At read operation, DISEL registers of not available channels return undefined value.

Write ’0’ to DISEL registers of not available channels.

Do not configure the same interrupt number in more than one enabled DMA channel.

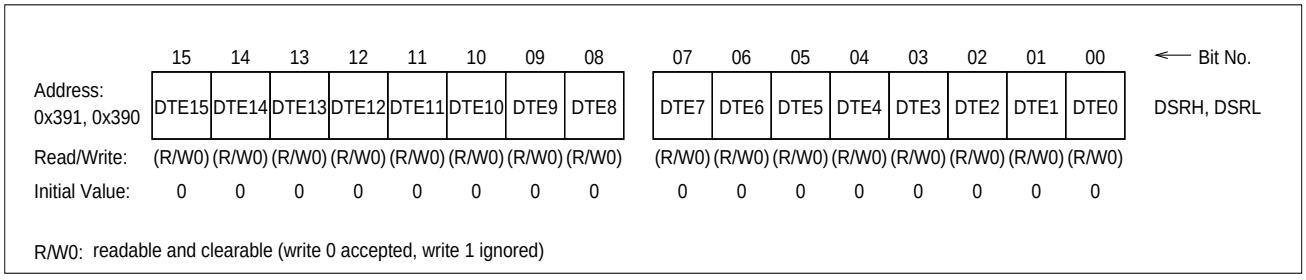
When updating a DISEL register ensure that the associated IRQ is deactivated or the DMA channel is disabled.

## 4.2.2 DMA Status Register (DSR)

This section describes the DMA Status Register (DSR).

### *4.2.2.1 DMA Status Register (DSR)*

Figure 4-4. DMA Status Register (DSR) configuration



### [bit 15 to 0] DTEx: Data Transfer End interrupt request

For each DMA channel a DTE bit exists, which indicates the end of the data transfer. This could be the case if the Data Count Register (DCT) reaches 0 or if a stop request from a peripheral resource is asserted. With setting a DTE bit by the hardware of the DMA controller, an interrupt is requested.

| DTEx bit | Function |
|---|---|
| 0<br>[Initial value] | Indicates no interrupt request present for channel x. |
| 1 | Indicates that DMA transfer is completed or stopped and an interrupt is requested. If DTEx is set the DMA transfer is disabled on this channel. |

**Note:**

Read access returns the status of all DTE bits of available channels in DSR.

At read operation, DTE bits of not available channels return undefined value.

A RMW-read instruction returns all DTE bits set to ’1’.

This bit is set by hardware only. Writing ’1’ to this bit is ignored.

Writing ’0’ clears this bit.

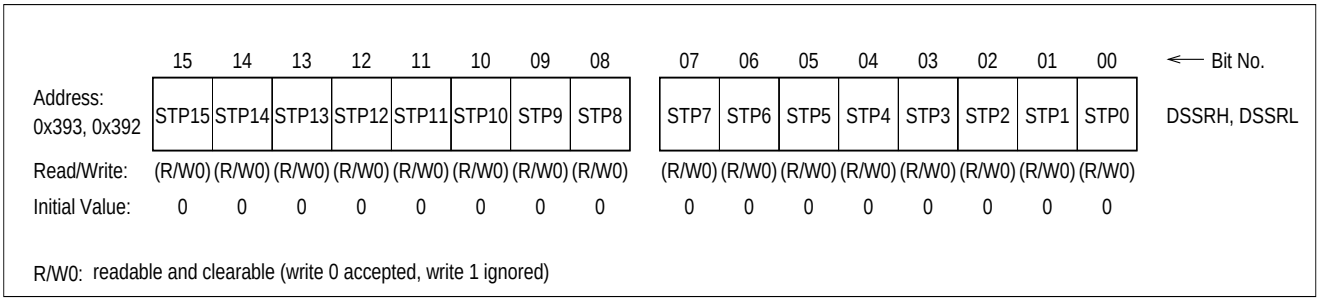
Write ’0’ to DTE bits of not available channels.

## 4.2.3 DMA Stop Status Register (DSSR)

This section describes the DMA Stop Status Register (DSSR)

### *4.2.3.1 DMA Stop Status Register (DSSR)*

Figure 4-5. DMA Stop Status Register (DSSR) configuration




### [bit 15 to 0] STPx: DMA Stop Status: occurrence of stop request

| STPx bit | Function |
|---|---|
| 0<br>[Initial value] | No STOP request occurred during DMA transfer on channel x, or STOP request was not enabled by the DMACS:SE bit. |
| 1 | Indicates that DMA transfer stopped due to STOP request issued by the peripheral resource during DMA transfer. |

**Note:**

Read access returns the status of all STP bits of available channels in DSSR.

At read operation, STP bits of not available channels return undefined value.

A RMW-read instruction returns all STP bits set to ’1’.

This bit is set by hardware only. Writing ’1’ to the bit is ignored.

Writing ’0’ clears the bit.

Write ’0’ to STP bits of not available channels.

All DMA channels support the STOP request, however not all IRQ sources (peripherals) provide this feature.

## 4.2.4 DMA Enable Register (DER)

This section describes the DMA Enable Register (DER).

### *4.2.4.1 DMA Enable Register (DER)*

Figure 4-6. DMA Enable Register (DER) configuration

| | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address: 0x395, 0x394 | EN15 | EN14 | EN13 | EN12 | EN11 | EN10 | EN9 | EN8 | EN7 | EN6 | EN5 | EN4 | EN3 | EN2 | EN1 | EN0 | DERH, DERL |
| Read/Write: | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Initial Value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

R/W: readable and writable

**[bit 15 to 0] ENx: DMA Enable**

| ENx bit | Function |
|---|---|
| 0<br>[Initial value] | The interrupt request from the resource is not processed as a DMA start request, but it is passed to the interrupt controller. |
| 1 | The interrupt request from the resource is processed as a DMA start request. In consequence, the interrupt is handled by the DMA controller. At completion of the DMA transfer (DCT reaches 0) or stop request from peripheral, the interrupt request is output to the interrupt controller. |

**Note:**

The register can be read and written.

At read/RMW-read operation, EN bits of not available channels return undefined value.

Write ’0’ to EN bits of not available channels.

Before entering any standby mode these bits must be cleared to zero. Otherwise interrupts associated to these DMA channels cannot be used for wakeup.

Do not change the configuration of a DMA channel, if its EN bit is set.

## 4.3 DMA Descriptor

For each DMA channel the DMA descriptor consists of 8 bytes. In addition each DMA channel offers 1 byte for extended configuration. DMA descriptor and extended configuration are used to set up a DMA transfer.

### Configuration of DMA Descriptors and Extended Configuration

Each DMA channel has its own DMA descriptor of 8 bytes and the extended configuration register IOABK. The DMA descriptors are located from addresses $000100_H$ to $00017F_H$. The extended configuration registers are located from $000A00_H$ to $000A0F_H$. The following table shows the relation of DMA channels to the descriptor and extended configuration addresses.

The descriptor and extended configuration areas of not available channels are not accessible. Write '0' to the descriptor area of not available channels. Read access returns undefined values.

Before changing the configuration of a DMA channel it has to be ensured that the associated IRQ is deactivated. Alternatively the EN bit of the channel to be updated must be cleared.

Table 4-1. Relation of DMA channel numbers to DMA descriptor and DMA extended configuration addresses

| DMA Channel | Descriptor Start Address | Descriptor End Address | IOABK |
|---|---|---|---|
| 0 | $000100_H$ | $000107_H$ | $000A00_H$ |
| 1 | $000108_H$ | $00010F_H$ | $000A01_H$ |
| 2 | $000110_H$ | $000117_H$ | $000A02_H$ |
| 3 | $000118_H$ | $00011F_H$ | $000A03_H$ |
| 4 | $000120_H$ | $000127_H$ | $000A04_H$ |
| 5 | $000128_H$ | $00012F_H$ | $000A05_H$ |
| 6 | $000130_H$ | $000137_H$ | $000A06_H$ |
| 7 | $000138_H$ | $00013F_H$ | $000A07_H$ |
| 8 | $000140_H$ | $000147_H$ | $000A08_H$ |
| 9 | $000148_H$ | $00014F_H$ | $000A09_H$ |
| 10 | $000150_H$ | $000157_H$ | $000A0A_H$ |
| 11 | $000158_H$ | $00015F_H$ | $000A0B_H$ |
| 12 | $000160_H$ | $000167_H$ | $000A0C_H$ |
| 13 | $000168_H$ | $00016F_H$ | $000A0D_H$ |
| 14 | $000170_H$ | $000177_H$ | $000A0E_H$ |
| 15 | $000178_H$ | $00017F_H$ | $000A0F_H$ |

The structure of each channel is as shown in Figure 4-7.

Figure 4-7. DMA Descriptor and Extended Configuration

| Address: | Extended Configuration | Access | Initial value: |
|---|---|---|---|
| 0x000A00 + ch | Bank select of I/O register address pointer (IOABK) | (R/W) | 0 |
| | DMA Descriptor | | |
| 0x000107 + 8*ch | Upper 8 bits of data counter (DCTH) | (R/W) | X |
| 0x000106 + 8*ch | Lower 8 bits of data counter (DCTL) | (R/W) | X |
| 0x000105 + 8*ch | Upper 8 bits of I/O register address pointer (IOAH) | (R/W) | X |
| 0x000104 + 8*ch | Lower 8 bits of I/O register address pointer (IOAL) | (R/W) | X |
| 0x000103 + 8*ch | DMA control register (DMACS) | (R/W) | X |
| 0x000102 + 8*ch | Upper 8 bits of buffer address pointer (BAPH) | (R/W) | X |
| 0x000101 + 8*ch | Middle 8 bits of buffer address pointer (BAPM) | (R/W) | X |
| 0x000100 + 8*ch | Lower 8 bits of buffer address pointer (BAPL) | (R/W) | X |

R/W: readable and writable

The terms "ch" and "8*ch" define the address offsets of the descriptors and extended configuration, depending on the channel index "ch".

### Each Register of DMA Descriptor

Each register contained in the DMA descriptor and extended configuration is explained in the following sections. The DMA descriptor registers must be initialized before setting DER:ENx to ’1’ because their initial values are undefined. The extended configuration registers are initialized at reset.

## 4.3.1 DMA IO Address Pointer Bank Select (IOABK)[1]

Each DMA channel has one IO Address Pointer Bank Select Register (IOABK0 ... IOABK15). This register defines which bank is used to address the IO space during DMA transfer.

1. IOA bank registers are not available for all devices. Please refer to the Datasheet.

#### 4.3.1.1 DMA IO Address Pointer Bank Select Register (IOABK)

Figure 4-8. DMA IO Address Pointer Bank Select Register (IOABK) configuration

| | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|
| Address: 0xA00...A0F | BK7 | BK6 | BK5 | BK4 | BK3 | BK2 | BK1 | BK0 | IOABK0...15 |
| Read/Write: | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Initial Value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

R/W: readable and writable

**[bit 7 to 0] BK: Bank select**

For each DMA channel an IOABKx register exists, which defines the bank address for IO register addressing during DMA transfer on this channel.

**Note:**

IOABK can be read and written.

The IOABK register is initialized to 0x00 at all resets.

At read operation, IOABK registers of not available channels return undefined value.

Write ’0’ to IOABK registers of not available channels.

During DMA transfer IOABK remains constant. A transfer across multiple banks is not possible.

### 4.3.2 Data Count Register (DCT)

This section describes the Data Count Register (DCT).

#### 4.3.2.1 Data Count Register (DCT)

The Data Count Register (DCT) is a 16-bit register to store the number of bytes to be transferred. After each data transfer, the Data Count Register is decremented by 1 at byte transfer or by 2 at word transfer.

When “0” is set to DCT, the maximum data transfer count (65536 bytes) is set.

When the Data Count Register becomes “0”, DMA transfer ends.

Figure 4-9 shows the configuration of the Data Count Register (DCT).

Figure 4-9. Data Count Register (DCT) Configuration

| | DCTH | | | | | | | | DCTL | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address: | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 | ← Bit No. |
| 8*ch + 0x107, 0x106 | C15 | C14 | C13 | C12 | C11 | C10 | C9 | C8 | C7 | C6 | C5 | C4 | C3 | C2 | C1 | C0 | DCT |
| Read/Write: | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Initial Value: | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | |

R/W: readable and writable

#### 4.3.2.2 Transfer type and decrement of DCT

The type of the transfer and the decrement value of DCT mainly depends on the DMACS:BW bit. However, also the configuration and status of IOA, BAP and DCT itself are considered.

Table 4-2 shows the transfers caused by different configuration or status of IOA, BAP, DCT and BW. The BAP increment/decrement operation depends on the DMACS:BPD bit.

Table 4-2. Transfer type resulting from descriptor configuration

| Configuration/Status | | | | Transfer type | | Post transfer update of | | | Transfer cycles[a] |
|---|---|---|---|---|---|---|---|---|---|
| BW | DCT | IOA | BAP | I/O | Buffer | DCT | IOA when DMACS:IF = 1 | BAP when DMACS:BF = 1 | |
| 0 (byte) | >0 | - | - | byte | byte | -1 | +1 | +/-1 | 2 |
| 1 (word) | >1 | even | even | word | word | -2 | +2 | +/-2 | 2 |
| 1 (word) | >1 | even | odd | word | 2 x byte | -2 | +2 | +/-2 | 3 |
| 1 (word) | >1 | odd | even | 2 x byte | word | -2 | +2 | +/-2 | 3 |
| 1 (word) | >1 | odd | odd | 2 x byte | 2 x byte | -2 | +2 | +/-2 | 4 |
| 1 (word) | =1 | - | - | byte | byte | -1 | +1 | +/-1 | 2 |

a. One transfer cycle is equal to one bus access

### 4.3.3 I/O Register Address Pointer (IOA)

This section describes the I/O Register Address Pointer (IOA).

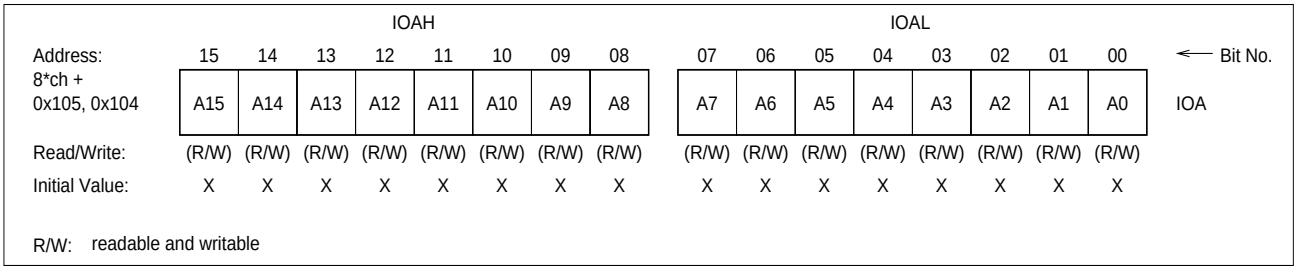
#### 4.3.3.1 I/O Register Address Pointer (IOA)

The I/O Register Address Pointer (IOA) is a 16-bit register and indicates the lower addresses (A15 to A0) of the I/O register. Its upper addresses (A23 to A16) can be set in IOABK register. The reset value of IOABK defaults to bank 0.

When “Update Performed” is specified with the DMACS:IF bit of the DMA Control Register, IOA is incremented by 1 at byte transfer; and incremented by 2 at word transfer. When “Update Not Performed” is specified with the DMACS:IF bit, IOA is fixed (IOA does not change).

Figure 4-10 shows the configuration of the I/O Register Address Pointer (IOA).

Figure 4-10. I/O Register Address Pointer (IOA) Configuration




### 4.3.4 DMA Control Register (DMACS)

This section describes the DMA Control Register (DMACS).

#### 4.3.4.1 DMA Control Register (DMACS)

The DMA Control Register (DMACS) is 8-bit long and is used:

- to specify, if the Buffer Address Pointer should be incremented or decremented (BPD)
- to specify whether to update or fix the Buffer Address Pointer and the I/O Register Address Pointer (BF, IF)
- to specify the transfer data format is byte or word (BW)

- to specify the transfer direction (DIR) and
- to specify, if STOP request should be accepted (SE).

Figure 4-11 shows the configuration of DMACS.

Figure 4-11. Configuration of DMACS

| Address: | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | ← Bit No. |
|---|---|---|---|---|---|---|---|---|---|
| 8*ch + 0x103 | − | − | BPD | IF | BW | BF | DIR | SE | DMACS |
| Read/Write: | (−) | (−) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Initial Value: | X | X | X | X | X | X | X | X | |

R/W: readable and writable
−: no access; read returns undefined value, write always 0 to this bit

### [bit 13] BPD: Buffer pointer decrement bit

| BPD bit | Function |
|---|---|
| 0 | In the case of BF=0, BAP is incremented after each transfer. |
| 1 | In the case of BF=0, BAP is decremented after each transfer.<br>When both BPD and BW are set to '1' (word transfers in reverse order), even numbered values should be specified for IOA, BAP and DCT. |

### [bit 12] IF: IOA update or fixed selection

| IF bit | Function |
|---|---|
| 0 | After each data transfer, IOA is incremented. |
| 1 | IOA is not updated (fixed I/O address). |

### [bit 11] BW: Transfer data length specification

| BW bit | Function |
|---|---|
| 0 | Byte transfers are issued. |
| 1 | Word transfers are issued. |

### [bit 10]: BF BAP update or fixed selection

| BF bit | Function |
|---|---|
| 0 | After each data transfer, BAP is updated. |
| 1 | BAP is not updated (fixed buffer address). |

### [bit 9] DIR: Data transfer direction

| DIR bit | Function |
|---|---|
| 0 | Transfer from address specified by IOABK and IOA to address specified by BAP (@IOABK.IOA -> @BAP). |
| 1 | Transfer from address specified by BAP to address specified by IOABK and IOA (@BAP -> @IOABK.IOA). |

### [bit 8] SE: DMA STOP request enable

| SE bit | Function |
|---|---|
| 0 | No reaction on DMA STOP request by peripheral. |
| 1 | DMA transfer can be stopped by a peripheral function (e.g. UART-RX). |

## 4.3.5 Buffer Address Pointer (BAP)

This section describes the Buffer Address Pointer (BAP).

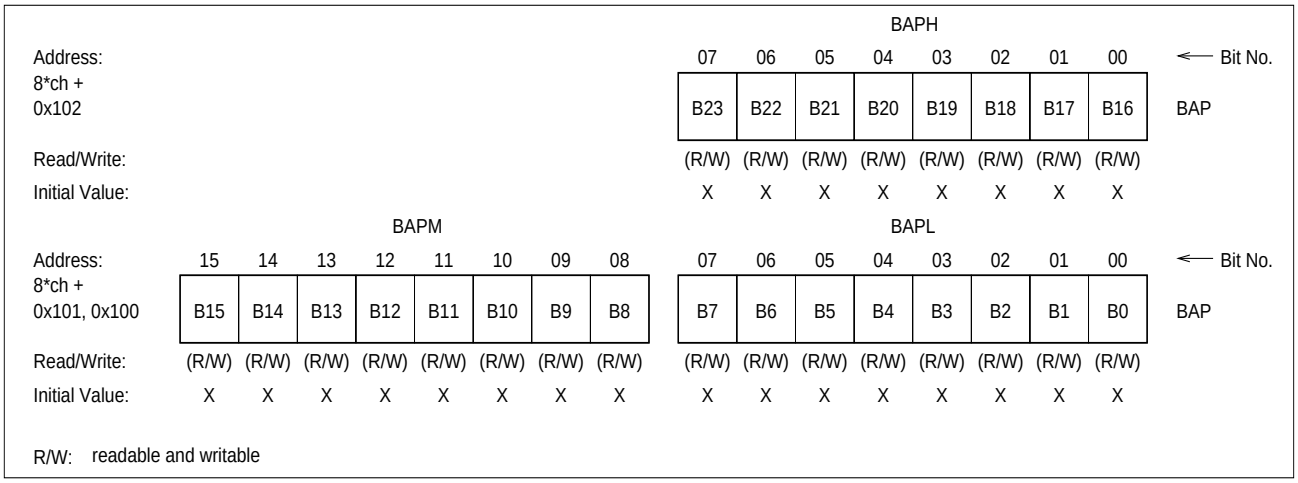
### *4.3.5.1 Buffer Address Pointer (BAP)*

The Buffer Address Pointer (BAP) is a 24-bit register and is used to store addresses that will be used for DMA transfer. Any address can be specified.

When "Update Performed" is specified using the DMACS:BF bit, the BAP lower 16 bits (BAPM, BAPL) are incremented by "1" at byte transfer and incremented by "2" at word transfer. Its upper 8 bits do not change. When DMACS:BPD is set, BAP is decremented by "1" or "2" instead of incremented. When “Update Not Performed” is specified with the DMACS:BF bit, BAP is fixed (BAP does not change).

Figure 4-12 shows the configuration of the Buffer Address Pointer (BAP).

Figure 4-12. Buffer Address Pointer (BAP)




**Notes:**

The area that can be specified using the I/O Address Bank Register (IOABK) and I/O Address Pointer (IOA) is "$000000_H$" to "$FFFFFF_H$". IOABK defaults to bank "$00_H$" after reset.

The area that can be specified using the Buffer Address Pointer (BAP) is "$000000_H$" to "$FFFFFF_H$".

The following addresses should not be specified for IOA and BAP: addresses of DMA controller internal registers (DISEL0...15, DSR, DSSR, DER); addresses of DMA descriptors "$000100_H$" to "$00017F_H$"; addresses of DMA IOABK registers "$000A00_H$" to "$000A0F_H$".

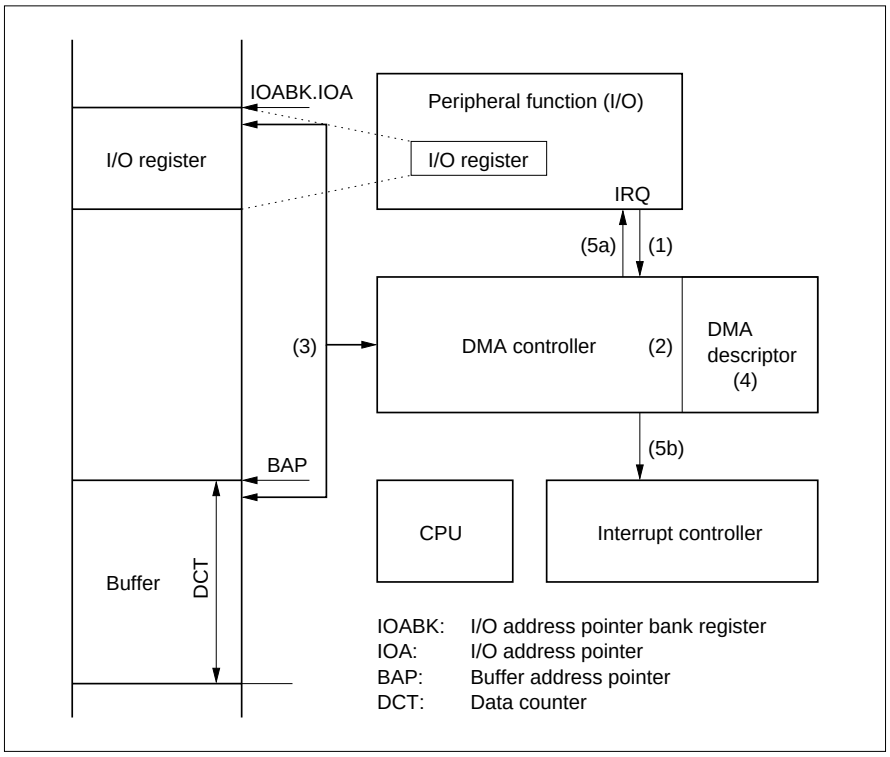
## 4.4 DMA Controller Operation

This section describes the DMA controller operation.

### DMA Controller Operation

Figure 4-13 shows the DMA controller operation.

Figure 4-13. DMAC Operation

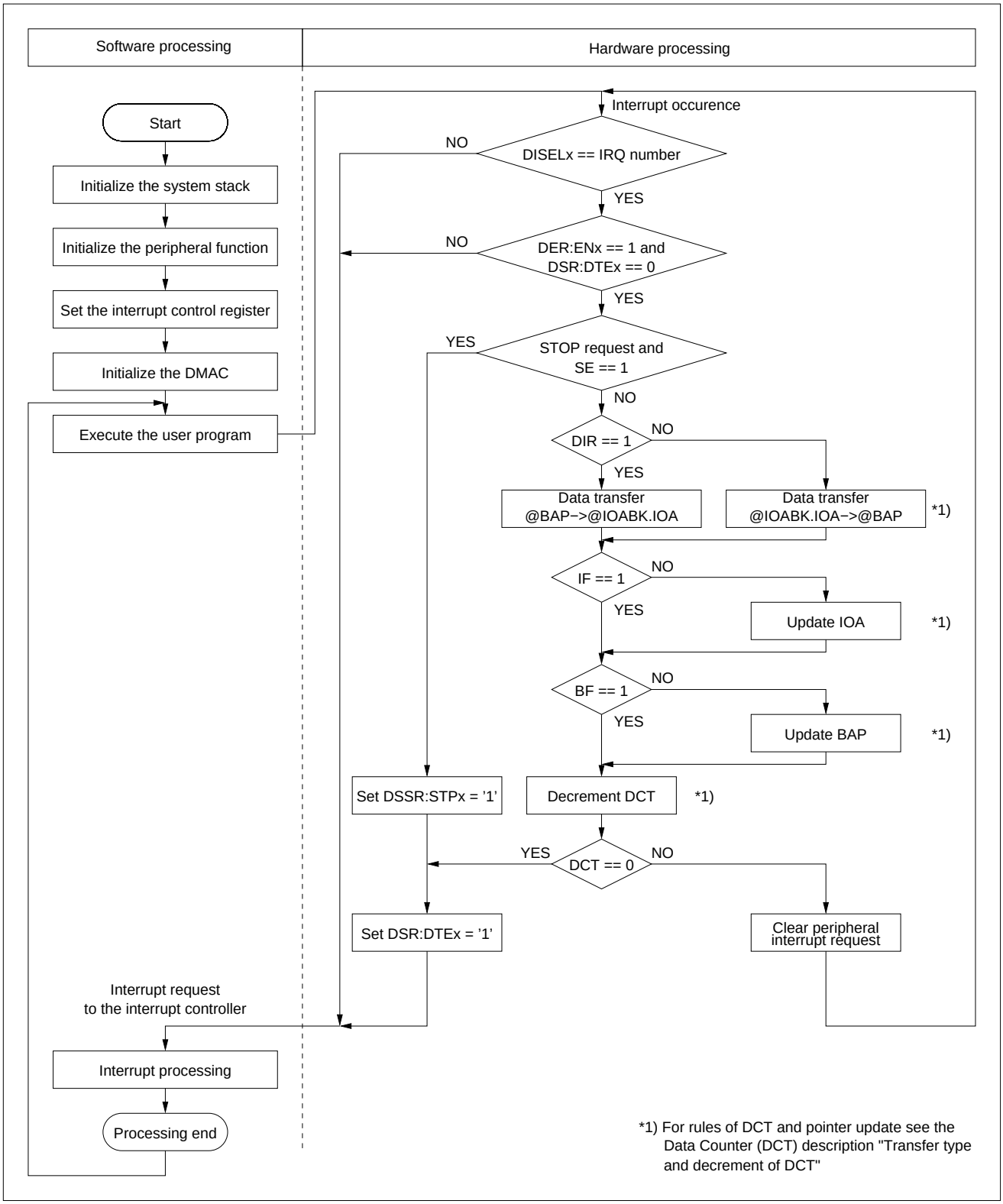



Data transfer using DMA is performed in the following order:

1. The peripheral resource (I/O) requests DMA transfer by asserting an interrupt.
2. When the corresponding bit of the DMA Enable Register (DER:EN) is "1" and the interrupt number matches the setting of the DISEL register, DMAC references from the descriptor the transfer source address, destination address, count and configuration of the channel.
3. DMA data transfer is started between I/O and memory.
4. DMA descriptors are updated.
5. After one item transferred (either Byte data or Word data)
   a. Transfer has not been completed (DCT does not reach 0):
   DMAC clears the DMA transfer request (interrupt) of the peripheral resource.
   b. At transfer end (DCT reaches 0):
   After completion of DMA transfer, the flag indicating completion of the transfer is set in the DMA Status Register (DSR:DTE bit), outputting an interrupt request to the interrupt controller.

## Procedure for using DMAC

Figure 4-14. Procedure for using DMAC



### Number of cycles for the data transfer

The number of transfer cycles (bus cycles during DMA transfer) is the sum of all single transfers until DCT reaches 0. For reference of the number of cycles for a single DMA transfer, see Table 4-2, column "Transfer cycles".

## 4.5 Examples of DMA transfers

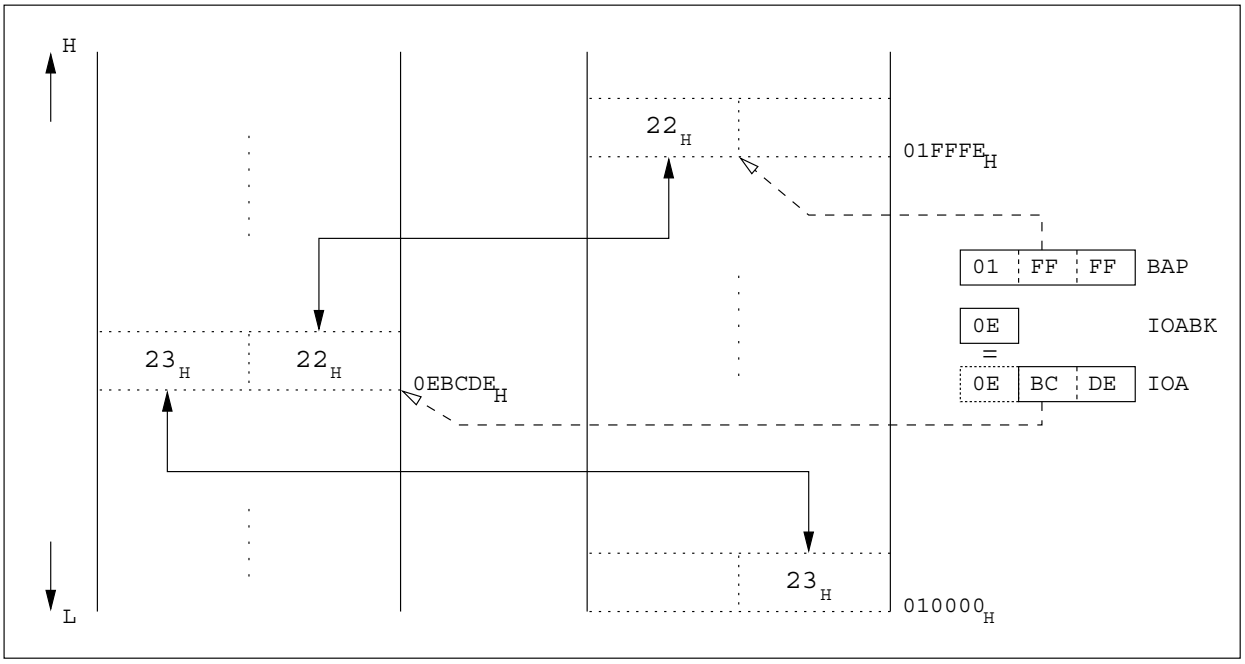
This section describes some of the lesser ordinary DMA transfer types or circumstances.

To simplify matters an even address somewhere in the memory bank selected by IOABK is used as IOA in the following examples. These examples still hold if the addresses of IOA and BAP are exchanged. The only exception is in the cases where BAP is to be decremented.

### Wrap around at bank boundary

A DMA transfer does not cross a bank boundary under any circumstances. If IOA or BAP are to be modified, their updated values fulfill this requirement. If a word has to be read from the top address of a bank, the lower byte is taken from the top address of the bank and the higher byte is taken from address 0 within the bank. Writing to the top address is handled accordingly (see Figure 4-15).

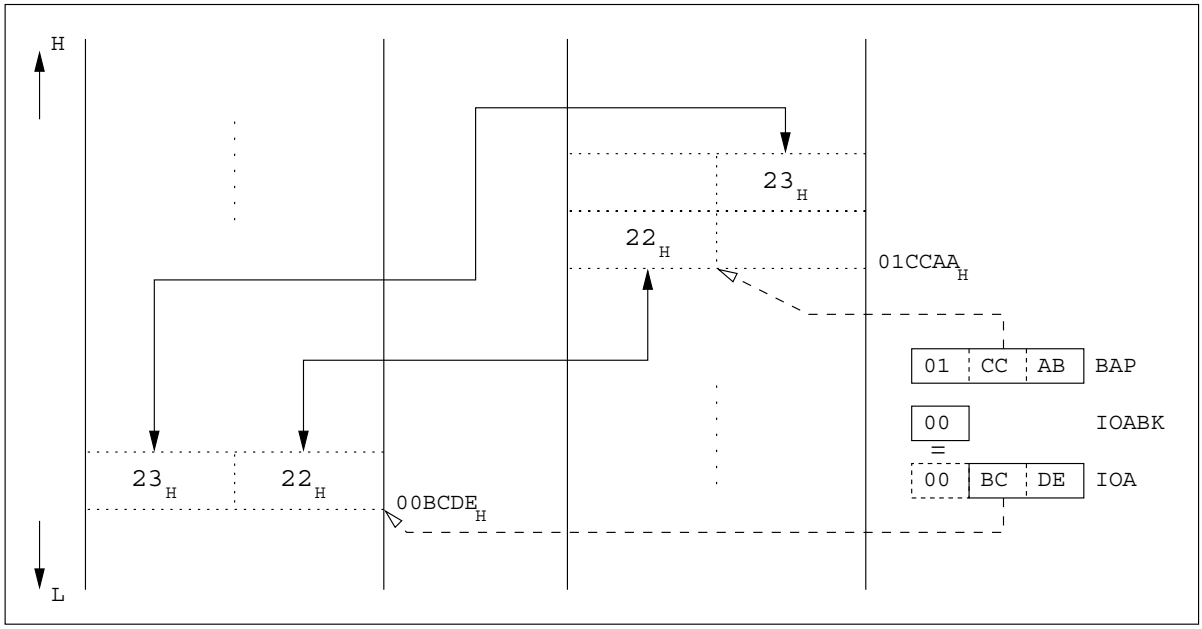
Figure 4-15. Transferring one word from/to the top address of bank $01_H$



### Word transfer to an odd address

A word transfer incorporating an odd address is split in byte transfers as stated in Table 4-2. The result of such a transfer is shown in Figure 4-16.

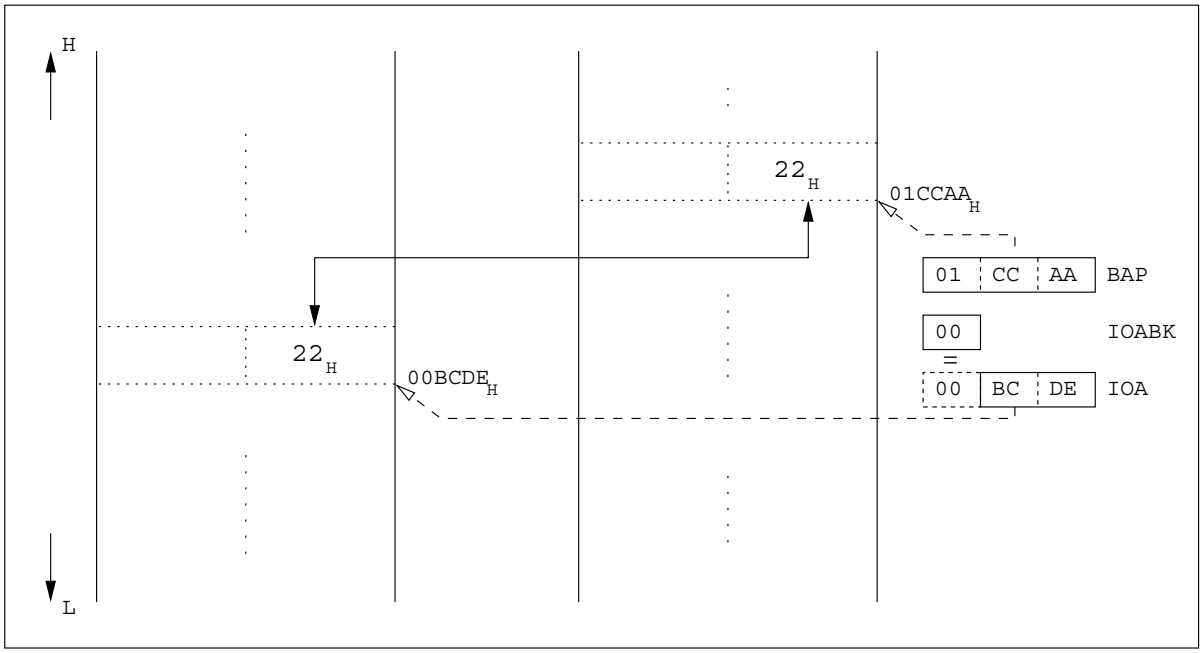
Figure 4-16. Transferring one word to/from an odd address



### Word transfer with odd byte count

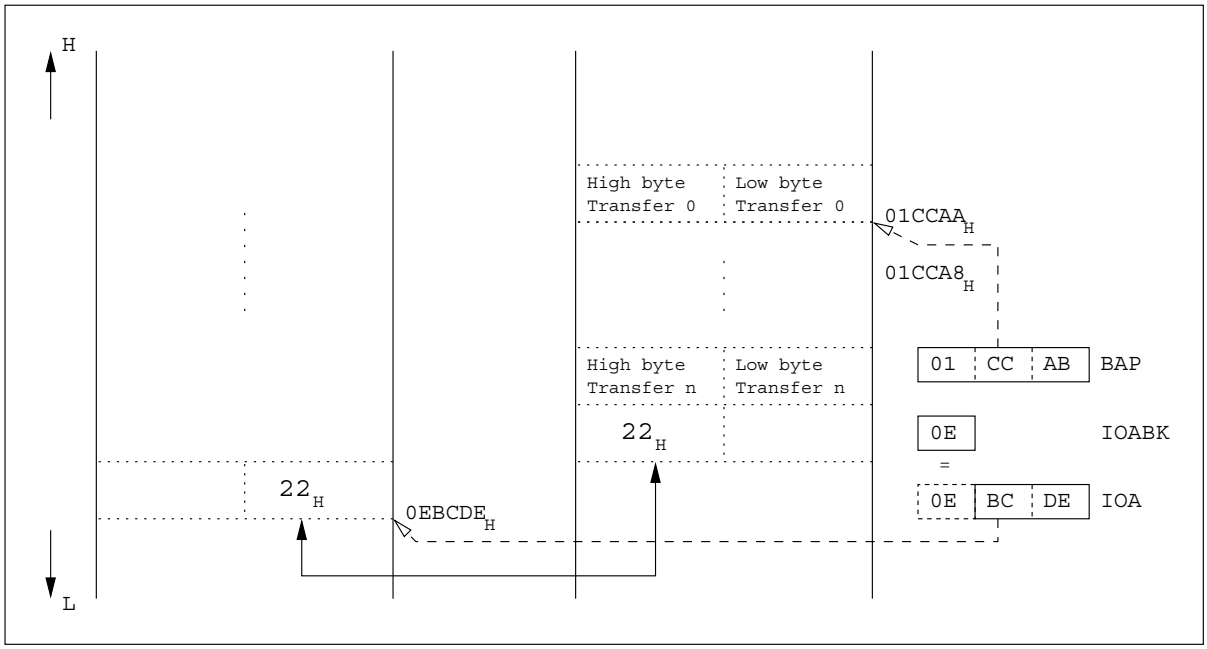
When DCT is set to an odd number of bytes and word transfers are used, the last transfer moves only one byte. In the case of BAP not being decremented (i.e. it is either fixed or to be incremented) the lower byte of the source is moved to the lower byte of the target (see Figure 4-17).

Figure 4-17. Transfer of the last byte of a channel set for word transfer




When BAP is to be decremented, the transferred byte is read from or written to the high byte of the word which is referred to by BAP (see Figure 4-18).

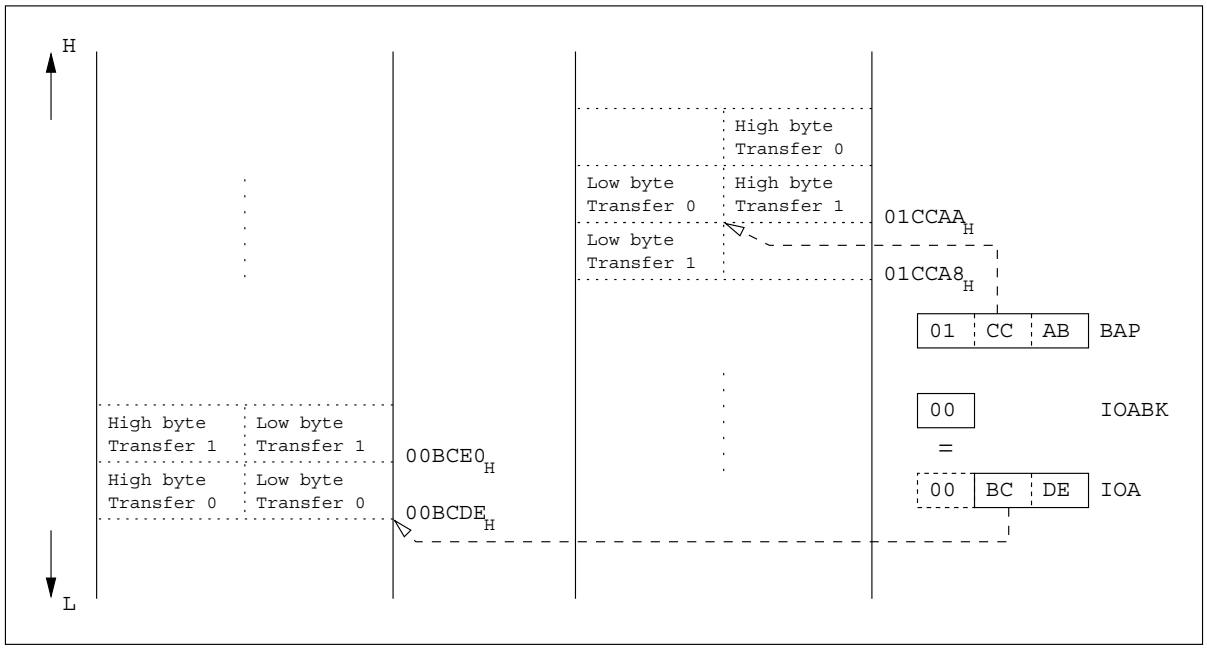
Figure 4-18. Transfer of the last byte of a channel set to word transfer and BAP decrement



### Word transfers with incrementing IOA and decrementing BAP

When IOA is set to be incremented and BAP is set to be decremented, the order of the transmitted words is reversed, but not the order of the bytes inside a word. Figure 4-19 exemplifies the memory contents after the completion of two transfers.

Figure 4-19. Word transfer with incrementing IOA and decrementing BAP



# 5. Delayed Interrupt

This chapter explains the functions and operations of the delayed interrupt.

5.1 Outline of Delayed Interrupt Module

5.2 Delayed Interrupt Register

5.3 Delayed Interrupt Operation

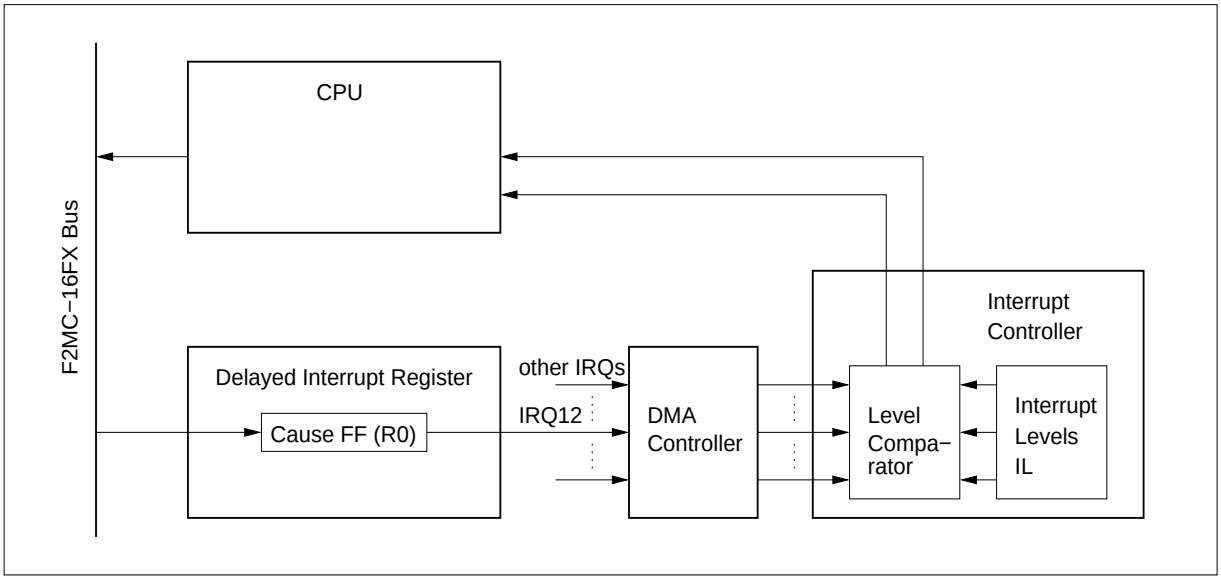
## 5.1 Outline of Delayed Interrupt Module

The Delayed Interrupt source module is used to generate interrupts for task switching. Using this module, interrupts to the $F^2MC$-16FX CPU can be requested and canceled by software.

### Block diagram of Delayed Interrupt

Figure 5-1 is a block diagram of the Delayed Interrupt source module.

Figure 5-1. Block diagram of Delayed Interrupt




### Notes on operation

The Delayed Interrupt signal is activated by writing "1" to the corresponding bit of DIRR and inactivated by writing "0" to the same bit. Therefore, the interrupt bit in DIRR should be cleared to "0" within the interrupt service routine. Otherwise the same interrupt is serviced again right after the first interrupt service is completed.

Compared to other hardware interrupts, the delayed interrupt has lowest priority in the case of same levels are configured in the ICR register. For the delayed interrupt, priority is independent from the interrupt number. The delayed interrupt uses the interrupt vector of INT #12.

## 5.2 Delayed Interrupt Register

DIRR controls request and cancellation of the Delayed Interrupt. Writing "1" to this register issues a delayed interrupt request, and writing "0" cancels the delayed interrupt request.

### Delayed Interrupt Cause Issuance/Cancellation register (DIRR: Delayed Interrupt Request Register)

Figure 5-2. Delayed Interrupt Cause/Cancel Register (DIRR)

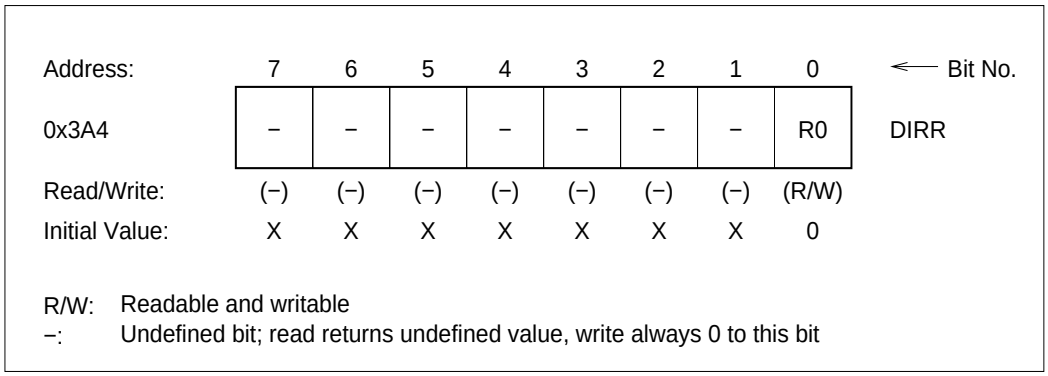



Table 5-1. Functional Explanation of Each Bit of the Delayed Interrupt Cause/Cancel Register (DIRR)

| Bit name | | Function |
|---|---|---|
| bit 7 to bit 1 | -:<br>Undefined bit | • When these bits are read, the values are undefined.<br>• Writing to these bits does not affect operation.<br>• Write always ’0’ to this bit |
| bit 0 | R0:<br>Delayed interrupt request output bit | • This bit sets the generation/cancel of a delayed interrupt request.<br>• When this bit is "1", a delayed interrupt request is output.<br>• When this bit is "0", the delayed interrupt request is cleared.<br>• This bit is cleared after reset. |

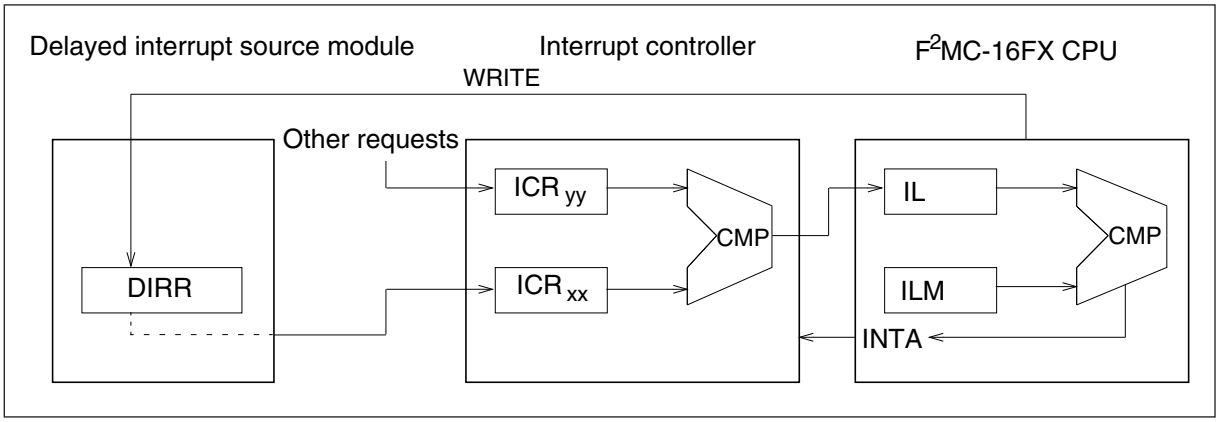
## 5.3 Delayed Interrupt Operation

When the CPU writes "1" to the relevant bit of DIRR by software, the request latch in the Delayed Interrupt source module is set and an interrupt request is issued to the interrupt controller.

### Delayed interrupt occurrence

When the CPU writes “1” to the relevant bit of DIRR by software, the request latch in the delayed interrupt source module is set and an interrupt request is issued to the interrupt controller. If this interrupt has the highest priority or if there is no other interrupt request, the interrupt controller issues an interrupt request to the $F^2$MC-16FX CPU. The $F^2$MC-16FX CPU compares the ILM bit of its internal CCR register and the interrupt request, and starts the hardware interrupt processing microprogram as soon as the current instruction is completed if the interrupt level of the request is higher than that of the ILM bit. Thus the interrupt processing routine for this interrupt is executed.

Figure 5-3. Delayed interrupt issuance



# 6. Clocks

This chapter describes the clocks used by F$^2$MC-16FX family micro controllers.

6.1 Clocks

6.2 Clock Control Registers

6.3 Clock Modes

6.4 Configuration of the PLL

6.5 Oscillation Stabilization Wait Time

6.6 Connection of an Oscillator or an External Clock to the Microcontroller

## 6.1 Clocks

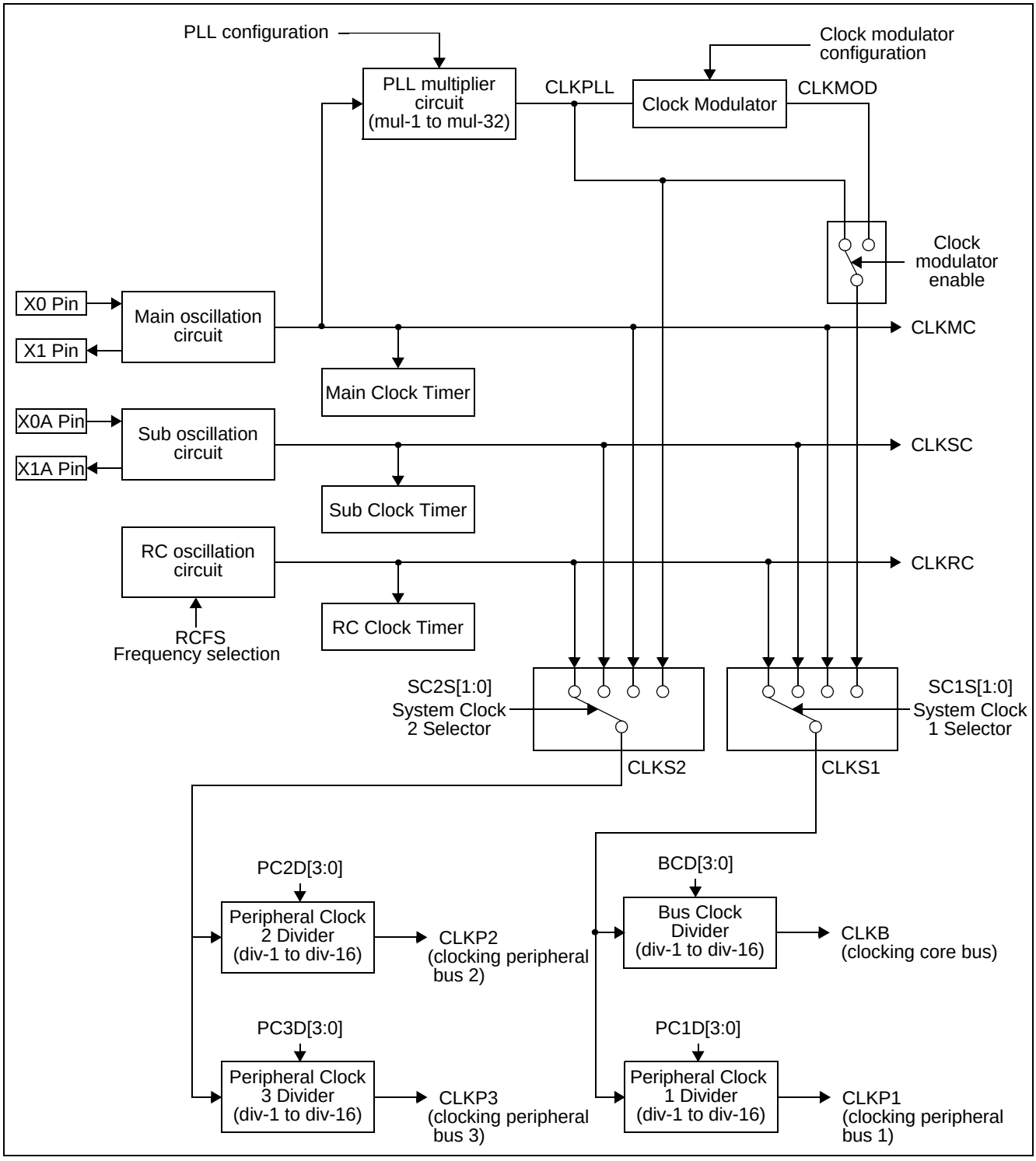
The F2MC-16FX MCU offers up to 4 different clock sources (RC clock, Main clock, PLL clock and Sub clock). Flexible clock dividers allow an independent setting of the Bus clock (for the internal bus with the CPU and memories) and for the Peripheral clocks frequencies.

The Sub oscillator is not included in all F2MC-16FX MCUs. The functions regarding the Sub oscillator and the Sub clock described in this hardware manual are not available for MCUs without Sub oscillator. The corresponding settings have no effect.

### Block Diagram

The following block diagram shows the clock sources, the generation of the internal clocks and the source clock timers of the F2MC-16FX MCU. The different clocks are described below. See also Overview on page 23 for a block diagram showing which modules are connected to which clocks.

Figure 6-1. Block diagram of the clock generation and source clocks timers




See datasheet for support of CLKP3.

### Source clocks

The following clock signals are the clock sources of the F2MC-16FX MCU.

- RC clock (CLKRC)

  The RC clock CLKRC is the output clock of the internal RC oscillator. The RC oscillator can generate two different clock frequencies (2MHz and 100kHz nominal, see datasheet) which can be selected with the RCFS (RC Clock Frequency Select) bit of the CKFCR register.

- Main clock (CLKMC)

  The Main clock CLKMC is the output clock of the main oscillation circuit. Either an external oscillator or an external clock can be connected to the main oscillation circuit.

- PLL clock (CLKPLL)

  The PLL clock CLKPLL is obtained by multiplying the Main clock CLKMC with the internal PLL clock multiplier circuit (PLL oscillation circuit). Multiplication rates from 1 to 32 are available.

This clock is only available when the Main clock CLKMC is active.

- Sub clock (CLKSC)

  The Sub clock CLKSC is the output clock of the sub oscillation circuit. Either an external oscillator or an external clock can be connected to the sub oscillation circuit.

  The Sub clock is not available for all F2MC-16FX MCUs.

### Internally derived clocks

The following clock signals are internally generated out of the source clocks.

- Modulated PLL clock (CLKMOD)

  The PLL clock CLKPLL can be modulated with the built-in clock modulator to reduce the electromagnetic emission. The output clock of this clock modulator is called CLKMOD.

- System clock 1 (CLKS1)

  The System clock 1 (CLKS1) is a master clock of the F2MC-16FX MCU. It feeds the clock divider for the Bus clock (CLKB) and the Peripheral Clock 1 (CLKP1) and defines the clock mode of the CPU.

  Depending on the SC1S[1:0] (System Clock 1 Select) bits of the CKSR register, one of the following 4 clocks can be selected as the System clock 1: CLKRC (RC clock), CLKMC (Main clock), CLKPLL/CLKMOD (unmodulated or modulated PLL clock) or CLKSC (Sub clock).

- Bus clock (CLKB)

  The Bus clock CLKB is the clock source for the internal bus, the CPU, internal memories, DMA controller and the external bus interface. It is stopped in all Standby modes (Sleep, Timer and Stop mode). It is generated by the programmable Bus clock divider out of the System clock 1 (CLKS1).

- Peripheral clock 1 (CLKP1)

  The Peripheral clock 1 (CLKP1) is the clock source for all peripheral resources that can operate with a modulated clock. It is stopped in Timer and Stop mode. It is generated by the programmable Peripheral clock divider 1 out of the System clock 1 (CLKS1).

- System clock 2 (CLKS2)

  The System clock 2 (CLKS2) is a second master clock which is never modulated. It feeds the clock divider for the Peripheral Clock 2 (CLKP2) only.

  Depending on the SC2S[1:0] (System Clock 2 Select) bits of the CKSR register, one of the following 4 clocks can be selected as the System clock 2: CLKRC (RC clock), CLKMC (Main clock), CLKPLL (unmodulated PLL clock) or CLKSC (Sub clock).

  The setting of these bits does not influence the clock mode of the CPU which is defined by SC1S[1:0] only.

- Peripheral clock 2 (CLKP2)

  The Peripheral clock 2 (CLKP2) is the clock source for peripheral resources that never operate with a modulated clock. It is stopped in Timer and Stop mode. It is generated by the programmable Peripheral clock divider 2 out of the System clock 2 (CLKS2).

- Peripheral clock 3 (CLKP3)

  The Peripheral clock 3 (CLKP3) is an optional second clock source for peripherals that never operate with a modulated clock and which need a different clock frequency than those connected to CLKP2. CLKP3 is stopped in Timer and Stop mode and is generated by the programmable Peripheral Clock Divider 3 from System clock 2 (CLKS2).

  CLKP3 clock is not available for all devices. Please refer to Datasheet.

  In case the USB function is used, CLKP3 needs to be set to 48MHz

### Source Clock Timers

The Main clock, RC clock and the Sub clock oscillators feed dedicated source clock timers (Main Clock Timer, RC Clock Timer and Sub Clock Timer) which are running independently of the selected System clock, Bus clock and Peripheral clock.

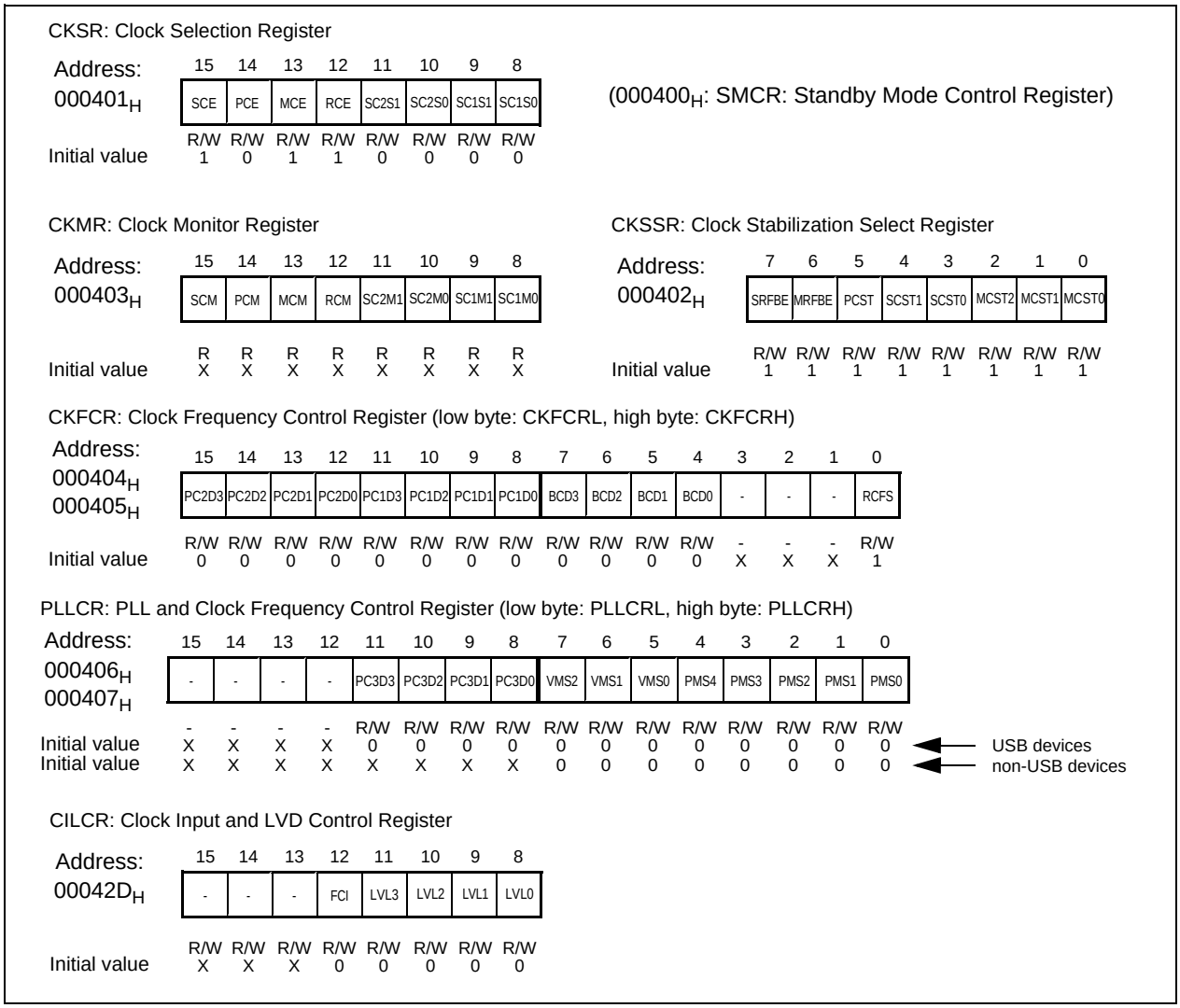
## 6.2 Clock Control Registers

This section lists the clock control registers and describes the function of each register in detail.

### Clock Control Registers

Figure 6-2 shows an overview of all clock control registers

Figure 6-2. Clock Control Registers



## 6.2.1 Clock Selection Register (CKSR)

The Clock Selection Register (CKSR) is used to control the System Clock selector 1 and 2 and the oscillation circuits.

### Configuration of the Clock Selection Register (CKSR)

Figure 6-3 shows the configuration of the Clock Selection Register (CKSR) and Table 6-1 describes the function of each bit.

Figure 6-3. Configuration of the Clock Selection Register (CKSR)

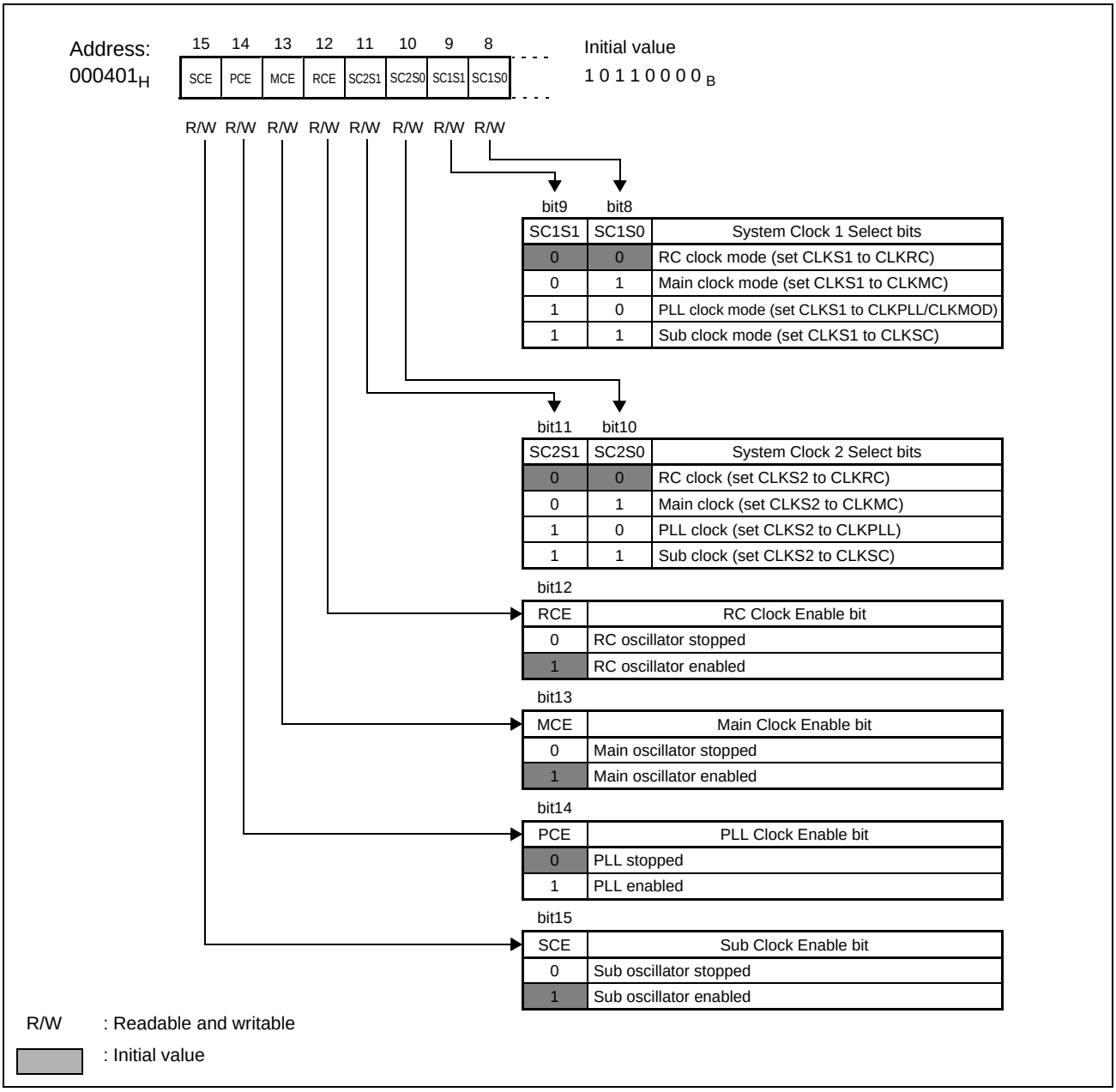


Table 6-1. Function Description of Each Bit of the Clock Selection Register (CKSR) (Sheet 1 of 2)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 8 - bit 9</td><td>SC1S0 and SC1S1:<br>System Clock 1 Select bits</td><td>
• These bits control the System Clock 1 Selector for the source of the Bus clock and the Peripheral clock 1 according to the following table:
<table>
<tr><th colspan="2">bit9 bit8</th><th></th><th></th></tr>
<tr><td>SC1S1</td><td>SC1S0</td><td>Clock mode</td><td>CLKS1 (System Clock 1)</td></tr>
<tr><td>0</td><td>0</td><td>RC clock mode</td><td>set to CLKRC (RC clock)</td></tr>
<tr><td>0</td><td>1</td><td>Main clock mode</td><td>set to CLKMC (Main clock)</td></tr>
<tr><td>1</td><td>0</td><td>PLL clock mode</td><td>set to CLKPLL or CLKMOD (PLL clock)</td></tr>
<tr><td>1</td><td>1</td><td>Sub clock mode</td><td>set to CLKSC (Sub clock)</td></tr>
</table>
• These bits control the CPU clock mode.<br>
• Any reset initializes these bits to "00" (RC clock mode).<br>
• A transition to a different clock mode is delayed until the selected clock is ready (means the corresponding clock ready monitor bit in the Clock Monitor Register CKMR is set).<br>
• If the selected clock is ready, then the clock mode transition will be executed by starting the synchronization mechanism. The System Clock 1 Monitor bits (SC1M) of the Clock Monitor register indicate when this clock switching has completed.<br>
• Writing a second value to this register before switching to the mode selected beforehand has completed cancels the first transition request.<br>
• A transition to a clock which is not available (oscillator failed or is not existent) is not possible.<br>
Note:<br>
• In Stop mode, the System Clock 1 Selector directly changes to the clock selected by the SC1S bits, independently of the corresponding clock ready monitor bit (in Stop mode, all clock monitor bits are cleared). Hence do not select an unavailable clock before switching to Stop mode.
</td></tr>
<tr><td>bit 10 - bit 11</td><td>SC2S0 and SC2S1:<br>System Clock 2 Select bits</td><td>
• These bits control the System Clock 2 Selector for the source of the Peripheral clock 2 according to the following table:
<table>
<tr><th>bit11</th><th>bit10</th><th></th></tr>
<tr><td>SC2S1</td><td>SC2S0</td><td>CLKS2 (System Clock 2)</td></tr>
<tr><td>0</td><td>0</td><td>set to CLKRC (RC clock)</td></tr>
<tr><td>0</td><td>1</td><td>set to CLKMC (Main clock)</td></tr>
<tr><td>1</td><td>0</td><td>set to CLKPLL (Unmodulated PLL clock)</td></tr>
<tr><td>1</td><td>1</td><td>set to CLKSC (Sub clock)</td></tr>
</table>
• Any reset initializes these bits to "00" (RC clock).<br>
• Changing to a different clock source is delayed until the selected clock is ready (means the corresponding clock ready monitor bit in the Clock Monitor Register CKMR is set).<br>
• If the selected clock is ready, then the clock source transition will be executed by starting the synchronization mechanism. The System Clock 2 Monitor bits (SC2M) of the Clock Monitor register indicate when this clock switching has completed.<br>
• Writing a second value to this register before switching to the mode selected beforehand has completed cancels the first transition request.<br>
• A transition to a clock which is not available (oscillator failed or is not existent) is not possible.<br>
Note:<br>
• In Stop mode, the System Clock 2 Selector directly changes to the clock selected by the SC2S bits, independent of the corresponding clock ready monitor bit (in Stop mode, all clock monitor bits are cleared). Hence do not select an unavailable clock before switching to Stop mode.
</td></tr>
</table>

Table 6-1. Function Description of Each Bit of the Clock Selection Register (CKSR) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 12 | RCE:<br>RC Clock Enable bit | • This bit is used to enable/stop the internal RC oscillator.<br>• Writing "1" to this bit enables the RC oscillator and writing "0" stops the oscillator.<br>• This bit is initialized to "1" (oscillator on) by each reset.<br>• The RC oscillation stabilization time is applied after enabling the RC oscillator. This time is specified in the data-sheet. After this time has elapsed, the RC clock monitor bit (RCM) in the CKMR register is set.<br>• Selecting CLKRC as clock source for CLKS1 or CLKS2 activates the RC oscillator independently of the setting of the RCE bit. The oscillator can only be stopped when both System clocks are running with a different clock source.<br>**Note:**<br>• Do not disable the RC oscillator if the Clock stop detection reset is enabled. Setting RCE to "0" in this case causes a Clock stop detection reset.<br>• Do not disable the RC oscillator if the clock source for the Watchdog is set to RC clock. Setting RCE to "0" in this case causes a Watchdog reset. |
| bit 13 | MCE:<br>Main Clock Enable bit | • This bit is used to enable/stop the Main oscillation circuit.<br>• Writing "1" to this bit enables the Main oscillator and writing "0" stops the oscillator.<br>• This bit is initialized to "1" (oscillator on) by each reset.<br>• The Main oscillation stabilization time is applied after enabling the Main oscillator. This time is defined by the MCST bits of the CKSSR register. After this time has elapsed, the Main clock monitor bit (MCM) in the CKMR register is set.<br>• Selecting CLKMC, CLKPLL or CLKMOD as clock source for CLKS1 or CLKS2 activates the Main oscillator inde-pendently of the setting of the MCE bit. The oscillator can only be stopped when both System clocks are running with different clock sources.<br>• Setting this bit to "0" also disables the PLL clock (independently of the PCE setting) and sets PCM to "0" unless the PLL clock is used as System clock 1 or 2.<br>**Note:**<br>• Do not disable the Main oscillator if the clock source for the Watchdog is set to Main clock. Setting MCE to "0" in this case causes a Watchdog reset. |
| bit 14 | PCE:<br>PLL Clock Enable bit | • This bit is used to enable/stop the PLL oscillation circuit.<br>• Writing "1" to this bit enables the PLL and writing "0" stops the PLL.<br>• MCE must also be set to "1" if the PLL should be enabled because the PLL is using the Main clock as input clock. For MCE = "0", the setting of PCE has no effect.<br>• This bit is initialized to "0" (PLL stopped) by each reset.<br>• The PLL stabilization time is applied after stabilization of the Main clock (MCM = "1") and enabling of the PLL. This time is defined by the PCST bit of the CKSSR register. After this time has elapsed, the PLL clock monitor bit (PCM) in the CKMR register is set.<br>• Selecting CLKPLL or CLKMOD as clock source for CLKS1 or CLKS2 activates the PLL independently of the set-ting of the PCE bit. The PLL can only be stopped when both System clocks are running with different clock sources. |
| bit 15 | SCE:<br>Sub Clock Enable bit | • This bit is used to enable/stop the Sub oscillation circuit.<br>• Writing "1" to this bit enables the Sub oscillator and writing "0" stops the oscillator.<br>• This bit is initialized to "1" (oscillator on) by each reset.<br>• The Sub oscillation stabilization time is applied after enabling the Sub oscillator. This time is defined by the SCST bits of the CKSSR register. After this time has elapsed, the Sub clock monitor bit (SCM) in the CKMR register is set.<br>• Selecting CLKSC as clock source for CLKS1 or CLKS2 activates the Sub oscillator independently of the setting of the SCE bit. The oscillator can only be stopped when both System clocks are running with a different clock source.<br>**Note:**<br>• Do not disable the Sub oscillator if the clock source for the Watchdog is set to Sub clock. Setting SCE to "0" in this case causes a Watchdog reset. |

## 6.2.2 Clock Monitor Register (CKMR)

The Clock Monitor Register (CKMR) is used to check the current status of the System clocks (Clock mode) and the status of the oscillation circuits.

### Configuration of the Clock Monitor Register (CKMR)

Figure 6-4 shows the configuration of the Clock Monitor Register (CKMR) and Table 6-2 describes the function of each bit.

Figure 6-4. Configuration of the Clock Monitor Register (CKMR)

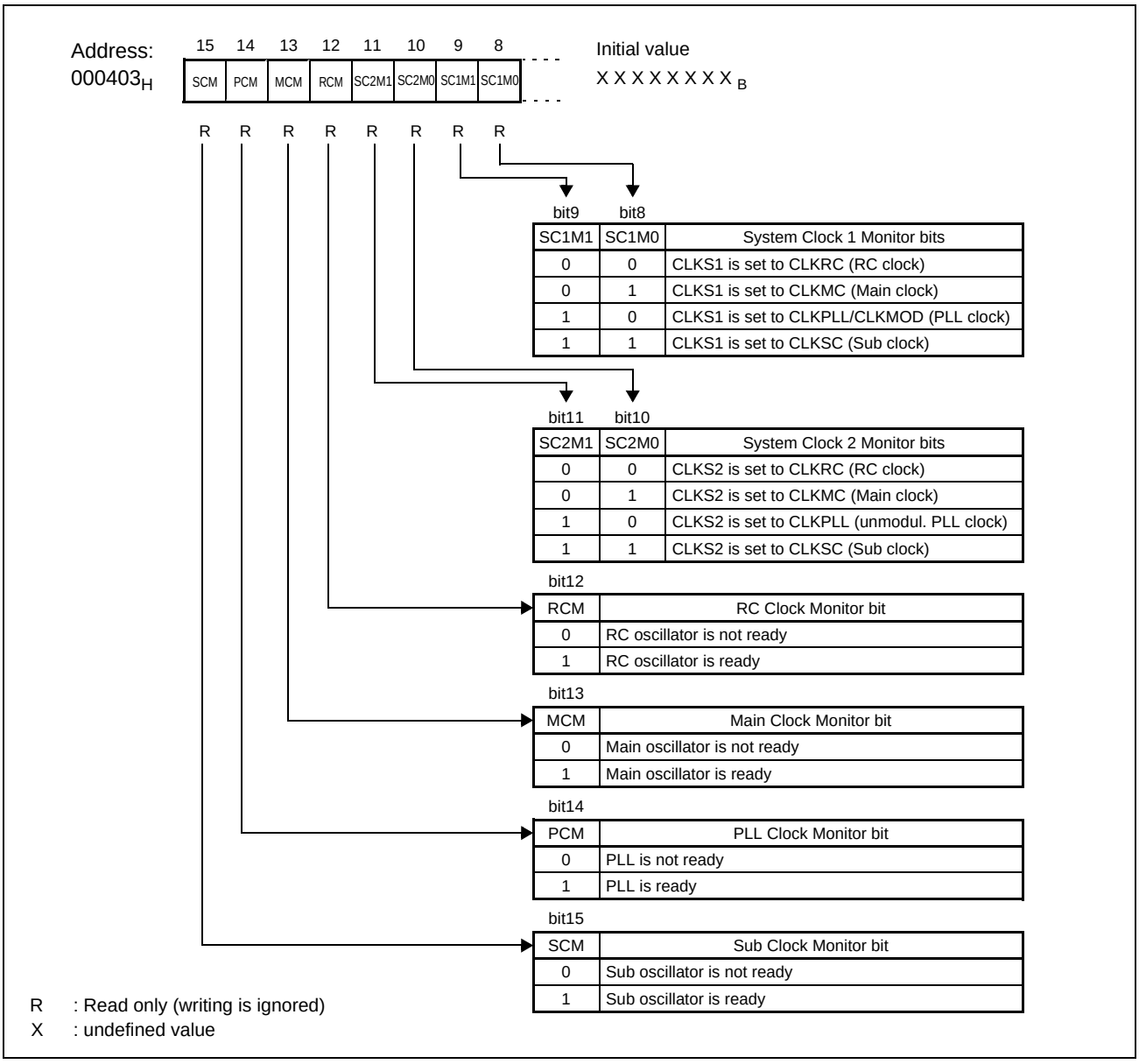


Table 6-2. Function Description of Each Bit of the Clock Monitor Register (CKMR) (Sheet 1 of 2)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 8 - bit 9</td><td>SC1M0 and SC1M1:<br>System Clock 1 Monitor bits</td><td>
• These bits indicate which clock is currently used for the System Clock 1 according to the following table:
<table>
<tr><th></th><th>bit9</th><th>bit8</th><th></th></tr>
<tr><th></th><th>SC1M1</th><th>SC1M0</th><th>System Clock 1 Monitor bits</th></tr>
<tr><td></td><td>0</td><td>0</td><td>CLKS1 is set to CLKRC (RC clock)</td></tr>
<tr><td></td><td>0</td><td>1</td><td>CLKS1 is set to CLKMC (Main clock)</td></tr>
<tr><td></td><td>1</td><td>0</td><td>CLKS1 is set to CLKPLL/CLKMOD (PLL clock)</td></tr>
<tr><td></td><td>1</td><td>1</td><td>CLKS1 is set to CLKSC (Sub clock)</td></tr>
</table>
• These bits show the currently active CPU clock mode.<br>
• The read value of the SC1M bits can differ from the value written to the SC1S System Clock 1 Select bits. This indicates that a requested clock mode transition has not been completed yet.<br>
• An ongoing transition of the clock mode (synchronization mechanism in effect) is indicated as follows: Active clock mode shown by the SC1M bits differs from the SC1S setting although clock selected by SC1S bits is ready (clock monitor bit ’1’). The SC1M bits are updated after completion of the clock mode transition.<br>
• Clock mode switching is not possible when the selected clock is not ready (clock monitor bit of the clock selected by the SC1S bits is "0").
</td></tr>
<tr><td>bit 10 - bit 11</td><td>SC2M0 and SC2M1:<br>System Clock 2 Monitor bits</td><td>
• These bits indicate which clock is currently used for the System Clock 2 according to the following table:
<table>
<tr><th></th><th>bit11</th><th>bit10</th><th></th></tr>
<tr><th></th><th>SC2M1</th><th>SC2M0</th><th>System Clock 2 Monitor bits</th></tr>
<tr><td></td><td>0</td><td>0</td><td>CLKS2 is set to CLKRC (RC clock)</td></tr>
<tr><td></td><td>0</td><td>1</td><td>CLKS2 is set to CLKMC (Main clock)</td></tr>
<tr><td></td><td>1</td><td>0</td><td>CLKS2 is set to CLKPLL (unmodul. PLL clock)</td></tr>
<tr><td></td><td>1</td><td>1</td><td>CLKS2 is set to CLKSC (Sub clock)</td></tr>
</table>
• The read value of the SC2M bits can differ from the value written to the SC2S System Clock 2 Select bits. This indicates that a requested clock mode transition has not been completed yet.<br>
• An ongoing transition of the clock mode (synchronization mechanism in effect) is indicated as follows: Active clock mode shown by the SC2M bits differs from the SC2S setting although clock selected by SC2S bits is ready (clock monitor bit ’1’). The SC2M bits are updated after completion of the clock mode transition.<br>
• Clock mode switching is not possible when the selected clock is not ready (clock monitor bit of the clock selected by the SC2S bits is "0").
</td></tr>
<tr><td>bit 12</td><td>RCM:<br>RC Clock Monitor bit</td><td>
• This bit indicates if the internal RC oscillator is ready or not.<br>
• RCM = "1" means that the RC oscillator is ready and can be used. If RCM = "1" although RCE was set to "0", then the RC oscillator has not been disabled because the RC clock is used for System Clock 1 or 2.<br>
• RCM = "0" means that the RC oscillator is either disabled or the RC oscillation stabilization time is in effect.<br>
• Any reset initializes this bit to "0" and stops the operation of the MCU. After the RC oscillation stabilization wait time, this bit is set to "1" and the operation of the MCU resumes with the execution of the reset sequence.
</td></tr>
<tr><td>bit 13</td><td>MCM:<br>Main Clock Monitor bit</td><td>
• This bit indicates if the main oscillator is ready or not.<br>
• MCM = "1" means that the main oscillator is ready and can be used. If MCM = "1" although MCE was set to "0", then the Main oscillator has not been disabled because the Main clock or PLL clock is used for System Clock 1 or 2.<br>
• MCM = "0" means that the main oscillator is either disabled or the main oscillation stabilization time is in effect.<br>
• A Power or External reset (RSTX falling edge) and a Main clock stop detection reset initializes this bit to "0". See section Startup after Power and External reset on page 162 for more details regarding the effect of RSTX.<br>
• A Sub clock stop detection, Software or Watchdog reset does not reset this bit.<br>
Note:<br>
The Main Clock Stabilization Time select bits CKSSR:MCST[2:0] are reset to "111" ($2^{18}$ CLKMC cycles) by any reset. Hence if any reset is asserted within $2^{18}$ CLKMC cycles after activation of the Main oscillator, then MCM will be cleared to ’0’.
</td></tr>
<tr><td>bit 14</td><td>PCM:<br>PLL Clock Monitor bit</td><td>
• This bit indicates if the PLL is ready or not.<br>
• PCM = "1" means that the PLL clock is ready and can be used. If PCM = "1" although PCE was set to "0", then the PLL has not been disabled because the PLL is used for System Clock 1 or 2.<br>
• PCM = "0" means that the PLL is either disabled or the PLL stabilization time is in effect.<br>
• Any reset initializes this bit to "0" (PLL disabled).
</td></tr>
</table>

Table 6-2. Function Description of Each Bit of the Clock Monitor Register (CKMR) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 15 | SCM:<br>Sub Clock Monitor bit | • This bit indicates if the sub oscillator is ready or not.<br>• SCM = "1" means that the sub oscillator is ready and can be used. If SCM = "1" although SCE was set to "0", then the Sub oscillator has not been disabled because the Sub clock is used for System Clock 1 or 2.<br>• SCM = "0" means that the sub oscillator is either disabled or the sub oscillation stabilization time is in effect.<br>• A Power or External reset and a Sub clock stop detection reset initializes this bit to "0".<br>• A Main clock stop detection, Software or Watchdog reset does not reset this bit.<br>**Note:**<br>The Sub Clock Stabilization Time select bits CKSSR:SCST[1:0] are reset to "11" ($2^{16}$ CLKSC cycles) by any reset. Hence if any reset is asserted within $2^{16}$ CLKSC cycles after activation of the Sub oscillator, then SCM will be cleared to '0'. |

## 6.2.3 Clock Stabilization Select Register (CKSSR)

The Clock Stabilization Select Register (CKSSR) is used to select the stabilization times for the oscillation circuits, the PLL and for controlling the feedback resistors of the Main and Sub oscillation circuits.

### Configuration of the Clock Stabilization Select Register (CKSSR)

Figure 6-5 shows the configuration of the Clock Stabilization Select register (CKSSR) and Table 6-3 describes the function of each bit.

Figure 6-5. Configuration of the Clock Stabilization Select Register (CKSSR)

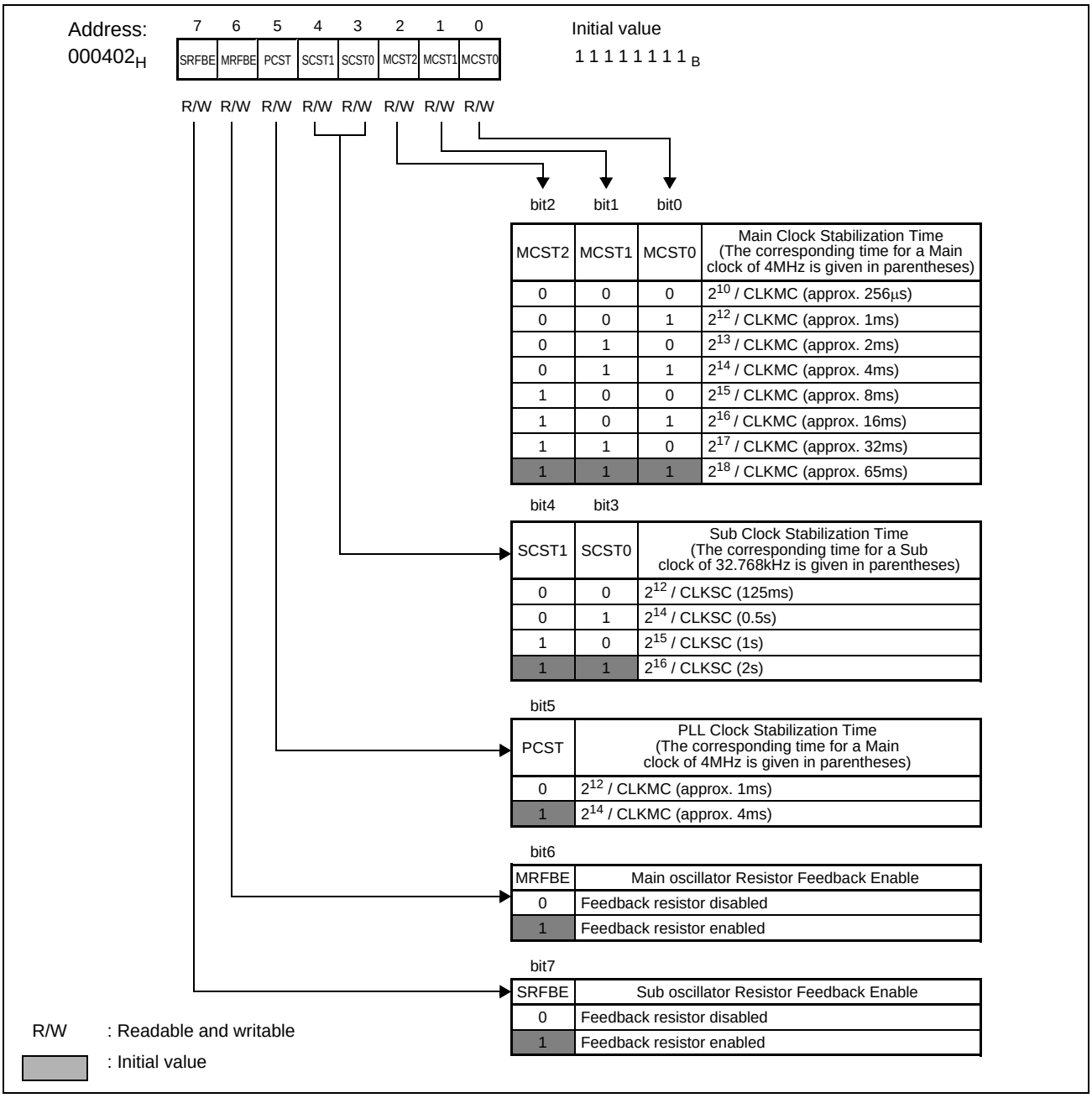


Table 6-3. Function Description of Each Bit of the Clock Stabilization Select Register (CKSSR) (Sheet 1 of 2)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 2</td><td>MCST0 to MCST2:<br>Main Clock Stabilization Time select bits</td><td>
• These bits select the stabilization time for the Main oscillation circuit according to the following table:
<table>
<tr><th>bit2</th><th>bit1</th><th>bit0</th><th></th></tr>
<tr><td>MCST2</td><td>MCST1</td><td>MCST0</td><td>Main Clock Stabilization Time (The corresponding time for a Main clock of 4MHz is given in parentheses)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>$2^{10}$ / CLKMC (approx. 256µs)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>$2^{12}$ / CLKMC (approx. 1ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>$2^{13}$ / CLKMC (approx. 2ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>$2^{14}$ / CLKMC (approx. 4ms)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>$2^{15}$ / CLKMC (approx. 8ms)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>$2^{16}$ / CLKMC (approx. 16ms)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>$2^{17}$ / CLKMC (approx. 32ms)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>$2^{18}$ / CLKMC (approx. 65ms)</td></tr>
</table>
• Any reset initializes these bits to "111".<br>
• These bits define after how many Main clock (CLKMC) cycles the Main Clock Monitor (MCM) bit will be set, allowing to use the Main clock as System clock.<br>
Note:<br>
The oscillation stabilization wait interval must be set to a value appropriate for the oscillator used.<br>
The initialization of these bits to "111" clears the CKMR:MCM bit in case the reset was asserted within $2^{18}$ CLKMC cycles after activation of the Main oscillator.
</td></tr>
<tr><td>bit 3 - bit 4</td><td>SCST0 to SCST1:<br>Sub Clock Stabilization Time select bits</td><td>
• These bits select the stabilization time for the Sub oscillation circuit according to the following table:
<table>
<tr><th>bit4</th><th>bit3</th><th></th></tr>
<tr><td>SCST1</td><td>SCST0</td><td>Sub Clock Stabilization Time (The corresponding time for a Sub clock of 32.768kHz is given in parentheses)</td></tr>
<tr><td>0</td><td>0</td><td>$2^{12}$ / CLKSC (125ms)</td></tr>
<tr><td>0</td><td>1</td><td>$2^{14}$ / CLKSC (0.5s)</td></tr>
<tr><td>1</td><td>0</td><td>$2^{15}$ / CLKSC (1s)</td></tr>
<tr><td>1</td><td>1</td><td>$2^{16}$ / CLKSC (2s)</td></tr>
</table>
• Any reset initializes these bits to "11".<br>
• These bits define after how many Sub clock (CLKSC) cycles the Sub Clock Monitor (SCM) bit will be set, allowing to use the Sub clock as System clock.<br>
Note:<br>
The oscillation stabilization wait interval must be set to a value appropriate for the oscillator used.<br>
The initialization of these bits to "11" clears the CKMR:SCM bit in case the reset was asserted within $2^{16}$ CLKSC cycles after activation of the Sub oscillator.
</td></tr>
<tr><td>bit 5</td><td>PCST:<br>PLL Clock Stabilization Time select bit</td><td>
• This bits selects the stabilization time for the PLL clock according to the following table:
<table>
<tr><th>bit5</th><th></th></tr>
<tr><td>PCST</td><td>PLL Clock Stabilization Time (The corresponding time for a Main clock of 4MHz is given in parentheses)</td></tr>
<tr><td>0</td><td>$2^{12}$ / CLKMC (approx. 1ms)</td></tr>
<tr><td>1</td><td>$2^{14}$ / CLKMC (approx. 4ms)</td></tr>
</table>
• Any reset initializes this bit to "1".<br>
• This bit defines after how many Main clock (CLKMC) cycles the PLL Clock Monitor (PCM) bit will be set, allowing to use the PLL clock as System clock.<br>
• The PLL clock stabilization time starts counting after stabilization of the Main clock (MCM = "1") and after setting PCE to "1"<br>
• Setting PCST to "1" is only permitted for a Main clock (CLKMC) frequency up to 8MHz.
</td></tr>
<tr><td>bit 6</td><td>MRFBE: Main oscillator Resistor Feedback Enable</td><td>
• This bit controls the feedback resistor of the Main oscillator.<br>
• Writing "1" to this bit enables the feedback resistor and writing "0" disables the resistor.<br>
• Any reset initializes this bit to "1" (resistor enabled).
</td></tr>
</table>

Table 6-3. Function Description of Each Bit of the Clock Stabilization Select Register (CKSSR) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 7 | SRFBE: Sub oscillator Resistor Feedback Enable | • This bit controls the feedback resistor of the Sub oscillator.<br>• Writing "1" to this bit enables the feedback resistor and writing "0" disables the resistor.<br>• Any reset initializes this bit to "1" (resistor enabled). |

## 6.2.4 Clock Frequency Control Register (CKFCR)

The Clock Frequency Control Register (CKFCR) is used to control the Peripheral clock dividers (1 and 2), the Bus clock divider and the RC oscillator frequency.

### Configuration of the Clock Frequency Control Register (CKFCR)

Figure 6-6 shows the configuration of the Clock Frequency Control Register (CKFCR) and Table 6-4 describes the function of each bit.

The register can be accessed 16-bit wide (CKFCR) and 8-bit wide (low byte: CKFCRL, high byte: CKFCRH).

Figure 6-6. Configuration of the Clock Frequency Control Register (CKFCR)

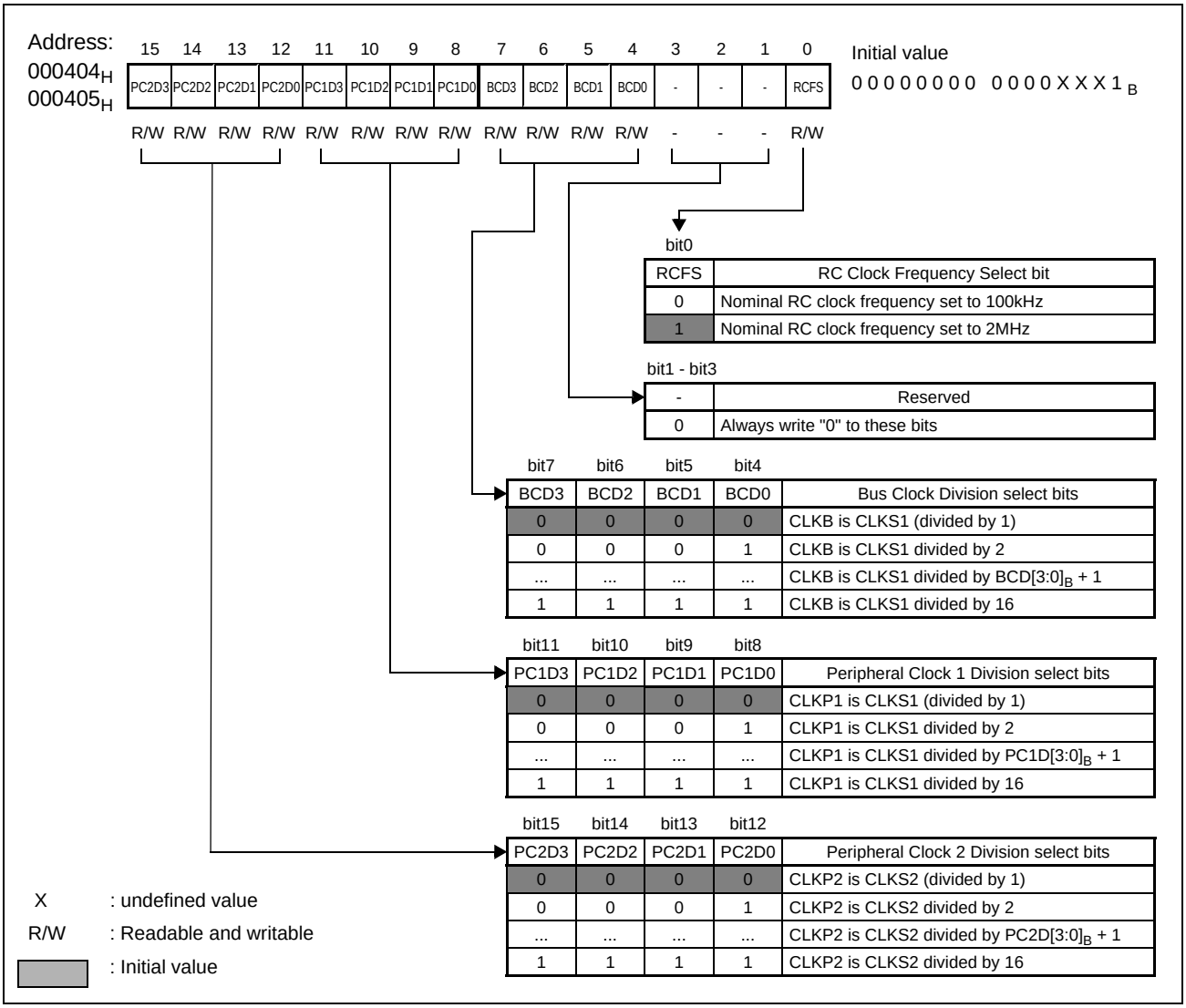


Table 6-4. Function Description of Each Bit of the Clock Frequency Control Register (CKFCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0</td><td>RCFS:<br>RC Clock Frequency Select bit</td><td>
• This bit is used to set the clock frequency of the internal RC oscillator.<br>
• Writing "1" to this bit sets the nominal RC clock frequency to 2MHz and writing "0" sets the nominal frequency to 100kHz.<br>
• For minimum and maximum frequencies in both settings, please refer to the Datasheet.<br>
• This bit is initialized to "1" (2MHz) by each reset.<br>
Note:<br>
The RC clock frequency can be changed at any time. However consider that the Clock stop detection circuit and the Watchdog reset can operate with the RC clock and that changing the RC clock frequency affects the behavior of these circuits. The Watchdog reset module has an operation mode where changing the RC clock frequency is not allowed and causes a Watchdog reset. Please read the corresponding description in the chapters about the Watchdog reset and the Clock stop detection reset for more details.
</td></tr>
<tr><td>bit 1 - bit 3</td><td>Reserved</td><td>
• Always write "0" to these bits.<br>
• The read value of these bits is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
<tr><td>bit 4 - bit 7</td><td>BCD0 to BCD3:<br>Bus Clock Division select bits</td><td>
• These bits control the clock divider for the Bus clock (CLKB) according to the following table:
<table>
<tr><th>bit7</th><th>bit6</th><th>bit5</th><th>bit4</th><th></th></tr>
<tr><th>BCD3</th><th>BCD2</th><th>BCD1</th><th>BCD0</th><th>Bus Clock Division select bits</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>CLKB is CLKS1 (divided by 1)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>CLKB is CLKS1 divided by 2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>CLKB is CLKS1 divided by 3</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>CLKB is CLKS1 divided by $BCD[3:0]_B + 1$</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>CLKB is CLKS1 divided by 15</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>CLKB is CLKS1 divided by 16</td></tr>
</table>
• These bits are initialized to "0000" (CLKB = CLKS1) by each reset.
</td></tr>
<tr><td>bit 8 - bit 11</td><td>PC1D0 to PC1D3:<br>Peripheral Clock 1 Division select bits</td><td>
• These bits control the clock divider for the Peripheral clock (CLKP1) according to the following table:
<table>
<tr><th>bit11</th><th>bit10</th><th>bit9</th><th>bit8</th><th></th></tr>
<tr><th>PC1D3</th><th>PC1D2</th><th>PC1D1</th><th>PC1D0</th><th>Peripheral Clock 1 Division select bits</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>CLKP1 is CLKS1 (divided by 1)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>CLKP1 is CLKS1 divided by 2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>CLKP1 is CLKS1 divided by 3</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>CLKP1 is CLKS1 divided by $PC1D[3:0]_B + 1$</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>CLKP1 is CLKS1 divided by 15</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>CLKP1 is CLKS1 divided by 16</td></tr>
</table>
• These bits are initialized to "0000" (CLKP1 = CLKS1) by each reset.
</td></tr>
<tr><td>bit 12 - bit 15</td><td>PC2D0 to PC2D3:<br>Peripheral Clock 2 Division select bits</td><td>
• These bits control the clock divider for the Peripheral clock (CLKP2) according to the following table:
<table>
<tr><th>bit11</th><th>bit10</th><th>bit9</th><th>bit8</th><th></th></tr>
<tr><th>PC2D3</th><th>PC2D2</th><th>PC2D1</th><th>PC2D0</th><th>Peripheral Clock 2 Division select bits</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>CLKP2 is CLKS2 (divided by 1)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>CLKP2 is CLKS2 divided by 2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>CLKP2 is CLKS2 divided by 3</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>CLKP2 is CLKS2 divided by $PC2D[3:0]_B + 1$</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>CLKP2 is CLKS2 divided by 15</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>CLKP2 is CLKS2 divided by 16</td></tr>
</table>
• These bits are initialized to "0000" (CLKP2 = CLKS2) by each reset.
</td></tr>
</table>

### 6.2.5 PLL and clock frequency Control Register (PLLCR)

The PLL and clock frequency Control Register (PLLCR) is used to control all functions of the PLL multiplier circuit and to control the Peripheral Clock Divider of CLKP3.

#### Configuration of the PLL and clock domain 3 frequency Control Register (PLLCR)

Figure 6-7 show the configuration of the PLL and clock domain 3 frequency Control Registers (PLLCR) and Table 6-5 describes the function of each bit.

The register can be accessed 16-bit wide (PLLCR) and 8-bit wide (low byte: PLLCRL, high byte PLLCRH).

Figure 6-7. Configuration of the PLL Control Register (PLLCR)

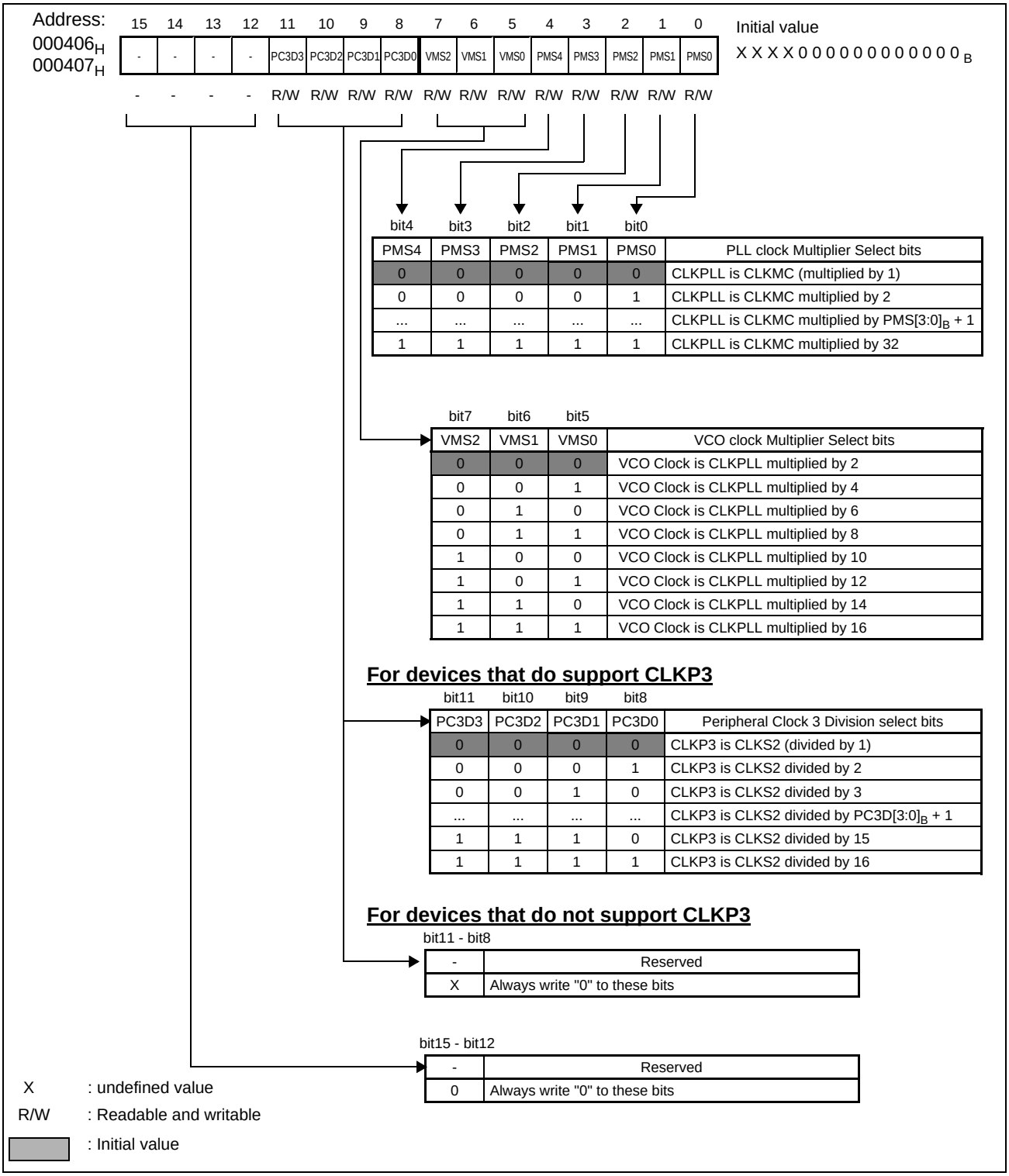


Table 6-5. Function Description of Each Bit of the PLL Control Register (PLLCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 4</td><td>PMS0 to PMS4:<br>PLL clock Multiplier Select bits</td><td>
• These bits control the PLL clock multiplier factor according to the following table:
<table>
<tr><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th><th></th></tr>
<tr><td>PMS4</td><td>PMS3</td><td>PMS2</td><td>PMS1</td><td>PMS0</td><td>PLL clock Multiplier Select bits</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>CLKPLL is CLKMC (multiplied by 1)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>CLKPLL is CLKMC multiplied by 2</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>CLKPLL is CLKMC multiplied by PMS[3:0]$_B$ + 1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>CLKPLL is CLKMC multiplied by 32</td></tr>
</table>
• These bits are initialized to "00000" by each reset.<br>
• The PLL multiplier setting defines the ratio between the Main clock frequency (CLKMC) and the PLL output frequency (CLKPLL).<br>
• These bits must be set before the PLL is enabled by writing "1" to the CKSR: PCE bit or by selecting the PLL clock as source for CLKS1 or CLKS2.<br>
• Do not change the setting of these bits after activation of the PLL.<br>
• The permitted range of settings depend on the used Main clock frequency and the permitted frequency range for CLKVCO and CLKS1/S2 (see Datasheet) .
</td></tr>
<tr><td>bit 5 - bit 7</td><td>VMS0 to VMS2:<br>VCO clock Multiplier Select bits</td><td>
• These bits control the VCO clock multiplier factor according to the following table:
<table>
<tr><th>bit6</th><th>bit5</th><th>bit4</th><th></th></tr>
<tr><td>VMS2</td><td>VMS1</td><td>VMS0</td><td>VCO clock Multiplier Select bits</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>VCO Clock is CLKPLL multiplied by 2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>VCO Clock is CLKPLL multiplied by 4</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>VCO Clock is CLKPLL multiplied by 6</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>VCO Clock is CLKPLL multiplied by 8</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>VCO Clock is CLKPLL multiplied by 10</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>VCO Clock is CLKPLL multiplied by 12</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>VCO Clock is CLKPLL multiplied by 14</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>VCO Clock is CLKPLL multiplied by 16</td></tr>
</table>
• These bits are initialized to "000" by each reset.<br>
• The VCO clock multiplier setting defines the relation between the PLL output clock frequency (CLKPLL) and the output frequency of the PLL internal VCO.<br>
• These bits must be set before the PLL is enabled by writing "1" to the CKSR: PCE bit or by selecting the PLL clock as source for CLKS1 or CLKS2.<br>
• Do not change the setting of these bits after activation of the PLL.<br>
The VCO clock multiplier factor should be set depending on the PLL input clock CLKMC and the PLL multiplier setting. See chapter 6.4 Configuration of the PLL for more details.
</td></tr>
<tr><td>bit 8- bit 11</td><td>PC3D0 to PC3D3:<br>Peripheral Clock 3 Division Select bits</td><td>
<u>For devices that do support CLKP3</u><br>
• These bits control the clock divider for the Peripheral clock (CLKP3) according to the following table:
<table>
<tr><th>bit11</th><th>bit10</th><th>bit9</th><th>bit8</th><th></th></tr>
<tr><td>PC3D3</td><td>PC3D2</td><td>PC3D1</td><td>PC3D0</td><td>Peripheral Clock 3 Division select bits</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>CLKP3 is CLKS2 (divided by 1)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>CLKP3 is CLKS2 divided by 2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>CLKP3 is CLKS2 divided by 3</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>CLKP3 is CLKS2 divided by PC3D[3:0]$_B$ + 1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>CLKP3 is CLKS2 divided by 15</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>CLKP3 is CLKS2 divided by 16</td></tr>
</table>
• These bits are initialized to "0000" (CLKP3 = CLKS2) by each reset<br><br>
<u>For devices that do not support CLKP3</u><br>
These bits are reserved.<br>
• Always write "0" to these bits.<br>
• The read value of these bits is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
<tr><td>bit 12 - bit 15</td><td>Reserved</td><td>
• Always write "0" to these bits.<br>
• The read value of these bits is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
</table>

## 6.2.6 Clock Input and LVD Control Register (CILCR)

The Clock Input and LVD Control Register (CILCR) is used to control additional functions of the oscillator circuit and the Low Voltage Detection Level.

### Configuration of the Clock Input and LVD Control Register (CILCR)

Figure 6-8 shows the configuration of the Clock Input and LVD Control Registers (CILCR) and Table 6-6 describes the function of each bit.

The register can be accessed 16-bit wide (VRCR) and 8-bit wide (low byte: VRCR, high byte CILCR).

Figure 6-8. Configuration of the Clock Input and LVD Control Register (CILCR)

Address: 00042D$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| - | - | - | FCI | LVL3 | LVL2 | LVL1 | LVL0 |
| - | - | - | R/W | R/W | R/W | R/W | R/W |

Initial value

X X X 0 0 0 0 0 $_B$

bit11 bit10 bit9 bit8

| LVL3 | LVL2 | LVL1 | LVL0 | Low voltage detector level |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Level 0 * |
| 0 | 0 | 0 | 1 | Level 1 * |
| 0 | 0 | 1 | 0 | Level 2 * |
| 0 | 0 | 1 | 1 | Level 3 * |
| 0 | 1 | 0 | 0 | Level 4 * |
| 0 | 1 | 0 | 1 | Level 5 * |
| 0 | 1 | 1 | 0 | Level 6 * |
| 0 | 1 | 1 | 1 | Level 7 * |
| 1 | 0 | 0 | 0 | Level 8 * |
| 1 | 0 | 0 | 1 | Level 9 * |
| 1 | 0 | 1 | 0 | Level 10 * |
| 1 | 0 | 1 | 1 | Level 11 * |
| 1 | 1 | 0 | 0 | Level 12 * |
| 1 | 1 | 0 | 1 | Level 13 * |
| 1 | 1 | 1 | 0 | Level 14 * |
| 1 | 1 | 1 | 1 | Level 15 * |

bit12

| FCI | Fast Clock Input Select |
|---|---|
| 0 | Oscillation mode |
| 1 | Fast external clock input mode |

bit15-13

| - | Reserved |
|---|---|
| 110 | Always write "110" to these bits. |

X : undefined value

R/W : Readable and writable

: Initial value

*** Note**: For the allowed settings and the analog detection level, please refer to the datasheet.

Table 6-6. Function Description of Each Bit of the Clock Input and LVD Control Register (CILCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 8 - bit 11</td><td>LVL0 to LVL3:<br>Low Voltage Detector Level Select bits</td><td>
• These bits control the analog threshold level for the Low Voltage Detector (LVD):
<table>
<tr><th>LVL3</th><th>LVL2</th><th>LVL1</th><th>LVL0</th><th>Low voltage detector level</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Level 0 *</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>Level 1 *</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>Level 2 *</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>Level 3 *</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>Level 4 *</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>Level 5 *</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>Level 6 *</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>Level 7 *</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>Level 8 *</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>Level 9 *</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>Level 10 *</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>Level 11 *</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>Level 12 *</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>Level 13 *</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>Level 14 *</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>Level 15 *</td></tr>
</table>
• These bits are initialized to "0000" by each reset.<br>
• The Low Voltage Detector circuit will issue a reset request when this function is enabled and the external supply voltage drops below the selected threshold value.<br><br>
* Note: For the allowed settings and the analog threshold level, please refer to the datasheet. For more details refer to Resets and Startup on page 159
</td></tr>
<tr><td>bit 12</td><td>FCI</td><td>
• This bit is initialized to "0" by each reset.<br>
• This bit selects the oscillation or Fast Clock Input mode of the Main Oscillator.
<table>
<tr><th>FCI</th><th>Fast Clock Input Control</th></tr>
<tr><td>0</td><td>Oscillation mode</td></tr>
<tr><td>1</td><td>Fast External Clock Input mode</td></tr>
</table>
• This bit must be set to "0" when connecting an oscillator (crystal/resonator) to the Main Oscillator Pin X0 and X1. It is also possible to connect an external clock with low frequency to the oscillator X0/X1 pins in this mode.<br>
• For inputting an external clock with higher frequency, this bit must be set to "1" before the device is switched to external (main) clock.<br>
• The frequency ranges for both settings are described in the datasheet.<br>
• This feature is not available on all devices. See datasheet for more details.
</td></tr>
<tr><td>bit 13 - bit 15</td><td>Reserved</td><td>
• Always write "110" to these bits.<br>
• The read value of these bits is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
</table>

## 6.3 Clock Modes

Four clock modes are provided: RC clock mode, Main clock mode, PLL clock mode and Sub clock mode.

### 6.3.1 Definition of clock modes

The clock mode of the MCU is defined by the source for the System clock 1 (CLKS1) which is selected by the SC1S[1:0] bits. CLKS1 is the master clock for the Bus clock (CLKB) that supplies the internal bus with CPU, memories, the DMA controller and the external bus interface, and for the Peripheral clock 1 (CLKP1) that supplies most peripheral modules.

The selection of the System clock 2 (CLKS2) can be made independent of the clock mode.

#### RC clock mode

In RC clock mode, the clock generated by the internal RC oscillator is used for the System clock 1 (CLKS1). Hence the Bus clock (CLKB) and the Peripheral clock 1 (CLKP1) are fed by the RC oscillator. The RC clock frequency can be selected with the RC Clock Frequency Select bit (CKFCR: RCFS) between nominal 100kHz and 2MHz.

The Main clock, PLL clock and Sub clock can be disabled with the MCE, PCE and SCE bits if they are not used for the System clock 2 or the watchdog. Disabling these clocks also disables the corresponding source clock timers.

#### Main clock mode

In Main clock mode, the output signal of the Main oscillator is used for the System clock 1 (CLKS1). Hence the Bus clock (CLKB) and the Peripheral clock 1 (CLKP1) are fed by CLKMC.

The RC clock, PLL clock and Sub clock can be disabled with the RCE, PCE and SCE bits if they are not used for the System clock 2, the watchdog or clock stop detect function. Disabling these clocks also disables the corresponding source clock timers.

#### PLL clock mode

In PLL clock mode, either the modulated or the unmodulated PLL clock is used for the System clock 1 (CLKS1). Hence the Bus clock (CLKB) and the Peripheral clock 1 (CLKP1) are fed by CLKPLL or CLKMOD. The setting of the clock modulator decides if CLKPLL or CLKMOD is used. See Clock Modulator on page 147 for more details.

The frequency of this clock signal depends on the setting of the PLL Control Register and the Main clock.

The RC clock and Sub clock can be disabled with the RCE and SCE bits if they are not used for the System clock 2, the watchdog or clock stop detect function. Disabling these clocks also disables the corresponding source clock timers. The Main clock cannot be disabled in PLL clock mode.

#### Sub clock mode

In Sub clock mode, the output signal of the Sub oscillator CLKSC is used for the System clock 1 (CLKS1). Hence the Bus clock (CLKB) and the Peripheral clock 1 (CLKP1) are fed by the CLKSC.

The RC clock, Main clock and PLL clock can be disabled with the RCE, MCE and PCE bits if they are not used for the System clock 2, the watchdog or clock stop detect function. Disabling these clocks also disables the corresponding source clock timers.

### 6.3.2 Clock source switching

Clock source switching means changing the clock source for CLKS1 or CLKS2.

#### Changing the clock source

- The clock source selection can be done independently for CLKS1 and CLKS2
- A clock source switching is done by writing a new value to the System clock select bits (SC1S[1:0] for System clock 1 and SC2S[1:0] for System clock 2)
- A transition to a different clock source is possible only when the selected clock is ready (means the corresponding ready flag RCM, MCM, PCM or SCM in the Clock Monitor Register CKMR is set).

- If the selected clock is ready, then the clock source transition will be executed by starting the synchronization mechanism. The System Clock Monitor bits (SC1M for System clock 1 and SC2M for System clock 2) of the Clock Monitor register change to the new value after completion of this clock switching.
- Writing a second value to the System clock select bits before switching to the mode selected before has completed, cancels the first transition request.
- If the selected clock is not available or not stabilized, then switching to the new clock mode will be delayed until the selected clock is available and stable (clock ready flag CKMR: RCM, MCM, PCM or SCM set).
- After Stop mode release by interrupt, the System Clock Selectors directly select the clocks which are defined by the SC1S/SC2S bits, irrespective of the clock ready monitor bits. Hence do not select an unavailable clock before switching to Stop mode because this clock is used for the restart by an interrupt.

#### Reset and clock source

The clock source for both System clocks (CLKS1 and CLKS2) is set to RC clock (CLKRC) by any reset. The RC clock stabilization time is applied after each reset.

#### Access to peripheral resources clocked with CLKP2 or CLKP3

Do not access (read or write) any resource clocked with the peripheral clock 2 (CLKP2) directly after changing the setting of the SC1S or SC2S bits. Do always make sure that the requested clock transition has completed by reading the SC1M/SC2M monitor bits. Access to resources clocked with CLKP2 is allowed only if SC1M[1:0] equals to SC1S[1:0] and SC2M[1:0] equals to SC2S[1:0].

### 6.3.3 Activating and Disabling source clocks

- After each reset, the Main oscillator, Sub oscillator and RC oscillator are enabled while the PLL multiplier circuit is disabled.
- Source clocks selected for the System clocks CLKS1 or CLKS2 are always activated.
- Source clocks not used for the System clocks can be enabled and disabled with the corresponding enable bits in the CKSR register.
- Disable a source clock for current saving by setting the corresponding enable bit in the CKSR register to "0".
- Enable a source clock if it is needed for a certain function or for a fast System clock changing by setting the corresponding enable bit in the CKSR register to "1". After stabilization of the activated clock, the corresponding Clock monitor bit of the CKMR register is set and indicates the clock as "ready".
- The Main clock must be enabled if the PLL clock should be enabled (The setting of PCE has no effect when MCE is set to "0").
- Setting MCE to "0" does not disable the Main oscillator when System clock 1 or 2 is set to Main or PLL clock because the Main clock is the input signal for the PLL multiplier circuit.

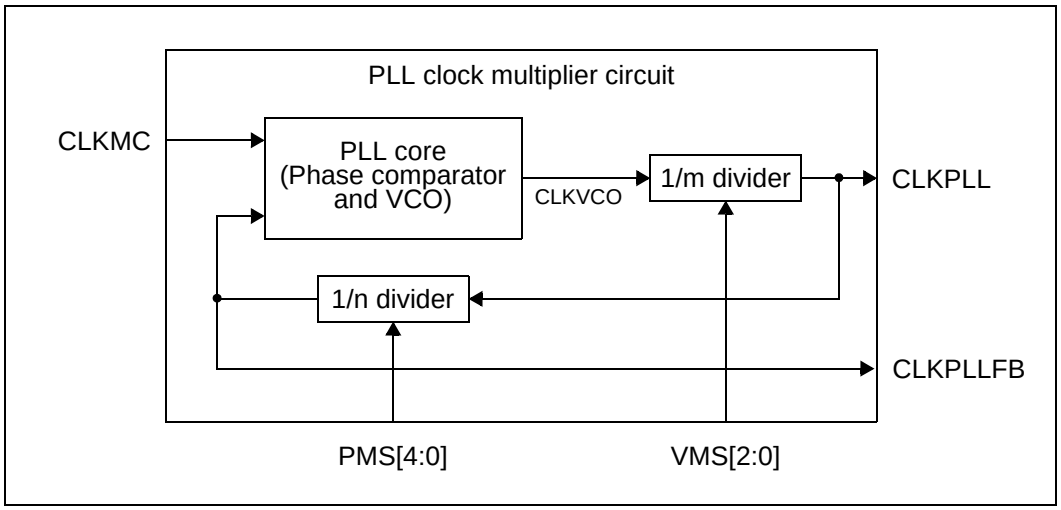
## 6.4 Configuration of the PLL

The PLL multiplier circuit is used to generate the PLL clock out of the Main clock. 32 different multiplier values (mul-1 to mul-32) are available.

### 6.4.1 Components of the PLL clock multiplier circuit

The following block diagram describes the modules of the PLL.

Figure 6-9. Block diagram of the PLL circuit




#### PLL core (Phase comparator and VCO)

The PLL core consists of a phase comparator which controls a voltage controlled oscillator (VCO). The phase comparator compares the input clock CLKMC (Main clock) with the output clock of the "1/n divider". The VCO generates the output clock CLKVCO which frequency depends on the setting of the "1/m divider" and the "1/n divider": CLKVCO = CLKMC * m * n. The allowed frequency range of CLKVCO is specified in the Datasheet.

#### "1/m divider" (VCO clock divider)

The "1/m divider" is controlled by the VMS[2:0] VCO clock Multiplier Select bits. It defines the frequency relation between the PLL output clock CLKPLL and the VCO output clock CLKVCO. The minimum division value is "2" to guarantee a PLL output clock with a duty cycle of 50%. It is also used to keep the VCO in its operating range.

These bits must be set depending on the Main oscillation frequency CLKMC and CLKPLL in a way that the resulting CLKVCO frequency is always within the permitted range as described in the datasheet. Otherwise the PLL will not lock and the resulting CLKPLL frequency is undefined.

Do not change the setting of these bits after activation of the PLL.

#### "1/n divider" (PLL clock divider)

The "1/n divider" is controlled by the PMS[4:0] PLL clock Multiplier Select bits. It defines the frequency multiplication value of the PLL clock multiplier circuit (relation between CLKMC and CLKPLL). Multiplication values from 1 to 32 are available. However the minimum and maximum permitted frequencies for CLKMC, CLKVCO and CLKPLL (->CLKS1/S2) must be adhered. Do not change the setting of these bits after activation of the PLL.

### 6.4.2 Setting the VCO clock Multiplier Select bits VMS[2:0]

The "1/m divider" is programmable to keep the VCO in its operating range as specified in the datasheet for a wide range of PLL clock frequencies. Write a value to the VMS[2:0] bits which results in a valid CLKVCO clock frequency. The table below shows recommended settings for a 4MHz Main clock. For other Main clock frequencies, it is necessary to adjust the VMS[2:0] setting based on the formula:

"CLKVCO = CLKMC * m * n" .

Table 6-7. Recommended settings for the PMS[3:0], VMS[2:0] bits for CLKMC=4MHz

| Main Oscillation frequency CLKMC | Requested PLL output frequency CLKPLL | Setting for PMS[4:0] bits (n division value) | Setting for VMS[2:0] bits (m division value) | VCO output frequency CLKVCO |
|---|---|---|---|---|
| 4 MHz | 4 MHz | "00000" (div 1) | "111" (div 16) | 64 MHz |
| 4 MHz | 8 MHz | "00001" (div 2) | "101" (div 12) | 96 MHz |
| 4 MHz | 12 MHz | "00010" (div 3) | "011" (div 8) | 96 MHz |
| 4 MHz | 16 MHz | "00011" (div 4) | "010" (div 6) | 96 MHz |
| 4 MHz | 20 MHz | "00100" (div 5) | "010" (div 6) | 120 MHz |
| 4 MHz | 24 MHz | "00101" (div 6) | "001" (div 4) | 96 MHz |
| 4 MHz | 28 MHz | "00110" (div 7) | "001" (div 4) | 112 MHz |
| 4 MHz | 32 MHz | "00111" (div 8) | "001" (div 4) | 128 MHz |
| 4 MHz | 36 MHz | "01000" (div 9) | "000" (div 2) | 72 MHz |
| 4 MHz | 40 MHz | "01001" (div 10) | "000" (div 2) | 80 MHz |
| 4 MHz | 44 MHz | "01010" (div 11) | "000" (div 2) | 88 MHz |
| 4 MHz | 48 MHz | "01011" (div 12) | "000" (div 2) | 96 MHz |
| 4 MHz | 52 MHz | "01100" (div 13) | "000" (div 2) | 104 MHz |
| 4 MHz | 56 MHz | "01101" (div 14) | "000" (div 2) | 112 MHz |
| 4 MHz | 60 MHz | "01110" (div 15) | "000" (div 2) | 120 MHz |
| 4 MHz | 64 MHz | "01111" (div 16) | "000" (div 2) | 128 MHz |
| 4 MHz | 68 MHz | "10000" (div 17) | "000" (div 2) | 136 MHz |
| 4 MHz | 72 MHz | "10001" (div 18) | "000" (div 2) | 144 MHz |
| 4 MHz | 76 MHz | "10010" (div 19) | "000" (div 2) | 152 MHz |
| 4 MHz | 80 MHz | "10011" (div 20) | "000" (div 2) | 160 MHz |
| 4 MHz | 84 MHz | "10100" (div 21) | "000" (div 2) | 168 MHz |
| 4 MHz | 88 MHz | "10101" (div 22) | "000" (div 2) | 176 MHz |
| 4 MHz | 92 MHz | "10110" (div 23) | "000" (div 2) | 184 MHz |
| 4 MHz | 96 MHz | "10111" (div 24) | "000" (div 2) | 192 MHz |
| 4 MHz | 100 MHz | "11000" (div 25) | "000" (div 2) | 200 MHz |

## 6.5 Oscillation Stabilization Wait Time

When the power is turned on, when stop mode is released or when a disabled clock is enabled, an oscillation stabilization wait time is required before the clock can be used.

### 6.5.1 Oscillation Stabilization Wait Interval

Ceramic and crystal oscillators which can be connected to the X0/X1 and X0A/X1A pins generally require several ms to stabilize at their natural frequency (oscillation frequency) when oscillation starts. The internal PLL multiplier circuit and the RC oscillator also need a certain stabilization time after activation. For this reason, CPU operation with the activated clock is not allowed immediately after oscillation starts but is allowed only after full oscillation stabilization.

After the oscillation stabilization wait interval has elapsed, the corresponding clock ready monitor bit will be set and the clock can be selected as System clock.

Because the oscillation stabilization time of the Main and Sub oscillator depends on the type of oscillator (crystal, ceramic, etc.), the proper oscillation stabilization wait interval for the oscillator used must be selected. An oscillation stabilization wait interval is selected by setting the Clock Stabilization Select Register (CKSSR). The stabilization time of the RC oscillator is fixed and the stabilization time of the PLL can be selected depending on the PLL and main clock frequency.

When the clock source is switched, the MCU runs with the previously selected clock until the newly selected clock is stabilized. When the corresponding clock ready flag is set, the MCU changes to the specified clock.

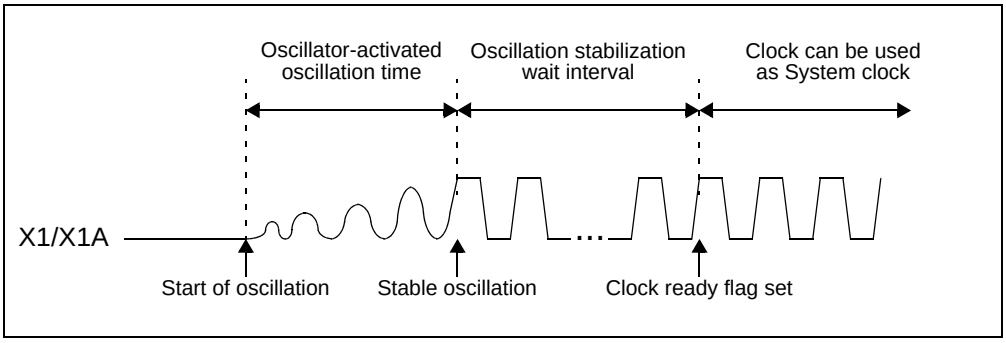
A Power reset or External reset clears all source clock timers and clock ready monitor bits and the corresponding oscillation stabilization wait interval is applied.

Software and Watchdog resets do not affect the source clock timers and Main and Sub clock monitor bits (no Main/Sub oscillation stabilization wait interval applied if clock was enabled before reset), but clear the RC and PLL clock monitor bits (PLL is disabled and RC oscillation stabilization time is applied).

Clock stop resets clear the source clock timer and clock monitor bit of the clock that caused the reset. This clock should not be used any more. The PLL is also disabled and the RC oscillation stabilization time applied.

Figure 6-10 shows the operation of the oscillators directly after activation.

Figure 6-10. Operation Immediately after Oscillation Starts




#### RC clock stabilization interval

The RC clock stabilization time is a fixed number of RC clock cycles (see datasheet). However after a Power reset or an External reset (RSTX falling edge), an additional wait time of 700 RC clock cycle (Power reset) or 700 RC clock cycles (External reset) is applied by the reset extension circuit (see section Startup after Power and External reset on page 162 for more details).

#### Main clock stabilization interval

The Main clock stabilization time can be selected with the MCST[2:0] bits of the CKSSR register. 8 settings are possible as described in Table 6-3.

### Sub clock stabilization interval

The Sub clock stabilization time can be selected with the SCST[1:0] bits of the CKSSR register. 4 settings are possible as described in Table 6-3.

### PLL clock stabilization interval

The PLL clock stabilization time can be selected with the PCST bit of the CKSSR register. 2 settings are possible as described in Table 6-3.

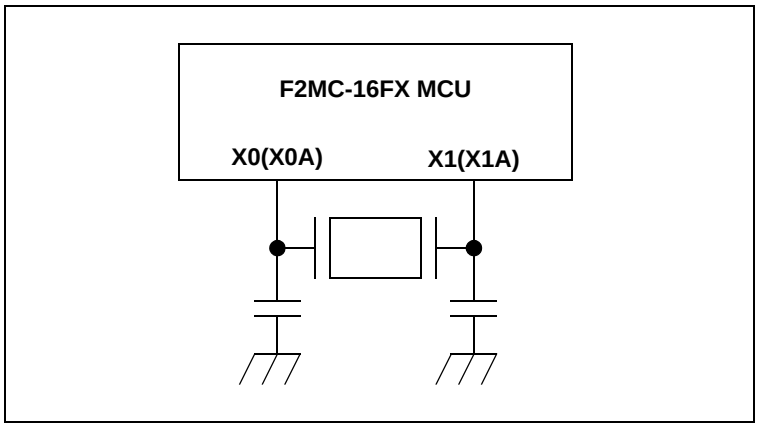
## 6.6 Connection of an Oscillator or an External Clock to the Microcontroller

The F2MC-16FX microcontroller contains up to two clock generation circuits for external oscillators, the Main oscillation circuit and the Sub oscillation circuit. Connecting an external oscillator to these circuits generates the Main clock and the Sub clock. Alternatively, externally generated clocks can be input to the micro controller.

### Example of connecting a crystal or ceramic oscillator to the microcontroller

Connect a crystal or ceramic oscillator as shown in the following diagram.

Figure 6-11. Example of Connecting a Crystal or Ceramic Oscillator to the Microcontroller




If this connection is used on devices, where the Fast Clock Input feature is available, the CILCR:FCI bit must be set to "0". For information about the availability of this feature, please refer to the datasheet.

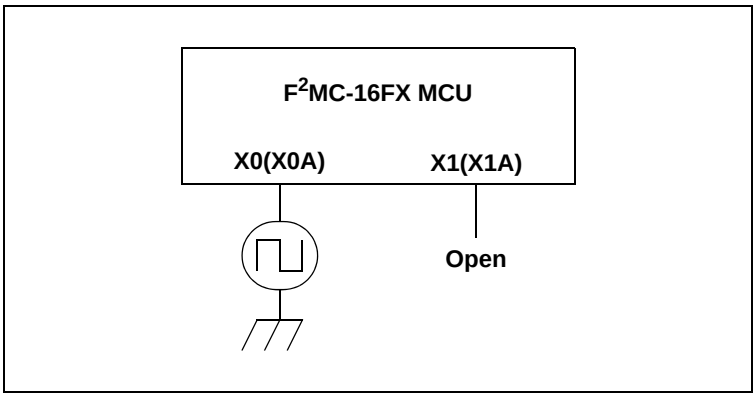
### Example of connecting an external clock to the microcontroller

As shown in the example in Figure 6-12, connect an external clock to pin X0 (X0A). Pin X1 (X1A) must be open.

For devices with the optional Fast Clock Input feature, it is possible to use a high speed clock frequency for the X0 pin (please consult the datasheet for the actual limits of the clock input frequency and the availability of the Fast Clock Input feature). This feature is activated by setting the CILCR:FCI bit to ’1’ before switching the device to the external Main clock.

Cypress recommends to always use the Fast Clock Input feature when connecting an external clock to the Main oscillator.

Figure 6-12. Example of Connecting an External Clock to the Microcontroller



# 7. Clock Modulator

This chapter provides an overview of the Clock Modulator and its features. It describes the register structure and operation of the Clock Modulator.

7.1 Overview

7.2 Register Description

7.3 Application Note

THIS FUNCTION IS UNDER EVALUATION.

PLEASE CONTACT CYPRESS BEFORE USING THIS FUNCTION.

## 7.1 Overview

This section describes an overview of the Clock Modulator.

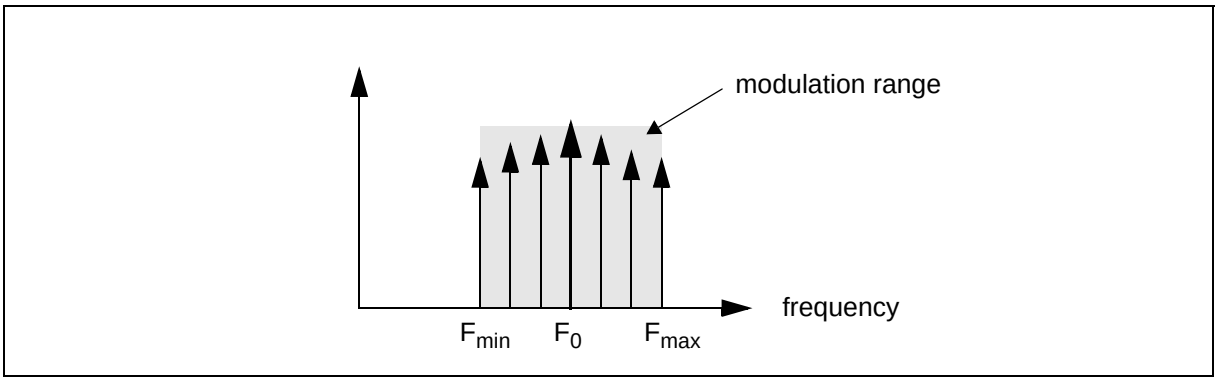
### Overview

The clock modulator is intended for the reduction of electromagnetic interference - EMI, by spreading the spectrum of the clock signal over a wide range of frequencies.

The module is fed with an unmodulated reference clock with frequency F0, provided by the PLL circuit (CLKPLL). This reference clock is frequency modulated, controlled by a random signal.

The mean frequency of the modulated clock is equal to the reference clock frequency F0.

Figure 7-1. Frequency spectrum of the modulated clock (fundamentals only)




### Modulation degree and frequency resolution

Maximum and minimum frequencies ($F_{max}$ and $F_{min}$) of the modulated clock are defined by the modulation degree parameter. Furthermore the resolution of the modulation range is selectable in 7 steps from low (1) to high (7). Higher resolution implies a finer granularity of discrete frequencies in the spectrum of the modulated clock but less possible modulation degrees.

In general the highest possible frequency resolution combined with the highest possible modulation degree results in the highest EMI reduction. But for some cases lower modulation degrees may result in a better EMI behavior. Please refer to the table of possible settings in Table 7-4.

## 7.2 Register Description

This section lists the clock modulator registers and describes the function of each register in detail.

### Clock modulator registers

Figure 7-2. Clock modulator registers

CMCR: Clock Modulator Control Register

| Address: 000418$_H$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Res. | Res. | Res. | Res. | Res. | MOD-RUN | MODEN | PDX |
| Initial value | - | R/W | R/W | R/W | - | R | R/W | R/W |
| | X | 0 | 0 | 1 | X | 0 | 0 | 0 |

CMPRL: Clock Modulator Parameter Register (lower)

| Address: 00041A$_H$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MP7 | MP6 | MP5 | MP4 | MP3 | MP2 | MP1 | MP0 |
| Initial value | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

CMPRH: Clock Modulator Parameter Register (upper)

| Address: 00041B$_H$ | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | Res. | Res. | MP13 | MP12 | MP11 | MP10 | MP9 | MP8 |
| Initial value | - | - | R/W | R/W | R/W | R/W | R/W | R/W |
| | X | X | 0 | 0 | 0 | 0 | 1 | 0 |

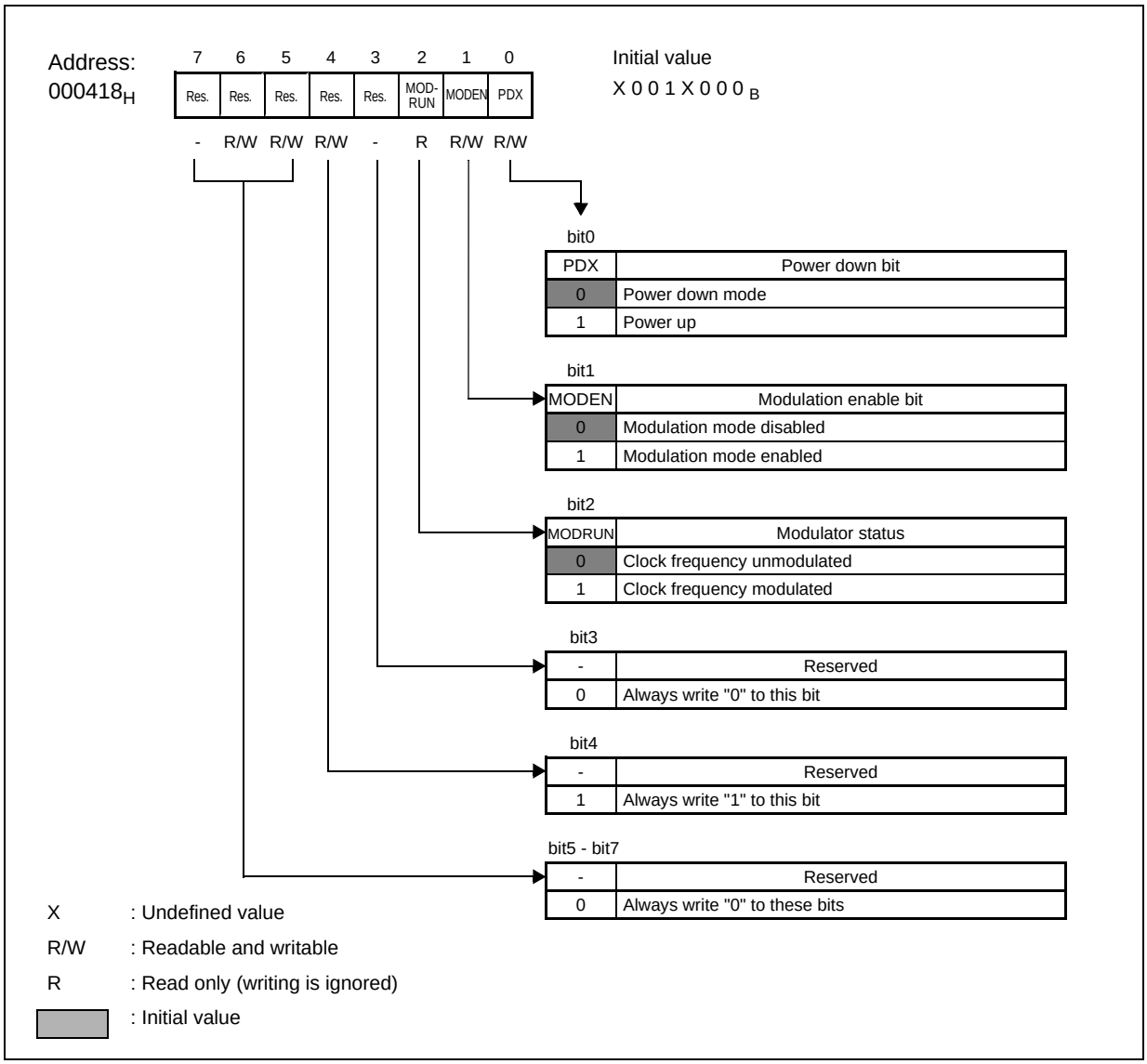
### 7.2.1 Clock Modulator Control Register (CMCR)

The Control Register (CMCR) has the following functions:

- Set the modulator to power down mode
- Modulator enable/disable
- Indicates the status of the modulator

### Clock modulator control register (CMCR)

Figure 7-3. Configuration of the clock modulator control register (CMCR)



## Clock modulator control register contents

Table 7-1. Function of each bit of the clock modulator control register

| Bit name | | Function |
|---|---|---|
| bit 0 | PDX:<br>Power down bit | "0": Power down mode<br>"1": Power up<br>• PDX is the power down signal for the modulator. Before the modulation can be enabled, this bit must be set to 1 and the startup time of 6 μs must be awaited. Please refer to the application note for a description of the recommended startup sequence.<br>• Before switching to power down mode (PDX=0), the modulator must be disabled -> MODEN=0 and MODRUN=0. |
| bit 1 | MODEN:<br>Modulation enable bit | "0": Modulation disabled.<br>"1": Modulation enabled.<br>• To enable the modulation, MODEN must be set to 1.<br>• Before the modulation can be enabled, the PLL must deliver a stable reference clock (PLL lock time must be elapsed).<br>• The specified PLL frequency range for modulation is 16 MHz to 84 MHz.<br>• Each PLL output frequency offers a set of possible modulation parameters. The selected setting (CMPR register) and the PLL frequency must match.<br>Please refer to the CMPR register description.<br>• Whenever the PLL output frequency is changed or the PLL is switched off e.g. in power down modes, the modulator must be disabled before -> MODEN=0 and MODRUN=0.<br>• Before the modulation can be enabled, the modulator must be switched from power down to active mode by setting PDX to 1 and the startup time of 6 μs must be awaited.<br>Please refer to the application note for a description of the recommended startup sequence.<br>• Before the modulation can be enabled, a proper setting must be selected via the parameter register CMPR.<br>• After enabling the modulation by setting MODEN to 1, the modulator is calibrated. During this time, the clock is unmodulated. Therefore the output clock does not switch immediately to modulated clock. The status of the clock (frequency modulated / unmodulated) is indicated by the MODRUN status bit. Please refer to the MODRUN bit description.<br>• Due to the synchronization of the MODEN signal and the synchronized switching to unmodulated clock, it takes less than 9 x T0 (input clock period) before the clock switches to unmodulated clock after the modulation is disabled. The modulation can be disabled at any time.<br>• Before changing the parameter register CMPR, the modulation must be disabled -> MODEN=0 and MODRUN=0. |
| bit 2 | MODRUN:<br>Modulator status bit | "0": MCU is running with unmodulated clock<br>"1": MCU is running with modulated clock<br>• MODRUN indicates the status of the modulator output clock. If the output clock is modulated, MODRUN is set to 1, otherwise MODRUN is set to 0.<br>• After enabling the modulation mode by setting MODEN to 1, the modulator is calibrated. During this time, the clock is unmodulated. Therefore it takes several μs before the output clock switches to modulated clock and the MODRUN bit is set to '1'. The calibration time is measured with the Main clock timer and takes 256-512 Main clock cycles (64-128μs at 4MHz Main clock frequency). Do not clear the Main clock timer during calibration.<br>• Due to the synchronization of the MODEN signal and the synchronized switching to unmodulated clock, it takes less than 9 x T0 (input clock period) before MODRUN changes to 0 and the clock switches to unmodulated clock after the modulation is disabled.<br>• The MODRUN bit is read only. Writing to MODRUN has no effect.<br>• Before changing the parameter register CMPR, the modulator must be disabled -> MODEN=0 and MODRUN=0. |
| bit 3 | Reserved | • Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected. |
| bit 4 | Reserved | • Always write 1 to this bit.<br>• Reading this bit returns the write value. |
| bit 5 - bit 6 | Undefined | • Always write 0 to these bits.<br>• Reading these bits returns the write value. |
| bit 7 | Reserved | • Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected. |

In the Table below the modulator states are summarized:

Table 7-2. States of the modulator

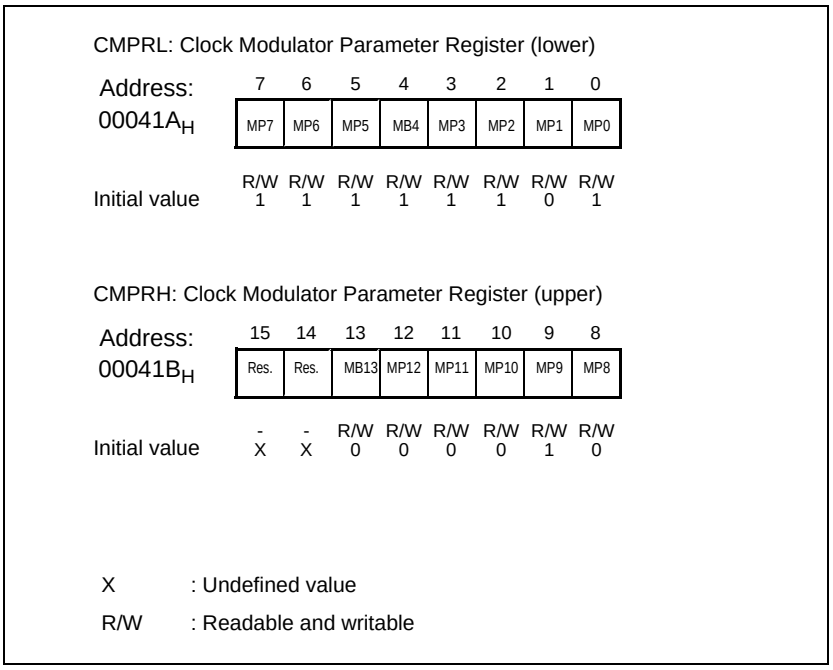
| | MODEN | PDX | MODRUN (read only) |
|---|---|---|---|
| modulator disabled | 0 | 0 | 0 |
| modulator power on, waiting modulator startup time (> 6 µs) | 0 | 1 | 0 |
| modulator enabled , modulator is calibrating, modulation not active | 1 | 1 | 0 |
| modulation is active | 1 | 1 | 1 |
| | others not allowed | | |

## 7.2.2 Clock Modulation Parameter Register (CMPR)

The Clock Modulation Parameter Register (CMPR) determines the modulation degree.

### Clock Modulation parameter register

Figure 7-4. Clock Modulation parameter register




- The modulation parameter determines the degree of modulation and the maximal and minimal occurring frequencies in the modulated clock. Please refer to the application note for a description of an approach to select the optimal setting.
- Each set of possible modulation parameters refers to a particular PLL frequency. The PLL frequency and the selected parameter must match. Please refer to the following table of possible settings.

### Note:

The modulation parameter must be changed only when the modulator is disabled and the RUN flag is 0 (MODEN=0, MOD-RUN=0).

### Clock Modulation parameter register contents

Table 7-3. Function of each bit of the clock modulation parameter register (CMPR)

| Bit name | | Function |
|---|---|---|
| bit 15, 14 | Undefined | |
| bit 13 to 0 | MP13 to 0:<br>Modulation Parameter bits | Depending on the PLL frequency the following modulation parameter settings are possible. The corresponding CMPR register value is stated in the most right column. |

The table below shows the recommended setting for several MCU clocks and modulation parameters.

Make sure the frequency range $f_{max}$ and $f_{min}$ of the clock modulator output clock CLKMOD of the selected modulation setting does not exceed the permitted frequency range of the MCU (see Figure 6-1 on page 120):

- The $f_{max}$ CLKMOD must be lower than the maximum permitted system clock frequency $f_{CLKS1}$ of the MCU devices (see datasheet: 'Internal Clock timing').
- If internal clock CLKB or CLKP1 is set to "divide by 1", then the $f_{max}$ CLOMOD must be lower than the permitted maximum frequency of $f_{CLKB}$ and $f_{CLKP1}$.
- Flash devices: The $f_{min}$ and $f_{max}$ of CLOMOD must be valid between the permitted frequency range of $f_{CLKS1}$ for the used Flash timing setting (see hardware manual: Table 33-4 on page 670).

Table 7-4. Modulation Parameter recommended settings (Sheet 1 of 5)

| input CLKPLL (MHz) | $f_{min}$ CLKMOD (MHz) | $f_{max}$ CLKMOD (MHz) | Modulation bandwidth (MHz) | modulation degree | frequency resolution | CMPR register value (hex) |
|---|---|---|---|---|---|---|
| 16 | 14.7 | 17.5 | 2 | 1 | 3 | 026F |
| | 13.9 | 18.8 | 4.1 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 13.2 | 20.3 | 6.2 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 12.5 | 22.1 | 8.5 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| | 11.9 | 24.1 | 11.1 | 1<br>5 | 11<br>3 | 036B<br>0A6B |
| | 11.4 | 26.6 | 14 | 1<br>2<br>3<br>6 | 13<br>7<br>5<br>3 | 03AA<br>04EA<br>06AA<br>0C6A |
| | 10.9 | 29.7 | 17.3 | 1 | 15 | 03E9 |
| | 10.5 | 33.6 | 21.3 | 2<br>4 | 9<br>5 | 0528<br>08A8 |

Table 7-4. Modulation Parameter recommended settings (Sheet 2 of 5)

| input CLKPLL (MHz) | $f_{min}$ CLKMOD (MHz) | $f_{max}$ CLKMOD (MHz) | Modulation bandwidth (MHz) | modulation degree | frequency resolution | CMPR register value (hex) |
|---|---|---|---|---|---|---|
| 20 | 18.3 | 22.1 | 2.5 | 1 | 3 | 026F |
| | 17.3 | 23.7 | 5.1 | 1 | 5 | 02AE |
| | | | | 2 | 3 | 046E |
| | 16.4 | 25.6 | 7.8 | 1 | 7 | 02ED |
| | | | | 3 | 3 | 066D |
| | 15.6 | 27.8 | 10.7 | 1 | 9 | 032C |
| | | | | 2 | 5 | 04AC |
| | | | | 4 | 3 | 086C |
| | 14.9 | 30.4 | 13.9 | 1 | 11 | 036B |
| | | | | 5 | 3 | 0A6B |
| | 14.2 | 33.6 | 17.5 | 1 | 13 | 03AA |
| | | | | 2 | 7 | 04EA |
| | | | | 3 | 5 | 06AA |
| | | | | 6 | 3 | 0C6A |
| | 13.6 | 37.6 | 21.7 | 1 | 15 | 03E9 |
| | 13 | 42.6 | 26.7 | 2 | 9 | 0528 |
| | | | | 4 | 5 | 08A8 |
| 24 | 21.8 | 26.6 | 3 | 1 | 3 | 026F |
| | 20.6 | 28.6 | 6.1 | 1 | 5 | 02AE |
| | | | | 2 | 3 | 046E |
| | 19.6 | 30.9 | 9.4 | 1 | 7 | 02ED |
| | | | | 3 | 3 | 066D |
| | 18.6 | 33.6 | 12.8 | 1 | 9 | 032C |
| | | | | 2 | 5 | 04AC |
| | | | | 4 | 3 | 086C |
| | 17.8 | 36.9 | 16.6 | 1 | 11 | 036B |
| | | | | 5 | 3 | 0A6B |
| | 17 | 40.8 | 21 | 1 | 13 | 03AA |
| | | | | 2 | 7 | 04EA |
| | | | | 3 | 5 | 06AA |
| | | | | 6 | 3 | 0C6A |
| | 16.3 | 45.6 | 26 | 1 | 15 | 03E9 |
| | 15.6 | 51.7 | 32 | 2 | 9 | 0528 |
| | | | | 4 | 5 | 08A8 |

Table 7-4. Modulation Parameter recommended settings (Sheet 3 of 5)

| input CLKPLL (MHz) | $f_{min}$ CLKMOD (MHz) | $f_{max}$ CLKMOD (MHz) | Modulation bandwidth (MHz) | modulation degree | frequency resolution | CMPR register value (hex) |
|---|---|---|---|---|---|---|
| 28 | 25.3 | 31.3 | 3.5 | 1 | 3 | 026F |
| | 24 | 33.6 | 7.1 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 22.7 | 36.4 | 10.9 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 21.6 | 39.6 | 15 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| | 20.6 | 43.4 | 19.4 | 1<br>5 | 11<br>3 | 036B<br>0A6B |
| | 19.7 | 48 | 24.5 | 1<br>2<br>3<br>6 | 13<br>7<br>5<br>3 | 03AA<br>04EA<br>06AA<br>0C6A |
| | 18.9 | 53.8 | 30.3 | 1 | 15 | 03E9 |
| | 18.1 | 61.2 | 37.3 | 2<br>4 | 9<br>5 | 0528<br>08A8 |
| 32 | 28.8 | 36 | 4 | 1 | 3 | 026F |
| | 27.2 | 38.7 | 8.1 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 25.9 | 41.9 | 12.5 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 24.6 | 45.6 | 17.1 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| | 23.5 | 50.1 | 22.2 | 1<br>5 | 11<br>3 | 036B<br>0A6B |
| | 22.5 | 55.5 | 28 | 1<br>2<br>3<br>6 | 13<br>7<br>5<br>3 | 03AA<br>04EA<br>06AA<br>0C6A |
| | 21.5 | 62.2 | 34.7 | 1 | 15 | 03E9 |
| | 20.6 | 70.8 | 42.7 | 2<br>4 | 9<br>5 | 0528<br>08A8 |

Table 7-4. Modulation Parameter recommended settings (Sheet 4 of 5)

| input CLKPLL (MHz) | $f_{min}$ CLKMOD (MHz) | $f_{max}$ CLKMOD (MHz) | Modulation bandwidth (MHz) | modulation degree | frequency resolution | CMPR register value (hex) |
|---|---|---|---|---|---|---|
| 36 | 32.2 | 40.8 | 4.5 | 1 | 3 | 026F |
| | 30.5 | 43.9 | 9.2 | 1 | 5 | 02AE |
| | | | | 2 | 3 | 046E |
| | 29 | 47.5 | 14 | 1 | 7 | 02ED |
| | | | | 3 | 3 | 066D |
| | 27.6 | 51.7 | 19.2 | 1 | 9 | 032C |
| | | | | 2 | 5 | 04AC |
| | | | | 4 | 3 | 086C |
| | 26.3 | 56.9 | 25 | 1 | 11 | 036B |
| | | | | 5 | 3 | 0A6B |
| | 25.2 | 63.1 | 31.4 | 1 | 13 | 03AA |
| | | | | 2 | 7 | 04EA |
| | | | | 3 | 5 | 06AA |
| | | | | 6 | 3 | 0C6A |
| | 24.1 | 70.8 | 39 | 1 | 15 | 03E9 |
| | 23.1 | 80.7 | 48 | 2 | 9 | 0528 |
| | | | | 4 | 5 | 08A8 |
| 40 | 35.6 | 45.6 | 5 | 1 | 3 | 026F |
| | 33.7 | 49.1 | 10.2 | 1 | 5 | 02AE |
| | | | | 2 | 3 | 046E |
| | 32 | 53.2 | 15.6 | 1 | 7 | 02ED |
| | | | | 3 | 3 | 066D |
| | 30.5 | 58 | 21.3 | 1 | 9 | 032C |
| | | | | 2 | 5 | 04AC |
| | | | | 4 | 3 | 086C |
| | 29.1 | 63.8 | 27.7 | 1 | 11 | 036B |
| | | | | 5 | 3 | 0A6B |
| | 27.8 | 70.8 | 34.9 | 1 | 13 | 03AA |
| | | | | 2 | 7 | 04EA |
| | | | | 3 | 5 | 06AA |
| | | | | 6 | 3 | 0C6A |
| | 26.7 | 79.6 | 43.3 | 1 | 15 | 03E9 |
| | 25.6 | 90.9 | 53.3 | 2 | 9 | 0528 |
| | | | | 4 | 5 | 08A8 |
| 44 | 39 | 50.5 | 5.5 | 1 | 3 | 026F |
| | 36.9 | 54.4 | 11.2 | 1 | 5 | 02AE |
| | | | | 2 | 3 | 046E |
| | 35.1 | 59 | 17.1 | 1 | 7 | 02ED |
| | | | | 3 | 3 | 066D |
| | 33.4 | 64.4 | 23.5 | 1 | 9 | 032C |
| | | | | 2 | 5 | 04AC |
| | | | | 4 | 3 | 086C |
| | 31.9 | 70.8 | 30.5 | 1 | 11 | 036B |
| | | | | 5 | 3 | 0A6B |
| | 30.5 | 78.7 | 38.4 | 1 | 13 | 03AA |
| | | | | 2 | 7 | 04EA |
| | | | | 3 | 5 | 06AA |
| | | | | 6 | 3 | 0C6A |
| | 29.2 | 88.6 | 47.6 | 1 | 15 | 03E9 |

Table 7-4. Modulation Parameter recommended settings (Sheet 5 of 5)

| input CLKPLL (MHz) | $f_{min}$ CLKMOD (MHz) | $f_{max}$ CLKMOD (MHz) | Modulation bandwidth (MHz) | modulation degree | frequency resolution | CMPR register value (hex) |
|---|---|---|---|---|---|---|
| 48 | 42.3 | 55.5 | 6 | 1 | 3 | 026F |
| | 40.1 | 59.8 | 12.2 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 38.1 | 64.8 | 18.7 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 36.3 | 70.8 | 25.6 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| | 34.6 | 78 | 33.3 | 1<br>5 | 11<br>3 | 036B<br>0A6B |
| | 33.1 | 86.8 | 41.9 | 1<br>2<br>3<br>6 | 13<br>7<br>5<br>3 | 03AA<br>04EA<br>06AA<br>0C6A |
| 52 | 45.6 | 60.5 | 6.5 | 1 | 3 | 026F |
| | 43.2 | 65.3 | 13.2 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 41.1 | 70.8 | 20.2 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 39.1 | 77.4 | 27.7 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| | 37.4 | 85.3 | 36 | 1<br>5 | 11<br>3 | 036B<br>0A6B |
| 56 | 48.8 | 65.6 | 7 | 1 | 3 | 026F |
| | 46.3 | 70.8 | 14.2 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 44 | 76.9 | 21.8 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 42 | 84.1 | 29.9 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| 60 | 52 | 70.8 | 7.5 | 1 | 3 | 026F |
| | 49.3 | 76.5 | 15.2 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 46.9 | 83.1 | 23.3 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| | 44.7 | 90.9 | 32 | 1<br>2<br>4 | 9<br>5<br>3 | 032C<br>04AC<br>086C |
| 64 | 55.2 | 76.1 | 8 | 1 | 3 | 026F |
| | 52.4 | 82.2 | 16.3 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| | 49.8 | 89.3 | 24.9 | 1<br>3 | 7<br>3 | 02ED<br>066D |
| 68 | 58.4 | 81.4 | 8.5 | 1 | 3 | 026F |
| | 55.4 | 88 | 17.3 | 1<br>2 | 5<br>3 | 02AE<br>046E |
| 72 | 61.5 | 86.8 | 9 | 1 | 3 | 026F |

## 7.3 Application Note

This chapter describes the startup/stop sequence of the clock modulator and the modulation parameter settings.

### Recommended startup/stop sequence

| | | |
|---|---|---|
| start | 1. | Switch modulator from power down to power up mode: Set PDX=1 |
| | 2. | Switch on PLL |
| | 3. | Wait PLL lock time (refer to the CKMR:PCM bit). |
| | 4. | Set CMPR register to a proper setting |
| | 5. | Enable clock modulation by setting MODEN=1<br>After the calibration is finished, the clock switches from unmodulated to modulated clock and the MODRUN flag changes to 1 |
| | | ... running... |
| | 6. | Disable modulator by setting MODEN=0 |
| stop | 7. | Wait until MODRUN changes to 0 |
| | 8. | Switch to power down mode by setting PDX=0 |
| | 9. | Disable PLL, switch to other operation mode (Standby mode, etc.) |

### Note:

Do not enable the modulator before the PLL lock time has elapsed. Do not disable the PLL while the modulator is running.

### Modulation parameter

It is not possible to recommend a particular modulation parameter setting to achieve a particular reduction in EMI. The best setting depends much on the actual application, the whole system and the requirements.

In order to find the optimal modulation parameter setting, the following approach is recommended.

| | | |
|---|---|---|
| 1. | define the required PLL frequency CLKPLL based on performance needs | e.g. 32 MHz |
| 2. | determine the maximal allowed CLKS1 clock frequency of the MCU (see data sheet). When the CLKB or CLKP1 divider is set to "divided by 1", select the maximal allowed CLKB/CLKP1 frequency. | e.g. 48 MHz |
| 3. | Determine the settings with the highest $f_{max}$ CLKMOD frequency which is still below the above determined maximal allowed clock frequency of the MCU. | e.g. CMPR = 0x032C, 0x04AC, 0x086C ($f_{max}$ = 45.6MHz) |
| 3. | Choose the setting with the highest frequency resolution. | e.g. CMPR = 0x032C: frequency resolution = 9, modulation degree = 1 |
| 4. | Perform EMI measurements | |
| 5. | If the EMI measurements does not fulfill the requirements, choose the next setting: | |
| | If available, select the next setting with the same maximum CLKMOD frequency range (higher modulation degree and lower frequency resolution).<br>This may improve the reduction of the fundamental < 100 MHz, but worsen the reduction in the upper frequency band > 100 MHz | e.g. CMPR = 0x04AC: frequency resolution = 5, modulation degree = 2 and CMPR = 0x086C: frequency resolution = 3, modulation degree = 4 |
| | Otherwise select the next settings with a lower maximum CLKMOD frequency range | e.g. CMPR = 0x02ED |
| 6. | repeat item 4. with the new setting and continue until the best setting is identified | |

### Note: NOT ALL SETTINGS ARE ALLOWED ON EVERY DEVICE!

Please consider the actual maximal allowed CLKS1 and CLKB/CLKP1 clock frequency of the MCU (refer to the data sheet).

### Note: SETTING RESTRICTIONS FOR FLASH DEVICES

When using the clock modulator, extra care is needed for Flash devices. A Flash timing setting must be used which is valid at the minimum and the maximum possible output frequency of the clock modulator.

# 8. Resets and Startup

This chapter describes the resets and the startup of for the F2MC-16FX family microcontrollers.

8.1 Resets

8.2 Reset, System clock and Stabilization Wait Times

8.3 Startup after Power and External reset

8.4 Boot ROM program execution and Operation mode and ROM Configuration Block

8.5 Reset Control Registers

8.6 Operation of the Clock stop detection function and reset

8.7 Operation of the low voltage reset function

## 8.1 Resets

If a reset is generated, the CPU immediately stops the current execution process and waits for the reset to be cleared. The CPU then begins with the Boot ROM program execution. Depending on the Mode pin settings, the Boot ROM program branches into different operating modes or starts the user program.

The five causes of a reset are as follows:

- Power reset (Power-on or Low voltage)
- External reset request via the RSTX pin
- Clock stop detection
- Software reset request
- Watchdog timer overflow

### 8.1.1 Causes of a reset

Table 8-1 lists the causes of a reset.

Table 8-1. Causes of a reset

| Type of reset | Cause | System clock CLKS1 and CLKS2 after reset release | Reading of mode pin setting | RAM contents |
|---|---|---|---|---|
| Power | When the power is turned on or when Vcc is below the Low Voltage Detector level (LVL in CILCR register). Please refer to the Datasheet for the analog values. | RC clock CLKRC | Yes | Not guaranteed |
| External pin | L level input to RSTX pin | | | Maintained[*1] |
| Clock stop detection | Failure of external oscillator | | No | Not guaranteed |
| Software | A "1" is written to the SRSTG bit of the Reset Configuration Register (RCR). | | | Maintained |
| Watchdog timer | Watchdog timer overflow. | | | |

The MCU is running with the RC clock set to nominal 2MHz after each reset.

### Power reset

A power reset is generated when the power is turned on with a power on rise time as specified in the datasheet (power-on reset) or when the low voltage detector detects that the power supply Vcc is below a certain value as specified in the LVL bits of the CILCR register (low voltage reset). Refer to the Datasheet for the analog LVL levels. A power reset is internally prolonged by the Power reset extension circuit to a minimum length of 700 RC clock cycles

The contents of internal RAM and all registers with initial value 'X' is undefined after a Power reset.

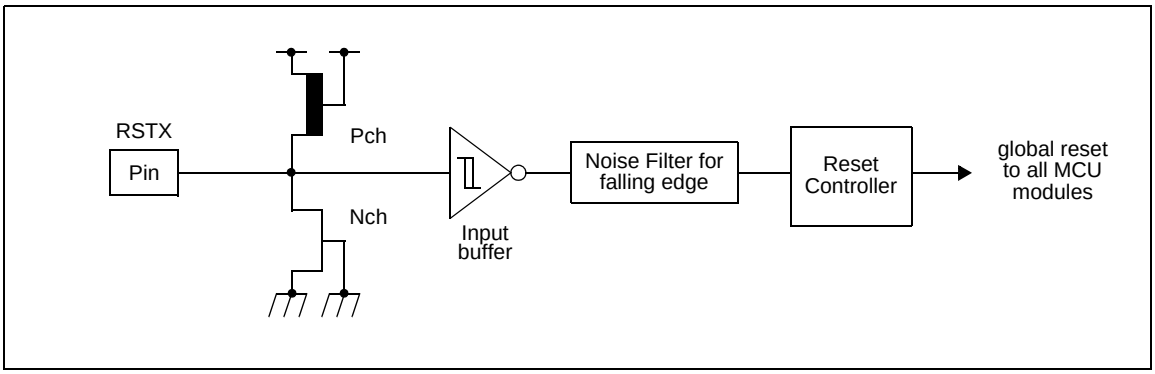
### External pin reset

An External pin reset is generated by a L level input to the external reset pin (RSTX pin). This signal is internally prolonged by the External reset extension circuit to a minimum length of 700 RC clock cycles.

Because the external reset is asynchronous, it becomes active with little delay (some 10ns) after assertion, disregarding whether the device is clocked or not.

The contents of internal RAM and all registers which are not reset (initial value 'X') is maintained except for the 4 bytes at the address 7FFC-7FFF.

Figure 8-1. Block diagram of external reset pin




### Clock stop detection reset

A Clock stop detection reset is generated upon a failure of the Main or Sub clock oscillator depending on the current operation mode and the setting of the Reset Configuration Register (RCR). See section 8.6 Operation of the Clock stop detection function and reset for more details.

The contents of internal RAM memory and all registers which are not reset (initial value 'X') cannot be guaranteed after a Clock stop detection reset.

### Software reset

A Software reset is an internal reset generated by writing "1" to the SRSTG bit of the Reset Configuration Register (RCR).

The contents of internal RAM and all registers which are not reset (initial value 'X') is maintained.

### Watchdog timer reset

A Watchdog timer reset is generated when the watchdog timer is activated but not cleared within the defined period as specified in the chapter Watchdog Timer and Watchdog Reset on page 225

The contents of internal RAM and all registers which are not reset (initial value 'X') is maintained.

### Note:

*1: Except for bytes at addresses 7FFC-7FFF.

The contents of the registers in the GPR bank 00 and 01 as well as the 4 above mentioned RAM bytes are undefined after each reset. The contents of the other GPR registers is either maintained or undefined (depending on reset cause as described above).

**Note:**

Status of pins at reset: all pins are in Hi-Z state. See also Table 12-11 on page 284.

## 8.2 Reset, System clock and Stabilization Wait Times

The F2MC-16FX family has five reset causes. The System clocks (CLKS1 and CLKS2) are set to RC clock after each reset. The stabilization wait time depends on the reset cause and the status of the RC oscillator when the reset occurs.

### 8.2.1 Reset causes and stabilization wait times

Table 8-2 summarizes which stabilization times are applied for the different reset causes.

Table 8-2. Reset cause and stabilization wait times

| Reset cause | Stabilization wait time |
|---|---|
| Power reset | 700 RC clock cycles Power reset extension time (approximately 350µs at 2MHz nominal RC clock frequency) plus the RC clock stabilization time (see datasheet).<br>This wait time starts AFTER reaching a valid power supply (external and internal voltage). |
| External reset | 700 RC clock cycles External reset extension time (approximately 350µs at 2MHz nominal RC clock frequency) plus the RC clock stabilization time (see datasheet).<br>The External reset extension time starts already at the falling edge of the RSTX input signal while the RC clock stabilization time is applied after RSTX release. |
| Clock stop detection reset | RC clock stabilization time is applied (see datasheet) |
| Software reset | |
| Watchdog timer reset | |

Each reset activates the RC oscillator, Main oscillator and Sub oscillator, stops the PLL and resets the Clock Stabilization Select Register (CKSSR).

A Power or External reset clears all clock ready monitor bits.

Other types of resets only clear the PLL and RC clock ready monitor bit, but not the Main and Sub clock ready monitor bits. Active clocks stay active and disabled clocks are activated and become ready after the stabilization time defined in the CKSSR register.

See Clocks on page 119, for more details about selected clocks and oscillation stabilization wait times.

### 8.2.2 Stabilization wait time in case of an External reset

The stabilization wait time in case of an External reset consists of two parts, the External reset extension time and the RC clock stabilization time. The External reset extension time starts counting with the assertion of RSTX (falling edge) while the RC clock stabilization time starts counting after RSTX release. Hence the start of the program execution depends on the assertion time of RSTX as follows:

#### External reset asserted for more than 700 RC clock cycles

The External reset extension time is already expired in this case. Hence the CPU starts executing the Boot ROM program after the RC clock stabilization time after clearing the External reset.

#### External reset asserted for less than 700 RC clock cycles

The External reset extension time is not expired in this case. Hence the execution of the Boot ROM program by the CPU is delayed until the External reset extension time plus the RC clock stabilization time is expired.

### 8.2.3 Stabilization of the Main oscillator

The initial value of the Main oscillation stabilization wait time selector is set to $2^{18}$ Main clock cycles (MCST[2:0] = "111"). This value can be overwritten directly after start of the User program execution (when the MCU is running in RC clock mode). Select a value suitable for the used oscillator.

The Main oscillation stabilization wait time is measured with the Main clock counter which starts counting when the Power and External reset extension time is expired. Hence if RSTX is asserted for more than 700 RC clock cycles, then the Main oscillation stabilization wait time takes place already when RSTX is still asserted.

This allows controlling the Main oscillation stabilization time by RSTX. Release RSTX after stabilization of the Main oscillator and set MCST[2:0] to a value that already expired. However before disabling the Main oscillator (by writing to the MCE bit or by going to Stop mode), a suitable value must be written to the MCST[2:0] bits because this value will be used after reactivation of the Main oscillator.

### 8.2.4 Stabilization of the Sub oscillator

The initial value of the Sub oscillation stabilization wait time selector is set to $2^{16}$ Sub clock cycles (SCST[1:0] = "11"). This value can be overwritten directly after start of the User program execution (when the MCU is still running in RC clock mode). Select a value suitable for the used oscillator.

The Sub oscillation stabilization wait time is measured with the Sub clock counter which starts counting when RSTX is cleared and the External reset and Power reset extension time is expired.

## 8.3 Startup after Power and External reset

After a Power or External reset event, the MCU waits for the stabilization of the power supply and the RC oscillator. Then the MCU starts executing the Boot ROM program with the RC clock as clock source (RC Run mode).

A transition to another clock mode (Main clock, PLL clock or Sub clock) is possible after stabilization of the respective clock. The stabilization times for the external oscillators can be adjusted to the characteristics of the connected oscillators.

### 8.3.1 Initialization after startup

After startup, the MCU must be reset to put all registers in the initial state. This is possible by applying an external reset or by using the Power reset functions.

#### Initialization by External reset

Applying a low level at the RSTX input during or after switching on the power supply initializes the complete MCU. Reset extenstion and stabilization wait times are applied as specified below.

#### Initialization by Power reset

The MCU can also be started without asserting an External reset by using the Power reset functions.

If a power-on voltage profile as specified in the datasheet under "Power On Reset timing" is applied, then the MCU will start running after reaching a power supply voltage level above the default threshold of the Low Voltage Detector (Level 0).

Reset extenstion and stabilization wait times are applied as specified below.

### 8.3.2 Reset extension

Power and External reset events are extended by the reset extension circuit to guarantee the stabilization of the voltage regulators and comparators before program execution starts.

#### Power reset extension

The Power reset extension counter is a 10 bit counter that is initialized by the power-on detector and by a low voltage reset and that is clocked by the RC clock CLKRC.

This counter keeps the internal Power reset signal PRST active for 700 RC clock cycles after the last power-on or low voltage event was cleared. PRST is deactivated if the power supply was stable for at least 700 RC clock cycles (approximately 350µs at 2MHz nominal RC clock frequency).

### External reset extension

The External reset extension counter is also a 10 bit counter that is initialized by a falling edge at the RSTX input pin and that is clocked by the RC clock CLKRC. A Power reset also initializes this counter to achieve a defined startup time even if the device is started by a Power reset only.

This counter keeps the signal ERST active for 700 RC clock cycles after the last falling edge of the RSTX input pin (approximately 350µs at 2MHz nominal RC clock frequency).

If RSTX is asserted for more than 700 RC clock cycles, then this function has no effect.

Figure 8-2 shows a block diagram of the reset extension circuit and Figure 8-3 shows a timing diagram of the External reset extension function.

Figure 8-2. Block diagram of reset extension circuit

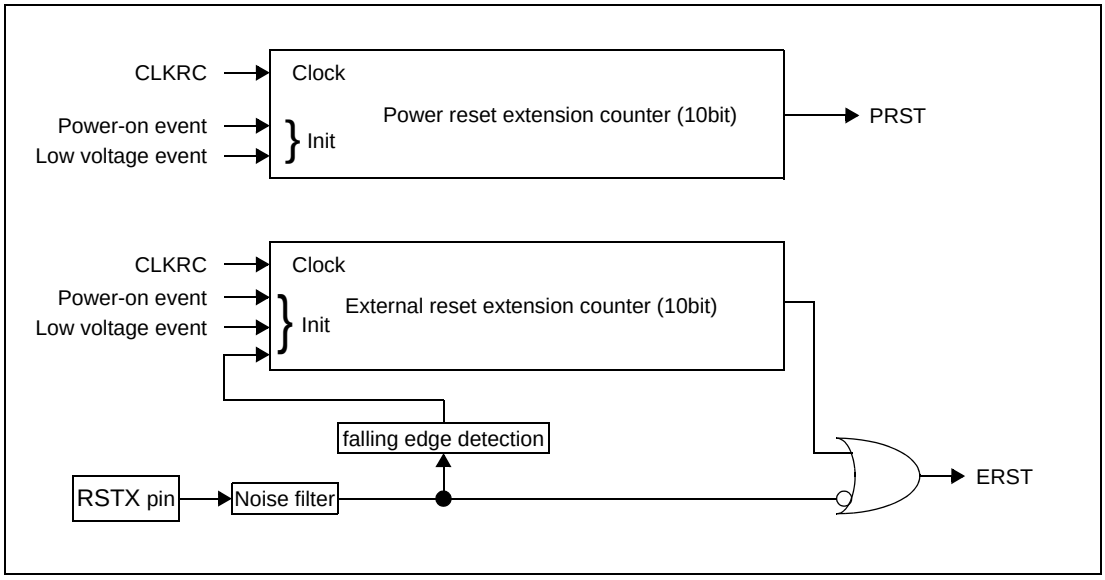
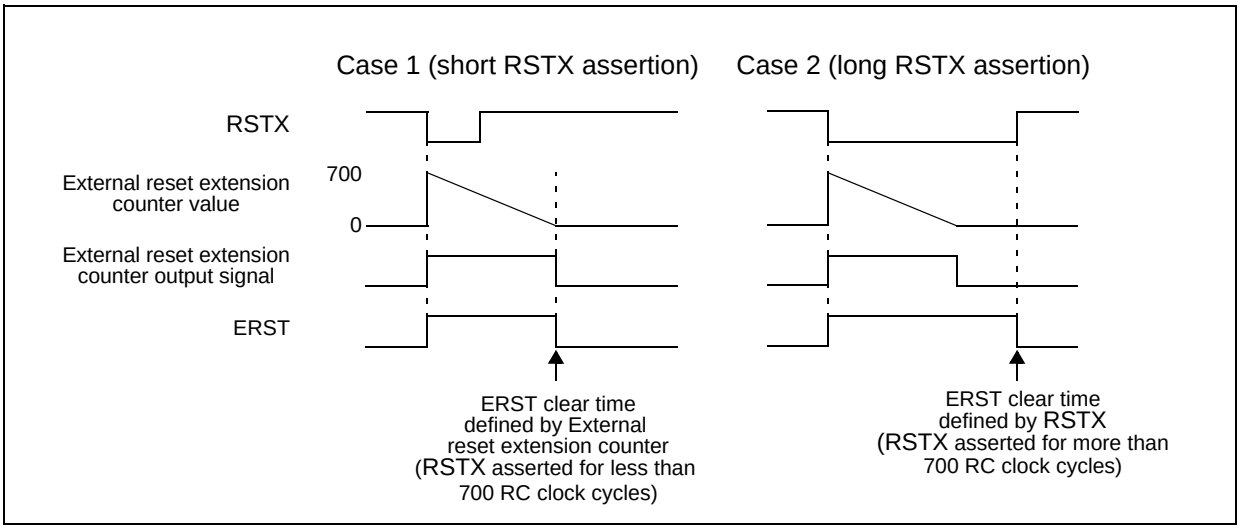


Figure 8-3. Function of the External reset extension circuit




## 8.3.3 Source clock timers and clock ready monitor bits

The Source clock timers (RC clock timer, Main clock timer and Sub clock timer) and the clock monitor bits (RCM, MCM, PCM and SCM) are all cleared by a Power or External reset event while the three oscillators are enabled.

### RC clock timer, Sub clock timer, RCM bit and SCM bit

The RC clock timer and Sub clock timer are cleared and stopped by PRST or ERST. They start counting when both PRST and ERST are released. Hence the RC clock ready monitor bit RCM and the Sub clock ready monitor bit SCM cannot be set as long as ERST or PRST are active.

### Main clock timer and MCM bit

The Main clock timer is cleared and stopped during the External or Power reset extension time only. Hence the Main clock timer is activated 700 RC clock cycles after a RSTX falling edge or 700 RC clock cycles after clearing a power-on or low voltage event. Thus the Main oscillation stabilization wait time takes place even if RSTX is still asserted.

### PCM bit

The PLL is always disabled and the PLL clock ready monitor bit PCM cleared by reset. The PLL must be enabled by software after User program start.

## 8.3.4 Start of program execution after Power or External reset

The RC clock timer starts counting after deactivation of PRST and ERST. After the RC clock stabilization time, the RC clock ready monitor bit RCM will be set and the internal reset signal for the CPU will be released. After reset release, the CPU is always in RC clock mode with the RC clock frequency set to nominal 2MHz and starts executing the Boot ROM program.

The low Voltage Detector level (LVL bits of CILCR register) is initialized to the lowest possible value. If a higher detection value should be needed, the register value can be changed accordingly.

See also section 8.2 Reset, System clock and Stabilization Wait Times for more details.

## 8.3.5 Transition to Main clock mode

A transition to Main clock mode is done by writing to the System clock select bits of the CKSR register.

The transition however is delayed until the Main clock is stabilized which is indicated by the Main Clock ready Monitor bit MCM (set to "1").

The stabilization time of the Main oscillator is defined by the CKSSR: MCST[2:0] bits as described in section 8.2 Reset, System clock and Stabilization Wait Times.

**Note:** Use of the Fast Clock Input feature, which is optional on some devices, requires setting the FCI bit of the CILCR register to 1 before switching to main clock mode. Please refer to the Datasheet for the input characteristics of the oscillator pin.

### 8.3.6 Transition to PLL clock mode

The PLL must be configured and enabled by software. The PLL stabilization wait time takes place after PLL activation and after stabilization of the Main clock (MCM="1").

A transition to PLL clock mode is possible when the PLL Clock ready Monitor bit PCM is set.

It is possible to directly switch from RC clock mode to PLL clock mode or via Main clock mode.

### 8.3.7 Transition to Sub clock mode

The Sub oscillator is enabled by each reset and the Sub clock stabilization time starts after PRST and ERST are released.

The stabilization time of the Sub oscillator is determined by the CKSSR: SCST[1:0] bits which define when the Sub Clock ready Monitor bit SCM will be set. A transition to Sub clock mode is delayed until SCM is set to "1".

## 8.4 Boot ROM program execution and Operation mode and ROM Configuration Block

When the reset signal is released, the MCU starts with the execution of the internal Boot ROM program. The Boot ROM program reads the status of the mode pins (MD2-MD0), which define the operation mode of the MCU. Depending on the mode pin setting, the MCU activates the Parallel Flash Programming mode, the Serial Communication mode or starts executing the User program.

After the activation of the MCU operation mode, further Boot ROM program execution is defined by the configuration stored in the so-called ROM Configuration Block in Flash/ROM, which also determines the Boot Vector (user program start address).

Finally, the Boot ROM program will start execution of the user program at the Boot Vector.

### 8.4.1 External bus

By default, the Boot Vector (user program start address) is read from the fixed address FF:FFDC$_H$.

Devices with external bus interface allow reading the Boot Vector from external memory. Depending on the mode pin setting, three different external bus configurations can be selected for reading the Boot Vector.

The Boot ROM program reads the Boot Vector and the mode byte from a fixed address. The bus configuration is changed according to the given mode byte and the user program is started at the given address (Boot Vector).

### 8.4.2 Mode pins

The mode pins MD[2:0] define the operation mode of the MCU. The mode pin setting is internally sampled and can only be changed by a Power or External reset event (the Mode pin data is sampled when the signals PRST and ERST defined in Figure 8-2 are both deactivated). Hence the operation mode cannot be changed by a Clock stop detection, Software or Watchdog timer reset event.

The sampled mode pin signals are read by the Boot ROM program after reset release, which determines the operation mode according to the following table.

Table 8-3. Mode Pin settings

| Mode pin setting | | | Operation mode name | Boot Vector Source | Configuration of external bus for Boot Vector read access | Remarks |
|---|---|---|---|---|---|---|
| MD2 | MD1 | MD0 | | | | |
| 0 | 0 | 0 | External Vector mode 0 | External memory | 8 bits, address/data multiplexed | Available only for devices with external bus interface |
| 0 | 0 | 1 | External Vector mode 1 | External memory | 16 bits, address/data multiplexed | Available only for devices with external bus interface |
| 0 | 1 | 0 | Serial Communication mode | - | - | Used for serial Flash programming |
| 0 | 1 | 1 | Internal Vector mode | Internal memory or fixed value | - | Available for all devices |
| 1 | 0 | 0 | Reserved (Test mode) | - | - | |
| 1 | 0 | 1 | Reserved (Test mode) | - | - | |
| 1 | 1 | 0 | External Vector mode 2 | External memory | 8 bits, address/data non-multiplexed | Available only for devices with external bus interface |
| 1 | 1 | 1 | Parallel Flash programming mode | - | - | Available only for Flash devices |

## 8.4.3 Operation modes

### External Vector modes (MD[2:0]="000", "001" or "110")

External vector modes are CPU run modes where the Boot Vector and a mode byte are read from an external memory at FF:FFDC$_H$ via the external bus interface. These modes are available only for devices with an external bus interface.

Optionally, a built-in monitor debugger kernel can be initialized, if the BDM Activation Marker is set in ROM Configuration Block. Note that neither internal ROM nor built-in monitor debugger kernel can be used in External Vector mode, if read security is enabled.

See External Bus Interface on page 235 for more details. For details how to configure the monitor debugger kernel, please refer to application note MCU-AN-390213-E.

### Internal Vector mode (MD[2:0]="011")

The Internal vector mode is a CPU run mode where the Boot Vector (user program start address) is read from an internal memory at FF:FFDC$_H$.

Optionally, a fixed Boot Vector can be selected, if the Fixed Vector Activation Marker is set in the ROM Configuration Block. The fixed Boot Vector allows implementation of boot loader applications.

After main reset and before starting user program, the Boot ROM temporarily scans dedicated UART channels for an external communication request as in Serial Communication mode[1]. Optionally, this scanning can be disabled, if the UART Scan Deactivation Marker is set in the ROM Configuration Block.

Optionally, a built-in monitor debugger kernel can be initialized, if the BDM Activation Marker is set in the ROM Configuration Block.

The Internal Vector mode is available for all MCU types and the fixed Boot Vector is given by DF:0080$_H$.

### Serial Communication mode (MD[2:0]="010")

This mode allows the reading and writing of any memory address by serial communicated read/write commands. Programming of the internal Flash is possible by executing tiny program in RAM.

1. Not supported by all derivatives.

Despite of dedicated read/write commands, the internal RAM area 0x7C00 to 0x7FFF might also be changed by other commands (e.g. CRC check).

If the ROM/Flash security feature is activated, then reading, writing and programming is only possible after transmitting the correct security key.

See Flash Memory on page 659 and ROM/Flash Security on page 713 for more details.

### Parallel Flash programming mode (MD[2:0]="111")

This mode allows the programming of the internal Flash on a parallel Flash programmer. This mode is available only for Flash devices.

See Flash Memory on page 659 for more details.

### Test mode (MD[2:0]="100" or "101")

These modes are reserved for test functions.

## 8.4.4 ROM Configuration Block (RCB)

The ROM Configuration Block is a fixed area in Flash A/ROM and if present Flash B, Data Flash A and Data Flash B, which enables the user to configure the behavior of the Boot ROM. User program should not allocate it for program except for specified configuration data.

The ROM Configuration Block A (RCBA) is located at address DF:$0000_H$ - DF:$007F_H$ (start address of Flash A or ROM). The ROM Configuration Block B (RCBB) is located at address DE:$0000_H$ - DE:$002F_H$ (start address of Flash B, if present). The ROM Configuration Block for the Data Flash A is the Data Flash Configuration Block A (DFCBA) located at address 0E:$FF00_H$ - 0E:$FF2F_H$ (if present). The ROM Configuration Block for the Data Flash B is the Data Flash B Configuration Block B (DFCBB) located at address 0E:$FE00_H$ - 0E:$FE2F_H$ (if present). They are read and processed by Boot ROM. Its content can be set and changed by any method, which writes or erases Flash memory.

RCBA, RCBB, DFCBA and DFCBB configure the security setting of belonging Flash/ROM or Data Flash. In addition, the RCBA also configures the boot sequence and background debug mode.

Table 8-4. Structure of ROM Configuration Block A (RCBA)

| Offset | Sub-block | Marker | Comment |
|---|---|---|---|
| $00_H$ | Security and Protection configuration block | MSBA | Secure Flash A/ROM by $99_H$, permit access by $66_H$ |
| $01_H$ | | reserved | |
| $02_H$-$11_H$ | | MSUKA | Unlock key for secured Flash A/ROM |
| $12_H$-$1B_H$ | | reserved | |
| $1C_H$-$1F_H$ | | FWPAMA | Activation of Flash write protection by $292D3A7B_H$ |
| $20_H$-$27_H$ | | FWPSMA | Sector selection for Flash write protection |
| $28_H$-$2F_H$ | | reserved | |
| $30_H$-$33_H$ | Boot Sequence configuration block | FBVAM | Activation of fixed Boot Vector in internal vector mode by $292D3A7B_H$ |
| $34_H$-$37_H$ | | USDM | Deactivation of UART scanning in internal vector mode by $292D3A7B_H$ |
| $38_H$-$3F_H$ | | reserved | |
| $40_H$-$43_H$ | Background Debugging configuration block | BDMAM | Activation of BDM by $292D3A7B_H$ |
| $44_H$-$7F_H$ | | reserved | BDM Configuration |

Table 8-5. Structure of ROM Configuration Block B (RCBB)

| Offset | Sub-block | Marker | Comment |
|---|---|---|---|
| 00H | Security and Protection configuration block | MSBB | Secure Flash B by $99_H$, permit access by $66_H$ |
| 01H | | reserved | |
| 02H-11H | | MSUKB | Unlock key for secured Flash B |
| 12H-1BH | | reserved | |
| 1CH-1FH | | FWPAMB | Activation of Flash write protection by 292D3A7BH |
| 20H-27H | | FWPSMB | Sector selection for Flash write protection |
| 28H-2FH | | reserved | |

Table 8-6. Structure of Data Flash Configuration Block A (DFCBA)

| Offset | Sub-block | Marker | Comment |
|---|---|---|---|
| 00H | Security and Protection configuration block | MSBDA | Secure Data Flash A by $99_H$, permit access by $66_H$ |
| 01H | | reserved | |
| 02H-11H | | MSUKDA | Unlock key for secured Data Flash A |
| 12H-2FH | | reserved | |

Table 8-7. Structure of Data Flash Configuration Block B (DFCBB)

| Offset | Sub-block | Marker | Comment |
|---|---|---|---|
| 00H | Security and Protection configuration block | MSBDB | Secure Data Flash B by $99_H$, permit access by $66_H$ |
| 01H | | reserved | |
| 02H-11H | | MSUKDB | Unlock key for secured Data Flash B |
| 12H-2FH | | reserved | |

## 8.4.5 Function description of RCB Markers

### *8.4.5.1 Security and Protection Configuration Block (SPCB)*

The read security feature prevents the unauthorized read-out of Flash/ROM contents and targets all cases other than read access by application in Internal Vector mode.

For details, see section Usage of the ROM/Flash Security on page 714.

The write protection feature prevents the unintended writing of Flash memory by application in Internal Vector mode and External vector mode. The write protection feature is only supported for Flash A and Flash B.

For details, see section Protecting sectors from being erased/written to on page 695.

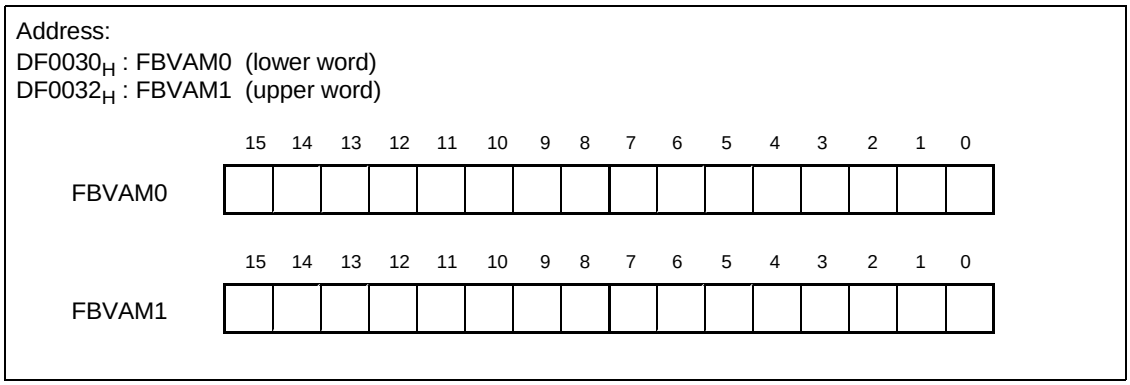
Note that read security setting also affects the boot sequence, if built-in monitor debugger kernel is activated.

### *8.4.5.2 Boot Sequence Configuration Block (BSCB)*

#### Enabling fixed Boot Vector

In Internal Vector mode, the Boot Vector (user program start address) can be set to the fixed value DF:$0080_H$ instead of reading value from FF:$FFDC_H$. This fixed Boot Vector starts program execution in a small Flash sector and allows implementation of a boot loader.

Figure 8-4. Configuration of the Fixed Boot Vector Activation Marker



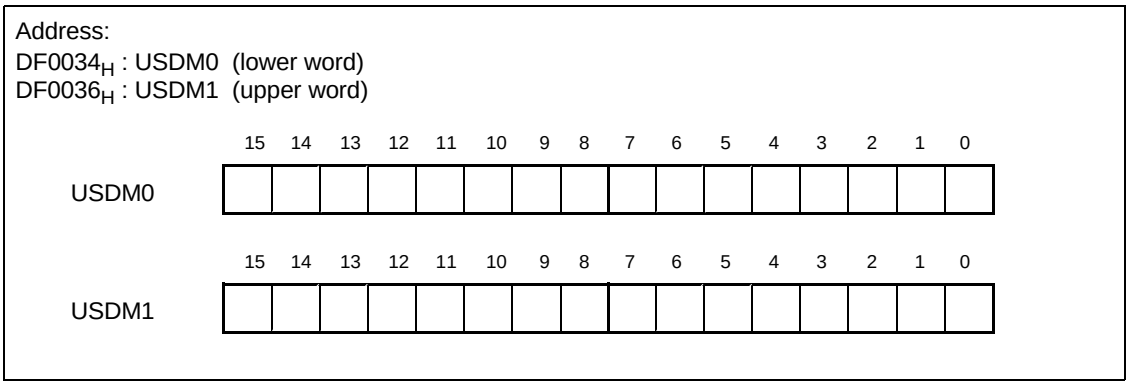

When the content of {FBVAM1, FBVAM0} = 292D3A7BH, then the fixed Boot Vector is used.

For any other value of {FBVAM1, FBVAM0}, the Boot Vector is read from the Boot Vector location.

### Disabling temporary UART scan

After main reset in Internal Vector mode, dedicated UART channels are scanned for a serial communication request as in Serial Communication mode without changing the mode pin setting. This scanning can be disabled to shorten the start-up time.

Figure 8-5. Configuration of the UART Scan Deactivation Marker




When the content of {USDM1, USDM0} = 292D3A7BH, then no temporary UART scanning is performed.

For any other value of {USDM1, USDM0}, the Boot ROM will scan dedicated UART channels for limited time after main reset.

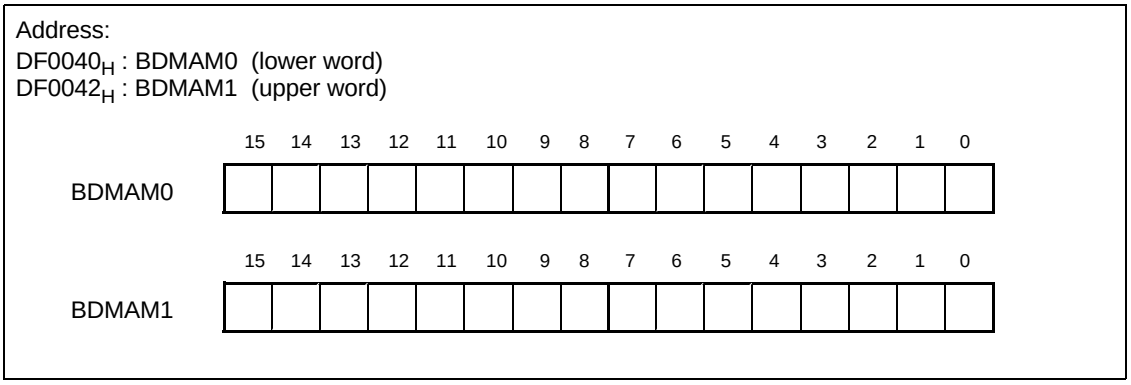
### 8.4.5.3 BDM Configuration Block (BDCB)

In Internal Vector mode and External Vector mode, a built-in monitor debugger kernel can be activated background debug mode – BDM). However, an external tool is required to operate the MCU in background debug mode (BDM) and Foreground Debug Mode (FDM). Furthermore, the debugger operation has to be refined by more settings in the BDM configuration block. Do not use this feature without appropriate tool support.

#### Enabling built-in monitor debugger kernel

In Internal Vector mode and External Vector mode, a built-in monitor debugger kernel can be activated.

Figure 8-6. Configuration of the BDM Activation Marker



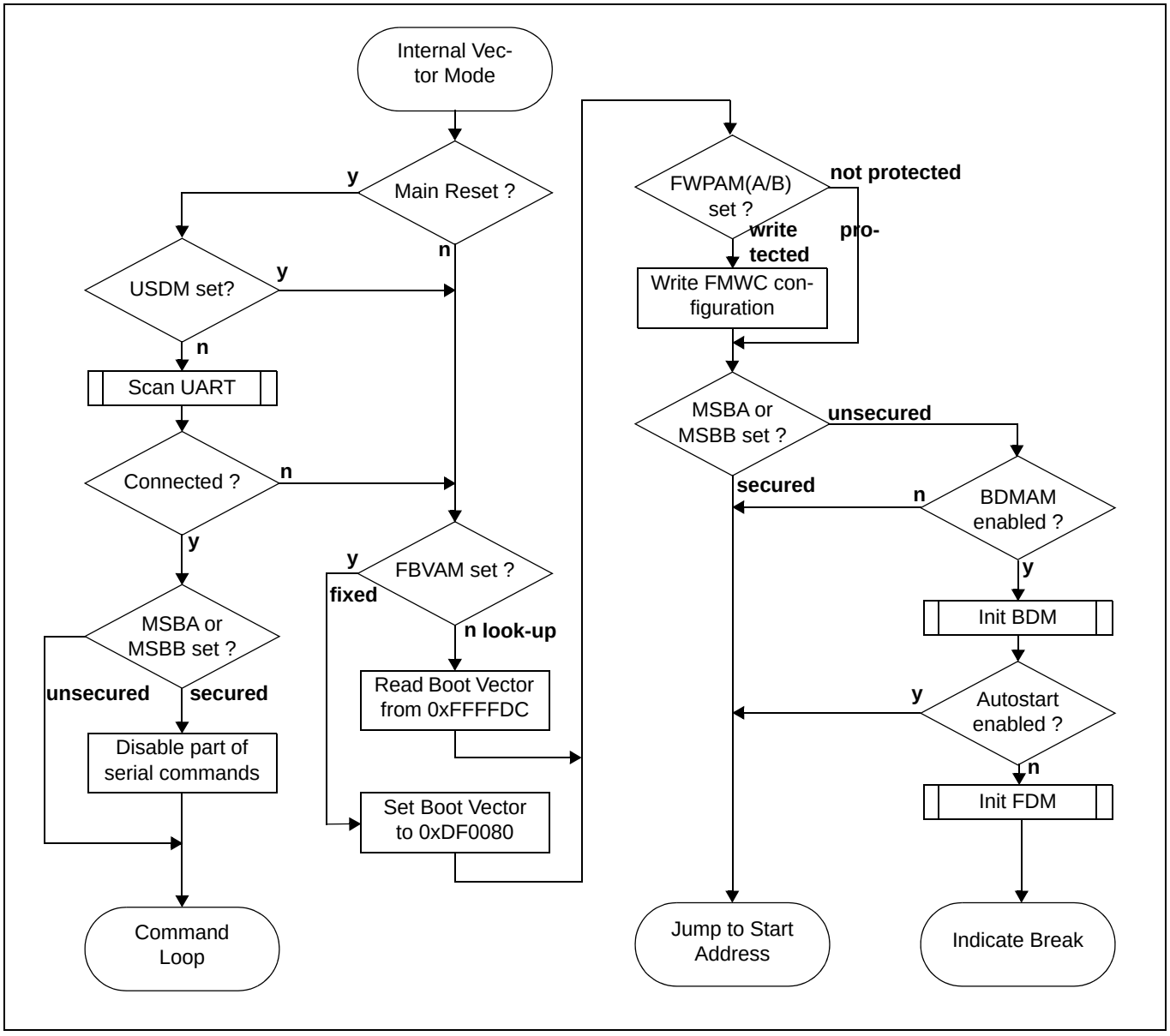

When the content of {BDMAM1, BDMAM0} = 292D3A7BH, the BDM is activated.

For any other value of {BDMAM1, BDMAM0}, the BDM is not activated.

Note that BDM is not activated, if ROM/Flash security is set.

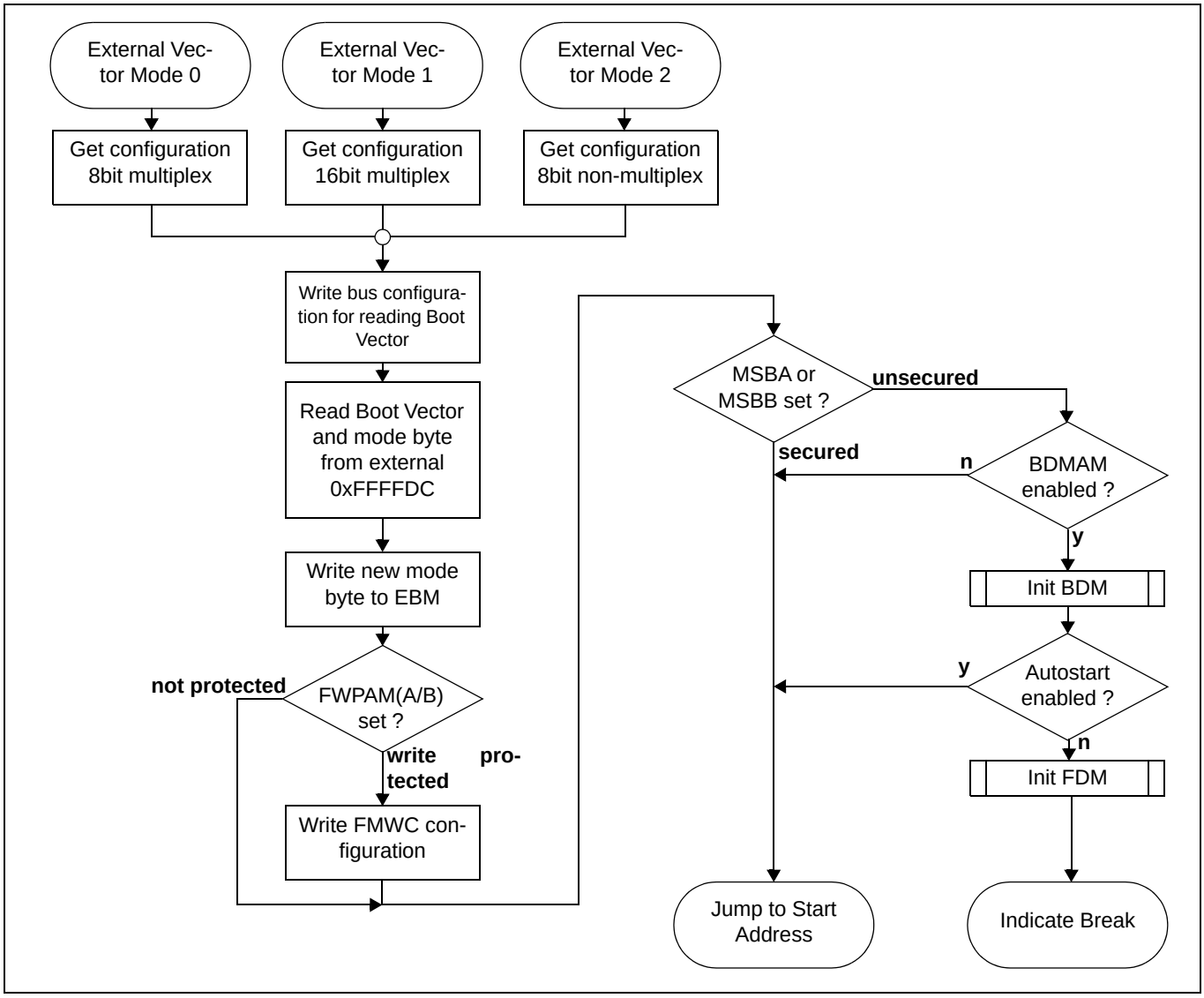
## 8.4.6 Flowchart Internal Vector Mode

Figure 8-7. Boot ROM sequence in Internal Vector Mode



## 8.4.7 Flowchart in External Vector Mode

Figure 8-8. Boot ROM sequence in External Vector Modes



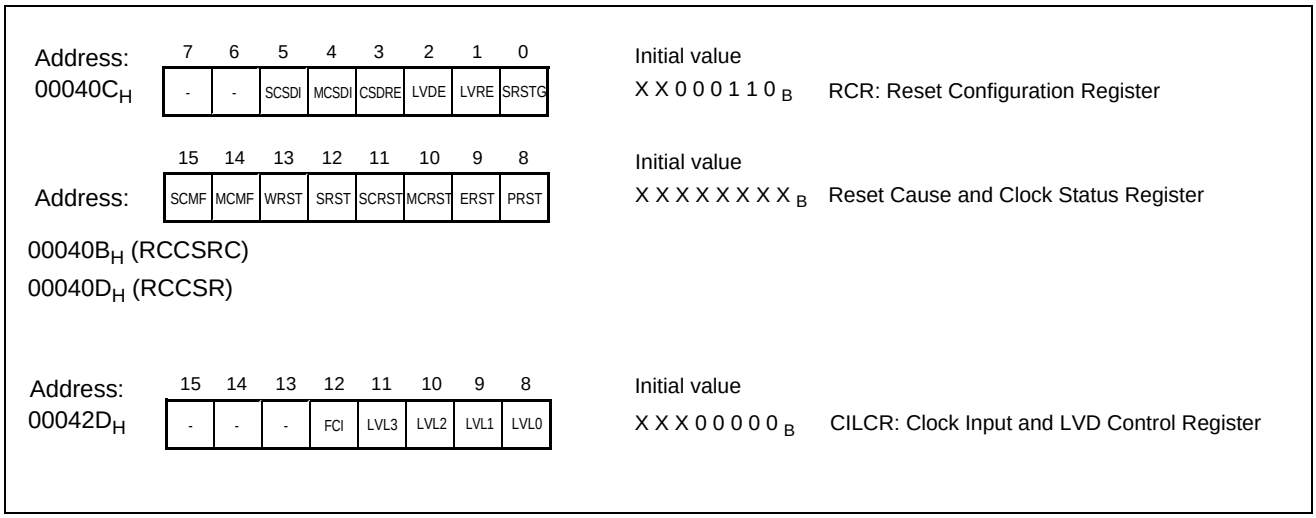
## 8.5 Reset Control Registers

This section lists the Reset Control Registers and describes the function of each register in detail.

### Reset Control Registers

The Reset Controller has two registers, the Reset Configuration Register (RCR) and the Reset Cause and Clock status Register (RCCSR/RCCSRC). Figure 8-9 shows an overview of the Reset Control register.

Figure 8-9. Reset Control Registers




### 8.5.1 Reset Configuration Register (RCR)

The Reset Configuration Register (RCR) is used to assert a Software reset, configure the low voltage reset and detector and to configure the Clock stop detection circuit.

#### Configuration of the Reset Configuration Register (RCR)

Figure 8-10 shows the configuration of the Reset Configuration Register (RCR) and Table 8-8 describes the function of each bit.

Figure 8-10. Configuration of the Reset Configuration Register (RCR)

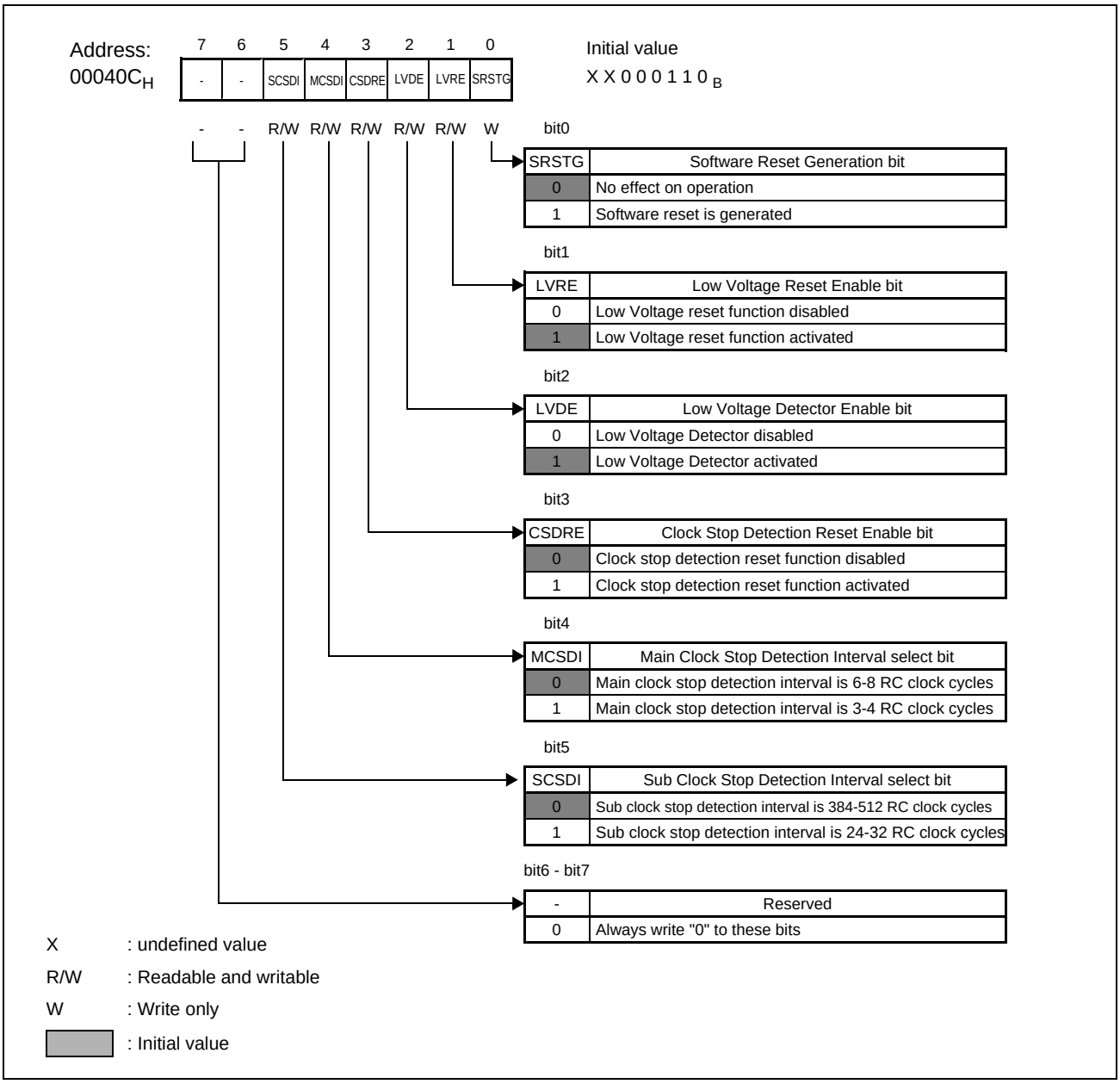


Table 8-8. Function Description of Each Bit of the Reset Configuration Register (RCR)

| Bit name | | Function |
|---|---|---|
| bit 0 | SRSTG:<br>Software Reset Generation bit | • When "1" is written to this bit, an internal Software reset is generated.<br>• Writing "0" to this bit has no effect.<br>• The read value of this bit is always "0". |
| bit 1 | LVRE:<br>Low Voltage Reset Enable bit | • This bit controls the Low voltage reset function which is one reset cause of the Power reset.<br>• Setting this bit to "1" enables the Low voltage reset function.<br>• Writing "0" to this bit disables the function.<br>• This bit is initialized to "1" (Low voltage reset active) by an External and a Power (power-on) reset only.<br>• Do not disable the low voltage detector (by writing "0" to the LVDE bit) as long as this bit is set to "1".<br>• Do not set this bit from "0" to "1" before the low voltage detector is enabled and stabilized.<br>• In devices in which the low voltage reset circuit is always enabled in the Internal vector mode, the setting of this bit has no effect for Mode pins set to "011" (Persistent Low Voltage Reset feature). |
| bit 2 | LVDE:<br>Low Voltage Detector Enable bit | • This bit controls the low voltage detector which is used for the Low voltage reset function.<br>• Setting this bit to "1" enables the low voltage detector.<br>• Writing "0" to this bit disables the low voltage detector.<br>• This bit is initialized to "1" (low voltage detector active) by an External and a Power (power-on) reset only.<br>• Do not set this bit to "0" as long as LVRE is set to "1".<br>• The low voltage detector is enabled by setting this bit to "1". However this circuit needs a stabilization time after activation as specified in the datasheet. Do not activate the Low voltage reset function before this time has elapsed.<br>• In devices in which the low voltage reset circuit is always enabled in the Internal vector mode, the setting of this bit has no effect for Mode pins set to "011" (Persistent Low Voltage Reset feature). |
| bit 3 | CSDRE:<br>Clock Stop Detection Reset Enable bit | • This bit enables the Clock stop detection reset function.<br>• This bit can only be written when the System clock 1 is set to RC clock.<br>• Setting this bit to "1" enables the Clock stop detection reset.<br>• Writing "0" to this bit disables the Clock stop detection reset.<br>• This bit is initialized to "0" (Clock stop detection reset disabled) by any reset.<br>• After activation, a Clock stop detection reset is asserted in the following three cases:<br>1. When a missing Main clock is detected while the Main or PLL clock is selected for the System clocks CLKS1 or CLKS2 or the Watchdog timer.<br>2.When a missing Sub clock is detected while the Sub clock is selected for the System clocks CLKS1 or CLKS2 or the Watchdog timer.<br>3.When "0" is written to the CKSR: RCE bit to disable the RC oscillator. |
| bit 4 | MCSDI:<br>Main Clock Stop Detection Interval select bit | • This bit controls the measurement interval of the Main clock stop detection circuit.<br>• This bit is initialized to "0" by any reset.<br>• Writing "0" to this bit sets the interval time to 6 - 8 RC clock cycles.<br>• Writing "1" to this bit sets the interval time to 3 - 4 RC clock cycles.<br>• The Main clock stop detection circuit sets the Main clock missing flag (RCCSR: MCMF) if no rising edge of the Main clock input signal (CLKMC) was observed within the selected interval time. |
| bit 5 | SCSDI:<br>Sub Clock Stop Detection Interval select bit | • This bit controls the measurement interval of the Sub clock stop detection circuit.<br>• This bit is initialized to "0" by any reset.<br>• Writing "0" to this bit sets the interval time to 384 - 512 RC clock cycles.<br>• Writing "1" to this bit sets the interval time to 24 - 32 RC clock cycles.<br>• The Sub clock stop detection circuit sets the Sub clock missing flag (RCCSR: SCMF) if no rising edge of the Sub clock input signal (CLKSC) was observed within the selected interval time. |
| bit 6 - bit 7 | Reserved | • Always write "0" to these bits.<br>• The read value of these bits is undefined.<br>• Read modify write operations to this register are not affected. |

## 8.5.2 Reset Cause and Clock Status Register (RCCSR/RCCSRC)

The RCCSR/RCCSRC register shows the reset cause and the status of the Main and Sub clock

### *8.5.2.1 Configuration of the Reset Cause and Clock Status Register (RCCSR/RCCSRC)*

The Reset Cause and Clock Status Register (RCCSR/RCCSRC) can be accessed at two addresses. A read access to address $00040B_H$ (RCCSRC) clears all bits of the register after reading while a read access to address $00040D_H$ (RCCSR) doesn't change the status of the register. Writing to the RCCSR/RCCSRC register is ignored. Figure 8-11 shows the configuration of the RCCSR/RCCSRC register and Table 8-9 describes the function of the bits.

Figure 8-11. Configuration of the Reset Cause and Clock Status Register (RCCSR/RCCSRC)

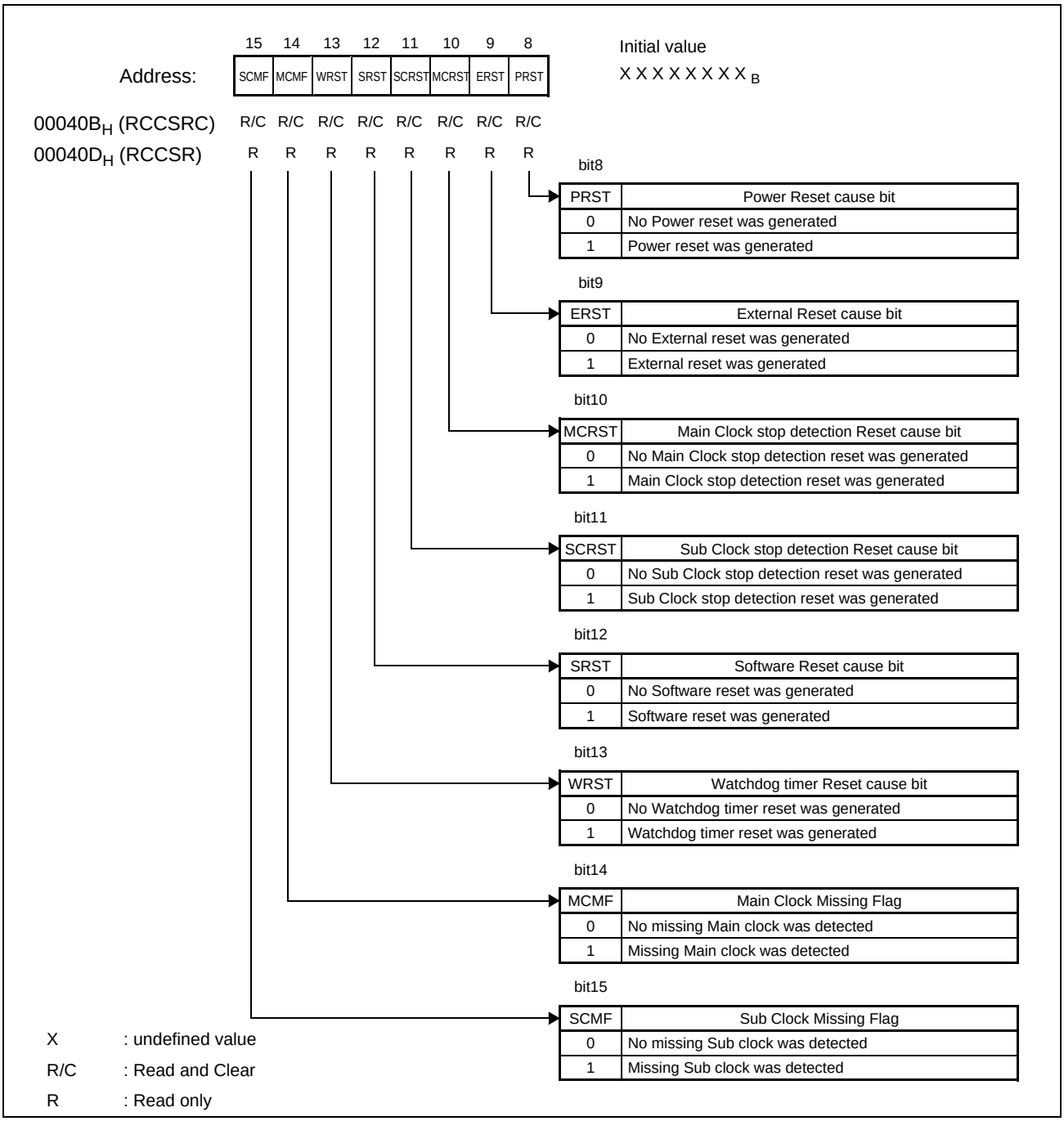


Table 8-9. Function Description of the Bits of the Reset Cause and Clock Status Register (RCCSR)

| Bit name | | Function |
|---|---|---|
| bit 8 | PRST:<br>Power Reset cause bit | • This bit indicates if a Power reset was generated.<br>• After Power-on, this bit is set to "1".<br>• This bit can only be cleared by a read access to RCCSRC at address $00040B_H$.<br>• This bit is set to "1" if a Power reset was generated. Possible reset causes are:<br>1. Power-on event: The power was turned on according to the profile described in the datasheet.<br>2. Low voltage event: The low voltage reset function was enabled and the power supply was below a value described in the datasheet. |
| bit 9 | ERST:<br>External Reset cause bit | • This bit indicates if an External reset was generated.<br>• This bit is set to "1" if an External reset was generated by asserting RSTX to "0".<br>• After a Power reset, this bit is also set to '1'.<br>• This bit can only be cleared by a read access to RCCSRC at address $00040B_H$. |
| bit 10 | MCRST:<br>Main Clock stop detection Reset cause bit | • This bit indicates if a Main Clock stop detection reset was generated.<br>• After a Power reset, this bit is not initialized.<br>• This bit can only be cleared by a read access to RCCSRC at address $00040B_H$.<br>• This bit is set to "1" if a Main Clock stop detection reset was generated. Possible reset causes are:<br>1. A missing Main clock was detected while the Main or the PLL clock was selected for the System clock CLKS1 or CLKS2 or the Watchdog timer.<br>2. "0" was written to the enable bit of the RC clock (CKSR: RCE) although the Clock stop detection reset was enabled. |
| bit 11 | SCRST:<br>Sub Clock stop detection Reset cause bit | • This bit indicates if a Sub Clock stop detection reset was generated.<br>• After a Power reset, this bit is not initialized.<br>• This bit can only be cleared by a read access to RCCSRC at address $00040B_H$.<br>• This bit is set to "1" if a Sub Clock stop detection reset was generated. Possible reset causes are:<br>1. A missing Sub clock was detected while the Sub clock was selected for the System clock CLKS1 or CLKS2 or the Watchdog timer.<br>2. "0" was written to the enable bit of the RC clock (CKSR: RCE) although the Clock stop detection reset was enabled. |
| bit 12 | SRST:<br>Software Reset cause bit | • This bit indicates if a Software reset was generated.<br>• After a Power reset, this bit is not initialized.<br>• This bit can only be cleared by a read access to RCCSRC at address $00040B_H$.<br>• This bit is set to "1" if a Software reset was generated by writing "1" to the RCR: SRSTG bit. |
| bit 13 | WRST:<br>Watchdog timer Reset cause bit | • This bit indicates if a Watchdog timer reset was generated.<br>• After a Power reset, this bit is not initialized.<br>• This bit can only be cleared by a read access to RCCSRC at address $00040B_H$.<br>• This bit is set to "1" if a Watchdog timer reset was generated. Possible reset causes are:<br>1. The Watchdog timer was activated but not cleared within the selected interval.<br>2. An illegal value was written to the Watchdog Timer Clear Pattern register.<br>3. "0" was written to the enable bit of the clock which is used as source clock of the Watchdog timer (CKSR: RCE, MCE or SCE) to disable this clock.<br>4. RC clock frequency was changed although Watchdog Timer Clock Selection bits were set to "01" (changing RC clock frequency not allowed).<br>5. Transition to Stop mode was requested although the WDTC:RSTP bit was set to '1'. |
| bit 14 | MCMF:<br>Main Clock Missing Flag | • This bit indicates if a missing Main clock was detected.<br>• After a Power reset, this bit is not initialized.<br>• This bit is set to "1" if the Main oscillator is enabled and no rising edge of the Main clock input signal (CLKMC) was observed within the interval time defined by the RCR: MCSDI bit.<br>• This bit is cleared by a read access to RCCSRC at address $00040B_H$. |
| bit 15 | SCMF:<br>Sub Clock Missing Flag | • This bit indicates if a missing Sub clock was detected.<br>• After a Power reset, this bit is not initialized.<br>• This bit is set to "1" if the Sub oscillator is enabled and no rising edge of the Sub clock input signal (CLKSC) was observed within the interval time defined by the RCR: SCSDI bit.<br>• This bit is cleared by a read access to RCCSRC at address $00040B_H$. |

### 8.5.2.2 Notes about reset cause bits

#### Multiple reset causes before reading reset cause register

When multiple reset causes are generated before the reset cause register is read out, the corresponding reset cause bits of the RCCSR/RCCSRC register are all set to "1". If, for example, an External reset request via the RSTX pin and the Watchdog timer reset occur at the same time, the ERST and the WRST bits are both set to "1". The following table show this correspondence.

Table 8-10. Correspondence between reset cause bits and reset causes

| Reset cause | PRST | ERST | MCRST | SCRST | SRST | WRST |
|---|---|---|---|---|---|---|
| Power reset (Power-on or low voltage) | 1 | 1 | X | X | X | X |
| External reset request via RSTX pin | * | 1 | * | * | * | * |
| Main clock stop detection reset | * | * | 1 | * | * | * |
| Sub clock stop detection reset | * | * | * | 1 | * | * |
| Software reset request | * | * | * | * | 1 | * |
| Watchdog timer overflow | * | * | * | * | * | 1 |

*: Previous state maintained

X: Undefined

#### Power reset

For PRST = ’1’, the software should be programmed so that it will ignore all other reset cause bits.

#### Clearing the reset cause bits

The reset cause bits are cleared only when reading the Reset Cause and Clock Status Register RCCSRC at address $00040B_H$. Any bit corresponding to a reset cause that has already been generated is not cleared when another reset is generated (a setting of "1" is retained).

After a Power reset, the register should be cleared by reading in order to initialize all bits.

#### Note:

If the power is turned on under conditions where power-on reset may not occur (power-on profile as specified in the datasheet not met), the value in RCCSR register is not guaranteed.

### 8.5.2.3 Notes about clock missing flags

For details about the clock missing flags MCMF and SCMF refer to section 8.6 Operation of the Clock stop detection function and reset.

## 8.5.3 Clock Input and LVD Control Register (CILCR)

The Clock Input and LVD Control Register (CILCR) is used to control additional functions of the oscillator circuit and the Low Voltage Detection Level.

#### Configuration of the Clock Input and LVD Control Register (CILCR)

Figure 8-12 shows the configuration of the Clock Input and LVD Control Registers (CILCR) and Table 8-11 describes the function of each bit.

The register can be accessed 16-bit wide (VRCR) and 8-bit wide (low byte: VRCR, high byte CILCR).

Figure 8-12. Configuration of the Clock Input and LVD Control Register (CILCR)

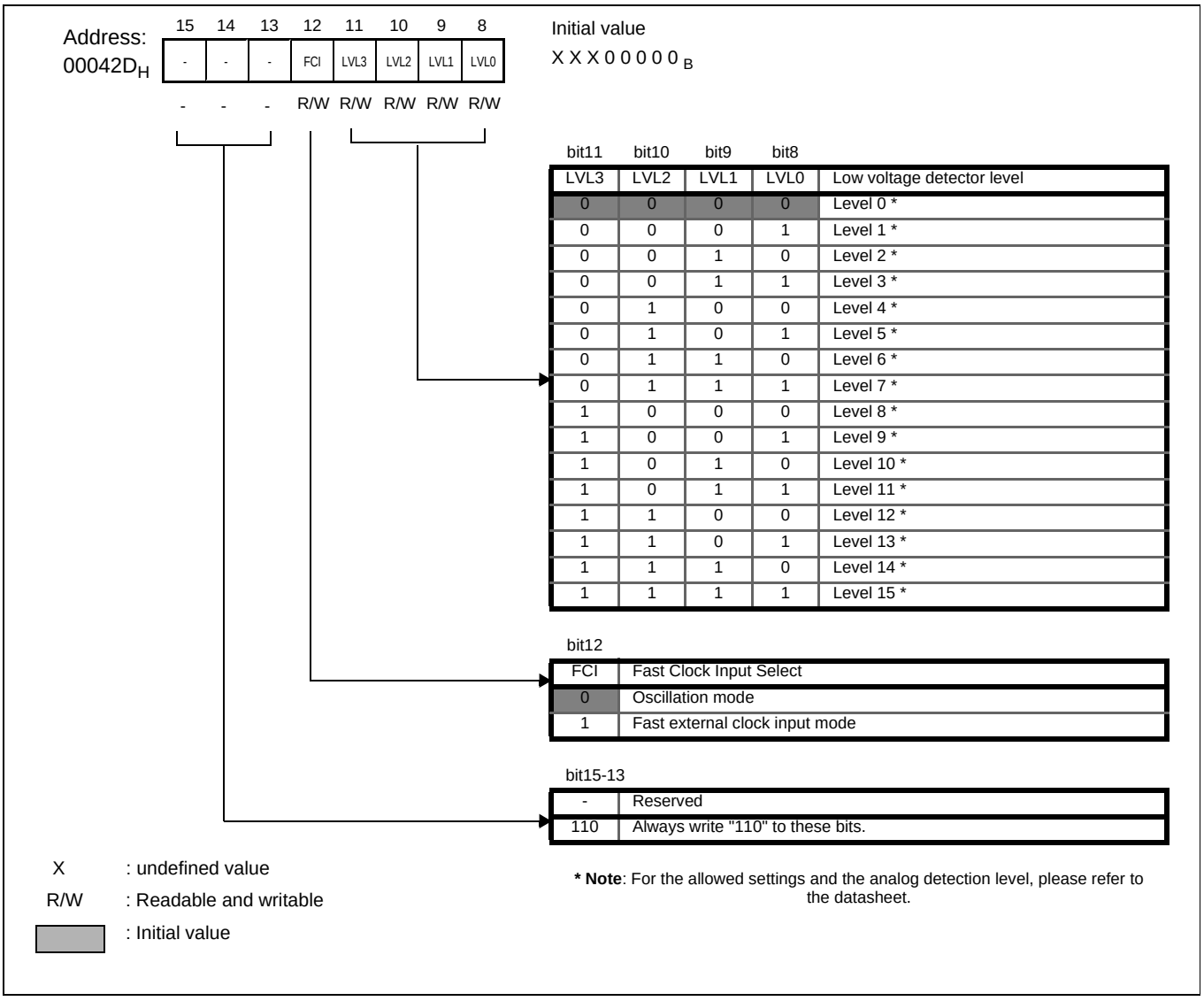


Table 8-11. Function Description of Each Bit of the Clock Input and LVD Control Register (CILCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 8 - bit 11</td><td>LVL0 to LVL3:<br>Low Voltage Detector Level Select bits</td><td>
• These bits control the analog threshold level for the Low Voltage Detector (LVD):
<table>
<tr><th>LVL3</th><th>LVL2</th><th>LVL1</th><th>LVL0</th><th>Low voltage detector level</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Level 0 *</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>Level 1 *</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>Level 2 *</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>Level 3 *</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>Level 4 *</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>Level 5 *</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>Level 6 *</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>Level 7 *</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>Level 8 *</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>Level 9 *</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>Level 10 *</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>Level 11 *</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>Level 12 *</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>Level 13 *</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>Level 14 *</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>Level 15 *</td></tr>
</table>
• These bits are initialized to "0000" by each reset.<br>
• The Low Voltage Detector circuit will issue a reset request when this function is enabled and the external supply voltage drops below the selected threshold value.<br><br>
* Note: For the allowed settings and the analog threshold level, please refer to the datasheet.
</td></tr>
<tr><td>bit 12</td><td>FCI</td><td>
• This bit is initialized to "0" by each reset.<br>
• This bit selects the oscillation or Fast Clock Input mode of the Main Oscillator.
<table>
<tr><th>FCI</th><th>Fast Clock Input Control</th></tr>
<tr><td>0</td><td>Oscillation mode</td></tr>
<tr><td>1</td><td>Fast External Clock Input mode</td></tr>
</table>
• This bit must be set to "0" when connecting an oscillator (crystal/resonator) to the Main Oscillator Pin X0 and X1. It is also possible to connect an external clock with low frequency to the oscillator X0 Pin in this mode.<br>
• For inputting an external clock with higher frequency, this bit must be set to "1" before the device is switched to external (main) clock. For more details about clocks refer to Clocks on page 119<br>
• The frequency ranges for both settings are described in the datasheet.<br>
• This feature is not available on all devices. See datasheet for more details.
</td></tr>
<tr><td>bit 13 - bit 15</td><td>Reserved</td><td>
• Always write "110" to these bits.<br>
• The read value of these bits is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
</table>

# 8.6 Operation of the Clock stop detection function and reset

This section describes the operation of the clock stop detection circuit that detects a failure of the external Main or Sub oscillator.

## 8.6.1 Function of the clock stop detection circuit

The clock stop detection circuit observes the oscillation of the Main and Sub clock. If no rising edge of the observed clock was detected within a selected interval although the clock was enabled, then the corresponding clock missing flag is set. This function is always active when the RC oscillator is enabled.

An additional Clock stop reset can be asserted if the failed clock was currently used for the System clocks CLKS1 or CLKS2 or the Watchdog timer to avoid a hang-up of the MCU. After such a reset the MCU always changes to RC clock mode where the CPU can check the reset cause and put the MCU into a safe state.

## 8.6.2 Configuration of the clock stop detection circuit

The clock stop detection circuit has 3 configuration bits, one for activating the clock stop reset function and two for selecting the detection interval.

### Clock Stop Detection Reset Enable bit (RCR: CSDRE)

The initial value of the CSDRE bit after any reset is "0" (Clock stop detection reset disabled). Setting this bit to "1" activates the clock stop detection reset function and writing "0" disables the function. However writing to the CSDRE bit is possible only when the System clock 1 (CLKS1) is set to RC clock.

### Clock Stop Detection Interval select bits (RCR: MCSDI and SCSDI)

A missing clock is detected when no rising edge of the observed clock was detected within a certain interval time. This interval is a certain number of RC clock cycles and therefore depends on the selected RC clock frequency. Two bits allow a selection of this interval to adjust the observation time to the selected RC clock frequency and the observed external clock frequency.

Table 8-12. Clock stop detection interval

| Observed clock | Select bit | Setting | Clock Stop Detection Interval | | |
|---|---|---|---|---|---|
| | | | RC clock cycles | Time for RC clock frequency of 2MHz (min - max) / (minimum external frequency) | Time for RC clock frequency of 100kHz (min - max) / (minimum external frequency) |
| Main clock | MCSDI | 0 (initial value) | 6 - 8 | 1.5µs - 8µs / (~700kHz) | 30µs - 160µs / (~35kHz) |
| | | 1 | 3 - 4 | 0.75µs - 4µs / (~1.4MHz) | 15µs - 80µs / (~70kHz) |
| Sub clock | SCSDI | 0 (initial value) | 384 - 512 | 96µs - 512µs / (~11kHz) | 1.92ms - 10.2ms / (~0.55kHz) |
| | | 1 | 24 - 32 | 6µs - 32µs / (~170kHz) | 12µs - 640µs / (~8.5kHz) |

The minimum time specifies after which time between two rising edges of the observed clock a stop detection may occur. The maximum time specifies after which time a clock stop will be detected. Use a setting with a minimum time that is longer than the cycle time of the observed clock (the observed clock frequency must be higher than the minimum external frequency given in Table 8-12).

Changing the RC clock frequency during operation of the clock stop detect function is allowed but changes the detection interval.

### 8.6.3 Clock missing flags MCMF and SCMF

The Main Clock Missing Flag MCMF indicates a missing Main clock and the Sub Clock Missing Flag SCMF indicates a missing Sub clock.

#### Status after power-on

The clock missing flags MCMF and SCMF are undefined after startup.

#### Clearing the clock missing flags

The MCMF and SCMF bits can be cleared by a read access to the RCCSRC register at address $00040B_H$.

#### Setting the clock missing flags

The Main clock stop detection circuit is enabled when the RC clock is running and the Main oscillator is active (active means the Main Clock Enable bit CKSR: MCE is "1" or the Main Clock Monitor bit CKMR: MCM is "1"). The Main Clock Missing Flag is set to "1" when no rising edge of the Main clock input signal was detected within the selected interval.

The Sub clock stop detection circuit is enabled when the RC clock is running and the Sub oscillator is active (active means the Sub Clock Enable bit CKSR: SCE is "1" or the Sub Clock Monitor bit CKMR: SCM is "1"). The Sub Clock Missing Flag is set to "1" when no rising edge of the Sub clock input signal was detected within the selected interval.

### 8.6.4 Clock stop detection reset

The Clock stop detection reset function generates a reset if a clock failure is detected while this clock is used as source for the System clocks (CLKS1 or CLKS2) or the Watchdog timer.

#### Activation of the Clock stop detection reset

The Clock stop detection reset is enabled by setting the RCR: CSDRE bit to "1".

#### Generating a Clock stop detection reset

A Clock stop detection reset is generated in the following cases. The Clock Stop Detection Reset Enable bit CSDRE must always be set to "1" (reset enabled).

- When a missing Main clock is detected while the Main clock is stabilized (CKMR:MCM = ’1’) and the Main or PLL clock is used for the System clocks CLKS1 or CLKS2 or the Watchdog timer. The MCRST (Main clock stop detection Reset cause) bit is set and a reset is generated.
- When a missing Sub clock is detected while the Sub clock is stabilized (CKMR:SCM = ’1’) and used for the System clocks CLKS1 or CLKS2 or the Watchdog timer. The SCRST (Sub clock stop detection Reset cause) bit is set and a reset is generated.
- When a "0" is written to the CKSR: RCE bit (disable RC clock). Both clock stop detection reset cause bits are set and a reset is generated. Do not disable the RC clock as long as the Clock stop detection reset is enabled.

#### Note:

A clock stop detection reset is generated only when the failing clock is used for the System clock 1 or 2 or for the Watchdog timer. In case a failing clock is only used by a resource (for example Main clock timer or Sub clock timer) while the System clocks are set to another clock, no reset is generated. Use the Watchdog timer for recovery in case this can lead to hang-up of the system.

#### Effect of a Clock stop detection reset

A Clock stop detection reset asserts the global reset signal, sets the corresponding Clock stop detection reset cause bit and puts the MCU into reset state.

The source clock timer and clock ready monitor bit of the failed clock is also cleared by the Clock stop detection reset.

After reset release the MCU restarts in RC clock mode and the reset cause bits can be read. It is recommended to disable the clock that caused the reset by setting the MCE or SCE bit to "0".

The contents of internal RAM memory and all registers which are not reset (initial value 'X') cannot be guaranteed after a Clock stop detection reset.

### 8.6.5 Clock stop detection reset and Standby modes

#### Sleep and Timer mode

The clock stop detection reset function is active in Sleep and Timer mode. This means a Main Clock stop reset can be asserted in Run, Sleep and Timer mode if System clock CLKS1 or CLKS2 is set to Main or PLL clock mode (as indicated by the CKMR register). A Sub Clock stop reset can be asserted in Run, Sleep and Timer mode if System clock CLKS1 or CLKS2 is set to Sub clock mode.

#### Stop mode

All oscillators including the RC oscillator are disabled in Stop mode, hence no clock stop detection is possible.

If an interrupt is asserted in Stop mode, then the MCU changes to Run mode with the clocks specified in the CKSR register. The transition to Run mode is delayed until the clock selected with the CKSR: SC1S[1:0] bits is stabilized. The operation of the System Clock 2 is delayed until the clock selected with the CKSR: SC2S[1:0] bits is stabilized.

#### Note:

The Clock stop detection reset function is disabled during these stabilization times to avoid an accidental assertion of the reset during clock stabilization. This means there will be no Clock stop detection reset if the clock failed during Stop mode or within the stabilization time of the oscillator.

Always set BOTH system clocks to RC clock (CKSR: SC1S[1:0] and SC2S[1:0] = "00") when changing to Stop mode to avoid this situation. Changing CLKS1 or CLKS2 to Main, PLL or Sub clock after start in RC clock mode will not be executed if the corresponding clock did not restart and hence could not set its clock ready monitor bit. The software should detect this situation by a time-out condition using the RC clock timer.

### 8.6.6 Clock stop detection reset and Watchdog timer

A Clock stop detection reset is also asserted when the Main or Sub clock fails while used as source for the Watchdog timer. During clock stabilization (after Stop mode release) however this reset function is disabled.

## 8.7 Operation of the low voltage reset function

This section describes the operation of the low voltage reset function.

### 8.7.1 Function of the low voltage reset

The low voltage reset function is using the low voltage detector that compares the power supply voltage $V_{CC}$ with an internally generated reference voltage. This reference voltage is programmable with the CILCR:LVL[3:0] bits.

If the reset function and the low voltage detector are enabled by the Low Voltage Reset Enable (LVRE) and the Low Voltage Detector Enable (LVDE) bits and $V_{CC}$ is below the selected detection level $V_{DL}$, then the Power reset extension counter is initialized and the PRST flag is set. See the datasheet for the specification of the selectable detection levels $V_{DL}$.

After recovery of the power supply voltage, the Power reset extension counter is released and the startup performed as described in section 8.3 Startup after Power and External reset.

### 8.7.2 Configuration of the low voltage detection and reset circuit

#### Low voltage reset

The low voltage reset is controlled by the Low Voltage Reset Enable (LVRE) bit. Setting this bit to "0" disables the low voltage reset function and setting the bit to "1" enables the function. After Power-on or when asserting RSTX, this bit is always set to "1" (reset function enabled).

The LVRE bit must be set to "0" when the low voltage detector should be disabled. When the low voltage detector is switched on, the low voltage detector stabilization time must be applied before setting LVRE to "1". Otherwise a low voltage reset could be asserted wrongly because the low voltage detector outputs a wrong value. See the datasheet for the specification of the low voltage detector stabilization time.

In devices in which the low voltage reset circuit is always enabled in the internal vector mode, the setting of the LVRE bit has no effect for Mode pins set to "011".

### Low voltage detector

The low voltage detector is controlled by the Low Voltage Detector Enable (LVDE) bit and the CILCR:LVL level select bits. Setting the LVDE bit to "0" disables the low voltage detector and setting the bit to "1" enables the detector. After Power-on or when asserting RSTX, this bit is always set to "1" (low voltage detector enabled). The CILCR:LVL level select bits are reset to "0000" (Level 0) by any reset.

When LVDE is set to "0", the LVRE bit must also be set to "0" at the same time or before. When the low voltage detector is switched on, the low voltage detector stabilization time must be applied before setting LVRE to "1".

In devices in which the low voltage reset circuit is always enabled in the internal vector mode, the setting of the LVDE bit has no effect for Mode pins set to "011" (low voltage detector is always active).

The detection level is always set to "Level 0" after reset. With this level, a reset can be generated when $V_{CC}$ falls below the minimum value required for safe operation of the MCU. This level can be increased after startup by setting the CILCR:LVL bits. This allows the generation of a Low Voltage reset already at higher $V_{CC}$ levels in case this is required by the system.

### Effect on current consumption

The low voltage detector draws a current when it is activated (see datasheet for details). This current flows independent of the selected operation mode. If this is not acceptable when using standby modes, disable the low voltage detector and reset before changing to standby mode. Please be aware that the low voltage reset function will not be available in this case.

In devices in which the low voltage reset circuit is always enabled in the internal vector mode, it is not possible to disable the low voltage detector and reset when Mode pins are set to "011".

### Restrictions when using the low voltage reset function

- After enabling the low voltage detector, the output value is undefined during the low voltage detector stabilization time. Hence the low voltage reset must not be used within this time.
- The low voltage reset function has restrictions regarding the AC specification as described in the datasheet.
- For devices or operation modes that allow disabling the low voltage reset function with the LVRE and LVDE bits, it is not possible to use the low voltage reset function to guarantee a safe startup after power-on. Either a Power-on reset or External reset as specified in the datasheet must be applied for starting the MCU in this case.

# 9. Standby Mode and Voltage Regulator Control Circuit

This chapter explains the functions and operations of the standby mode control circuit and the control of the internal voltage regulator.
The regulator control can be used to optimize power consumption, especially in standby modes.

9.1 Overview of the CPU Operating Modes

9.2 Standby Mode Control Register (SMCR)

9.3 Voltage Regulator Control Register (VRCR)

9.4 Standby Modes

9.5 Mode Change Table and operation status

9.6 Usage Notes on Standby Mode

9.7 Voltage Regulator Operation

## 9.1 Overview of the CPU Operating Modes

The $F^2$MC-16FX MCU has the following CPU operating modes:

- Run mode
- Sleep mode
- Timer mode
- Stop mode

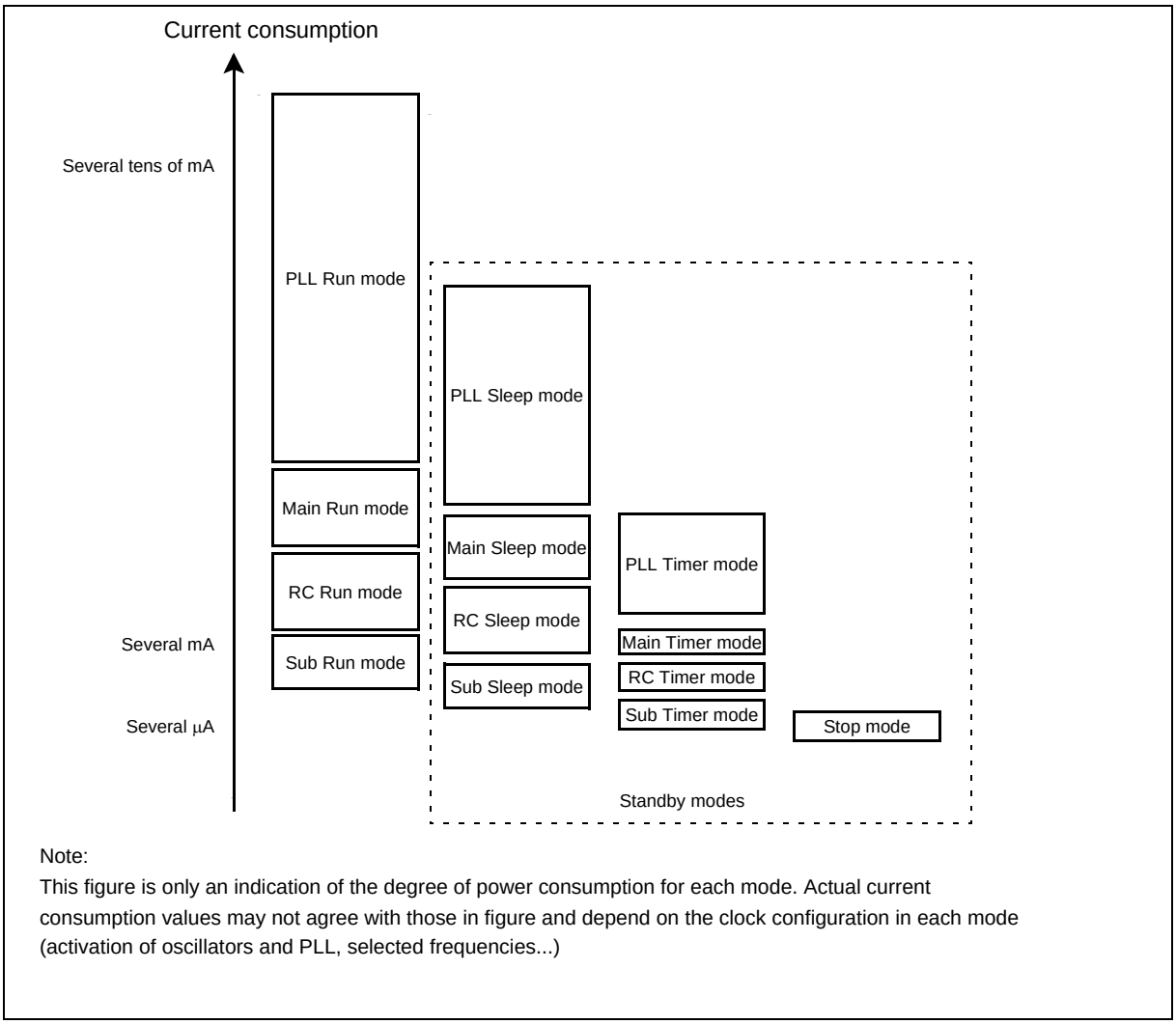
Sleep mode, Timer mode and Stop mode are called Standby modes. In Stop mode, all clocks are disabled. In the other CPU operating modes, the Clock source and active oscillators/PLL are selectable.

### 9.1.1 CPU operating modes and current consumption

Figure 9-1 shows the relationship between the CPU operating modes and current consumption.

The power consumption of the operating modes can be influenced by the settings of the internal voltage regulator.

Figure 9-1. CPU operating mode and current consumption

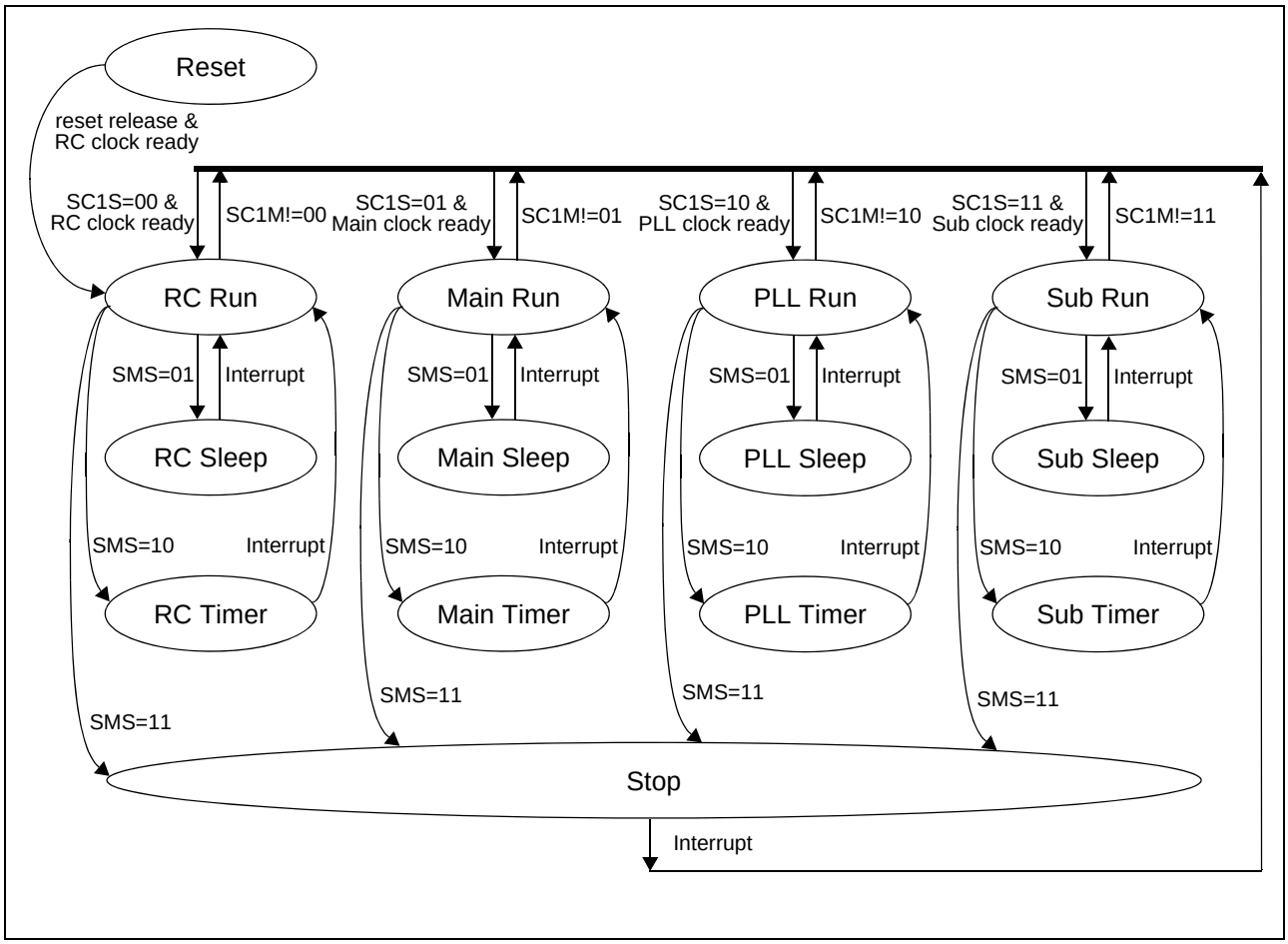



Note:
This figure is only an indication of the degree of power consumption for each mode. Actual current consumption values may not agree with those in figure and depend on the clock configuration in each mode (activation of oscillators and PLL, selected frequencies...)

## 9.1.2 Transitions between different operating modes

The mode change diagram in Figure 9-2 shows possible transitions between the different operating modes.

Figure 9-2. Mode change diagram




## 9.1.3 Run Mode

The Run mode is always defined by the clock selected as source for the System clock 1 (CLKS1). The System clock 2 (CLKS2) which is selected by the System clock 2 selector is independent of this Run mode.

Hence modules connected to the Peripheral clock 2 (CLKP2) can operate with a different clock source because the Peripheral clock 2 is derived from the System clock 2.

### RC Run mode

In this mode, the internal RC oscillation clock CLKRC is used for the System clock 1, operating the CPU and most peripheral functions. Two frequency settings are possible for the RC oscillator (nominal 2MHz or 100kHz). Independent clock dividers for the Bus clock operating the CPU (CLKB) and the Peripheral clock 1 operating most peripheral functions (CLKP1) are fed by the RC clock and allow flexible frequency settings.

The status of the main oscillator, the PLL multiplier circuit and the sub oscillator depend on the setting of the MCE, PCE and SCE bits when they are not used as System clock 2.

This mode is always active after reset release.

### Main Run mode

In this mode, the Main oscillation clock CLKMC is used for the System clock 1, operating the CPU and most peripheral functions. Independent clock dividers for the Bus clock operating the CPU (CLKB) and the Peripheral clock 1 operating most peripheral functions (CLKP1) are fed by the Main clock and allow flexible frequency settings.

The status of the PLL multiplier circuit, the RC and sub oscillator depend on the setting of the PCE, RCE and SCE bits when they are not used as System clock 2.

### PLL Run mode

In this mode, the PLL clock that is a multiple of the Main oscillation clock is used for the System clock 1, operating the CPU and most peripheral functions. Independent clock dividers for the Bus clock operating the CPU (CLKB) and the Peripheral clock 1 operating most peripheral functions (CLKP1) are fed by the PLL output clock and allow flexible frequency settings. Either the unmodulated (CLKPLL) or modulated (CLKMOD) PLL clock can be used.

The status of the RC and sub oscillator depend on the setting of the RCE and SCE bits when they are not used as System clock 2.

### Sub Run mode

In this mode, the Sub oscillation clock CLKSC is used for the System clock 1, operating the CPU and most peripheral functions. Independent clock dividers for the Bus clock operating the CPU (CLKB) and the Peripheral clock 1 operating most peripheral functions (CLKP1) are fed by the Sub clock and allow flexible frequency settings.

The status of the main oscillator, the PLL multiplier circuit and the RC oscillator depend on the setting of the MCE, PCE and RCE bits when they are not used as System clock 2.

**Reference:** For more details, see Clocks on page 119 and Clock Modulator on page 147.

## 9.1.4 Sleep Mode

In this mode, the standby control circuit stops the Bus clock (CLKB) what disables the CPU, internal memories, the DMA controller and the external bus interface, thereby reducing power consumption.

### RC Sleep mode

The RC Sleep mode is activated to stop the CLKB clock in the RC clock mode. Resources connected to the Peripheral clock 1 operate on the RC clock.

Resources connected to the Peripheral clock 2 are operating with the clock selected by the System clock 2 selector.

The status of the main oscillator, the PLL multiplier circuit and the sub oscillator depend on the setting of the MCE, PCE and SCE bits when they are not used as System clock 2.

### Main Sleep mode

The Main Sleep mode is activated to stop the CLKB clock in the Main clock mode. Resources connected to the Peripheral clock 1 operate on the Main clock.

Resources connected to the Peripheral clock 2 are operating with the clock selected by the System clock 2 selector.

The status of the PLL multiplier circuit, the RC and Sub oscillator depend on the setting of the PCE, RCE and SCE bits when they are not used as System clock 2.

### PLL Sleep mode

The PLL Sleep mode is activated to stop the CLKB clock in the PLL clock mode. Resources connected to the Peripheral clock 1 operate on the PLL clock.

Resources connected to the Peripheral clock 2 are operating with the clock selected by the System clock 2 selector.

The status of the RC and Sub oscillator depend on the setting of the RCE and SCE bits when they are not used as System clock 2.

### Sub Sleep mode

The Sub Sleep mode is activated to stop the CLKB clock in the Sub clock mode. Resources connected to the Peripheral clock 1 operate on the Sub clock.

Resources connected to the Peripheral clock 2 are operating with the clock selected by the System clock 2 selector.

The status of the Main oscillator, the PLL multiplier circuit and the RC oscillator depend on the setting of the MCE, PCE and RCE bits when they are not used as System clock 2.

### 9.1.5 Timer Mode

In this mode, the standby control circuit stops supplying the System clocks CLKS1 and CLKS2 what further reduces power consumption by stopping the Bus clock and also all Peripheral clocks. This stops all functions, excluding oscillators, the PLL and the corresponding source clock timers.

#### RC Timer mode

In RC Timer mode, the RC oscillator and RC clock timer is always active. The status of the PLL, the Main and Sub oscillator and corresponding source clock timers however depend on the setting of the PCE, MCE and SCE bits.

#### Main Timer mode

In Main Timer mode, the Main oscillator and Main clock timer is always active. The status of the PLL, the RC and Sub oscillator and corresponding source clock timers however depend on the setting of the PCE, RCE and SCE bits.

#### PLL Timer mode

In PLL Timer mode, the Main oscillator, the PLL and the Main clock timer is always active. The status of the RC and Sub oscillator and corresponding source clock timers however depend on the setting of the RCE and SCE bits.

#### Sub Timer mode

In Sub Timer mode, the Sub oscillator and Sub clock timer is always active. The status of the PLL, the Main and RC oscillator and corresponding source clock timers however depend on the setting of the PCE, MCE and RCE bits.

#### Note:

The different Timer modes may have the same behavior if more than one of the PCE, MCE, SCE and RCE bits is set to "1". However, the run mode that the device transits to, upon wakeup from timer mode by interrupt, depends on which of the four timer modes the device was in. Refer to Table 9-4.

### 9.1.6 Stop Mode

In this mode, the standby control circuit causes all oscillations to stop. All functions are inactivated.

#### Note:

Because the Stop mode turns all oscillation clocks off, data can be retained by the lowest power consumption.

## 9.2 Standby Mode Control Register (SMCR)

This register switches to standby mode, controls the pin functions in Timer and Stop mode and sets the voltage regulator to Low Power mode.

### Standby mode control register (SMCR)

Figure 9-3 shows the configuration of the standby mode control register (SMCR).

Figure 9-3. Configuration of the standby mode control register (SMCR)

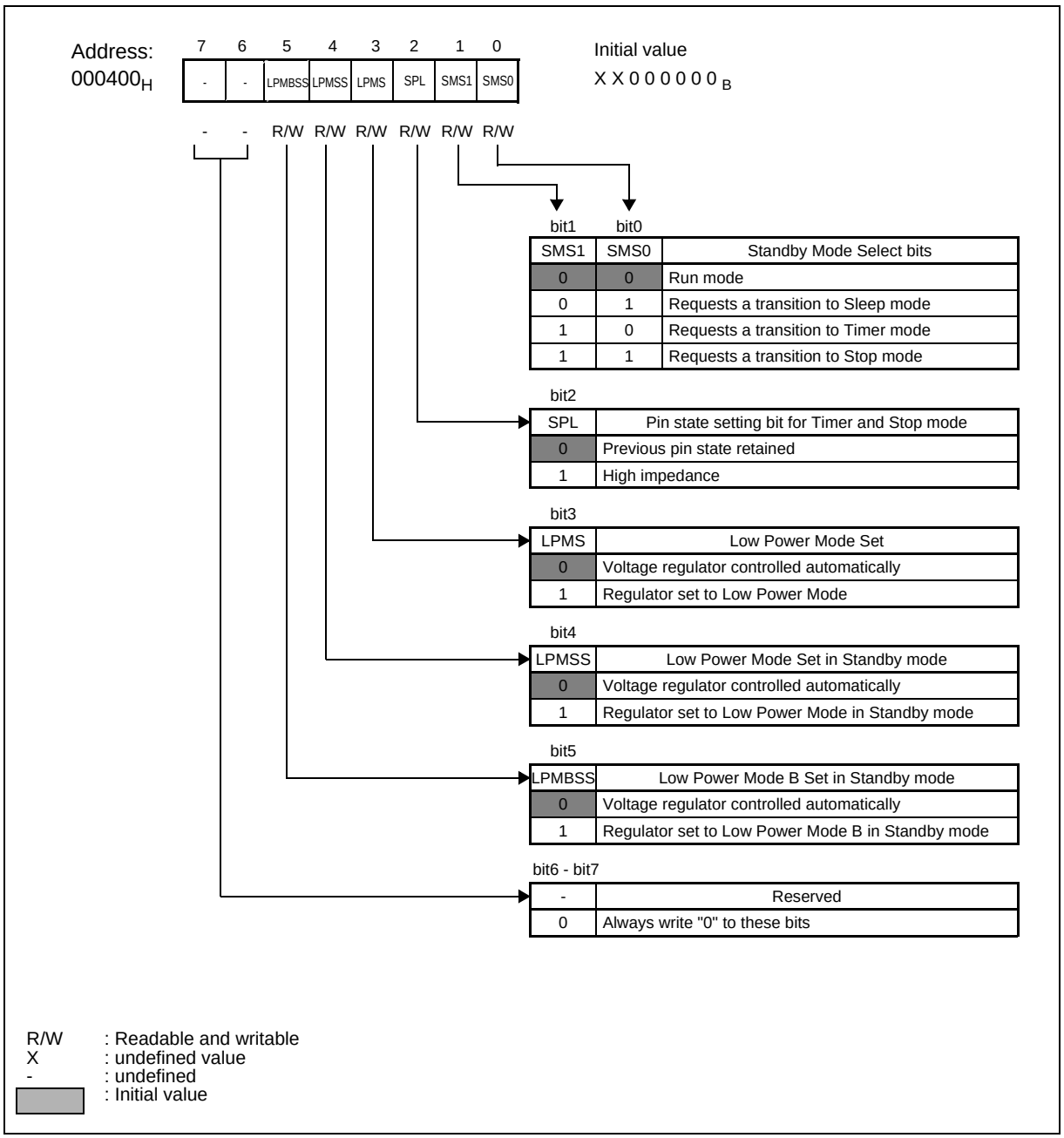


Table 9-1. Function description of each bit of the standby mode control register (SMCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 1</td><td>SMS0 and SMS1:<br>Standby Mode Select bits</td><td>
• These bits request switching to a Standby mode according to the following table:
<table>
<tr><th colspan="2">bit1 bit0</th><th></th></tr>
<tr><td>SMS1</td><td>SMS0</td><td>Standby Mode Select bits</td></tr>
<tr><td>0</td><td>0</td><td>Run mode</td></tr>
<tr><td>0</td><td>1</td><td>Requests a transition to Sleep mode</td></tr>
<tr><td>1</td><td>0</td><td>Requests a transition to Timer mode</td></tr>
<tr><td>1</td><td>1</td><td>Requests a transition to Stop mode</td></tr>
</table>
• When "01" is written to these bits, a switch to Sleep mode is requested.<br>
• When "10" is written to these bits, a switch to Timer mode is requested.<br>
• When "11" is written to these bits, a switch to Stop mode is requested.<br>
• After writing one of these three values, the bits are locked. It is not possible to select another Standby mode or Run mode.<br>
• These bits are cleared to "00" by any pending hardware interrupt request (level smaller than 7), an NMI (if available) and by any reset.<br>
• The read value of these bits shows the status of the Standby mode transition request. Reading a value other than "00" means that the transition to the selected Standby mode is still pending while reading "00" means that there is no Standby mode transition request pending.
</td></tr>
<tr><td>bit 2</td><td>SPL:<br>Pin state setting bit (for Timer and Stop mode)</td><td>
• The setting of this bit effects only the Timer mode and Stop mode.<br>
• When this bit is set to "0", the level of the external pins is retained.<br>
• When this bit is set to "1", the status of the external pins changes to high-impedance.<br>
• This bit is initialized to "0" by a reset.
</td></tr>
<tr><td>bit 3</td><td>LPMS:<br>Low Power Mode Set bit</td><td>
• This bit controls the voltage regulator in any mode. Refer to section 9.7 Voltage Regulator Operation for details.<br>
• This bit is cleared by any reset.<br>
• When this bit is set to "0" (initial value), then the voltage regulator is controlled automatically (regulator in High Power Mode when RC oscillator or Main oscillator is active).<br>
• Setting this bit to "1" forces the voltage regulator immediately to the Low Power Mode A (unless Stop mode is entered where regulator is always set to Low Power Mode B). This means the regulator output voltage is defined by the VRCR:LPMA[2:0] bits.<br>
• Setting this bit is permitted only in certain modes described in section 9.7 Voltage Regulator Operation.<br>
• After changing this bit back to "0", a stabilization time of ~40µs must be applied by the application program before changing to a mode with a higher current consumption.
</td></tr>
<tr><td>bit 4</td><td>LPMSS:<br>Low Power Mode Set bit for Standby modes</td><td>
• This bit controls the voltage regulator in Standby modes (Sleep and Timer mode with RC or Main clock). Refer to section 9.7 Voltage Regulator Operation for details.<br>
• This bit is cleared by any reset.<br>
• When this bit is set to "0" (initial value), then the voltage regulator is controlled automatically (regulator in High Power Mode when RC oscillator or Main oscillator is active).<br>
• Setting this bit to "1" forces the voltage regulator to the Low Power Mode A after entering Sleep or Timer mode. This means the regulator output voltage is defined by the VRCR:LPMA[2:0] bits.<br>
• Setting this bit is permitted only in certain modes described in section 9.7 Voltage Regulator Operation.<br>
• When this bit is set to "1", then a wakeup from Sleep or Timer mode will be automatically delayed by ~40µs for switching the regulator back to High Power Mode.
</td></tr>
<tr><td>bit 5</td><td>LPMBSS:<br>Low Power Mode B Set bit for Standby modes</td><td>
• This bit controls the voltage regulator in Standby modes (Sleep and Timer mode). Refer to section 9.7 Voltage Regulator Operation for details.<br>
• This bit is cleared by any reset.<br>
• When this bit is set to "0" (initial value), then the Low Power Mode B is only used in Stop mode. In all other modes, the regulator is always in High Power mode or in Low Power mode A.<br>
• Setting this bit to "1" forces the voltage regulator to the Low Power Mode B after entering Sleep or Timer mode (if Low Power mode was activated by hardware or LPMS/LPMSS bits). This means the regulator output voltage is defined by the VRCR:LPMB[2:0] bits.<br>
• Setting this bit is permitted only in certain modes described in section 9.7 Voltage Regulator Operation.<br>
• When this bit is set to "1", then a wakeup from Sleep or Timer mode will be automatically delayed by ~40µs for switching the regulator back to High Power Mode or Low Power Mode A<br>
Note:<br>
Do not change this bit, because this function is currently under evaluation by Cypress.
</td></tr>
</table>

Table 9-1. Function description of each bit of the standby mode control register (SMCR)

| Bit name | | Function |
|---|---|---|
| bit 6 - bit 7 | Reserved | • Always write "0" to these bits.<br>• The read value of these bits is undefined.<br>• Read modify write operations to this register are not affected. |

**Note:**

All input pins are automatically disabled in Timer and Stop mode, except activated external interrupt input pins, the NMI input pin and the HRQ hold request input pin of the external bus. The setting of the PIE port input enable bits have no effect in Timer and Stop mode.

## 9.3 Voltage Regulator Control Register (VRCR)

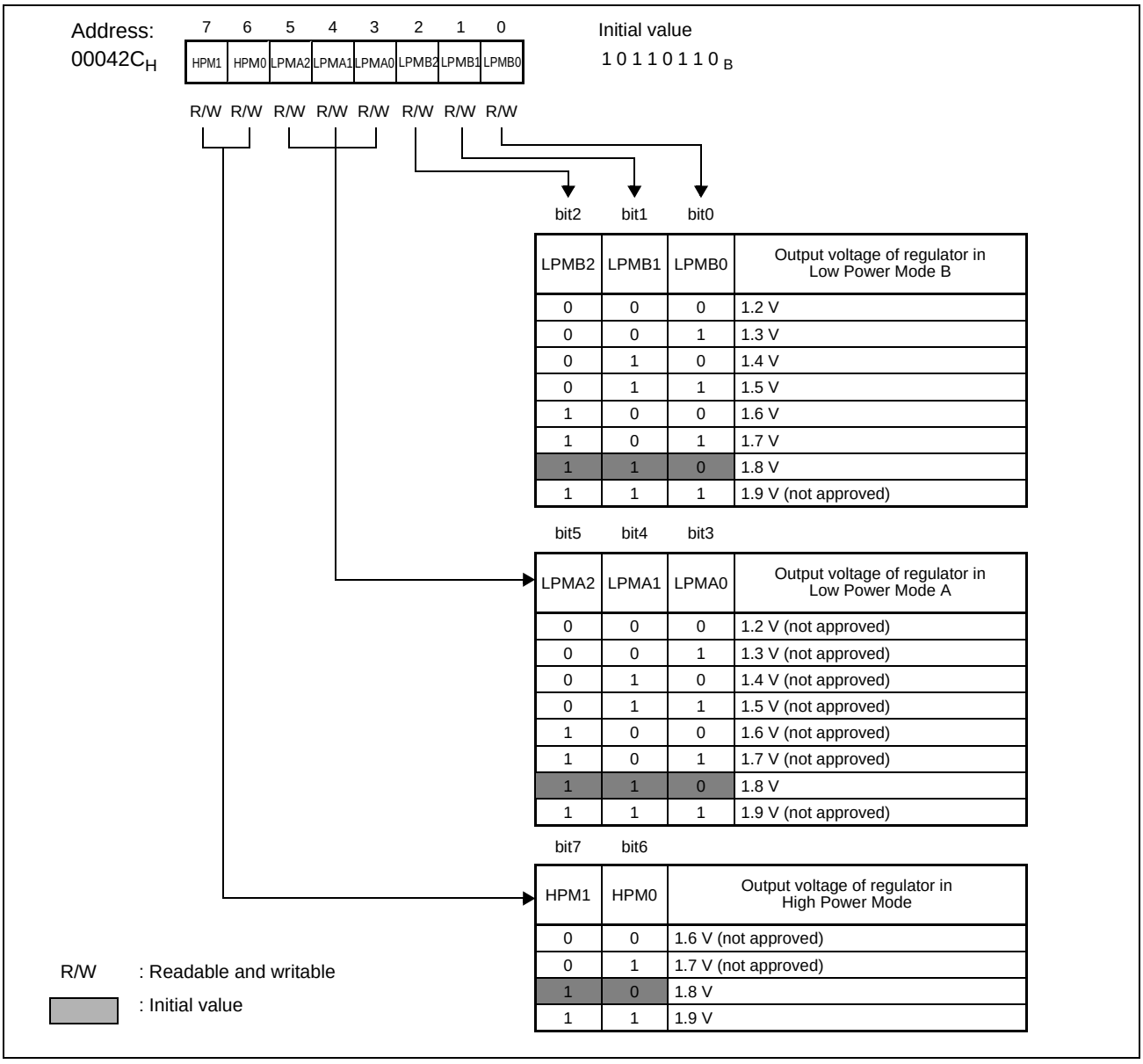
The voltage regulator control register defines the output voltage of the internal Voltage Regulator in different operation modes

### Voltage Regulator Control Register (VRCR)

Figure 9-4 shows the configuration of the Voltage Regulator Control Register (VRCR) and Table 9-2 describes the function of each bit.

Refer to section 9.7 Voltage Regulator Operation for more details.

Figure 9-4. Configuration of the Voltage Regulator Control Register (VRCR)




| LPMB2 | LPMB1 | LPMB0 | Output voltage of regulator in Low Power Mode B |
|---|---|---|---|
| 0 | 0 | 0 | 1.2 V |
| 0 | 0 | 1 | 1.3 V |
| 0 | 1 | 0 | 1.4 V |
| 0 | 1 | 1 | 1.5 V |
| 1 | 0 | 0 | 1.6 V |
| 1 | 0 | 1 | 1.7 V |
| 1 | 1 | 0 | 1.8 V |
| 1 | 1 | 1 | 1.9 V (not approved) |

| LPMA2 | LPMA1 | LPMA0 | Output voltage of regulator in Low Power Mode A |
|---|---|---|---|
| 0 | 0 | 0 | 1.2 V (not approved) |
| 0 | 0 | 1 | 1.3 V (not approved) |
| 0 | 1 | 0 | 1.4 V (not approved) |
| 0 | 1 | 1 | 1.5 V (not approved) |
| 1 | 0 | 0 | 1.6 V (not approved) |
| 1 | 0 | 1 | 1.7 V (not approved) |
| 1 | 1 | 0 | 1.8 V |
| 1 | 1 | 1 | 1.9 V (not approved) |

| HPM1 | HPM0 | Output voltage of regulator in High Power Mode |
|---|---|---|
| 0 | 0 | 1.6 V (not approved) |
| 0 | 1 | 1.7 V (not approved) |
| 1 | 0 | 1.8 V |
| 1 | 1 | 1.9 V |

Table 9-2. Function Description of Each Bit of the Voltage Regulator Control Register (VRCR) (Sheet 1 of 2)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 2</td><td>LPMB0 to LPMB2:<br>Low Power Mode B select bits</td><td>
• These bits select the output voltage of the regulator in Low Power Mode B according to the following table:
<table>
<tr><th></th><th>bit2</th><th>bit1</th><th>bit0</th><th></th></tr>
<tr><th>LPMB2</th><th>LPMB1</th><th>LPMB0</th><th>Output voltage of regulator in Low Power Mode B</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1.2 V</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1.3 V</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1.4 V</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1.5 V</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1.6 V</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1.7 V</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1.8 V</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1.9 V (not approved)</td></tr>
</table>
• Any reset initializes these bits to "110" (1.8V).<br>
• The Low Power Mode B of the regulator is used in Stop mode. It can also be selected with the SMCR:LPMBSS bit under certain conditions.<br>
Note:<br>
The usage of the 1.9V setting is not approved.
</td></tr>
<tr><td>bit 3 - bit 5</td><td>LPMA0 to LPMA2:<br>Low Power Mode A select bits</td><td>
• These bits select the output voltage of the regulator in Low Power Mode A according to the following table:
<table>
<tr><th></th><th>bit5</th><th>bit4</th><th>bit3</th><th></th></tr>
<tr><th>LPMA2</th><th>LPMA1</th><th>LPMA0</th><th>Output voltage of regulator in Low Power Mode A</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1.2 V (not approved)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1.3 V (not approved)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1.4 V (not approved)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1.5 V (not approved)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1.6 V (not approved)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1.7 V (not approved)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1.8 V</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1.9 V (not approved)</td></tr>
</table>
• Any reset initializes these bits to "110" (1.8V).<br>
• The Low Power Mode A of the regulator is used in Sub modes (Sub Run, Sub Sleep and Sub Timer mode) when the Main and RC oscillators are both disabled. It can also be selected with the SMCR:LPMS and LPMSS bits under certain conditions.<br>
Note:<br>
Do not change these bits, because permitted settings are currently under evaluation by Cypress.
</td></tr>
</table>

Table 9-2. Function Description of Each Bit of the Voltage Regulator Control Register (VRCR) (Sheet 2 of 2)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 6 - bit 7</td><td>HPM0 to HPM1:<br>High Power Mode select bits</td><td>
• These bits select the output voltage of the regulator in High Power Mode according to the following table:
<table>
<tr><th></th><th>bit7</th><th>bit6</th><th></th></tr>
<tr><th></th><th>HPM1</th><th>HPM0</th><th>Output voltage of regulator in High Power Mode</th></tr>
<tr><td></td><td>0</td><td>0</td><td>1.6 V (not approved)</td></tr>
<tr><td></td><td>0</td><td>1</td><td>1.7 V (not approved)</td></tr>
<tr><td></td><td>1</td><td>0</td><td>1.8 V</td></tr>
<tr><td></td><td>1</td><td>1</td><td>1.9 V</td></tr>
</table>
• Any reset initializes these bits to "10" (1.8V).<br>
• The High Power Mode of the regulator is the standard operation mode which is used when either the RC or the Main oscillator is enabled (unless the Low Power mode is selected with the SMCR:LPMS or LPMSS bits).<br>
• These bits must be set to "11" (1.9V) together with the FMCS:RD19V bit of the Flash interface before changing to a clock frequency which is permitted only at 1.9V core voltage.<br>
Note:<br>
The usage of the 1.6V and 1.7V settings is not approved.
</td></tr>
</table>

## 9.4 Standby Modes

The standby modes include the Sleep (RC Sleep, Main Sleep, PLL Sleep, Sub Sleep), Timer (RC Timer, Main Timer, PLL Timer, Sub Timer) and Stop modes.

### Operation status during standby mode

Table 9-3 shows the status of the clocks, CPU, Peripherals and external pins for the different standby modes.

Table 9-3. Operation status during standby mode

| Standby mode | | Condition for switch | RC clock | Main clock | PLL clock | Sub clock | Bus clock | Peripheral clock 1 | Peripheral clock 2 | Pins | Release event |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Sleep mode | RC Sleep mode | SC1M=00 SMS=01 | Active | depends on MCE and SC2M bits | depends on PCE, MCE and SC2M bits | depends on SCE and SC2M bits | Stopped | RC clock | depends on SC2M bits | Active | Power, External, Watchdog or Clock stop reset, Interrupt |
| | Main Sleep mode | SC1M=01 SMS=01 | depends on RCE and SC2M bits | Active | depends on PCE and SC2M bits | depends on SCE and SC2M bits | | Main clock | | | |
| | PLL Sleep mode | SC1M=10 SMS=01 | depends on RCE and SC2M bits | Active | Active | depends on SCE and SC2M bits | | PLL clock | | | |
| | Sub Sleep mode | SC1M=11 SMS=01 | depends on RCE and SC2M bits | depends on MCE and SC2M bits | depends on PCE, MCE and SC2M bits | Active | | Sub clock | | | |
| Timer mode | RC Timer mode | SC1M=00 SMS=10 | Active | depends on MCE and SC2M bits | depends on PCE, MCE and SC2M bits | depends on SCE and SC2M bits | | Stopped(*1) | | depends on SPL bit (Hi-Z or last value retained) | |
| | Main Timer mode | SC1M=01 SMS=10 | depends on RCE and SC2M bits | Active | depends on PCE and SC2M bits | depends on SCE and SC2M bits | | | | | |
| | PLL Timer mode | SC1M=10 SMS=10 | depends on RCE and SC2M bits | Active | Active | depends on SCE and SC2M bits | | | | | |
| | Sub Timer mode | SC1M=11 SMS=10 | depends on RCE and SC2M bits | depends on MCE and SC2M bits | depends on PCE, MCE and SC2M bits | Active | | | | | |
| Stop mode | Stop mode | SMS=11 | Stopped | | | | | Stopped | | | Power or External reset, Interrupt |

*1: The RC clock timer, Main clock timer and Sub clock timer operate if the corresponding clock was not stopped

SC1M: System Clock 1 Monitor bits of clock monitor register (CKMR)

SC2M: System Clock 2 Monitor bits of clock monitor register (CKMR)

SMS: Standby Mode Select bits of standby mode control register (SMCR)

SPL: Pin state setting bit of standby mode control register (SMCR)

Hi-Z: High-impedance

### 9.4.1 Sleep mode (RC Sleep, Main Sleep, PLL Sleep, Sub Sleep mode)

This mode causes the CPU operating clock (Bus clock CLKB) to stop while other components continue to operate. Transition to Sleep mode is done by writing to the standby mode control register (SMCR). Depending on the currently active clock mode (as indicated by the SC1M System Clock 1 Monitor bits), the MCU switches either to RC Sleep, Main Sleep, PLL Sleep or Sub Sleep mode.

### 9.4.1.1 Functions in Sleep Mode

#### CPU, internal memory and data retention

In Sleep mode, the clock supplying the CPU and the internal memory (CLKB) is stopped. However the contents of dedicated registers, such as accumulator, and the internal RAM are retained.

#### Resources

The resource clocks CLKP1 and CLKP2 are active in Sleep mode and the activated resources are operating in their last configuration.

#### Source clock timers

The oscillators and corresponding source clock timers (RC clock timer, Main clock timer and Sub clock timer) are not affected by Sleep mode. They keep running according to their configuration before transition to Sleep mode.

#### Resets and Interrupts

Resets and Interrupts are active in Sleep mode and can be used to release the Sleep mode.

#### External Bus and Hold function

The external bus is stopped in Sleep mode, however the external bus hold function is active (setting external bus pins to Hi-Z controlled by HRQ pin if hold function is enabled).

#### Status of pins

During Sleep mode, all pins (excluding those used for the external bus) retain their previous function.

### 9.4.1.2 Switching to Sleep Mode

Writing "01" to the SMS bits of the standby mode control register (SMCR) requests a switch to a Sleep mode. Depending on the current clock mode as indicated by the SC1M bits of the clock monitor register (CKMR), a transition to the corresponding Sleep mode is requested (RC Run -> RC Sleep, Main Run -> Main Sleep, PLL Run -> PLL Sleep, Sub Run -> Sub Sleep mode). The transition to Sleep mode takes place within a few clock cycles after the request.

**Note:**

To make sure instructions following the SMS write instruction are executed after wakeup from Sleep mode (and not before transition to Sleep mode), poll the SMS bits after setting to "01". Branch to the next instruction only when the SMS bits are cleared to "00". Reading "01" means that the transition to the Sleep mode has not been performed yet (transition request is still pending).

**Note:**

When a transition to Sleep Mode should be performed after changing the clock mode (by writing a new value to the System Clock 1 Select "SC1S" or System Clock 2 Select "SC2S" bits of the CKSR register), make sure the clock mode transition has been performed before writing the SMS bits. Clock mode transitions are delayed by the clock stabilization and synchronization mechanism. Always confirm the correct clock mode by reading the SC1M/SC2M (System Clock 1/2 Monitor) bits of the CKMR register before setting the SMS bits.

Only change the setting of the SC1S/SC2S bits when the SMS bits are "00" (no Standby mode transition request pending).

#### Interrupt request when switching to Sleep mode

When a hardware interrupt is pending, then writing to the SMS bits has no effect.

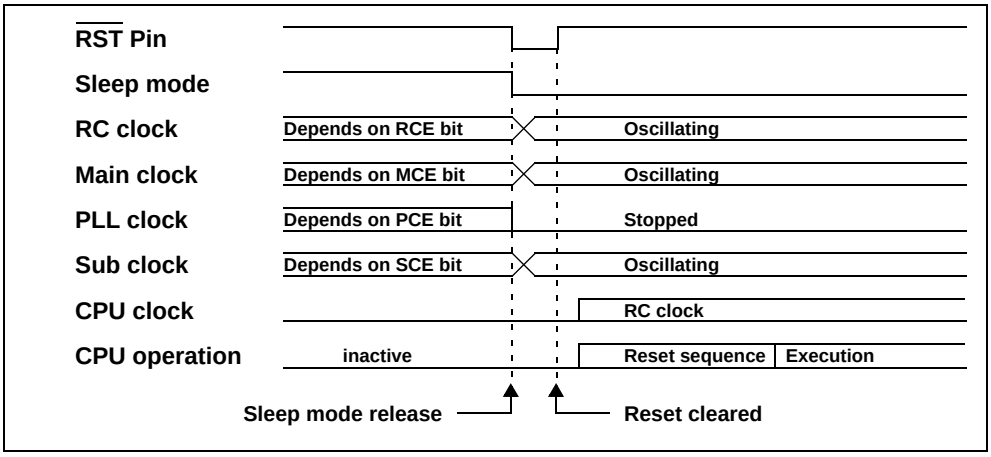
### 9.4.1.3 Release of Sleep Mode

The standby mode control circuit releases Sleep modes when a reset or an interrupt occurs. See also Table 9-4 for an overview.

### Return by a reset

In case of a reset (Power, External, Clock stop detection or Watchdog reset), the MCU changes to the RC Run mode, independent of the last selected clock mode.

Figure 9-5. Release of the Sleep Mode by External Reset




### Return by an interrupt

If an NMI interrupt or another interrupt request of higher than level seven is issued from a peripheral circuit during a Sleep mode, the MCU leaves the Sleep mode and returns to the corresponding Run mode (RC Sleep -> RC Run, Main Sleep -> Main Run, PLL Sleep -> PLL Run, Sub Sleep -> Sub Run). After the Sleep mode is released, the interrupt is handled as an ordinary interrupt.

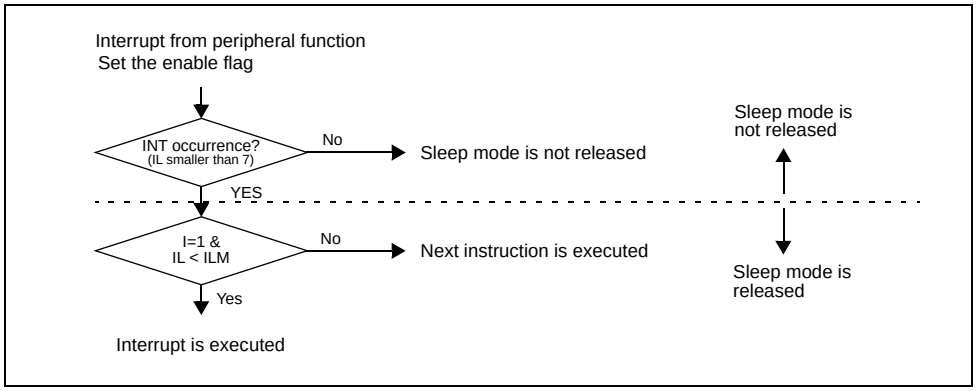
If the interrupt is accepted according to the setting of the I flag of the condition code register (CCR), the interrupt level mask register (ILM) and interrupt control register (ICR), then the CPU finishes the last instructions which were stored in the pipeline and then executes the interrupt routine. After return from this interrupt, the CPU executes the instruction following the last instruction executed before entering the interrupt routine.

If the interrupt is not accepted, the CPU directly executes the instruction following the last instruction executed before entering the Sleep mode.

In case the voltage regulator was manually set to Low Power mode during Sleep mode (by setting the SMCR:LMPSS bit), the wakeup is delayed by typ. 40µs.

Figure 9-6 shows the release of a Sleep mode when an interrupt occurs (not NMI).

Figure 9-6. Release of Sleep Mode by Interrupt



## 9.4.2 Timer Mode (RC Timer, Main Timer, PLL Timer, Sub Timer mode)

This mode causes all functions, excluding oscillators, PLL and source clock timers, to stop. Transition to Timer mode is done by writing to the standby mode control register (SMCR). Depending on the currently active clock mode (as indicated by the SC1M System Clock 1 Monitor bits), the MCU switches either to RC Timer, Main Timer, PLL Timer or Sub Timer mode.

### *9.4.2.1 Functions in Timer Mode*

#### CPU, internal memory and data retention

In Timer mode, the clock supplying the CPU and the internal memory (CLKB) is stopped. However the contents of dedicated registers, such as accumulator, and the internal RAM are retained.

#### Resources

The resource clocks CLKP1 and CLKP2 are stopped in Timer mode and all resources are frozen in their last state.

#### Source clock timers

The oscillators and corresponding source clock timers (RC clock timer, Main clock timer and Sub clock timer) are not affected in Timer mode. They keep running according to their configuration before transition to Timer mode.

#### Resets and Interrupts

Resets and Interrupts are active in Timer mode and can be used to release the Timer mode.

#### External Bus and Hold function

The external bus is stopped in Timer mode, however the external bus hold function is active (setting external bus pins to Hi-Z controlled by HRQ pin if hold function is enabled).

#### Status of pins

The SPL bit of the standby mode control register (SMCR) controls whether the external pins in the Timer mode retain the state they had immediately before switching to the Timer mode or go to the high-impedance state.

### *9.4.2.2 Switching to Timer Mode*

Writing "10" to the SMS bits of the standby mode control register (SMCR) requests a switch to a Timer mode. Depending on the current clock mode as indicated by the SC1M bits of the clock monitor register (CKMR), a transition to the corresponding Timer mode is requested (RC Run -> RC Timer, Main Run -> Main Timer, PLL Run -> PLL Timer, Sub Run -> Sub Timer mode). The transition to Timer mode takes place within a few clock cycles after the request.

#### Note:

To make sure instructions following the SMS write instruction are executed after wakeup from Timer mode (and not before transition to Timer mode), poll the SMS bits after setting to "10". Branch to the next instruction only when the SMS bits are cleared to "00". Reading "10" means that the transition to the Timer mode has not been performed yet (transition request is still pending).

#### Note:

When a transition to Timer Mode should be performed after changing the clock mode (by writing a new value to the System Clock 1 Select "SC1S" or System Clock 2 Select "SC2S" bits of the CKSR register), make sure the clock mode transition has been performed before writing the SMS bits. Clock mode transitions are delayed by the clock stabilization and synchronization mechanism. Always confirm the correct clock mode by reading the SC1M/SC2M (System Clock 1/2 Monitor) bits of the CKMR register before setting the SMS bits.

Only change the setting of the SC1S/SC2S bits when the SMS bits are "00" (no Standby mode transition request pending).

#### Interrupt request when switching to Timer mode

When a hardware interrupt is pending, then writing to the SMS bits has no effect.

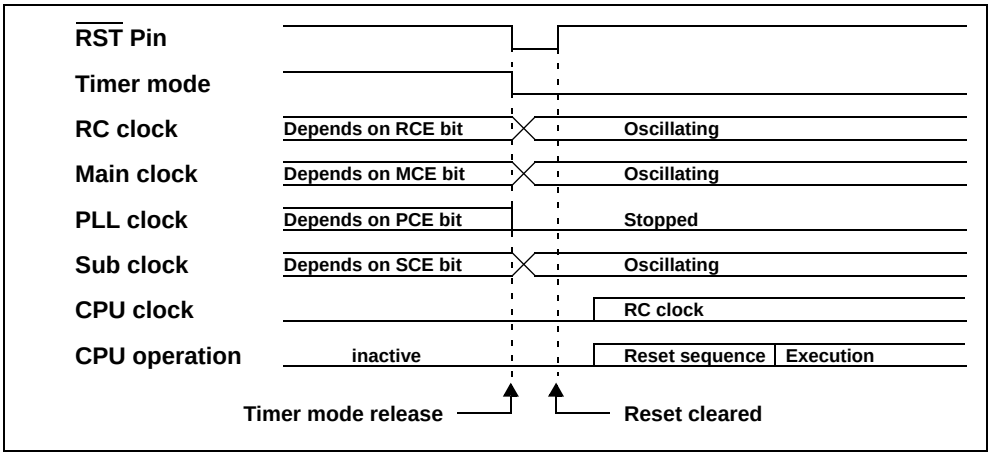
### 9.4.2.3 Release of Timer Mode

The standby control circuit releases Timer modes when a reset or an interrupt occurs. See also Table 9-4 for an overview.

#### Return by a reset

In case of a reset (Power, External, Clock stop detection or Watchdog reset), the MCU changes to the RC Run mode, independent of the last selected clock mode.

Figure 9-7. Release of the Timer Mode by External Reset




#### Return by an interrupt

If an NMI interrupt is asserted or if a source clock timer or External interrupt generates an interrupt request of higher than level seven during a Timer mode, the Timer mode is released and the MCU returns to the corresponding Run mode (RC Timer -> RC Run, Main Timer -> Main Run, PLL Timer -> PLL Run, Sub Timer -> Sub Run). After the Timer mode is released, the interrupt is handled as an ordinary interrupt.

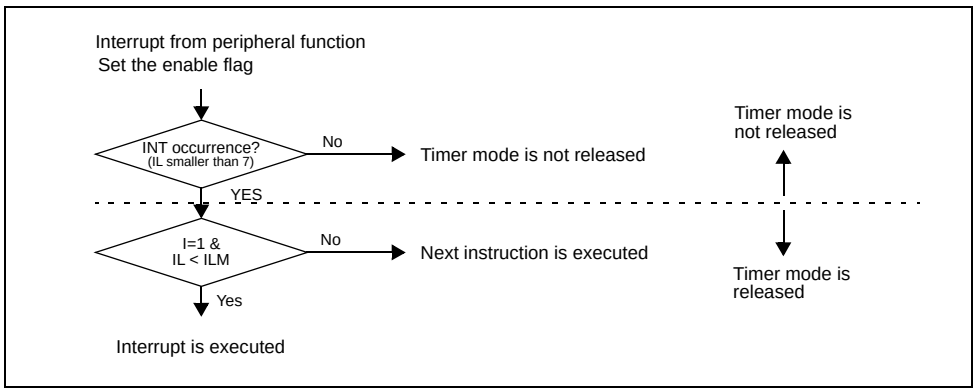
If the interrupt is accepted according to the setting of the I flag of the condition code register (CCR), the interrupt level mask register (ILM) and interrupt control register (ICR), then the CPU finishes the last instructions which were stored in the pipeline and then executes the interrupt routine. After return from this interrupt, the CPU executes the instruction following the last instruction executed before entering the interrupt routine.

If the interrupt is not accepted, the CPU directly executes the instruction following the last instruction executed before entering the Timer mode.

In case the voltage regulator was manually set to Low Power mode during Timer mode (by setting the SMCR:LMPSS bit), the wakeup is delayed by typ. 40µs.

Figure 9-6 shows the release of a Timer mode when an interrupt occurs (not NMI).

Figure 9-8. Release of Timer Mode by Interrupt



## 9.4.3 Stop Mode

Because this mode causes all oscillators to stop and inactivates all functions, data can be retained by the lowest power consumption.

### *9.4.3.1 Functions in Stop Mode*

#### CPU, internal memory and data retention

In Stop mode, the clock supplying the CPU and the internal memory (CLKB) is stopped. However the contents of dedicated registers, such as accumulator, and the internal RAM are retained.

#### Resources

The resource clocks CLKP1 and CLKP2 are stopped and all resources are frozen in their last state.

#### Source clock timers and clocks

The source clock timers (RC clock timer, Main clock timer and Sub clock timer) are all stopped and cleared together with all oscillators and the PLL. All oscillator ready flags and the PLL ready flag are cleared.

#### Resets and Interrupts

Resets and Interrupts are active in Stop mode and can be used to release the Stop mode.

#### External Bus and Hold function

The external bus is stopped in Stop mode, however the external bus hold function is active (setting external bus pins to Hi-Z controlled by HRQ pin if hold function is enabled).

#### Status of pins

The SPL bit of the standby mode control register (SMCR) controls whether the external pins in the Stop mode retain the state they had immediately before switching to the Stop mode or go to the high-impedance state.

### *9.4.3.2 Switching to Stop Mode*

Writing "11" to the SMS bits of the standby mode control register (SMCR) requests a switch to Stop mode, independent of the current clock mode. The transition to Stop mode takes place within a few clock cycles after the request.

#### Note:

To make sure instructions following the SMS write instruction are executed after wakeup from Stop mode (and not before transition to Stop mode), poll the SMS bits after setting to "11". Branch to the next instruction only when the SMS bits are cleared to "00". Reading "11" means that the transition to the Stop mode has not been performed yet (transition request is still pending).

#### Note:

Setting the SMS bits to "11" (Stop mode) is allowed together with changing the clock mode (by writing a new value to the SC1S or SC2S System Clock Select bits of the CKSR register). The new clock setting becomes effective immediately after leaving the Stop mode. In difference to the Sleep and Timer mode, it is not necessary to read the SC1M or SC2M (System Clock Monitor) bits of the CKMR register and wait for the clock mode transition before setting the SMS bits. However do not change the setting of the SC1S/SC2S bits after writing to the SMS bits (when a Standby mode transition request is pending).

Both System clocks should be set to RC clock when switching to Stop mode to ensure a reliable MCU startup even when the external clock fails. This is mandatory when the Clock stop detection reset function is used.

Never set the SC1S or SC2S bits to a clock which does not exist (for example "Sub clock mode" when device has no sub oscillator or no crystal/resonator is connected to sub oscillator pins).

### Interrupt request when switching to Stop mode

When a hardware interrupt is pending, then writing to the SMS bits has no effect.

### *9.4.3.3 Release of Stop Mode*

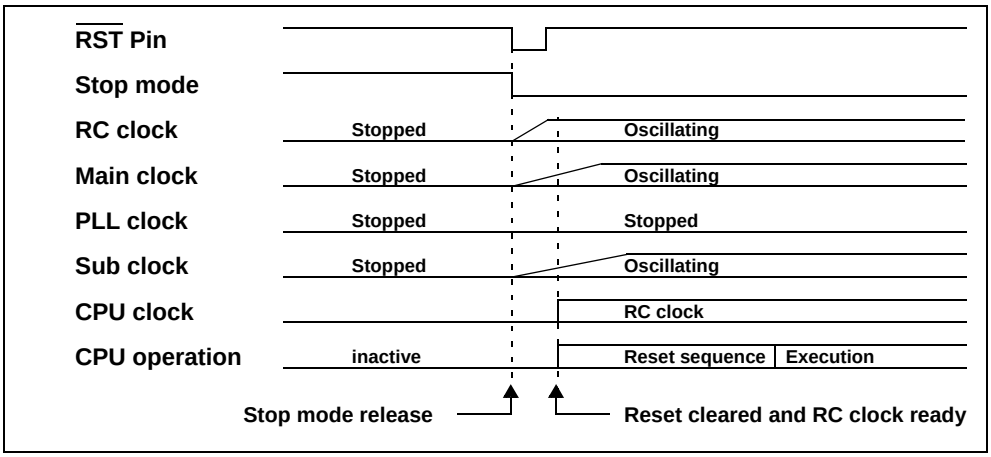
The standby control circuit releases the Stop mode when a reset or an interrupt occurs. See also Table 9-4 for an overview.

### Return by a reset

In case of a Power reset or External reset, the MCU changes to the RC Run mode, independent of the last selected clock mode.

Other types of reset (Clock stop, Software or Watchdog) are not possible in Stop mode.

Figure 9-9. Release of the Stop Mode by External Reset




### Return by an interrupt

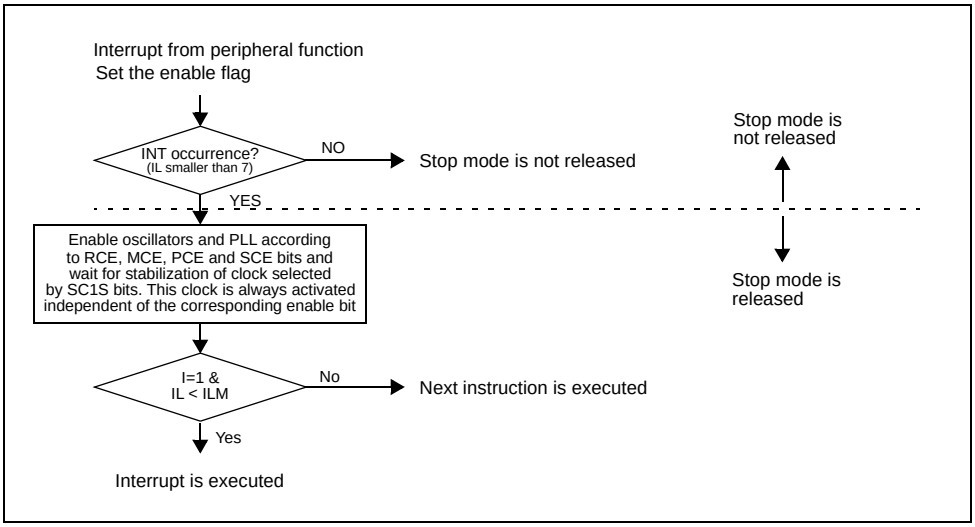
If an NMI interrupt is asserted or if an External interrupt generates an interrupt request of higher than level seven during Stop mode, the Stop mode is released and the MCU returns to the Run mode selected by the SC1S System Clock 1 Select bits (after applying the appropriate clock stabilization wait time). Oscillators are enabled according to the configuration of the CKSR register. After the Stop mode is released, the interrupt is handled as an ordinary interrupt.

If the interrupt is accepted according to the setting of the I flag of the condition code register (CCR), the interrupt level mask register (ILM) and interrupt control register (ICR), then the CPU finishes the last instructions which were stored in the pipeline and then executes the interrupt routine. After return from this interrupt, the CPU executes the instruction following the last instruction executed before entering the interrupt routine.

If the interrupt is not accepted, the CPU directly executes the instruction following the last instruction executed before entering the Stop mode.

Figure 9-10 shows the release of a Stop mode when an interrupt occurs (not NMI).

Figure 9-10. Release of Stop Mode by Interrupt




## 9.5 Mode Change Table and operation status

Table 9-4 show the mode change table of the F$^2$MC-16FX MCU and Table 9-5 shows the operation status in each operating mode.

### Mode change table

Table 9-4. Mode change table

| Current mode (indicated by SC1M bits) | After Reset | After interrupt | By CPU command |
|---|---|---|---|
| RC Run | RC Run | RC Run | Main Run, PLL Run, Sub Run, RC Sleep, RC Timer, Stop |
| Main Run | | Main Run | RC Run, PLL Run, Sub Run, Main Sleep, Main Timer, Stop |
| PLL Run | | PLL Run | RC Run, Main Run, Sub Run PLL Sleep, PLL Timer, Stop |
| Sub Run | | Sub Run | RC Run, Main Run, PLL Run Sub Sleep, Sub Timer, Stop |
| RC Sleep | | RC Run | - |
| Main Sleep | | Main Run | - |
| PLL Sleep | | PLL Run | - |
| Sub Sleep | | Sub Run | - |
| RC Timer | | RC Run | - |
| Main Timer | | Main Run | - |
| PLL Timer | | PLL Run | - |
| Sub Timer | | Sub Run | - |
| Stop | | RC Run, Main Run PLL Run, Sub Run (depending on last value written to SC1S bits) | - |

### Operation status in each operating mode

Table 9-5 lists the operation status in each operating mode.

Table 9-5. Operation status in each operating mode

| Operation mode | RC oscillator and RC clock timer | Main oscillator and Main clock timer | PLL clock | Sub oscillator and Sub clock timer | Clock source for Bus Clock CLKB | Clock source for Peripheral Clock | |
|---|---|---|---|---|---|---|---|
| | | | | | | CLKP1 | CLKP2 |
| RC run | Active | Depending on MCE and SC2M bits | Depending on PCE, MCE and SC2M bits | Depending on SCE and SC2M bits | RC clock | RC clock | Depending on SC2M bits |
| RC sleep | | | | | Stopped | | |
| RC timer | | | | | | Stopped | |
| Main run | Depending on RCE and SC2M bits | Active | Depending on PCE and SC2M bits | Depending on SCE and SC2M bits | Main clock | Main clock | Depending on SC2M bits |
| Main sleep | | | | | Stopped | | |
| Main timer | | | | | | Stopped | |
| PLL run | Depending on RCE and SC2M bits | Active | Active | Depending on SCE and SC2M bits | PLL clock | PLL clock | Depending on SC2M bits |
| PLL sleep | | | | | Stopped | | |
| PLL timer | | | | | | Stopped | |
| Sub run | Depending on RCE and SC2M bits | Depending on MCE and SC2M bits | Depending on PCE, MCE and SC2M bits | Active | Sub clock | Sub clock | Depending on SC2M bits |
| Sub sleep | | | | | Stopped | | |
| Sub timer | | | | | | Stopped | |
| Stop | Stopped | | | | | | |
| External reset activated | RC oscillator on, RC clock Timer stopped | Main oscillator on, Main clock Timer active | Stopped | Sub oscillator on, Sub clock Timer stopped | Stopped | | |

## 9.6 Usage Notes on Standby Mode

Note the following items when using the standby modes:

- Switching to a standby mode and Interrupt
- Release of the standby mode by an Interrupt
- Release of the Stop mode by an Interrupt
- Oscillation stabilization wait time after Stop mode release
- Clock mode switching

### Switching to a Standby Mode and Interrupt

When there is a pending interrupt request to the CPU, writing to the SMS bits in the SMCR register has no effect. In other words, the desired mode transition is not performed even after the interrupt service is completed. If the interrupt request is already cleared and if there are no other pending requests, the desired mode transition can be performed. Note that this behavior is not dependent on whether the setting in the PS register allows the CPU to accept the interrupt request.

### Release of the Standby Mode by an Interrupt

If an NMI or an interrupt request of interrupt priority level higher than seven is issued from a peripheral function during Sleep, Timer or Stop mode, the standby mode is released, which does not depend on whether the CPU accepts the interrupt.

After the release of the standby mode by an interrupt, normal processing is performed.

If the interrupt is accepted, the CPU first executes the remaining instructions in the pipeline as to the state before entering the standby mode. Then it continues with normal interrupt processing.

If the interrupt is not accepted, the CPU continues with the execution of the instructions in the pipeline and the following instructions.

In case the voltage regulator was manually set to Low Power mode during Sleep or Timer mode (by setting the SMCR:LMPSS bit), the wakeup is delayed by typ. 40μs for the stabilization of the voltage regulator.

### Release of the Stop Mode by an Interrupt

The Stop mode can be released by an external interrupt or by an NMI. As an external interrupt input cause, a H-level signal, L-level signal, rising edge, or falling edge can be selected.

### Oscillation stabilization wait time after Stop mode release

Because all oscillators are halted in the Stop mode, an oscillation stabilization wait time is required. The setting of the SC1S and SC2S bits of the CKSR register define which clock is used after wakeup from Stop mode by interrupt. Oscillators and the PLL clock are enabled depending on the RCE, MCE, PCE, SCE, SC1S and SC2S bits and stabilization times are applied according to the setting of the clock stabilization select register (CKSSR). The CPU starts operating after stabilization of the System clock 1, selected by the SC1S bit. Modules using the System clock 2 start operating after stabilization of the System clock 2. The clocks selected by the SC1S and SC2S bits are always enabled, independent of the setting of the clock enable bits (RCE, MCE, PCE and SCE).

**Note:**

After Stop mode release, the CPU starts operating after stabilization of the System clock 1 selected by the SC1S bit. However another clock can be selected as System clock 2 with a longer stabilization time. If the CPU tries to access a peripheral resource clocked by the Peripheral clock 2, then the CPU will be put into hold state until CLKP2 is stabilized and the access can be completed.

After Stop mode release, the SC2M bits indicate the clock used before stop mode until the newly selected clock is ready (clock ready monitor bit of clock selected by SC2S bits is "1") because this register can only be updated with System clock 2.

**Note:**

Never set the SC1S or SC2S bits to a clock which does not exist (for example "Sub clock mode" when device has no sub oscillator or no crystal/resonator is connected to sub oscillator pins) because program execution is delayed until the clock selected by SC1S is stabilized. It is recommended to always select the RC clock for both System clocks when changing to Stop mode.

If Stop mode is released by reset, the clock mode is always set to RC clock and the reset sequence is executed after stabilization of the RC oscillator. All registers and functions are reset to the initial state.

### Clock mode switching

When the clock mode is switched by writing to the SC1S/SC2S bits, do not switch to Sleep or Timer mode before the clock mode switching is completed. Confirm the completion of clock mode switching by referring to the SC1M/SC2M bits of the clock monitor register (CKMR).

A transition to Stop mode however is possible without waiting for the clock mode switching.

Only change the setting of the SC1S/SC2S bits when the SMS bits are "00" (no Standby mode transition request pending).

## 9.7 Voltage Regulator Operation

The F$^2$MC-16FX MCUs are equipped with an on-chip voltage regulator which generates the power supply for the core logic out of the external power supply. This regulator has different operation modes with programmable output voltage. This feature can be used to optimize power consumption, especially in Standby modes.

This chapter describes the different operation modes of the regulator, how to change the operation mode and how to set the output voltage.

### High Power mode

The "High Power mode" is the default operation mode of the voltage regulator. In this mode, the regulator is able to provide the maximum current consumed by the MCU in any mode. The output voltage for this mode is individually programmable.

The default output voltage is 1.8V (adjusted to the process technology) and the current consumption of the regulator itself is several hundred microampere.

The voltage regulator is operating in this mode when either the RC oscillator or the Main oscillator is active (unless the SMCR:LMPS or SMCR:LPMSS bits are set).

**Low Power mode**

The "Low Power Mode" is the alternative operation mode of the voltage regulator. In the Low Power Mode, the regulator is able to provide a maximum transient current of ~100µA. The current consumption of the regulator itself is reduced to ~10µA. No Flash programming/erasing is permitted in this mode.

Two different Low Power Modes are available. The output voltage for each mode is individually programmable. The default output voltage for both modes is 1.8V:

- Low Power Mode A

  The voltage regulator is operating in this mode when the RC oscillator and the Main oscillator are both stopped and the Sub oscillator is active (Sub Run/Sleep/Timer mode with disabled RC and Main oscillator).

  This mode can also be selected by setting the SMCR:LPMS or SMCR:LPMSS bits even if the RC or Main oscillator is enabled.

- Low Power Mode B

  The voltage regulator is operating in this mode when all oscillators are stopped (Stop mode).

  This mode can also be selected by setting the SMCR:LPMBSS bit.

**Note:**

Selecting this mode using the SMCR:LPMBSS bit is currently under evaluation and must not be used.

## 9.7.1 Changing the Voltage Regulator operation mode

The voltage regulator operation modes are automatically controlled by hardware depending on the selected CPU operation mode. However in some modes and under certain conditions, it is permitted to manually change the regulator from High Power mode to Low Power mode for further current saving.

### *9.7.1.1 Overview*

The current consumption of the MCU core depends on the operation mode of the MCU (Run, Sleep, Timer or Stop mode), the activated oscillators and PLL and the selected frequencies (CLKS1, CLKS2, CLKB, CLKP1 and CLKP2).

The automatic voltage regulator control logic sets the regulator to the High Power mode whenever the Main or the RC oscillator is activated. In this High Power mode, the regulator is able to drive any current consumed by the core. This guarantees a safe MCU operation independent of the selected CPU operation mode and the RC or Main clock frequency.

Certain applications however require a very small MCU current consumption in dedicated Low Power CPU operation modes. This can be achieved by manually setting the voltage regulator to the Low Power mode even if the Main or RC oscillator is active. This reduces the MCU current consumption by several 100µA.

Using this feature is permitted only for CPU operation modes with a small dynamic current consumption.

In the automatic voltage regulator control, the stabilization time of the regulator takes place during the stabilization of the oscillators. Hence there is no additional delay for stabilization of the voltage regulator.

### *9.7.1.2 Setting Voltage regulator to Low Power mode by user program*

The voltage regulator can be set to the Low Power mode by controlling the SMCR:LPMS and SMCR:LPMSS bits, depending on the CPU operation mode where power consumption should be reduced.

**Switching to the Low Power mode with the SMCR:LPMS bit**

Setting the SMCR:LPMS bit to "1" immediately switches the voltage regulator to the Low Power mode. This is permitted only in RC Run mode (100kHz) with disabled PLL and a maximum Main clock frequency of 4MHz. The regulator stays in Low Power mode until this bit is cleared, independent of the MCU operation mode (Run or Standby mode).

The MCU must be set to the RC Run mode and also CLKS2 must be set to RC or Sub clock by setting the SC1S/SC2S bits and confirming the transition by reading the SC1M/SC2M bits. Then the SMCR:LPMS bit can be set to "1" and the voltage regulator switches to the Low Power mode.

The regulator must be switched back to the High Power mode before changing the CPU operation mode to a setting with a higher current consumption. This must be done in the following sequence:

- Clear the SMCR:LPMS bit.
- Apply a wait time of 40µs for stabilization of the voltage regulator.
- Switch to the new operation mode (change SC1S/SC2S bits, enable PLL, etc.).

### Switching to the Low Power mode with the SMCR:LPMSS bit

This bit can be used to switch the voltage regulator to Low Power mode in certain Sleep and Timer modes.

Setting the SMCR:LPMSS bit to "1" switches the voltage regulator to the Low Power mode with the next transition to a Standby mode. This bit has no effect as long as the MCU is in Run mode. Setting the SMCR:LPMSS bit is possible before or together with setting the SMCR:SMS Standby mode request bits.

A wakeup interrupt switches the voltage regulator automatically back to the High Power mode (if the Main or RC oscillator is enabled). The transition to Run mode is automatically delayed by the stabilization time of the regulator (~40µs).

Setting SMCR:LPMS to "1" overrules the setting of the SMCR:LPMSS bit.

### Switching to the Low Power mode B with the SMCR:LPMBSS bit

**Note:** Do not change this bit, because this function is under evaluation by Cypress.

Setting the SMCR:LPMBSS bit to "1" switches the voltage regulator to the Low Power mode B instead of Low Power mode A after the next transition to a Standby mode. This means the output voltage is controlled by the VRCR:LPMB[2:0] bits instead of the VRCR:LPMA[2:0] bits. Setting the SMCR:LPMBSS bit is possible before or together with setting the SMCR:SMS Standby mode request bits.

This bit only has an effect when the device is in Standby mode and the regulator is set to Low Power mode by hardware or the SMCR:LPMS or SMCR:LPMSS bits.

A wakeup interrupt switches the voltage regulator back to the High Power mode or Low Power mode A. The transition to Run mode is automatically delayed by the stabilization time of the regulator (~40µs).

## *9.7.1.3 Permitted configurations for using the Low Power mode of the voltage regulator*

The voltage regulator can be switched to the Low Power mode (setting SMCR:LPMS or LMPSS to "1") in the following configurations:

Table 9-6. Permitted configurations for using the Low Power mode A

<table>
<tr><th>Operation mode</th><th>RC oscillator</th><th>Main oscillator</th><th>PLL</th><th>Sub oscillator</th></tr>
<tr><td>RC run (*1)</td><td rowspan="2">Active (set to 100kHz)</td><td rowspan="2">Active with $f_{osc} \leq 4$MHz or disabled</td><td rowspan="6">Disabled</td><td rowspan="3">Active or disabled</td></tr>
<tr><td>RC sleep (*1)</td></tr>
<tr><td>RC timer</td><td>Active (set to 100kHz or 2MHz)</td></tr>
<tr><td>Main timer</td><td rowspan="4">Active (set to 100kHz or 2MHz) or disabled</td><td>Active with $f_{osc} \leq 4$MHz</td></tr>
<tr><td>Sub run (*1)</td><td rowspan="3">Active with $f_{osc} \leq 4$MHz or disabled</td><td rowspan="3">Active</td></tr>
<tr><td>Sub sleep (*1)</td></tr>
<tr><td>Sub timer</td></tr>
</table>

*1: System clock 2 must be set to RC clock or Sub clock. 2MHz setting of RC clock is not permitted as System clock 2 in Run or Sleep mode.

## 9.7.2 Setting the Voltage Regulator output voltage

The output voltage of the regulator can be programmed to adjust the core voltage depending on the required performance or current consumption.

### *9.7.2.1 Voltage setting in High Power mode*

The default output voltage of the regulator is 1.8V. This is the standard core voltage for the 0.18µm process and suitable for most applications which do not require the maximum performance. See the Datasheet for the maximum internal clock frequencies (CLKS1, CLKS2, CLKB, CLKP1 and CLKP2) depending on the core voltage.

#### Changing the Core voltage to 1.9V

For applications which require the maximum performance, it is required to set the core voltage to 1.9V by setting the VRCR:HPM bits to "11" before changing to the fast clock.

The regulator must be set to 1.9V before enabling the PLL with the high frequency.

For Flash devices, it is also necessary to set the FMCS:RD19V bit(s) to ’1’ before changing to the fast clock. The order of the write access to the VRCR and FMCS register does not matter.

#### 1.6V and 1.7V settings

These settings are not approved by Cypress.

### *9.7.2.2 Voltage setting in Low Power modes*

The default output voltage of the regulator in Low Power mode A and B is 1.8V.

#### Changing the Core voltage in Low Power mode A

Changing the output voltage of the regulator in the Low Power mode A is not approved by Cypress.

#### Changing the Core voltage in Low Power mode B

For applications which require a small current consumption in Stop mode, it is possible to reduce the core voltage to 1.2V by setting the VRCR:LPMB[2:0] bits to "000" before entering Stop mode. This reduces the leakage current of the MCU.

See datasheet for absolute values of the Stop mode current at 1.2V and 1.8V core voltage.

# 10. Source Clock Timers

This chapter explains the functions and operations of the three source clock timers (RC clock timer, Main clock timer and Sub clock timer).

10.1 Overview

10.2 RC Clock Timer

10.3 Main Clock Timer

10.4 Sub Clock Timer

## 10.1 Overview

The MB96300 Super series offers 3 independent source clock timers (RC clock timer, Main clock timer and Sub clock timer) which can issue interrupts at specified intervals and which are used to measure the oscillation stabilization time.

### RC clock timer

The RC clock timer is clocked by the output signal of the internal RC oscillator (CLKRC) and is always running when the RC oscillator is enabled. It is also used to measure the RC clock stabilization wait time.

### Main clock timer

The Main clock timer is clocked by the output signal of the Main oscillation circuit (CLKMC) and is always running when the Main oscillator is enabled. It is also used to measure the Main clock stabilization wait time.

### Sub clock timer

The Sub clock timer is clocked by the output signal of the Sub oscillation circuit (CLKSC) and is always running when the Sub oscillator is enabled. It is also used to measure the Sub clock stabilization wait time.

## 10.2 RC Clock Timer

The RC clock timer consists of a 23-bit counter and a control register. The 23-bit counter divides the RC clock CLKRC. The RC clock timer issues interrupts at specified intervals based on carry signals of the RC clock counter.

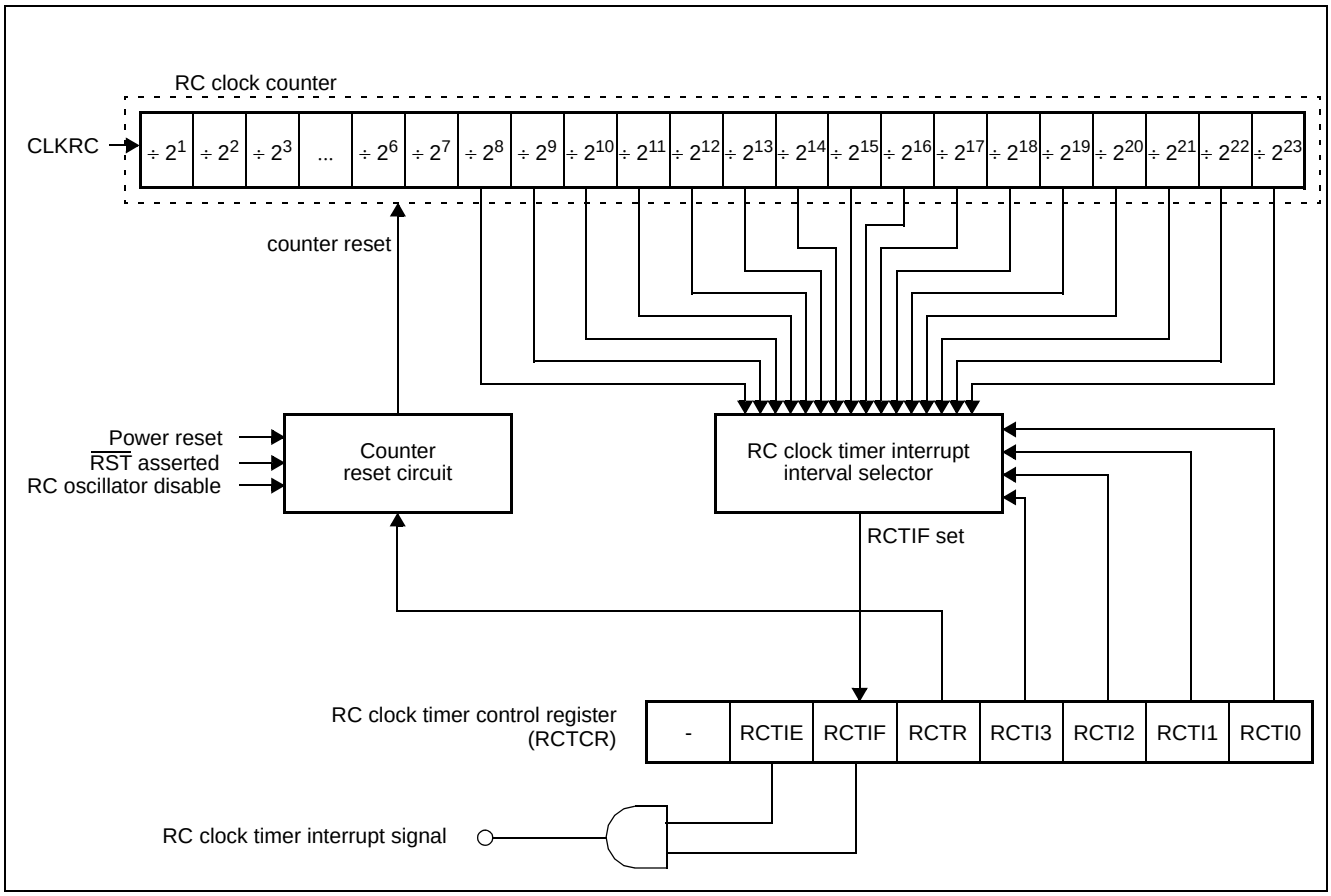
### Outline of the RC clock timer

A Power or External reset initializes the RC clock timer to zero. The RC clock timer is also initialized when the RC clock is disabled (at transition to Stop mode or by setting RCE to "0"), or by writing "0" to the RCTR bit of the RCTCR register. The RC clock timer continues counting as long as the RC clock is supplied. The RC clock timer is used to measure the RC clock stabilization wait time and can be used to generate interval interrupts.

### Block diagram of RC clock timer

Figure 10-1 shows a block diagram of the RC clock timer.

Figure 10-1. Block diagram of RC clock timer




## 10.2.1 RC Clock Timer Control Register (RCTCR)

The RC Clock Timer Control Register (RCTCR) is used to control the RC clock timer interval interrupt function and to reset the RC clock timer.

### Configuration of the RC Clock Timer Control Register (RCTCR)

Figure 10-2 shows the configuration of the RC Clock Timer Control Register (RCTCR) and Table 10-1 describes the function of each bit.

Figure 10-2. Configuration of the RC Clock Timer Control Register (RCTCR)

Address: $000408_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | RCTIE | RCTIF | RCTR | RCTI3 | RCTI2 | RCTI1 | RCTI0 |
| - | R/W | R/W | W | R/W | R/W | R/W | R/W |

Initial value $X\,0\,0\,1\,0\,0\,0\,0_B$

bit3 bit2 bit1 bit0

| RCTI3 | RCTI2 | RCTI1 | RCTI0 | RC Clock Timer Interrupt interval (The corresponding time for the nominal RC clock frequency of 2MHz and 100kHz is given in parentheses) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $2^8$ / CLKRC (approx. 128µs/2.5ms) |
| 0 | 0 | 0 | 1 | $2^9$ / CLKRC (approx. 256µs/5.1ms) |
| 0 | 0 | 1 | 0 | $2^{10}$ / CLKRC (approx. 512µs/10.2ms) |
| 0 | 0 | 1 | 1 | $2^{11}$ / CLKRC (approx. 1ms/20.5ms) |
| 0 | 1 | 0 | 0 | $2^{12}$ / CLKRC (approx. 2ms/41ms) |
| 0 | 1 | 0 | 1 | $2^{13}$ / CLKRC (approx. 4ms/82ms) |
| 0 | 1 | 1 | 0 | $2^{14}$ / CLKRC (approx. 8ms/164ms) |
| 0 | 1 | 1 | 1 | $2^{15}$/ CLKRC (approx. 16ms/328ms) |
| 1 | 0 | 0 | 0 | $2^{16}$ / CLKRC (approx. 32ms/655ms) |
| 1 | 0 | 0 | 1 | $2^{17}$ / CLKRC (approx. 65ms/1.3s) |
| 1 | 0 | 1 | 0 | $2^{18}$ / CLKRC (approx. 131m/2.6s) |
| 1 | 0 | 1 | 1 | $2^{19}$ / CLKRC (approx. 262ms/5.2s) |
| 1 | 1 | 0 | 0 | $2^{20}$ / CLKRC (approx. 524ms/10.4s) |
| 1 | 1 | 0 | 1 | $2^{21}$ / CLKRC (approx. 1.05s/21s) |
| 1 | 1 | 1 | 0 | $2^{22}$ / CLKRC (approx. 2.1s/42s) |
| 1 | 1 | 1 | 1 | $2^{23}$ / CLKRC (approx. 4.2s/84s) |

bit4

| RCTR | RC Clock Timer Reset bit | |
|---|---|---|
| | Read | Write |
| 0 | always reads 1 | reset all bits to 0 |
| 1 | | no effect |

bit5

| RCTIF | RC Clock Timer Interrupt Flag | |
|---|---|---|
| | Read | Write |
| 0 | no interrupt | clear this bit |
| 1 | interrupt requested | no effect |

bit6

| RCTIE | RC Clock Timer Interrupt Enable bit |
|---|---|
| 0 | Disable Interrupt |
| 1 | Enable Interrupt |

bit7

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

X : undefined value
W : Write only (read always returns "1")
R/W : Readable and writable
: Initial value

Table 10-1. Function Description of Each Bit of the RC Clock Timer Control Register (RCTCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 3</td><td>RCTI0 to RCTI3:<br>RC Clock Timer Interrupt interval select bits</td><td>
• These bits control the RC clock timer interrupt interval according to the following table:
<table>
<tr><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th><th></th></tr>
<tr><td>RCTI3</td><td>RCTI2</td><td>RCTI1</td><td>RCTI0</td><td>RC Clock Timer Interrupt interval (The corresponding time for the nominal RC clock frequency of 2MHz and 100kHz is given in parentheses)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>$2^8$ / CLKRC (approx. 128µs/2.5ms)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>$2^9$ / CLKRC (approx. 256µs/5.1ms)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>$2^{10}$ / CLKRC (approx. 512µs/10.2ms)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>$2^{11}$ / CLKRC (approx. 1ms/20.5ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>$2^{12}$ / CLKRC (approx. 2ms/41ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>$2^{13}$ / CLKRC (approx. 4ms/82ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>$2^{14}$ / CLKRC (approx. 8ms/164ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>$2^{15}$/ CLKRC (approx. 16ms/328ms)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>$2^{16}$ / CLKRC (approx. 32ms/655ms)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>$2^{17}$ / CLKRC (approx. 65ms/1.3s)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>$2^{18}$ / CLKRC (approx. 131m/2.6s)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>$2^{19}$ / CLKRC (approx. 262ms/5.2s)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>$2^{20}$ / CLKRC (approx. 524ms/10.4s)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>$2^{21}$ / CLKRC (approx. 1.05s/21s)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>$2^{22}$ / CLKRC (approx. 2.1s/42s)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>$2^{23}$ / CLKRC (approx. 4.2s/84s)</td></tr>
</table>
• These bits are initialized to "0000" by any reset.<br>
• When data is written to these bits, bit 5 (RCTIF) should be cleared at the same time.<br>
• Please note that above calculated times are based on the nominal RC clock frequency. The actual frequency and interval times however vary. Please see the datasheet for more details regarding accuracy of the RC clock frequency.
</td></tr>
<tr><td>bit 4</td><td>RCTR:<br>RC Clock Timer Reset bit</td><td>
• This bit clears all bits of the RC clock counter.<br>
• Writing "0" clears the RC clock counter.<br>
• Writing "1" has no effect.<br>
• "1" is always read from this bit.<br>
Note:<br>
When clearing the RC clock timer by writing "0" to this bit, also clear the RC clock timer interrupt flag by writing "0" to the RCTIF bit.
</td></tr>
<tr><td>bit 5</td><td>RCTIF:<br>RC Clock Timer Interrupt Flag</td><td>
• This is an interrupt request flag for the RC clock timer.<br>
• This bit is set to "1" for each interval specified with the RCTI[3:0] bits.<br>
• When this bit is set to "1", an interrupt request is issued if the interrupt enable bit RCTIE is set to "1".<br>
• This bit is cleared by writing "0" and by any reset.<br>
• This bit should always be cleared when the RC clock timer is reset<br>
• The RC clock timer interrupt interval bits RCTI[3:0] are reset to "0000" ($2^8$ CLKRC cycles) by any reset. Hence, at $2^8$ CLKRC cycles after any reset, the RCTIF bit is set to "1".<br>
• Writing "1" has no effect.<br>
• "1" is always read by a read-modify-write instruction.
</td></tr>
<tr><td>bit 6</td><td>RCTIE:<br>RC Clock Timer Interrupt Enable bit</td><td>
• This bit is used to enable interval interrupts based on the RC clock timer.<br>
• Writing "1" to this bit enables interrupts and writing "0" disables interrupts.<br>
• This bit is reset to "0" by any reset.
</td></tr>
<tr><td>bit 7</td><td>Reserved</td><td>
• Always write "0" to this bit.<br>
• The read value of this bit is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
</table>

### 10.2.2 Operations of RC Clock Timer

The RC clock timer functions as an interval timer for generating interrupts at specified intervals.

#### RC clock counter

The RC clock counter consists of a 23-bit counter that is clocked with the RC clock CLKRC. When the RC clock is active, the RC clock counter always keeps counting. The RC clock frequency is set by the CKFCR: RCFS bit.

The RC clock counter is cleared when the RC clock is stopped (transition to Stop mode or RC clock disabled with CKSR: RCE bit and confirmed by the CKMR: RCM bit), by writing "0" to the RCTR bit of the RCTCR register, by a Power reset and an External reset.

#### Note:

The RC clock counter is cleared and stopped as long as the External reset is asserted. This also stops the RC clock stabilization time counter. Hence the RC clock stabilization time is delayed by RSTX and starts counting after deasserting the External reset. This behavior differs from the Main clock counter.

#### Interval interrupt function

Interrupts are generated at specified intervals according to the carry signals of the RC clock counter. The RCTIF flag is set at the intervals specified with the RCTI[3:0] bits of the RCTCR register. The interval time starts when the RC clock timer is cleared and starts counting.

The RC clock stabilization time counter is connected to the RC clock counter. Hence when the RC clock is stopped (transition to Stop mode or RC clock disabled with CKSR: RCE bit and confirmed by the CKMR: RCM bit), then the RC clock counter is immediately cleared.

## 10.3 Main Clock Timer

The Main clock timer consists of a 23-bit counter and a control register. The 23-bit counter divides the Main clock CLKMC. The Main clock timer issues interrupts at specified intervals based on carry signals of the Main clock counter.

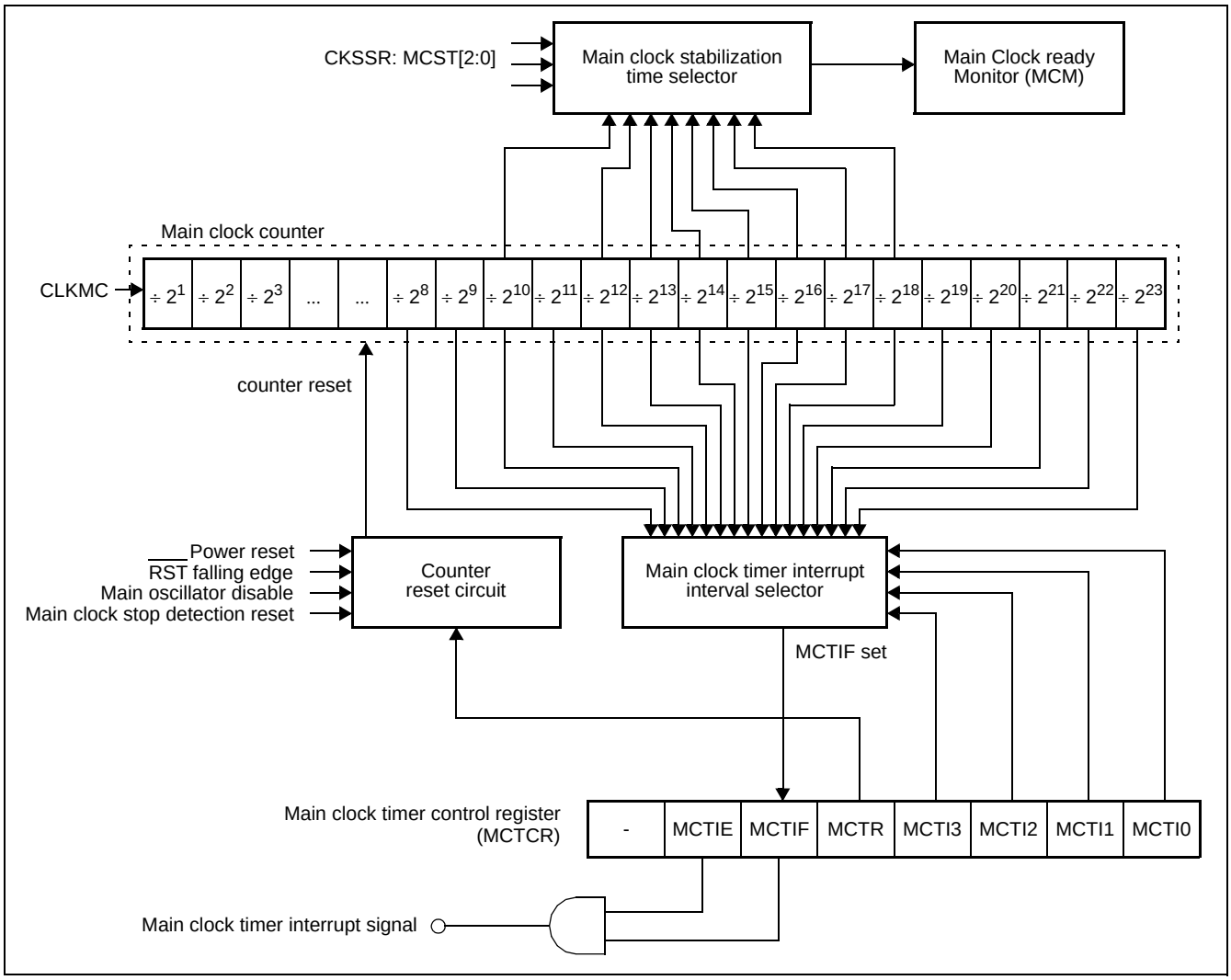
#### Outline of the Main clock timer

A Power reset or a falling edge at the $\overline{\text{RST}}$ input pin initializes the Main clock timer to zero. The Main clock timer is also initialized when the Main clock is disabled (at transition to Stop mode or by setting MCE to "0"), or by writing "0" to the MCTR bit of the MCTCR register. The Main clock timer continues counting as long as the Main clock is supplied. The Main clock timer is used to measure the Main clock stabilization wait time and can be used to generate interval interrupts.

#### Block diagram of Main clock timer

Figure 10-3 shows a block diagram of the Main clock timer.

Figure 10-3. Block diagram of Main clock timer




## 10.3.1 Main Clock Timer Control Register (MCTCR)

The Main Clock Timer Control Register (MCTCR) is used to control the Main clock timer interval interrupt function and to reset the Main clock timer.

### Configuration of the Main Clock Timer Control Register (MCTCR)

Figure 10-4 shows the configuration of the Main Clock Timer Control Register (MCTCR) and Table 10-2 describes the function of each bit.

Figure 10-4. Configuration of the Main Clock Timer Control Register (MCTCR)

Address: 000409$_H$

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| - | MCTIE | MCTIF | MCTR | MCTI3 | MCTI2 | MCTI1 | MCTI0 |
| - | R/W | R/W | W | R/W | R/W | R/W | R/W |

Initial value: X 0 0 1 0 0 0 0 $_B$

bit11 bit10 bit9 bit8

| MCTI3 | MCTI2 | MCTI1 | MCTI0 | Main Clock Timer Interrupt interval (The corresponding time for a Main clock of 4MHz is given in parentheses) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $2^8$ / CLKMC (approx. 64μs) |
| 0 | 0 | 0 | 1 | $2^9$ / CLKMC (approx. 128μs) |
| 0 | 0 | 1 | 0 | $2^{10}$ / CLKMC (approx. 256μs) |
| 0 | 0 | 1 | 1 | $2^{11}$ / CLKMC (approx. 512μs) |
| 0 | 1 | 0 | 0 | $2^{12}$ / CLKMC (approx. 1ms) |
| 0 | 1 | 0 | 1 | $2^{13}$ / CLKMC (approx. 2ms) |
| 0 | 1 | 1 | 0 | $2^{14}$ / CLKMC (approx. 4ms) |
| 0 | 1 | 1 | 1 | $2^{15}$/ CLKMC (approx. 8ms) |
| 1 | 0 | 0 | 0 | $2^{16}$ / CLKMC (approx. 16ms) |
| 1 | 0 | 0 | 1 | $2^{17}$ / CLKMC (approx. 32ms) |
| 1 | 0 | 1 | 0 | $2^{18}$ / CLKMC (approx. 65ms) |
| 1 | 0 | 1 | 1 | $2^{19}$ / CLKMC (approx. 131ms) |
| 1 | 1 | 0 | 0 | $2^{20}$ / CLKMC (approx. 262ms) |
| 1 | 1 | 0 | 1 | $2^{21}$ / CLKMC (approx. 524ms) |
| 1 | 1 | 1 | 0 | $2^{22}$ / CLKMC (approx. 1.049s) |
| 1 | 1 | 1 | 1 | $2^{23}$ / CLKMC (approx. 2.097s) |

bit12

| MCTR | Main Clock Timer Reset bit | |
|---|---|---|
| | Read | Write |
| 0 | always reads 1 | reset all bits to 0 |
| 1 | | no effect |

bit13

| MCTIF | Main Clock Timer Interrupt Flag | |
|---|---|---|
| | Read | Write |
| 0 | no interrupt | clear this bit |
| 1 | interrupt requested | no effect |

bit14

| MCTIE | Main Clock Timer Interrupt Enable bit |
|---|---|
| 0 | Disable Interrupt |
| 1 | Enable Interrupt |

bit15

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

X : undefined value
W : Write only (read always returns "1")
R/W : Readable and writable
: Initial value

Table 10-2. Function Description of Each Bit of the Main Clock Timer Control Register (MCTCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 8 - bit 11</td><td>MCTI0 to MCTI2:<br>Main Clock Timer Interrupt interval select bits</td><td>
• These bits control the Main clock timer interrupt interval according to the following table:
<table>
<tr><th>bit11</th><th>bit10</th><th>bit9</th><th>bit8</th><th></th></tr>
<tr><th>MCTI3</th><th>MCTI2</th><th>MCTI1</th><th>MCTI0</th><th>Main Clock Timer Interrupt interval (The corresponding time for a Main clock of 4MHz is given in parentheses)</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>$2^8$ / CLKMC (approx. 64μs)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>$2^9$ / CLKMC (approx. 128μs)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>$2^{10}$ / CLKMC (approx. 256μs)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>$2^{11}$ / CLKMC (approx. 512μs)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>$2^{12}$ / CLKMC (approx. 1ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>$2^{13}$ / CLKMC (approx. 2ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>$2^{14}$ / CLKMC (approx. 4ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>$2^{15}$/ CLKMC (approx. 8ms)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>$2^{16}$ / CLKMC (approx. 16ms)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>$2^{17}$ / CLKMC (approx. 32ms)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>$2^{18}$ / CLKMC (approx. 65ms)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>$2^{19}$ / CLKMC (approx. 131ms)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>$2^{20}$ / CLKMC (approx. 262ms)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>$2^{21}$ / CLKMC (approx. 524ms)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>$2^{22}$ / CLKMC (approx. 1.049s)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>$2^{23}$ / CLKMC (approx. 2.097s)</td></tr>
</table>
• These bits are initialized to "0000" by any reset.<br>
• When data is written to these bits, bit 13 (MCTIF) should be cleared at the same time.
</td></tr>
<tr><td>bit 12</td><td>MCTR:<br>Main Clock Timer Reset bit</td><td>
• This bit clears all bits of the Main clock counter.<br>
• Writing "0" clears the Main clock counter.<br>
• Writing "1" has no effect.<br>
• "1" is always read from this bit.<br>
Note:<br>
When clearing the Main clock timer by writing "0" to this bit, also clear the Main clock timer interrupt flag by writing "0" to the MCTIF bit.
</td></tr>
<tr><td>bit 13</td><td>MCTIF:<br>Main Clock Timer Interrupt Flag</td><td>
• This is an interrupt request flag for the Main clock timer.<br>
• This bit is set to "1" for each interval specified with the MCTI[3:0] bits.<br>
• When this bit is set to "1", an interrupt request is issued if the interrupt enable bit MCTIE is set to "1".<br>
• This bit is cleared by writing "0" and by any reset.<br>
• This bit should always be cleared when the Main clock timer is reset<br>
• The Main clock timer interrupt interval bits MCTI[3:0] are reset to "0000" ($2^8$ CLKMC cycles) by any reset. Hence, at $2^8$ CLKMC cycles after any reset, the MCTIF bit is set to "1".<br>
• Writing "1" has no effect.<br>
• "1" is always read by a read-modify-write instruction.
</td></tr>
<tr><td>bit 14</td><td>MCTIE:<br>Main Clock Timer Interrupt Enable bit</td><td>
• This bit is used to enable interval interrupts based on the Main clock timer.<br>
• Writing "1" to this bit enables interrupts and writing "0" disables interrupts.<br>
• This bit is reset to "0" by any reset.
</td></tr>
<tr><td>bit 15</td><td>Reserved</td><td>
• Always write "0" to this bit.<br>
• The read value of this bit is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
</table>

### 10.3.2 Operations of Main Clock Timer

The Main clock timer functions as an interval timer for generating interrupts at specified intervals and as a timer for waiting for the Main oscillation to stabilize.

#### Main clock counter

The Main clock counter consists of a 23-bit counter that is clocked with the Main clock CLKMC. When the Main clock is active, the Main clock counter always keeps counting.

The Main clock counter is cleared when the Main clock is stopped (transition to Stop mode or Main clock disabled with CKSR: MCE bit and confirmed by the CKMR: MCM bit), by writing "0" to the MCTR bit of the MCTCR register, by a Power reset, by an External reset and by a Main clock stop detection reset.

#### Note:

An External reset only clears and stops the Main clock counter by detecting a falling edge at the $\overline{\text{RST}}$ input pin. After 700 RC clock cycles (when the external reset extension time is expired), the Main clock counter reset is released and the counter starts counting even if the External reset is still asserted. Hence the Main clock stabilization time takes place already when $\overline{\text{RST}}$ is still active. This behavior differs from the RC clock counter and the Sub clock counter.

#### Interval interrupt function

Interrupts are generated at specified intervals according to the carry signals of the Main clock counter. The MCTIF flag is set at the intervals specified with the MCTI[3:0] bits of the MCTCR register. The interval time starts when the Main clock timer is cleared and starts counting.

When the Main clock is stopped (transition to Stop mode or Main clock disabled with CKSR: MCE bit and confirmed by the CKMR: MCM bit), the Main clock timer is used as a timer for waiting for the Main oscillation to stabilize upon recovery. Therefore, the Main clock counter is immediately cleared when the Main clock is stopped.

## 10.4 Sub Clock Timer

The Sub clock timer consists of a 17-bit counter and a control register. The 17-bit counter divides the Sub clock CLKSC. The Sub clock timer issues interrupts at specified intervals based on carry signals of the Sub clock counter.

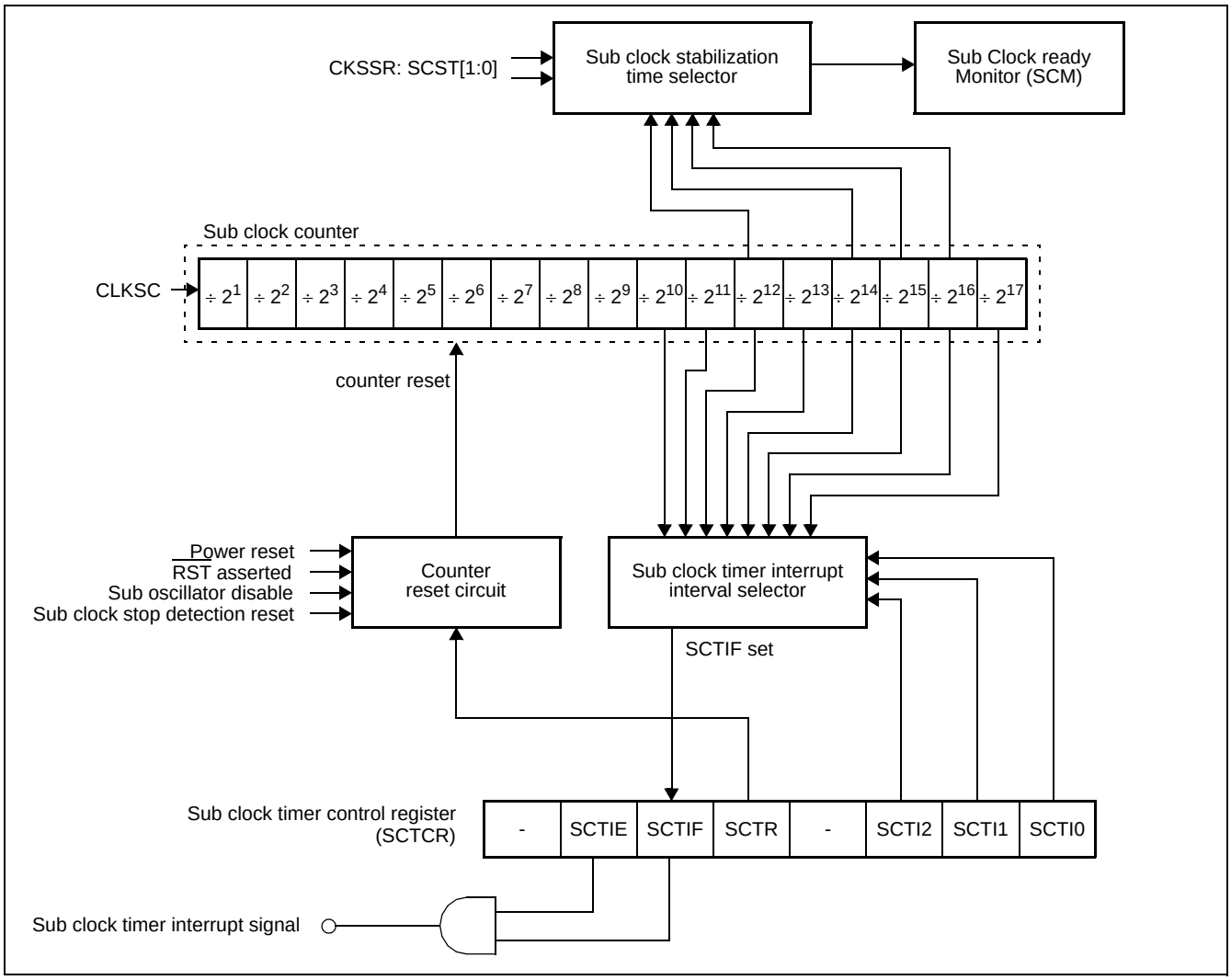
#### Outline of the Sub clock timer

A Power or External reset initializes the Sub clock timer to zero. The Sub clock timer is also initialized when the Sub clock is disabled (at transition to Stop mode or by setting SCE to "0"), or by writing "0" to the SCTR bit of the SCTCR register. The Sub clock timer continues counting as long as the Sub clock is supplied. The Sub clock timer is used to measure the Sub clock stabilization wait time and can be used to generate interval interrupts.

#### Block diagram of Sub clock timer

Figure 10-5 shows a block diagram of the Sub clock timer.

Figure 10-5. Block diagram of Sub clock timer




## 10.4.1 Sub Clock Timer Control Register (SCTCR)

The Sub Clock Timer Control Register (SCTCR) is used to control the Sub clock timer interval interrupt function and to reset the Sub clock timer.

### Configuration of the Sub Clock Timer Control Register (SCTCR)

Figure 10-6 shows the configuration of the Sub Clock Timer Control Register (SCTCR) and Figure 10-3 describes the function of each bit.

Figure 10-6. Configuration of the Sub Clock Timer Control Register (SCTCR)

Address: 00040A$_H$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | SCTIE | SCTIF | SCTR | - | SCTI2 | SCTI1 | SCTI0 |
| - | R/W | R/W | W | - | R/W | R/W | R/W |

Initial value: X 0 0 1 X 0 0 0 $_B$

bit2 bit1 bit0

| SCTI2 | SCTI1 | SCTI0 | Sub Clock Timer Interrupt interval (The corresponding time for a Sub clock of 32.768kHz is given in parentheses) |
|---|---|---|---|
| 0 | 0 | 0 | $2^{10}$ / CLKSC (31.25 ms) |
| 0 | 0 | 1 | $2^{11}$ / CLKSC (62.5 ms) |
| 0 | 1 | 0 | $2^{12}$ / CLKSC (125 ms) |
| 0 | 1 | 1 | $2^{13}$ / CLKSC (250 ms) |
| 1 | 0 | 0 | $2^{14}$ / CLKSC (500 ms) |
| 1 | 0 | 1 | $2^{15}$ / CLKSC (1 s) |
| 1 | 1 | 0 | $2^{16}$ / CLKSC (2 s) |
| 1 | 1 | 1 | $2^{17}$ / CLKSC (4 s) |

bit3

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

bit4

| SCTR | Sub Clock Timer Reset bit | |
|---|---|---|
| | Read | Write |
| 0 | always reads 1 | reset all bits to 0 |
| 1 | | no effect |

bit5

| SCTIF | Sub Clock Timer Interrupt Flag | |
|---|---|---|
| | Read | Write |
| 0 | no interrupt | clear this bit |
| 1 | interrupt requested | no effect |

bit6

| SCTIE | Sub Clock Timer Interrupt Enable bit |
|---|---|
| 0 | Disable Interrupt |
| 1 | Enable Interrupt |

bit7

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

X : undefined value
W : Write only (read always returns "1")
R/W : Readable and writable
: Initial value

Table 10-3. Function Description of Each Bit of the Sub Clock Timer Control Register (SCTCR)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 2</td><td>SCTI0 to SCTI2:<br>Sub Clock Timer Interrupt interval select bits</td><td>
• These bits control the Sub clock timer interrupt interval according to the following table:
<table>
<tr><th colspan="3">bit2 bit1 bit0</th><th></th></tr>
<tr><td>SCTI2</td><td>SCTI1</td><td>SCTI0</td><td>Sub Clock Timer Interrupt interval (The corresponding time for a Sub clock of 32.768kHz is given in parentheses)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>$2^{10}$ / CLKSC (31.25 ms)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>$2^{11}$ / CLKSC (62.5 ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>$2^{12}$ / CLKSC (125 ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>$2^{13}$ / CLKSC (250 ms)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>$2^{14}$ / CLKSC (500 ms)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>$2^{15}$ / CLKSC (1 s)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>$2^{16}$ / CLKSC (2 s)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>$2^{17}$ / CLKSC (4 s)</td></tr>
</table>
• These bits are initialized to "000" by any reset.<br>
• When data is written to these bits, bit 5 (SCTIF) should be cleared at the same time.
</td></tr>
<tr><td>bit 3</td><td>Reserved</td><td>• Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected.</td></tr>
<tr><td>bit 4</td><td>SCTR:<br>Sub Clock Timer Reset bit</td><td>• This bit clears all bits of the Sub clock counter.<br>• Writing "0" clears the Sub clock counter.<br>• Writing "1" has no effect.<br>• "1" is always read from this bit.<br>Note:<br>When clearing the Sub clock timer by writing "0" to this bit, also clear the Sub clock timer interrupt flag by writing "0" to the SCTIF bit.</td></tr>
<tr><td>bit 5</td><td>SCTIF:<br>Sub Clock Timer Interrupt Flag</td><td>• This is an interrupt request flag for the Sub clock timer.<br>• This bit is set to "1" for each interval specified with the SCTI[2:0] bits.<br>• When this bit is set to "1", an interrupt request is issued if the interrupt enable bit SCTIE is set to "1".<br>• This bit is cleared by writing "0" and by any reset.<br>• This bit should always be cleared when the Sub clock timer is reset<br>• Writing "1" has no effect.<br>• "1" is always read by a read-modify-write instruction.</td></tr>
<tr><td>bit 6</td><td>SCTIE:<br>Sub Clock Timer Interrupt Enable bit</td><td>• This bit is used to enable interval interrupts based on the Sub clock timer.<br>• Writing "1" to this bit enables interrupts and writing "0" disables interrupts.<br>• This bit is reset to "0" by any reset.</td></tr>
<tr><td>bit 7</td><td>Reserved</td><td>• Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected.</td></tr>
</table>

## 10.4.2 Operations of Sub Clock Timer

The Sub clock timer functions as an interval timer for generating interrupts at specified intervals and as a timer for waiting for the Sub oscillation to stabilize.

### Sub clock counter

The Sub clock counter consists of a 17-bit counter that is clocked with the Sub clock CLKSC. When the Sub clock is active, the Sub clock counter always keeps counting.

The Sub clock counter is cleared when the Sub clock is stopped (transition to Stop mode or Sub clock disabled with CKSR: SCE bit and confirmed by the CKMR: SCM bit), by writing "0" to the SCTR bit of the SCTCR register, by a Power reset, an External reset and by a Sub clock stop detection reset.

**Note:**

The Sub clock counter is cleared and stopped as long as the External reset is asserted. Hence the Sub clock stabilization time is delayed by $\overline{RST}$ and starts counting after deasserting the External reset. This behavior differs from the Main clock counter.

### Interval interrupt function

Interrupts are generated at specified intervals according to the carry signals of the Sub clock counter. The SCTIF flag is set at the intervals specified with the SCTI[2:0] bits of the SCTCR register. The interval time starts when the Sub clock timer is cleared and starts counting.

When the Sub clock is stopped (transition to Stop mode or Sub clock disabled with CKSR: SCE bit and confirmed by the CKMR: SCM bit), the Sub clock timer is used as a timer for waiting for the Sub oscillation to stabilize upon recovery. Therefore, the Sub clock counter is immediately cleared when the Sub clock is stopped.

# 11. Watchdog Timer and Watchdog Reset

This chapter explains the functions and operations of the Watchdog timer and reset.

11.1 Outline of Watchdog Timer and Reset

11.2 Watchdog Timer Control Registers

11.3 Watchdog Timer Operation

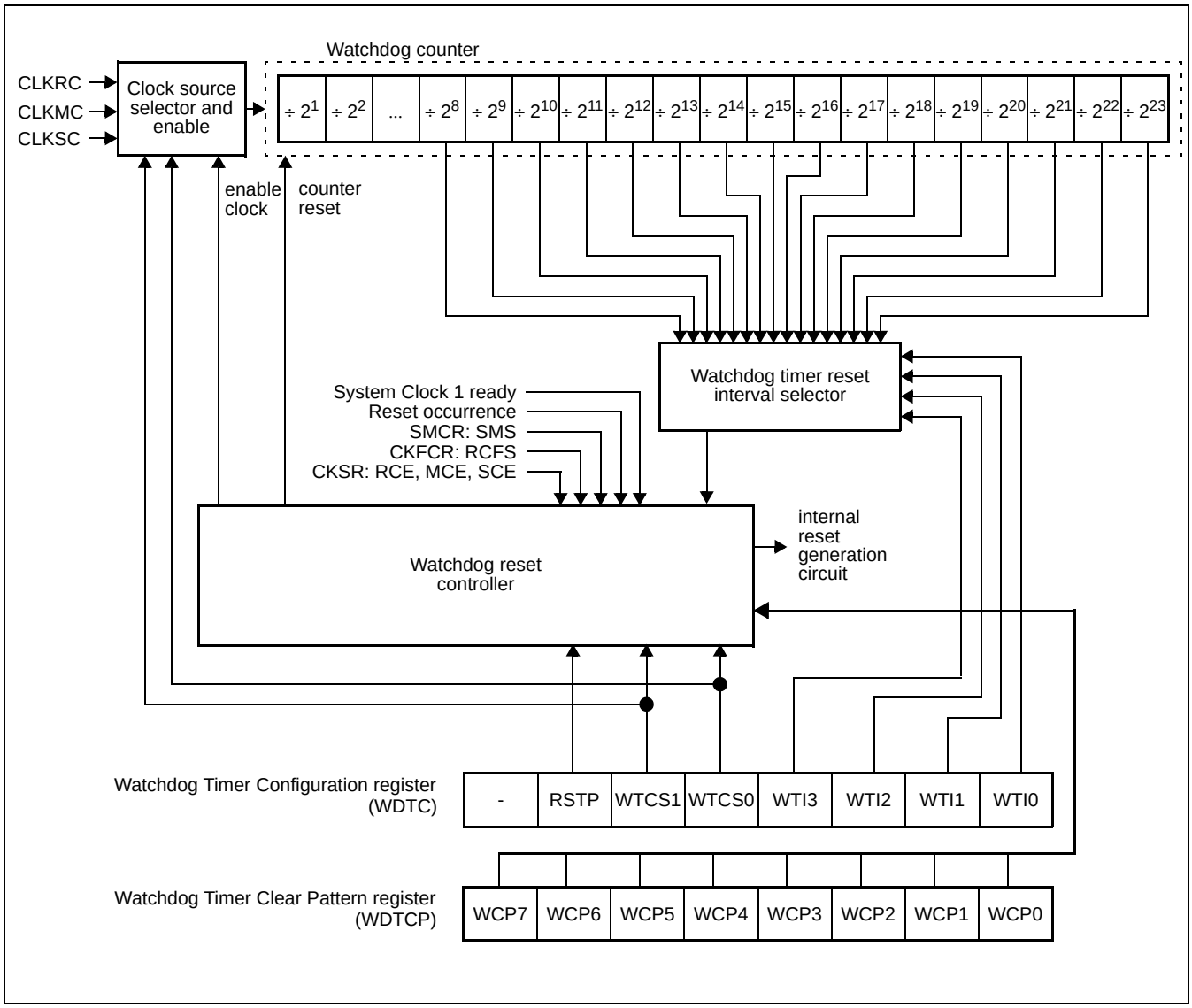
## 11.1 Outline of Watchdog Timer and Reset

The watchdog circuit consists of a 23-bit watchdog counter, control registers and the watchdog reset controller. The 23-bit watchdog counter uses either the RC clock, the Main clock or the Sub clock as clock source.

### Watchdog circuit block diagram

Figure 11-1 shows a block diagram of the watchdog circuit.

Figure 11-1. Watchdog circuit block diagram



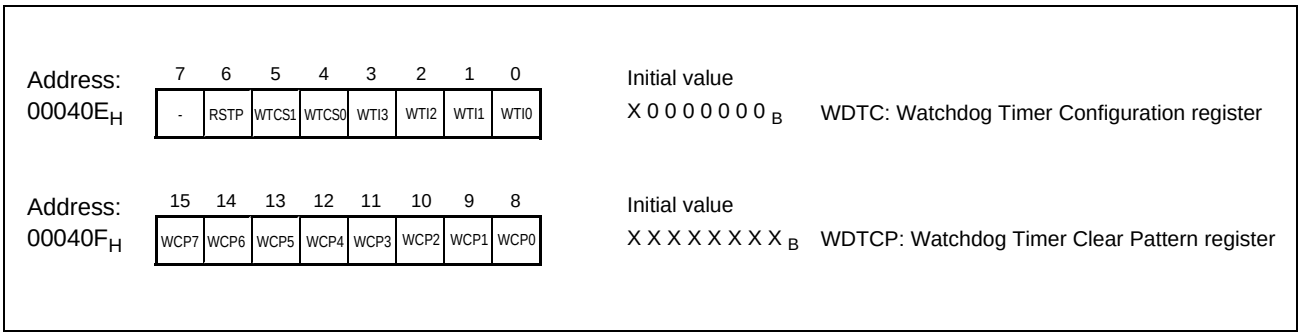

## 11.2 Watchdog Timer Control Registers

This section lists the Watchdog Timer Control Registers and describes the function of each register in details.

### Watchdog Timer Control Registers

The Watchdog Timer has two control registers, the Watchdog Timer Configuration register (WDTC) and the Watchdog Timer Clear Pattern register (WDTCP). Figure 11-2 shows an overview of the Watchdog Timer Control register.

Figure 11-2. Watchdog Timer Control Registers



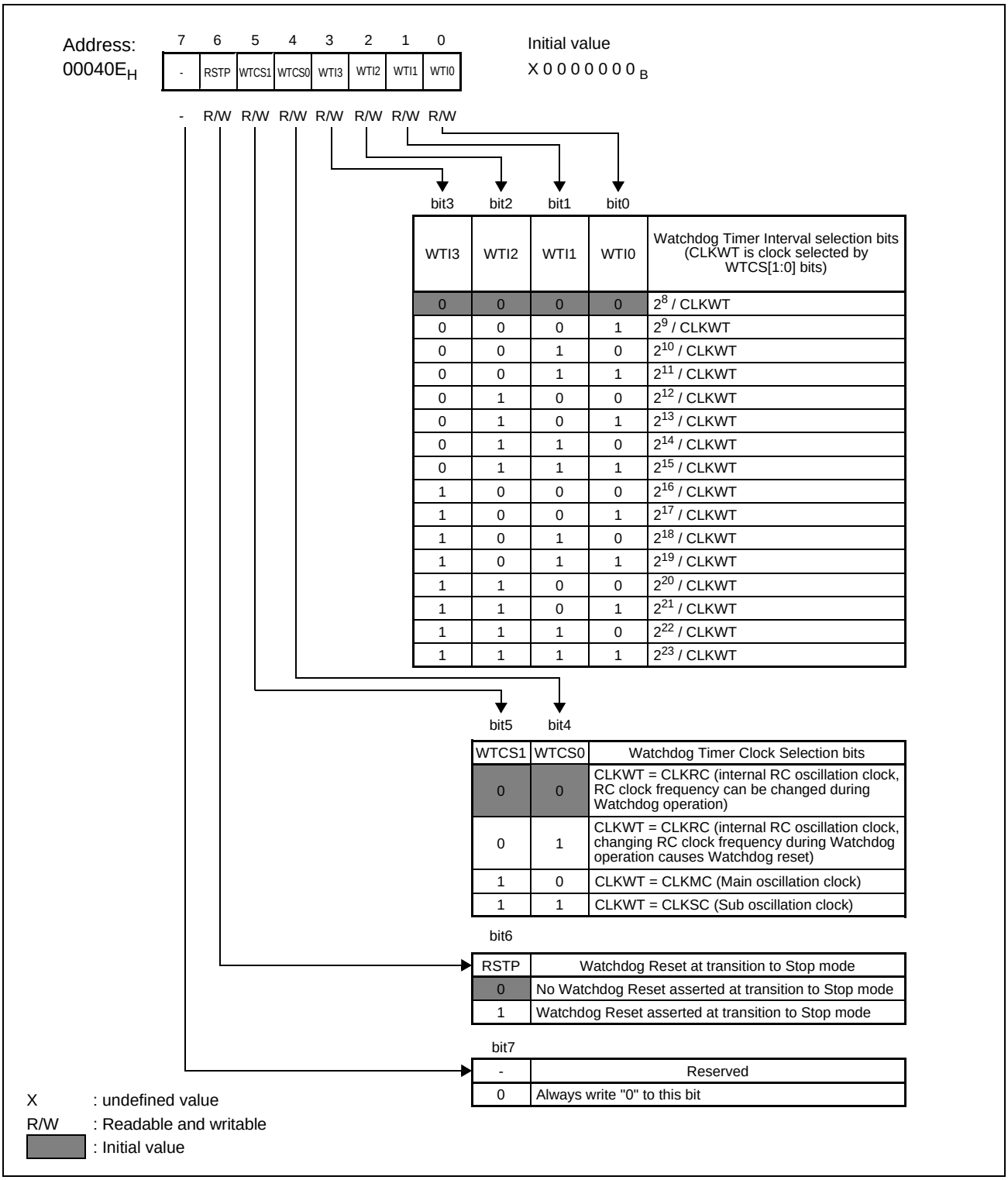

## 11.2.1 Watchdog Timer Configuration register (WDTC)

The Watchdog Timer Configuration register (WDTC) is used to select the clock source and the watchdog interval and to activate some special functions of the Watchdog Timer.

### Configuration of the Watchdog Timer Configuration register (WDTC)

Figure 11-3 shows the configuration of the Watchdog Timer Configuration register (WDTC) and Table 11-1 describes the function of each bit.

Figure 11-3. Configuration of the Watchdog Timer Configuration Register (WDTC)




| WTI3 | WTI2 | WTI1 | WTI0 | Watchdog Timer Interval selection bits (CLKWT is clock selected by WTCS[1:0] bits) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $2^8$ / CLKWT |
| 0 | 0 | 0 | 1 | $2^9$ / CLKWT |
| 0 | 0 | 1 | 0 | $2^{10}$ / CLKWT |
| 0 | 0 | 1 | 1 | $2^{11}$ / CLKWT |
| 0 | 1 | 0 | 0 | $2^{12}$ / CLKWT |
| 0 | 1 | 0 | 1 | $2^{13}$ / CLKWT |
| 0 | 1 | 1 | 0 | $2^{14}$ / CLKWT |
| 0 | 1 | 1 | 1 | $2^{15}$ / CLKWT |
| 1 | 0 | 0 | 0 | $2^{16}$ / CLKWT |
| 1 | 0 | 0 | 1 | $2^{17}$ / CLKWT |
| 1 | 0 | 1 | 0 | $2^{18}$ / CLKWT |
| 1 | 0 | 1 | 1 | $2^{19}$ / CLKWT |
| 1 | 1 | 0 | 0 | $2^{20}$ / CLKWT |
| 1 | 1 | 0 | 1 | $2^{21}$ / CLKWT |
| 1 | 1 | 1 | 0 | $2^{22}$ / CLKWT |
| 1 | 1 | 1 | 1 | $2^{23}$ / CLKWT |

| WTCS1 | WTCS0 | Watchdog Timer Clock Selection bits |
|---|---|---|
| 0 | 0 | CLKWT = CLKRC (internal RC oscillation clock, RC clock frequency can be changed during Watchdog operation) |
| 0 | 1 | CLKWT = CLKRC (internal RC oscillation clock, changing RC clock frequency during Watchdog operation causes Watchdog reset) |
| 1 | 0 | CLKWT = CLKMC (Main oscillation clock) |
| 1 | 1 | CLKWT = CLKSC (Sub oscillation clock) |

| RSTP | Watchdog Reset at transition to Stop mode |
|---|---|
| 0 | No Watchdog Reset asserted at transition to Stop mode |
| 1 | Watchdog Reset asserted at transition to Stop mode |

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

Table 11-1. Function Description of Each Bit of the Watchdog Timer Configuration register (WDTC) (Sheet 1 of 2)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 3</td><td>WTI0 to WTI3:<br>Watchdog Timer Interval select bits</td><td>
• These bits select the Watchdog Timer Interval according to the following table:
<table>
<tr><th colspan="4">bit3 bit2 bit1 bit0</th><th colspan="3">Watchdog Timer Interval depending on selected clock source</th></tr>
<tr><th>WTI3</th><th>WTI2</th><th>WTI1</th><th>WTI0</th><th>RC clock selected (corresponding time for nominal RC clock frequency of 2MHz/100kHz)</th><th>Main clock selected (corresponding time for Main clock frequency of 4MHz)</th><th>Sub clock selected (corresponding time for Sub clock frequency of 32.768kHz)</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>$2^8$ / CLKRC (~128µs/2.5ms)</td><td>$2^8$ / CLKMC (~64µs)</td><td>$2^8$ / CLKSC (7.8ms)</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>$2^9$ / CLKRC (~256µs/5.1ms)</td><td>$2^9$ / CLKMC (~128µs)</td><td>$2^9$ / CLKSC (15.6ms)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>$2^{10}$ / CLKRC (~512µs/10.2ms)</td><td>$2^{10}$ / CLKMC (~256µs)</td><td>$2^{10}$ / CLKSC (31.25ms)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>$2^{11}$ / CLKRC (~1ms/20.5ms)</td><td>$2^{11}$ / CLKMC (~512µs)</td><td>$2^{11}$ / CLKSC (62.5ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>$2^{12}$ / CLKRC (~2ms/41ms)</td><td>$2^{12}$ / CLKMC (~1ms)</td><td>$2^{12}$ / CLKSC (125ms)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>$2^{13}$ / CLKRC (~4ms/82ms)</td><td>$2^{13}$ / CLKMC (~2ms)</td><td>$2^{13}$ / CLKSC (250ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>$2^{14}$ / CLKRC (~8ms/164ms)</td><td>$2^{14}$ / CLKMC (~4ms)</td><td>$2^{14}$ / CLKSC (500ms)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>$2^{15}$ / CLKRC (~16ms/328ms)</td><td>$2^{15}$ / CLKMC (~8ms)</td><td>$2^{15}$ / CLKSC (1s)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>$2^{16}$ / CLKRC (~32ms/655ms)</td><td>$2^{16}$ / CLKMC (~16ms)</td><td>$2^{16}$ / CLKSC (2s)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>$2^{17}$ / CLKRC (~65ms/1.3s)</td><td>$2^{17}$ / CLKMC (~32ms)</td><td>$2^{17}$ / CLKSC (4s)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>$2^{18}$ / CLKRC (~131ms/2.6s)</td><td>$2^{18}$ / CLKMC (~65ms)</td><td>$2^{18}$ / CLKSC (8s)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>$2^{19}$ / CLKRC (~262ms/5.2s)</td><td>$2^{19}$ / CLKMC (~131ms)</td><td>$2^{19}$ / CLKSC (16s)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>$2^{20}$ / CLKRC (~524ms/10.4s)</td><td>$2^{20}$ / CLKMC (~262ms)</td><td>$2^{20}$ / CLKSC (32s)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>$2^{21}$ / CLKRC (~1.05s/21s)</td><td>$2^{21}$ / CLKMC (~524ms)</td><td>$2^{21}$ / CLKSC (64s)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>$2^{22}$ / CLKRC (~2.1s/42s)</td><td>$2^{22}$ / CLKMC (~1.049s)</td><td>$2^{22}$ / CLKSC (128s)</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>$2^{23}$ / CLKRC (~4.2s/84s)</td><td>$2^{23}$ / CLKMC (~2.097s)</td><td>$2^{23}$ / CLKSC (256s)</td></tr>
</table>
• These bits are initialized to "0000" by any reset.<br>
• It is not possible to change the setting of these bits after activation of the Watchdog reset function (by first writing to the Watchdog Timer Clear Pattern register). Configuring these bits and activating the Watchdog reset function with one 16 bit write access however is possible.<br>
• After a reset, these bits can be written again.
</td></tr>
<tr><td>bit 4 - bit 5</td><td>WTCS0 to WTCS1:<br>Watchdog Timer Clock Selection bits</td><td>
• These bits select the clock source for the Watchdog Timer according to the following table:
<table>
<tr><th colspan="2">bit4 bit3</th><th></th></tr>
<tr><th>WTCS1</th><th>WTCS0</th><th>Watchdog Timer Clock Selection bits</th></tr>
<tr><td>0</td><td>0</td><td>CLKWT = CLKRC (internal RC oscillation clock, RC clock frequency can be changed during Watchdog operation)</td></tr>
<tr><td>0</td><td>1</td><td>CLKWT = CLKRC (internal RC oscillation clock, changing RC clock frequency during Watchdog operation causes Watchdog reset)</td></tr>
<tr><td>1</td><td>0</td><td>CLKWT = CLKMC (Main oscillation clock)</td></tr>
<tr><td>1</td><td>1</td><td>CLKWT = CLKSC (Sub oscillation clock)</td></tr>
</table>
• These bits are initialized to "00" by any reset.<br>
• It is not possible to change the setting of these bits after activation of the Watchdog reset function (by first writing to the Watchdog Timer Clear Pattern register). Configuring these bits and activating the Watchdog reset function with one 16 bit write access however is possible.<br>
• After a reset, these bits can be written again.<br>
• Setting WTCS[1:0] to "00" selects the internal RC oscillator and it is possible to change the RC clock frequency (by writing to CKFCR: RCFS bit) after activation of the Watchdog reset function. Please note that changing the RC clock frequency also changes the Watchdog interval time.<br>
• Setting WTCS[1:0] to "01" also selects the internal RC oscillator, but it is NOT allowed to change the RC clock frequency (by writing to CKFCR: RCFS bit) after activation of the Watchdog reset function. Changing the RC clock frequency in this case causes a Watchdog reset.
</td></tr>
</table>

Table 11-1. Function Description of Each Bit of the Watchdog Timer Configuration register (WDTC) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 6 | RSTP:<br>Watchdog Reset at transition to Stop mode | • This bit is initialized to "0" by any reset.<br>• It is not possible to change the setting of this bit after activation of the Watchdog reset function (by first writing to the Watchdog Timer Clear Pattern register). Configuring this bit and activating the Watchdog reset function with one 16 bit write access however is possible.<br>• After a reset, this bit can be written again.<br>• When this bit is set to "0", then transition to Stop mode is allowed<br>• When this bit is set to "1", then writing "11" to SMCR: SMS (Stop mode) causes a Watchdog reset.<br>• Set this bit to "1" only when the application is not using the Stop mode. |
| bit 7 | Reserved | • Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected. |

## 11.2.2 Watchdog Timer Clear Pattern register (WDTCP)

The Watchdog Timer Clear Pattern register (WDTCP) is used to activate the Watchdog reset function and to clear the Watchdog counter.

### Configuration of the Watchdog Timer Clear Pattern register (WDTCP)

Figure 11-4 shows the configuration of the Watchdog Timer Clear Pattern register (WDTCP) and Table 11-2 describes the function of the bits.

Figure 11-4. Configuration of the Watchdog Timer Clear Pattern Register (WDTCP)

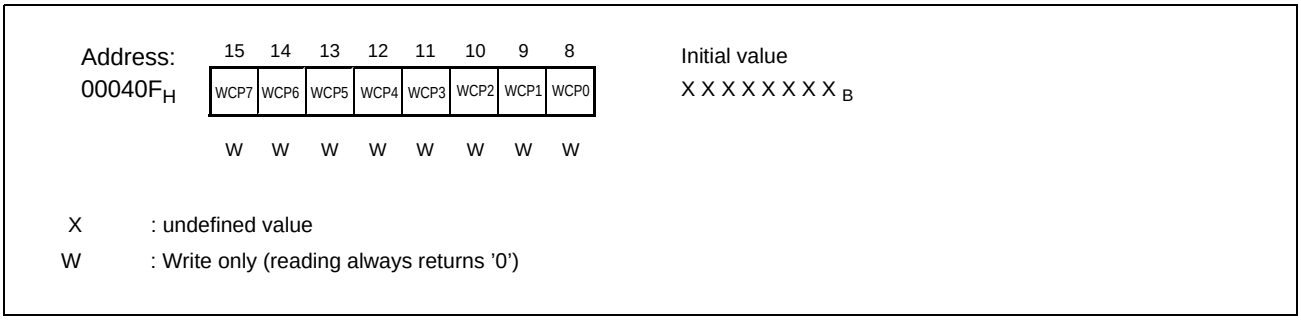



Table 11-2. Function Description of the Bits of the Watchdog Timer Clear Pattern register (WDTCP)

| Bit name | | Function |
|---|---|---|
| bit 8 - bit 15 | WCP0 to WCP7:<br>Watchdog Clear Pattern bits | • The Watchdog reset function is activated by writing to this register.<br>• Once the Watchdog reset function is activated, it can only be disabled by reset (any type of reset).<br>• After writing to this register, it is no longer possible to change the contents of the Watchdog Timer Configuration register (WDTC).<br>• Depending on the value written to the WDTCP register at the first time after reset, the Watchdog Timer operates in one of the following Watchdog operation modes:<br>1. "Clear by 00h, assert reset for other values": Write 00h to WDTCP:<br>In this mode, the Watchdog counter is cleared by writing 00h to the WDTCP register. Writing values other than 00h causes a Watchdog reset.<br>2. "Clear by complementary data" Write any value except 00h to WDTCP:<br>In this mode, the value written to the WDTCP register at the first time is memorized. Clearing the Watchdog counter is possible only by writing the complementary value what replaces the last register value. Writing the same value as before has no effect and writing other values causes a Watchdog reset. See the next chapter for more details.<br>• Reading this register always returns 00h. |

## 11.3 Watchdog Timer Operation

The Watchdog timer and reset function can be used to detect a hang-up of the user program.

If the Watchdog counter is not cleared within the specified time due to, for example, a program hang-up, the Watchdog timer resets the system.

### 11.3.1 Activation and Deactivation of Watchdog Timer

The Watchdog Timer is activated by writing to the Watchdog Timer Clear Pattern register WDTCP. The data written to this register the first time after reset and the data written to the Watchdog Timer Configuration register WDTC at the same time or before determines the Watchdog function.

Once the Watchdog Timer is activated it can only be deactivated by reset (any type of reset).

### 11.3.2 Clearing the Watchdog counter

The Watchdog counter can only be cleared by writing the appropriate data to the WDTCP register. This data depends on the Watchdog operation mode. If the counter is not cleared before the interval selected by the WDTC: WTI[3:0] bits expires, then a Watchdog reset is asserted.

### 11.3.3 Stopping the Watchdog counter

The Watchdog counter can only be stopped (not cleared) when the MCU goes to Stop mode.

#### Transition to Stop mode

All oscillators are disabled in Stop mode, hence the Watchdog counter is also stopped (not cleared).

#### Release of Stop mode

Upon wakeup by interrupt, the watchdog counter starts running when the clock selected as source for the watchdog is stabilized and the CPU clock CLKB is reactivated (clock ready flags set of clocks selected by WDTC:WTCS[1:0] and CKSR:SC1S[1:0] select bit).

#### Note:

Do not use the Stop mode if the Watchdog must always be active. It is recommended to use the RC Timer mode instead as the mode with the smallest current consumption while disabling the Main and Sub oscillator and running the watchdog with the RC clock. Set WDTC: RSTP to "1" to force a Watchdog reset in case of an accidental transition to Stop mode.

### 11.3.4 Selecting the clock source for the Watchdog Counter

The Watchdog counter can operate with the RC clock (CLKRC), the Main clock (CLKMC) or the Sub clock (CLKSC) as clock source. Select a clock source depending on the requirements of the system by setting the WDTC: WTCS[1:0] bits. After activation of the Watchdog Timer it is not possible to change the clock source any more.

When the RC clock should be used as clock source and the system requires changing the RC clock frequency during operation, then set the WDTC: WTCS[1:0] bits to "00". Be aware that changing the RC clock frequency also changes the Watchdog Timer interval.

When the RC clock should be used as clock source but changing the RC clock frequency during operation is not intended, then set the WDTC: WTCS[1:0] bits to "01". In case of an accidental change of the RC clock frequency, a Watchdog reset will be asserted.

The Watchdog counter can only operate if the selected clock is stabilized (CKMR: RCM, MCM or SCM bit is ’1’). During clock stabilization (after Stop mode release or when the Watchdog counter was enabled before stabilization of the clock source), the Watchdog counter is stopped.

**Caution:**

The Watchdog counter can only operate if the selected clock is stabilized (CKMR: RCM, MCM or SCM bit is ’1’). During clock stabilization (after Stop mode release or when the Watchdog counter was enabled before stabilization of the clock source), the Watchdog counter is stopped. Make sure that the clock is stabilized, before using it for the Watchdog.

### 11.3.5 Standby modes

In Sleep and Timer mode the Watchdog Timer operates according to its configuration and is neither cleared nor stopped. In Stop mode the Watchdog Timer is stopped but not cleared. After Stop mode release, the Watchdog timer resumes operation when both, the Watchdog timer clock source and the clock source for CLKS1 are stabilized.

### 11.3.6 Disabling the source oscillator

Setting the clock enable bit (CKSR: RCE, MCE, SCE) of the oscillator used as clock source for the Watchdog counter to "0" (disable oscillator) causes a Watchdog reset.

### 11.3.7 Setting the Watchdog Timer interval

The Watchdog Timer interval can be selected with the WDTC: WTI[3:0] bits. After activation of the Watchdog Timer it is not possible to change the interval any more.

When using the RC oscillator as clock source, consider the frequency variation of this oscillator when calculating the interval length. The absolute times in Table 11-1 are based on the nominal oscillation frequency. Please see the datasheet for more details regarding the accuracy of the RC clock frequency.

### 11.3.8 Selecting the operation mode of the Watchdog Timer

The first write access to the WDTCP register after reset activates the Watchdog Timer and defines the operation mode. Depending on the data written to WDTCP at this first write access, one of the following two operation modes is selected:

#### Clear by writing "00h" and assert Watchdog reset when writing other values

This mode is selected by writing "00h" to the WDTCP register at the first write access after reset.

In this mode, the Watchdog counter is cleared by writing "00h" to the WDTCP register. Writing values other than "00h" causes a Watchdog reset.

#### Clear by writing complementary data and assert Watchdog reset when writing invalid data

This mode is selected by writing any value except "00h" to the WDTCP register at the first write access after reset.

The data written at this first write access is stored in the WDTCP register.

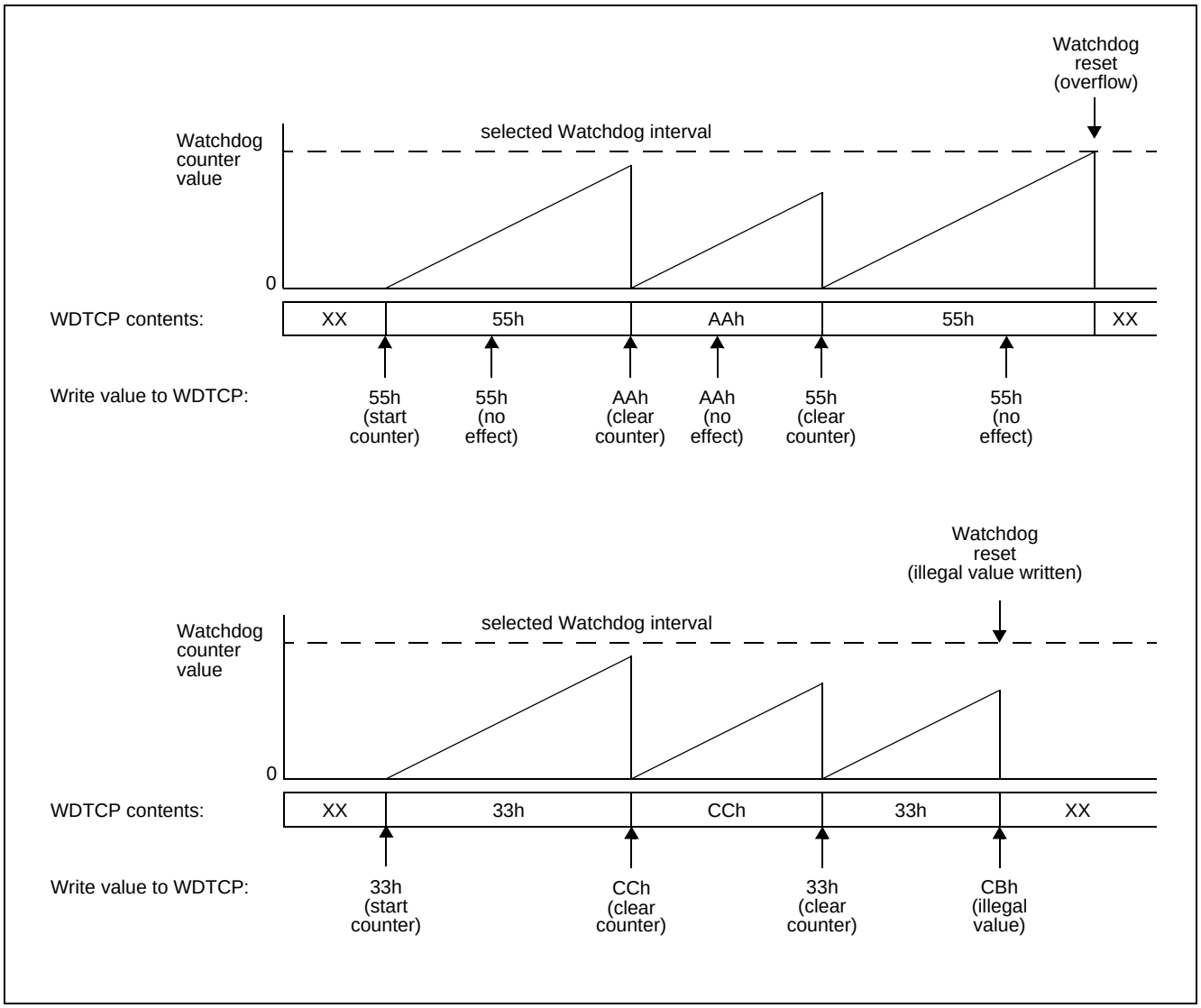
Writing the same data as currently stored in the WDTCP register has no effect.

Writing the complementary data of the current WDTCP contents clears the Watchdog counter and replaces the current value in the WDTCP register. To clear the Watchdog Timer the next time, the write data must be complemented again every time (original data - complementary data - original data - complementary data...)

Writing any data to the WDTCP register that differs from the current register contents or the complementary value causes a Watchdog reset.

The following figure shows an example of the Watchdog Timer operation in this mode.

Figure 11-5. Watchdog Timer operation in complementary mode




## 11.3.9 Watchdog Timer reset causes

A Watchdog Timer reset can be asserted in the following cases:

- The Watchdog Timer was not cleared within the selected interval.
- Invalid data was written to the WDTCP register.
- ’0’ was written to the enable bit of the clock which is used as source clock for the Watchdog Timer (CKSR: RCE, MCE or SCE).
- The RC clock frequency was changed although WDTC:WTCS was set to "01".
- Transition to Stop mode was requested although WDTC:RSTP was set to ’1’.

# 12. External Bus Interface

This chapter explains the functions and operations of the external bus interface.

12.1 Outline of External Bus

12.2 External bus configuration registers

12.3 External Memory Access Control Signal Operation

12.4 Notes on using the external bus

12.5 External Boot Vector fetch

12.6 Pin status in different MCU states

## 12.1 Outline of External Bus

The External Bus interface of the F$^2$MC-16FX MCU provides various access methods and access areas.

### 12.1.1 Features of the External Bus Interface

This chapter summarizes the features of the External Bus Interface

#### Features of the External Bus Interface

- 16 data lines
- Up to 24 address lines (depending on the device)
- Multiplexed data/address lines
- Non-multiplexed address/data lines (available for devices with high pin-count)
- Up to 6 chip select signals, four of them programmable in memory area
- Wait state request
- External bus master possible
- Support for WRXL, WRXH protocol and for WRX, UBX, LBX protocol
- Timing programmable

### 12.1.2 Terminology

This chapter explains some basic terms of the External Bus Interface.

#### Bus mode

Bus mode means the mode for controlling the internal ROM operation and external access function. The bus mode depends on the EAE[5:0] and ERE bits of the External Bus Mode register (EBM).

#### Access mode

Access mode means the mode for controlling the external bus format.

The data width of the external bus is specified by the BW-bit of the External Area Configuration registers (EACL[5:0]).

If supported by the device, it is possible to select between address/data multiplex and address/data non-multiplex by setting the NMS bit of the External Bus Mode register. This setting is valid for all external areas.

### External Boot Vector

Devices with an external bus interface offer the possibility to fetch the Boot Vector (User program start address) and a mode byte from an external memory instead of using the internal data (Internal Vector mode). The mode byte which is read together with the Boot Vector is copied to the External Bus Mode register (EBM) and overwrites the initial setting.

## 12.1.3 External bus areas

Up to 6 external bus areas with dedicated Chip select signals can be configured independently.

The 16MB address area can be divided into 6 external address areas with corresponding Chip select signals.

### External bus areas

The 16MB address area of the F$^2$MC-16FX MCU can be divided into 6 External bus areas with dedicated configuration registers and Chip select signals. The address range of the external bus areas 0 and 1 is fixed, while the address range of the other areas can be programmed within the upper 15MB address space by using the External Area Select (EAS) Registers and the External Area Size Select bits in the External Area Configuration (EACH) registers.

Memory access outside activated address areas does not show any activity on the I/O cells associated to the external bus interface.

Table 12-1. Address range of external bus areas

| External area | available address area | corresponding chip select | default address area |
|---|---|---|---|
| External area 0 | h’00.00f0...h’00.00ff | CS0 | h’00.00f0...h’00.00ff (fixed) |
| External area 1 | h’00.0c00...(RAM-Start - 1) | CS1 | h’00.0c00...(RAM-Start - 1) (fixed) |
| External area 2 | h’10.0000...h’ff.ffff | CS2 | h’10.0000...h’3f.ffff (3MByte) |
| External area 3 | | CS3 | h’40.0000...h’7f.ffff (4MByte) |
| External area 4 | | CS4 | h’80.0000...h’bf.ffff (4MByte) |
| External area 5 | | CS5 | h’c0.0000...h’ff.ffff (=4MByte incl. external Reset Vector) |

*) RAM-Start is the start address of the internal RAM (device dependent). Please see the Datasheet for RAM memory size and start address.

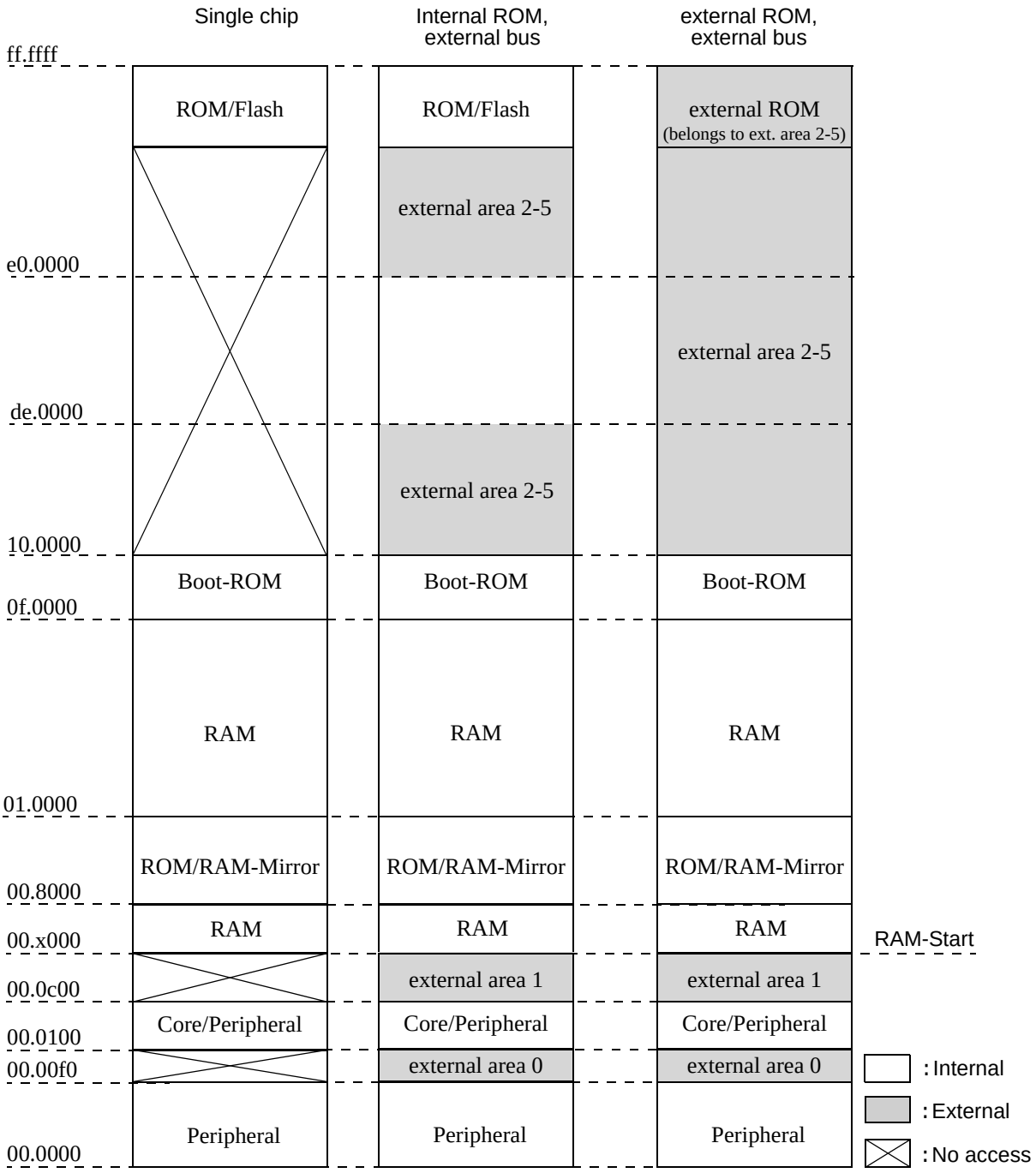
The following chapter explains the external bus areas with their default address ranges.

## 12.1.4 Memory Space in Each Bus Mode

Figure 12-1 shows the correspondence between the access areas and physical addresses for each bus mode.

**Memory Space in Each Bus Mode**

Figure 12-1. Relationship between Access Areas and Physical Addresses for Each Bus Mode



## 12.2 External bus configuration registers

The external bus contains various registers to change the external bus configuration for a wide range of external bus protocols:

- External Bus Mode register (EBM)
- External Bus Clock and Function register (EBCF)
- External Bus Address output Enable register [2:0] (EBAE[2:0])
- External Bus Control Signal register (EBCS)
- External Area [5:0] Configuration register (EACH/EACL[5:0])
- External Area Select register [5:2] (EAS[5:2])

### Arrangement of External bus configuration registers in memory

Table 12-2. External bus registers

| Address | upper byte | lower byte |
|---|---|---|
| $00:06E0/1_H$ | EACH0 | EACL0 |
| $00:06E2/3_H$ | EACH1 | EACL1 |
| $00:06E4/5_H$ | EACH2 | EACL2 |
| $00:06E6/7_H$ | EACH3 | EACL3 |
| $00:06E8/9_H$ | EACH4 | EACL4 |
| $00:06EA/B_H$ | EACH5 | EACL5 |
| $00:06EC/D_H$ | EAS3 | EAS2 |
| $00:06EE/F_H$ | EAS5 | EAS4 |
| $00:06F0/1_H$ | EBCF | EBM |
| $00:06F2/3_H$ | EBAE1 | EBAE0 |
| $00:06F4/5_H$ | EBCS | EBAE2 |

### Overview of External bus configuration registers

Figure 12-2. External bus configuration registers

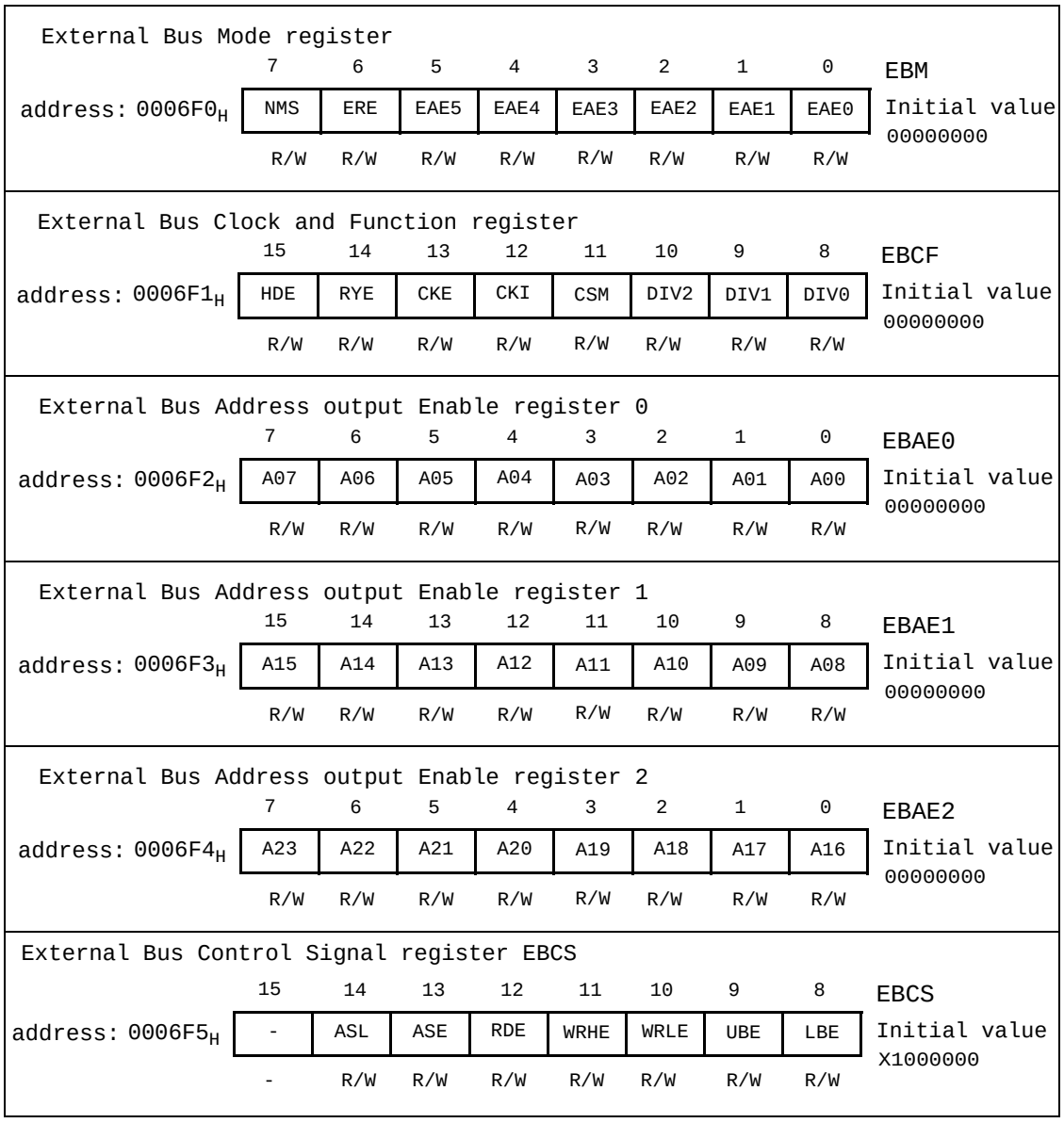


Figure 12-3. External bus configuration registers

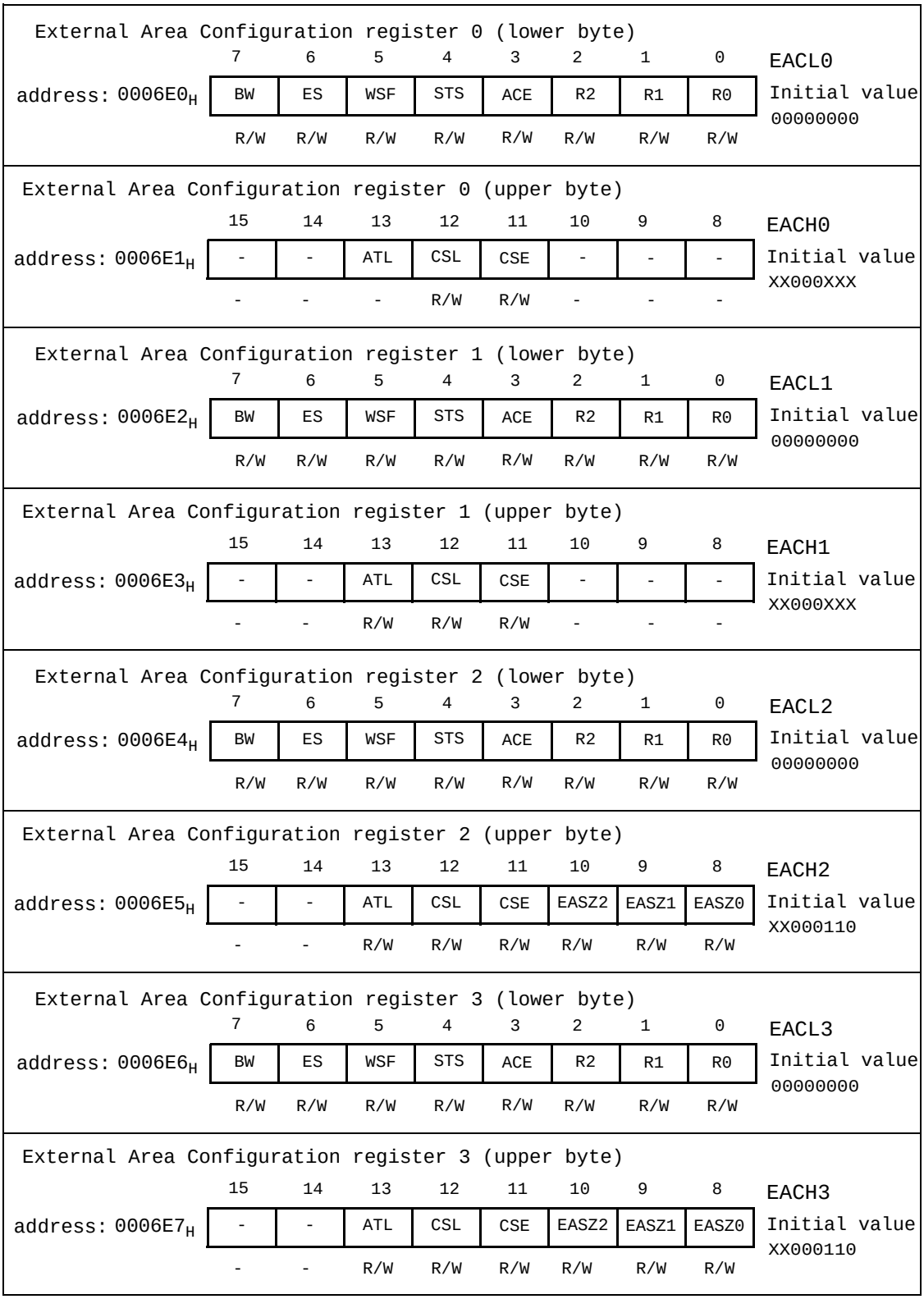
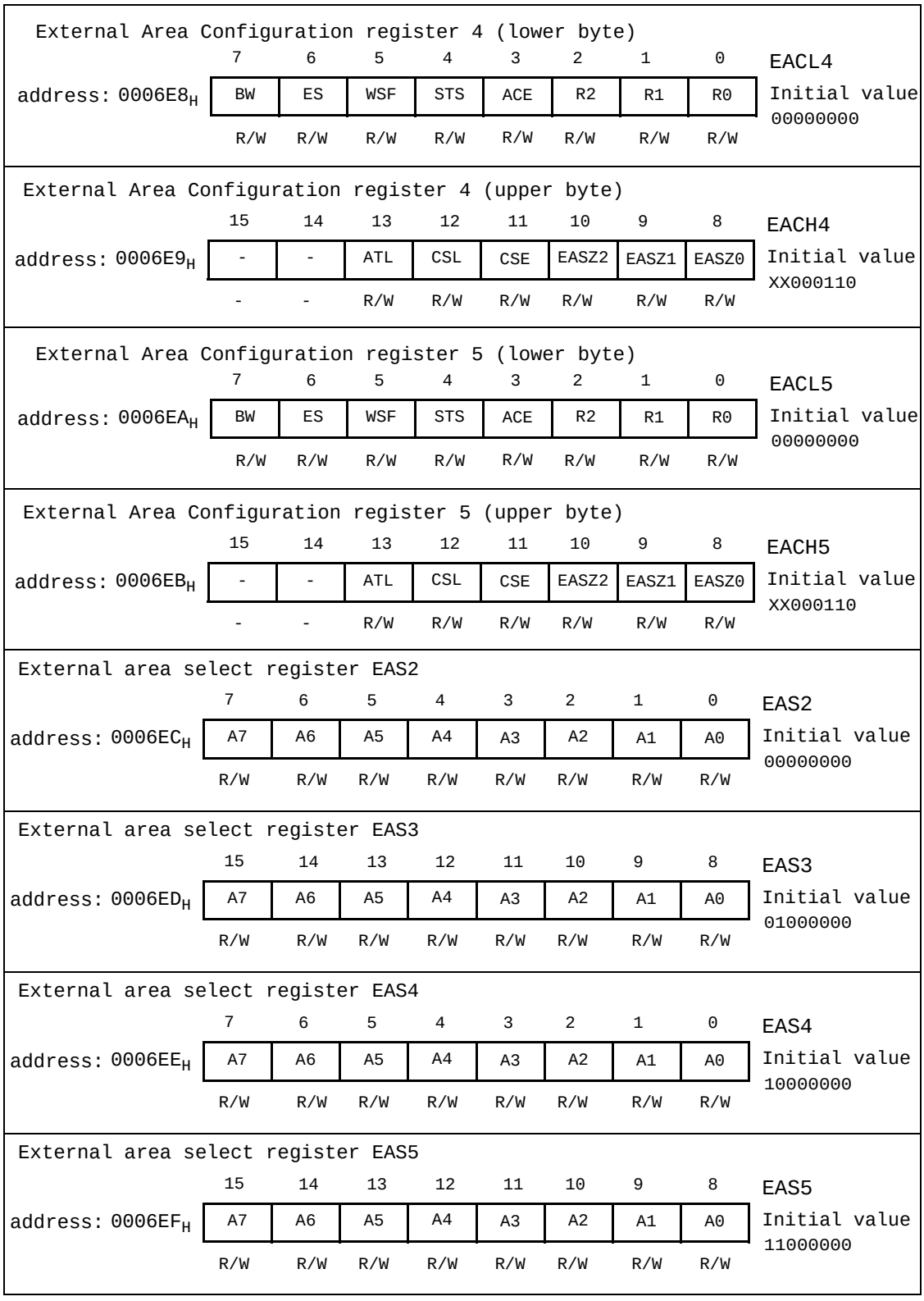


Figure 12-4. External bus configuration registers



## 12.2.1 External Bus Mode registers (EBM)

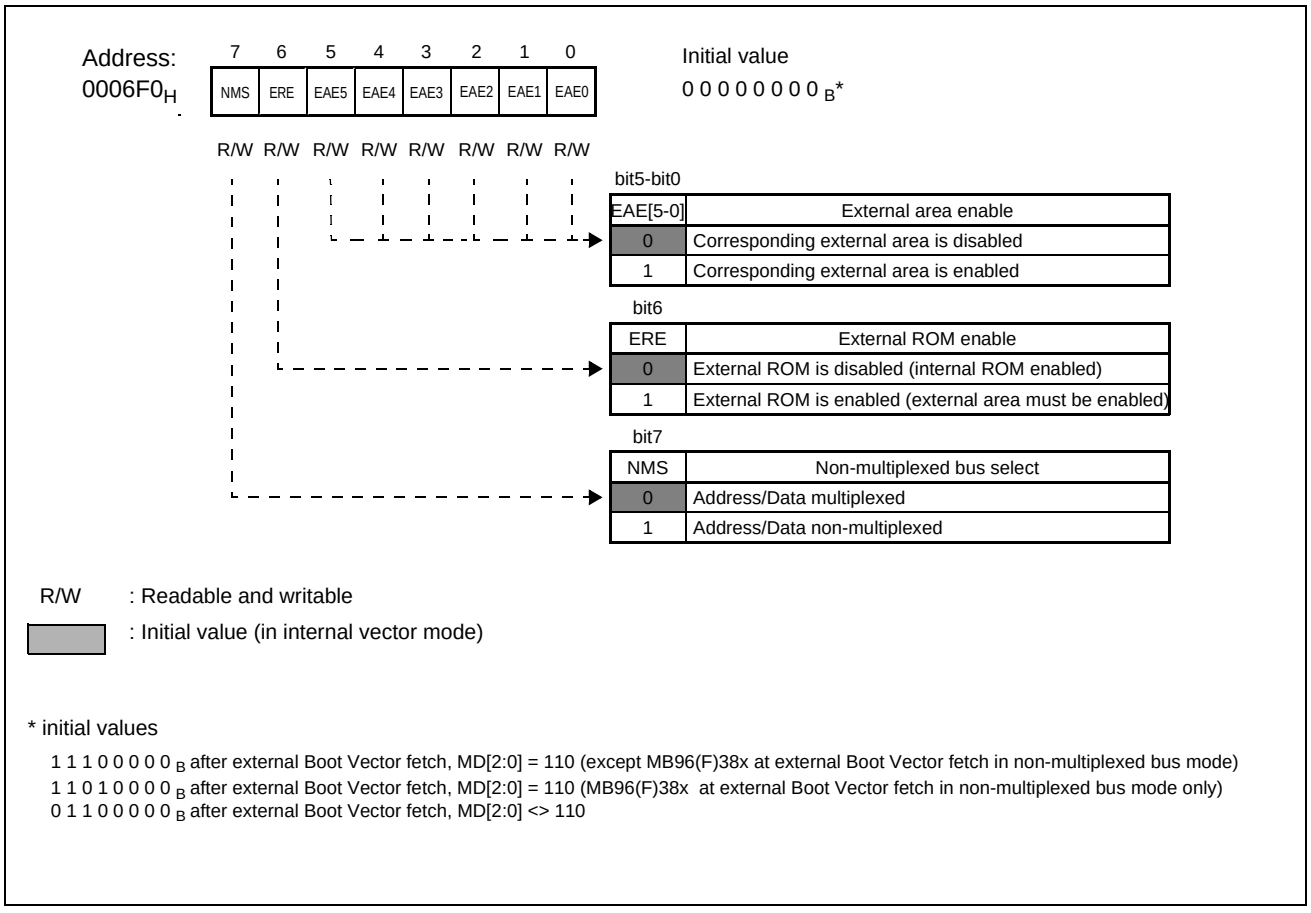
The External Bus Mode register defines the bus mode of the external bus interface. For the External Boot Vector fetch, the external bus mode byte is read externally from the address $FFFFDF_H$ and written to the register directly after reading the Boot Vector.

Following settings can be done in the EBM register:

- activation of all external bus areas
- switching between internal ROM access and external access
- multiplexed/non-multiplexed external bus access mode

### External Bus Mode register

Figure 12-5. External Bus Mode register (EBM)




* initial values

1 1 1 0 0 0 0 0 B after external Boot Vector fetch, MD[2:0] = 110 (except MB96(F)38x at external Boot Vector fetch in non-multiplexed bus mode)
1 1 0 1 0 0 0 0 B after external Boot Vector fetch, MD[2:0] = 110 (MB96(F)38x at external Boot Vector fetch in non-multiplexed bus mode only)
0 1 1 0 0 0 0 0 B after external Boot Vector fetch, MD[2:0] <> 110

| Bit name | | Function |
|---|---|---|
| bit [5:0] | EAE[5:0]:<br>External area enable | • The EAE (External Area Enable) bits are used to activate the External address areas<br>• '0' - the corresponding external address area is disabled. All other settings for this area have no effect.<br>• '1' - the corresponding external address area is enabled. The setting of the corresponding EACH/EACL and EAS registers define the characteristics of the address area. Configure these registers before setting the EAE bit to '1'.<br>• External ROM can only be used if an external area is enabled accordingly.<br>• The initial value of EAE[5:0] bits is '0'.<br>• Before the external Boot Vector fetch EAE[5] is set to '1', (except MB96(F)38x for startup in non-multiplexed bus mode where EAE[4] is set to '1'.<br>• After the Boot Vector fetch, the EBM register is overwritten by the externally read External bus mode byte. See also Chapter 12.5 External Boot Vector fetch. |
| bit 6 | ERE: External ROM enable | • This bit selects if the internal ROM is accessed or an external access is performed for the address area of the internal ROM<br>• '0' - the internal ROM (Flash) is accessed (Internal ROM mode)<br>• '1' - the access is redirected to the external bus (External ROM mode)<br>• An external bus access is performed only if an external address area is activated with the EAE bit which includes the address of the current access.<br>• The initial value is '0'. During external Boot Vector fetch, this bit is set to '1'. After the external Boot Vector fetch, the EBM register is overwritten by the externally read External bus mode byte. See also Chapter 12.5 External Boot Vector fetch. |
| bit 7 | NMS:<br>Non-multiplexed bus mode | • This bit selects the non-multiplexed or multiplexed bus mode for all external areas.<br>• '0' - selects multiplexed bus mode for all external areas<br>• '1' - selects non-multiplexed bus mode for all external areas<br>• It depends on the device which modes are available.<br>• The initial value is '0'. During external Boot Vector fetch, this bit is set according to the access mode selected by the mode pins MD[2:0]. After the Boot Vector fetch, the EBM register is overwritten by the externally read External bus mode byte. See also Chapter 12.5 External Boot Vector fetch. |

In External Vector mode, this register is configured by automatically reading the External bus mode byte (after reading the Boot Vector) externally from address $FFFFDF_H$ and writing the data to the register (see Chapter 12.5 External Boot Vector fetch for more information).

The External bus mode register can be rewritten by software at any time.

If the MCU starts in Internal Vector mode, this register keeps its default values.

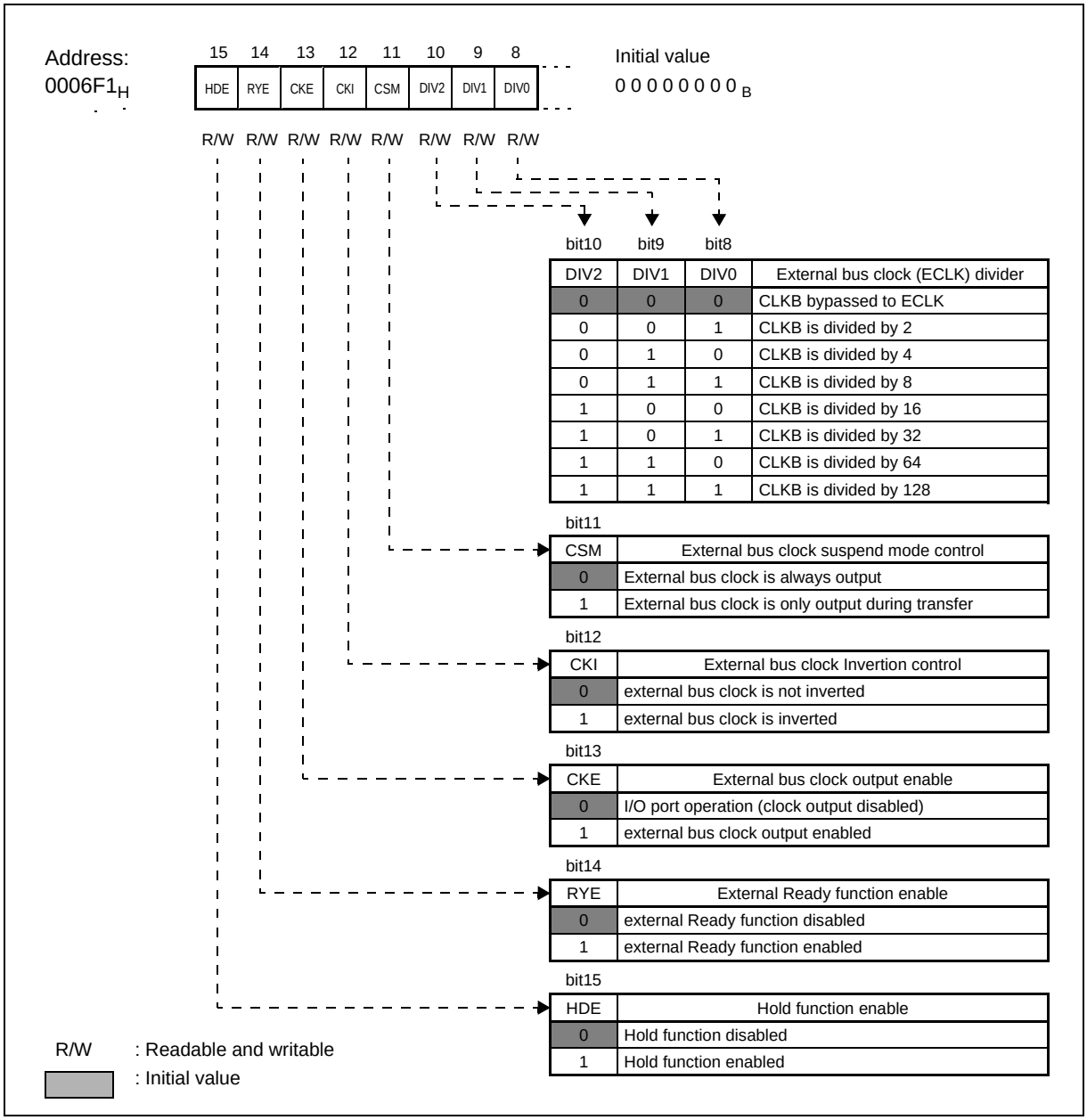
### 12.2.2 External Bus Clock and Function register (EBCF)

The External Bus Clock and Function register controls the external bus clock, the External Ready and the Hold function:

- Hold-function
- External Ready function
- Clock output enable
- Clock active edge
- Clock output mode
- Clock divider

### External Bus Clock and Function register

Figure 12-6. External Bus Clock and Function register (EBCF)




This register is not changed when the MCU starts in external vector mode.

The External Bus function is not guaranteed when changing the clock configuration bits (EBCF:DIV) while in operation. See also CKFCR:BCD bits configuration in chapter Clock Frequency Control Register (CKFCR) on page 132.

Table 12-3. Function description of each bit of the External bus clock and Function register

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit[10: 8]</td><td>DIV[2:0]:<br>External bus clock divider setting</td><td>
• These bits control the clock divider for the external bus clock signal pin (ECLK). The reference clock is the CPU clock (CLKB) selected by Clock control registers. All operations of the external bus are based on the divided clock (ECLK) even if the output of this clock is not enabled. The clock divider can only be changed for all external areas in common.
<table>
<tr><th>DIV[2:0]</th><th>External bus clock (ECLK) divider setting</th></tr>
<tr><td>000</td><td>CLKB bypassed to ECLK (divided by 1)</td></tr>
<tr><td>001</td><td>ECLK = CLKB / 2</td></tr>
<tr><td>010</td><td>ECLK = CLKB / 4</td></tr>
<tr><td>011</td><td>ECLK = CLKB / 8</td></tr>
<tr><td>100</td><td>ECLK = CLKB / 16</td></tr>
<tr><td>101</td><td>ECLK = CLKB / 32</td></tr>
<tr><td>110</td><td>ECLK = CLKB / 64</td></tr>
<tr><td>111</td><td>ECLK = CLKB / 128</td></tr>
</table>
• The initial value is ’000’ (ECLK = CLKB)<br>
• If the clock divider is set to another value than ’bypassed’ the transfer on the external bus needs multiple system bus cycles. Therefore the system bus is stopped during external bus transfer. This means that all internal operations on the system bus (CPU, DMA) wait until the external bus access it completed.
</td></tr>
<tr><td>bit 11</td><td>CSM:<br>External bus clock suspend mode</td><td>
• This bit controls the oscillation of the external bus clock (ECLK) on the corresponding pin.<br>
• The initial value is ’0’.<br>
• CSM = ’0’: The external bus clock (ECLK) is always oscillating and output also in Hold state.<br>
• CSM = ’1’: The behavior depends on the setting of the EBCF:DIV[2:0] bits:<br>
- EBCF:DIV[2:0] ≠ "000": The external bus clock (ECLK) is set to inactive level when the external bus is inactive (clock suspend mode). In Hold state, the ECLK pin is set to high-impedance.<br>
- EBCF:DIV[2:0] = "000": The external bus clock is also oscillating when no access to the external bus is done (no clock suspend mode available). However in Hold state, the ECLK pin is set to high-impedance as for the other settings of DIV[2:0].<br>
• See section 12.4 Notes on using the external bus for more details.<br>
• The inactive level depends on the setting of CKI (External Clock Inverting Control bit).<br>
• The setting of this bit influences only the output of the clock to the corresponding pin.
</td></tr>
</table>

Table 12-3. Function description of each bit of the External bus clock and Function register

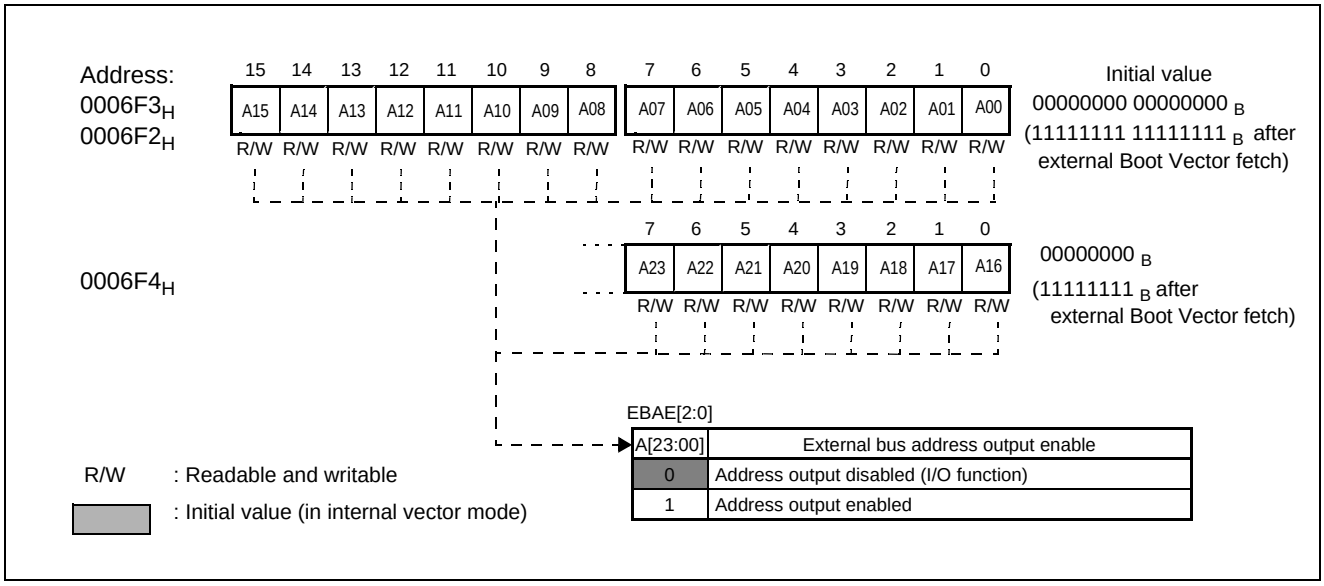
| Bit name | | Function |
|---|---|---|
| bit 12 | CKI:<br>External bus clock inverting | • This bit controls the active edge of the external bus clock.<br>• '0' - the inactive level is '0' and the rising edge is the active edge.<br>• '1' - the inactive level is '1' and the falling edge is the active edge.<br>• The initial value is '0'.<br>• The output of the external bus clock to the CLK-pin depends also on the CKE and CSM bits.<br>• The setting of this bit influences only the output of the clock to the corresponding pin, but not the internal function. |
| bit 13 | CKE:<br>External bus clock output enable | • This bit controls the output of the external bus clock signal pin (CLK).<br>• '0' - The external bus clock is not output to the corresponding pin. The CLK-pin is controlled by the corresponding DDR-register.<br>• '1' - The external bus clock is output to the corresponding pin.<br>• The initial value is '0'.<br>• Do not enable the external bus clock before all other clock settings are completed. |
| bit 14 | RYE:<br>External Ready function enable | • This bit enables the external ready function of the external bus.<br>• '0' - external Ready function for all external areas is disabled.<br>• '1' - external Ready function for all external areas is enabled.<br>• The initial value is '0'.<br>• The input of the corresponding pin for RDY is not enabled by setting this bit. This must be done by software in the Input enable register (PIER) and the Data direction register (DDR) by setting the corresponding PIER to '1' and the DDR to '0'.<br>• For a detailed description of the Ready function see chapter 12.3.3 Ready Function. |
| bit 15 | HDE:<br>Hold function enable | • This bit enables the Hold function of the external bus.<br>• '0' - Hold function is disabled.<br>• '1' - Hold function is enabled.<br>• The initial value is '0'.<br>• If the Hold function is enabled, the output of Hold Acknowledge pin ($\overline{\text{HAK}}$) is always enabled. The input of the corresponding pin of HRQ must be enabled by Software in the Input enable register (PIER) and Data direction register (DDR) by setting the corresponding PIER to '1' and the DDR to '0'.<br>• For a detailed description of Hold function see chapter 12.3.4 Hold Function. |

## 12.2.3 External Bus Address output Enable registers (EBAE[2:0])

The External Bus Address output Enable registers controls the output function of each address line.

### External Bus Address output Enable register

Figure 12-7. External Bus Address output Enable register EBAE[2:0]




| Bit name | | Function |
|---|---|---|
| bit [23:00] | A[23:00]:<br>External bus address output enable | • These bits enable the output function of the corresponding address line.<br>• In multiplexed external bus mode, EBAE[1:0] control the address output of the shared AD[15:00] address/data pins. In non-multiplexed external bus mode, these registers control the address output on the exclusive address pins A[15:00].<br>• '0' - the corresponding address line is not output to the pin. Instead the I/O function of the pin is enabled or another resource can be used.<br>• '1' - the corresponding address line is output to the pin during an external bus access.<br>• The initial value is '0', but changed to '1' in external Boot Vector fetch.<br>• If the device is started in external vector mode, all address outputs are enabled. |

The output of the address is disabled by default, hence it must be enabled before the external bus can be used. If the MCU is started in external vector mode, the Boot-ROM program enables all address lines before reading the external Boot Vector.

All pins used for address outputs must be enabled by the corresponding bit of the EBAE register. This includes shared address/data pins in the multiplexed external bus mode.

The address output of all address and address/data pins is active only during an external bus access. The address is not driven in a pause between two external bus accesses. Hence during an access pause, the state of the address pins depend on the setting of the corresponding port function DDR and PDR registers. Depending on the setting of these bits, the address pin can be set to Hi-Z or driven to '1' or '0'. Activating the internal pull-up resistor is also possible.

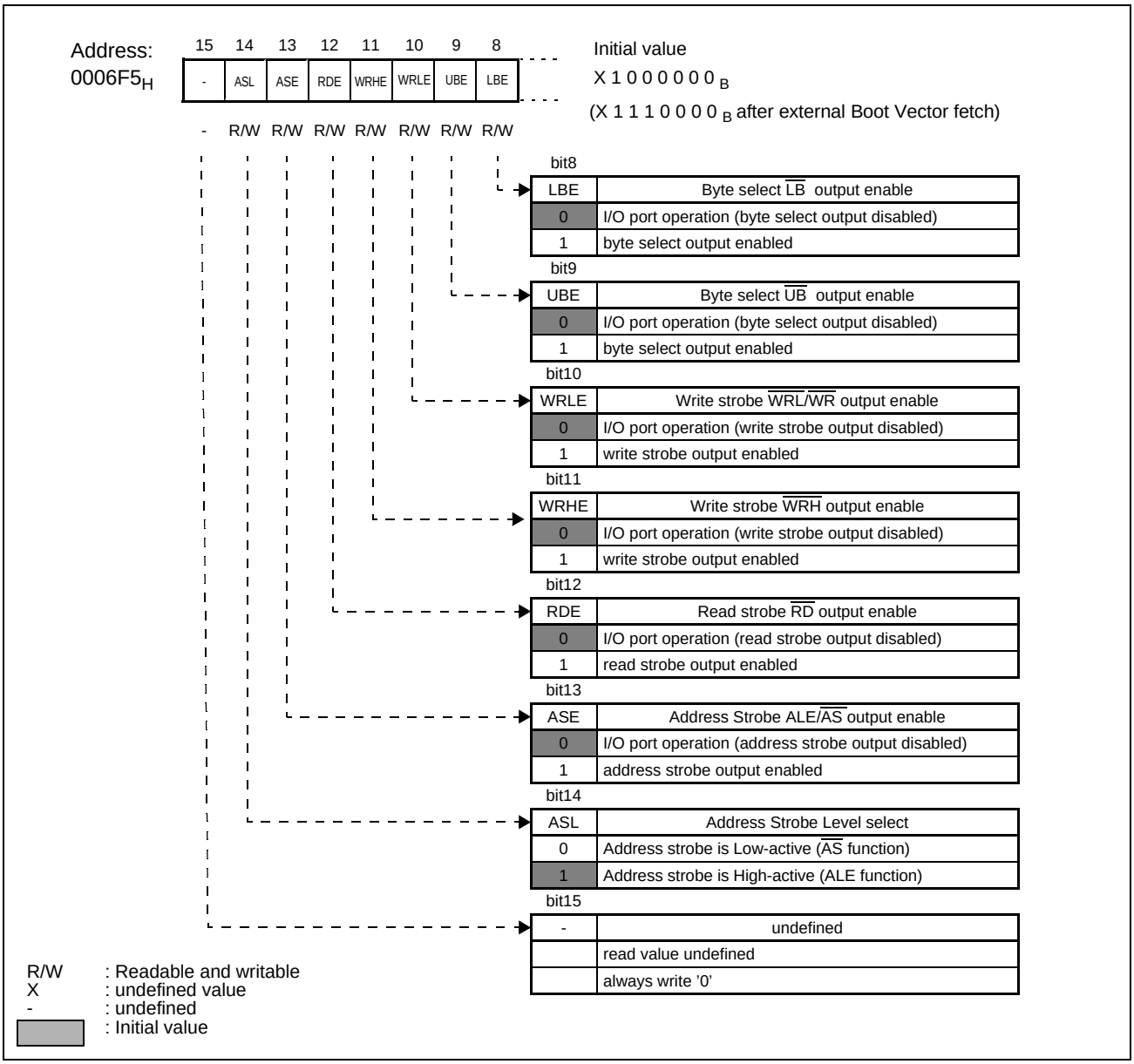
## 12.2.4 External Bus Control Signal register (EBCS)

This register is used to configure control signals of the external bus:

- Activation of Write strobe, Read strobe and Byte select outputs
- Activation of the Address strobe output and selection of the active level

### External Bus Control Signal register

Figure 12-8. External Bus Control Signal register (EBCS)



| Bit name | | Function |
|---|---|---|
| bit 8 | LBE:<br>Byte Select $\overline{LB}$ output enable | • This bit enables the output of the $\overline{LB}$ byte select signal.<br>• '0' - output of the $\overline{LB}$ byte select signal is disabled (I/O port function enabled)<br>• '1' - output of the $\overline{LB}$ byte select signal is enabled.<br>• The initial value is '0'. Hence it is not possible to use $\overline{LB}$ controlled external devices for reading the reset vector.<br>• For LBE ='1', the $\overline{LB}$ pin is forced to OUTPUT, independent of the corresponding bit in the Data Direction Register (DDR). |
| bit 9 | UBE:<br>Byte Select $\overline{UB}$ output enable | • This bit enables the output of the $\overline{UB}$ byte select signal.<br>• '0' - output of the $\overline{UB}$ byte select signal is disabled (I/O port function enabled)<br>• '1' - output of the $\overline{UB}$ byte select signal is enabled.<br>• The initial value is '0'. Hence it is not possible to use $\overline{UB}$ controlled external devices for reading the reset vector.<br>• For UBE ='1', the $\overline{UB}$ pin is forced to OUTPUT, independent of the corresponding bit in the Data Direction Register (DDR). |
| bit 10 | WRLE:<br>Write strobe $\overline{WRL}$/$\overline{WR}$ output enable | • This bit enables the output of the $\overline{WRL}$/$\overline{WR}$ write strobe signal.<br>• '0' - output of the $\overline{WRL}$/$\overline{WR}$ write strobe signal is disabled (I/O port function enabled).<br>• '1' - output of the $\overline{WRL}$/$\overline{WR}$ write strobe signal is enabled.<br>• The initial value is '0'.<br>• For WRLE ='1', the $\overline{WRL}$/$\overline{WR}$ pin is forced to OUTPUT, independent of the corresponding bit in the Data Direction Register (DDR). |
| bit 11 | WRHE:<br>Write strobe $\overline{WRH}$ output enable | • This bit enables the output of the $\overline{WRH}$ write strobe signal.<br>• '0' - output of the $\overline{WRH}$ write strobe signal is disabled (I/O port function enabled).<br>• '1' - output of the $\overline{WRH}$ write strobe signal is enabled.<br>• The initial value is '0'.<br>• For WRHE ='1', the $\overline{WRH}$ pin is forced to OUTPUT, independent of the corresponding bit in the Data Direction Register (DDR). |
| bit 12 | RDE:<br>Read strobe $\overline{RD}$ output enable | • This bit enables the output of the $\overline{RD}$ read strobe signal.<br>• '0' - output of the $\overline{RD}$ read strobe signal is disabled (I/O port function enabled).<br>• '1' - output of the $\overline{RD}$ read strobe signal is enabled.<br>• The initial value is '0', but changed to '1' in external Boot Vector fetch.<br>• For RDE ='1', the $\overline{RD}$ pin is forced to OUTPUT, independent of the corresponding bit in the Data Direction Register (DDR). |
| bit 13 | ASE:<br>Address Strobe ALE/$\overline{AS}$ output enable | • This bit enables the output of the ALE/$\overline{AS}$ address strobe signal.<br>• '0' - output of the ALE/$\overline{AS}$ address strobe signal is disabled (I/O port function enabled).<br>• '1' - output of the ALE/$\overline{AS}$ address strobe signal is enabled.<br>• The initial value is '0', but changed to '1' in external Boot Vector fetch mode.<br>• For ASE ='1', the ALE/$\overline{AS}$ pin is forced to OUTPUT, independent of the corresponding bit in the Data Direction Register (DDR). |
| bit 14 | ASL:<br>Address Strobe Level select | This bit selects the active level of the address strobe signal.<br>• '0' - active level is 'L', inactive level is 'H' ($\overline{AS}$ function).<br>• '1' - active level is 'H', inactive level is 'L' (ALE function).<br>• The initial value is '1'. |
| bit 15 | undefined | • Read value is undefined.<br>• Always write '0' to this bit.<br>• RMW operations are not affected. |

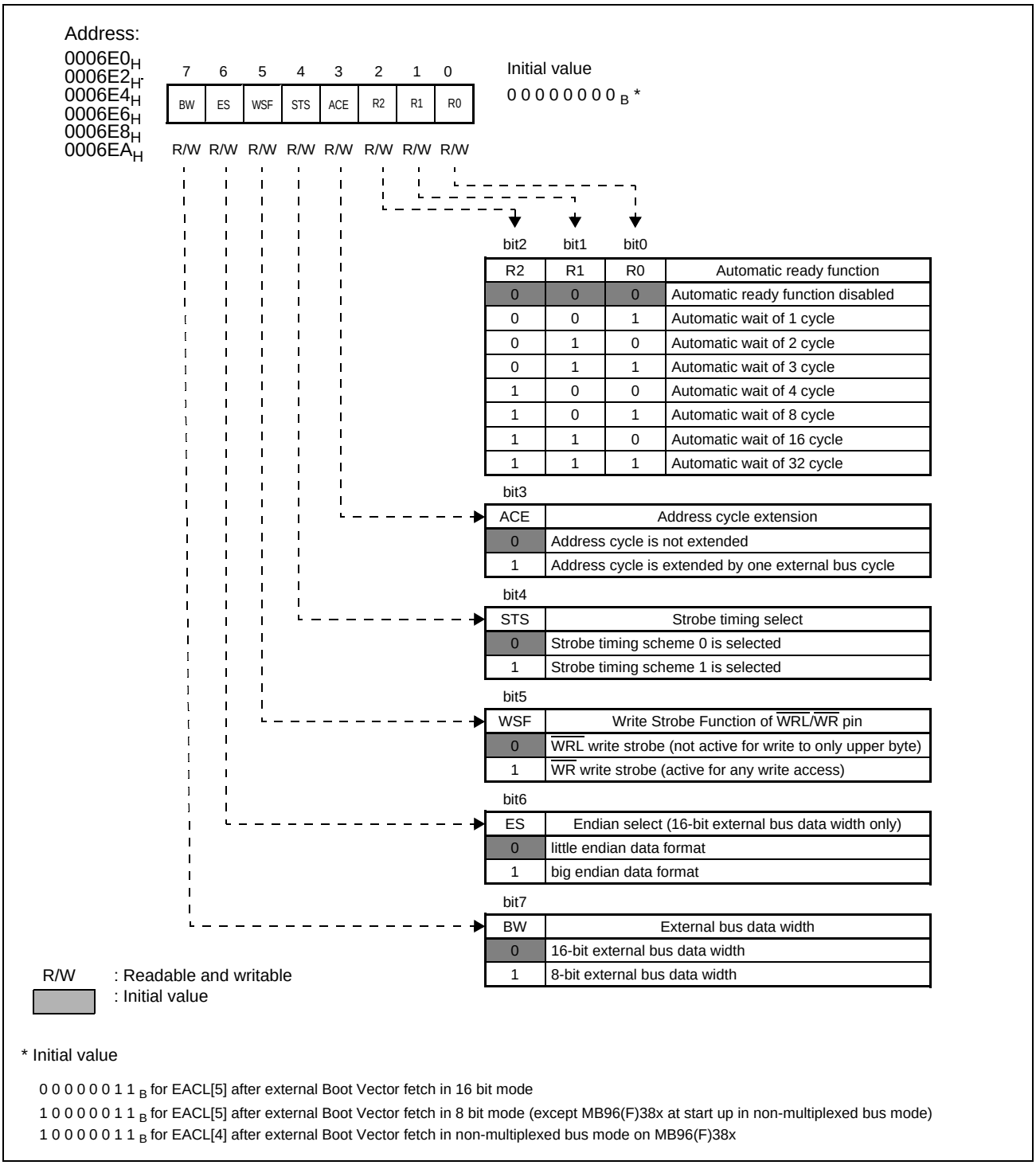
### 12.2.5 External Area Configuration registers (EACH/EACL[5:0])

The External Area Configuration registers are used to configure the following settings for each external area:

- Automatic ready function
- Address cycle length
- Strobe signal timing
- Write strobe function
- Little/big endian data format
- External bus data width
- External area size select
- Chip select output enable
- Chip select level select
- Access type limitation (code read from external area can be prohibited)

### External Area Configuration Register (lower byte) EACL[5:0]

Figure 12-9. External Area Configuration register (lower byte) EACL[5:0]



<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit [2:0]</td><td>R[2:0]:<br>automatic ready function</td><td>
• These bits enable the automatic ready function and select the number of wait cycles for the External area.
<table>
<tr><th></th><th>bit2</th><th>bit1</th><th>bit0</th><th></th></tr>
<tr><th></th><th>R2</th><th>R1</th><th>R0</th><th>Automatic ready function</th></tr>
<tr><td></td><td>0</td><td>0</td><td>0</td><td>Automatic ready function disabled</td></tr>
<tr><td></td><td>0</td><td>0</td><td>1</td><td>Automatic wait of 1 cycle</td></tr>
<tr><td></td><td>0</td><td>1</td><td>0</td><td>Automatic wait of 2 cycle</td></tr>
<tr><td></td><td>0</td><td>1</td><td>1</td><td>Automatic wait of 3 cycle</td></tr>
<tr><td></td><td>1</td><td>0</td><td>0</td><td>Automatic wait of 4 cycle</td></tr>
<tr><td></td><td>1</td><td>0</td><td>1</td><td>Automatic wait of 8 cycle</td></tr>
<tr><td></td><td>1</td><td>1</td><td>0</td><td>Automatic wait of 16 cycle</td></tr>
<tr><td></td><td>1</td><td>1</td><td>1</td><td>Automatic wait of 32 cycle</td></tr>
</table>
• The initial value is ’000’. R[2:0] of EACR[5] however are changed to ’011’ by the Boot ROM program in case of an external Boot Vector fetch,except om MB96(F)38x at external Boot Vector fetch in non-multiplexed bus mode where R[2:0] of EACR[4] is set to ’011’.
</td></tr>
<tr><td>bit 3</td><td>ACE:<br>Address cycle extension</td><td>
• This bit selects the length of the address cycle in multiplexed external bus mode.<br>
• ’0’ - Standard address cycle timing.<br>
• ’1’ - The address cycle is extended by one external bus clock cycle.<br>
• The initial value is ’0’.<br>
• The setting of this bit has no effect in non-multiplexed external bus mode.<br>
• See timing diagrams in Section 12.3 External Memory Access Control Signal Operation for more details.
</td></tr>
<tr><td>bit 4</td><td>STS:<br>Strobe timing select</td><td>
• This bit selects the timing of the address and write strobe signals to adjust the signal timing for different external devices.<br>
• The initial value is ’0’.<br>
• See timing diagrams in Section 12.3 External Memory Access Control Signal Operation for more details regarding the two timings.
</td></tr>
<tr><td>bit 5</td><td>WSF:<br>Write Strobe function of $\overline{\text{WRL}}$/$\overline{\text{WR}}$ pin</td><td>
• This bit selects the Write strobe function of the $\overline{\text{WRL}}$/$\overline{\text{WR}}$ pin for the corresponding external area.<br>
• ’0’ - $\overline{\text{WRL}}$ function activated: The signal is not activated during a write access to the upper byte. Use this setting for external devices with separate write strobe inputs for the upper and lower bytes (WRH/$\overline{\text{WRL}}$ controlled).<br>
• ’1’ - $\overline{\text{WR}}$ function activated: The signal is activated during any write access. Use this setting for external devices with common write enable input for upper and lower bytes ($\overline{\text{UB}}$/$\overline{\text{LB}}$ controlled).<br>
• The initial value is ’0’.
</td></tr>
<tr><td>bit 6</td><td>ES:<br>endian select</td><td>
• This bit selects the data format in 16-bit mode for the address area indicated by the corresponding chip select signal<br>
• ’0’ - little endian<br>
• ’1’ - big endian<br>
• The initial value is ’0’.<br>
• see Table 12-4 for details
</td></tr>
<tr><td>bit 7</td><td>BW:<br>external bus data width</td><td>
This bit selects the data width of the external bus for the corresponding address area.<br>
• ’0’ - 16-bit data width selected.<br>
• ’1’ - 8-bit data width selected.<br>
• The initial value is ’0’. BW of EACR[5] however is changed to ’1’ by the Boot ROM program in case of an 8 bit wide external reset vector fetch, except om MB96(F)38x at ext. Boot Vector fetch in non-multiplexed bus mode where BW of EACR[4] is set to ’1’.
</td></tr>
</table>

The following table describes the effect of the endian selection for different external bus formats:

Table 12-4. Endian selection

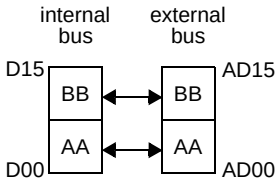
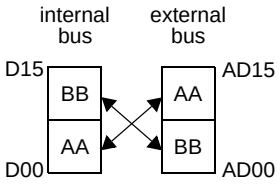
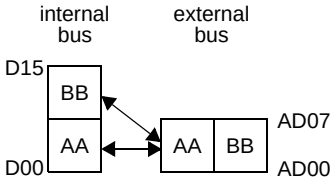
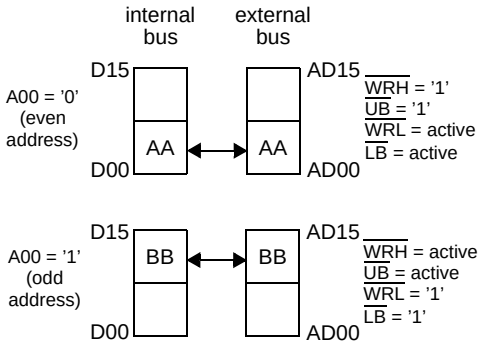
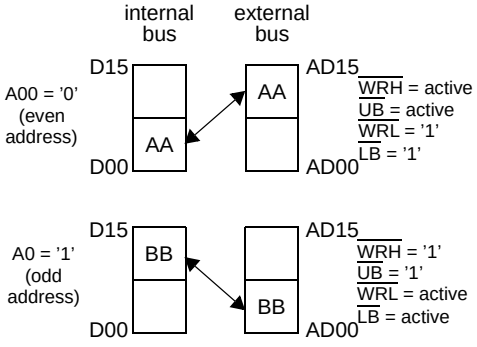
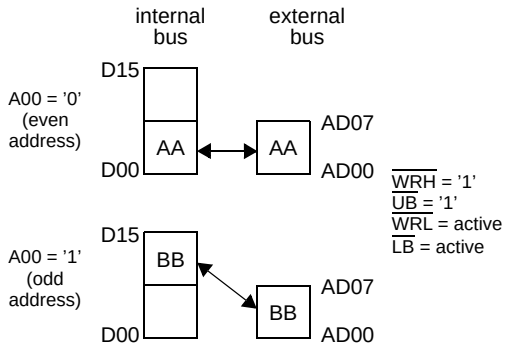
<table>
<tr><th rowspan="2">access width on internal bus</th><th rowspan="2">selected bus width of the external bus</th><th colspan="2">sequence on external bus</th></tr>
<tr><th>little endian (ES = ’0’)</th><th>big endian (ES = ’1’)</th></tr>
<tr><td rowspan="2">16-bit</td><td rowspan="2">16-bit</td><td>
</td><td>
</td></tr>
<tr><td colspan="2">• $\overline{WRH}$ and $\overline{WRL}$ are both activated during any write access.<br>• $\overline{UB}$ and $\overline{LB}$ are both activated during any write and read access.<br>• The address is always even (A00 = ’0’)</td></tr>
<tr><td rowspan="2">16-bit</td><td rowspan="2">8-bit</td><td colspan="2">
</td></tr>
<tr><td colspan="2">• The internal 16 bit access is split into two 8 bit accesses:<br>1. Transfer AA: External address equals to internal address (A00= ’0’)<br>2. Transfer BB: External address incremented by 1 (A00= ’1’)<br>• $\overline{WRL}$ is activated during any write access, $\overline{WRH}$ is fixed to ’1’.<br>• $\overline{LB}$ is activated during any write and read access, $\overline{UB}$ is fixed to ’1’.</td></tr>
</table>

Table 12-4. Endian selection

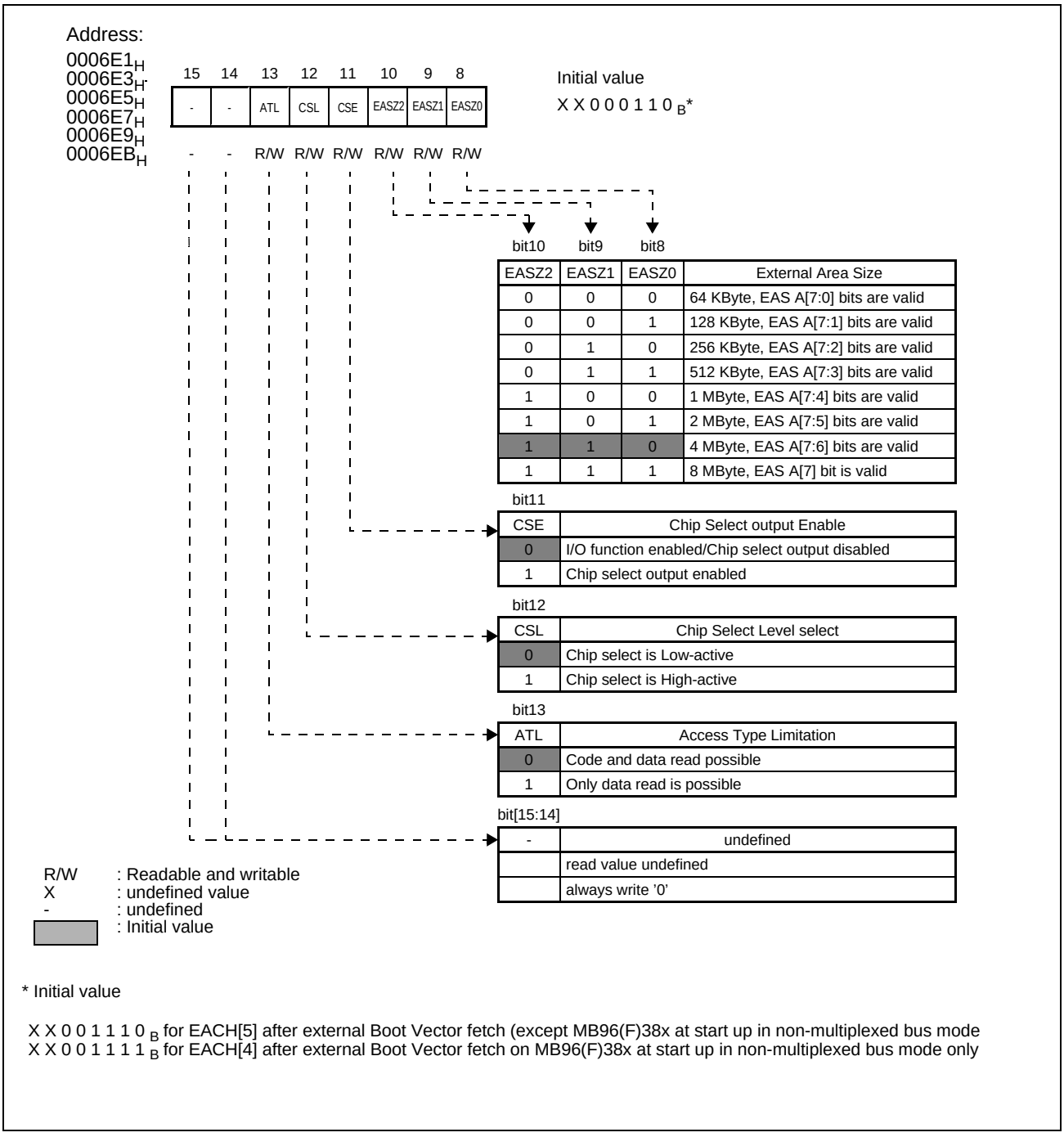
<table>
<tr><th rowspan="2">access width on internal bus</th><th rowspan="2">selected bus width of the external bus</th><th colspan="2">sequence on external bus</th></tr>
<tr><th>little endian (ES = ’0’)</th><th>big endian (ES = ’1’)</th></tr>
<tr><td>8-bit</td><td>16-bit</td><td>
</td><td>
</td></tr>
<tr><td colspan="2">• WRH and UB are used when data is accessed at the external bus data pins AD[15:08].<br>• WRL and LB are used when data is accessed at the external bus data pins AD[07:00].</td></tr>
<tr><td>8-bit</td><td>8-bit</td><td colspan="2">
</td></tr>
<tr><td colspan="2">• WRL is activated during any write access, WRH is fixed to ’1’.<br>• LB is activated during any write and read access, UB is fixed to ’1’.</td></tr>
</table>

- The endian selection affects the data ports AD[15:00] of the external bus, the write strobe signals WRH/WRL and the byte select signals UB/LB.
- The endian selection does NOT affect other signals as for example the address. In multiplexed bus mode, only the data cycle of the address/data pins is affected.

■ For WSF = ’1’, The WRL/WR pin changes its function to WR (asserted to ’0’ when either WRL or WRH is asserted).

### External Area Configuration Register (upper byte) EACH[5:0]

Figure 12-10. External Area Configuration register (upper byte) EACH[5:0]




Note: Bits 8 to 10 for Extra Area Size are not defined for EACH0 and EACH1 registers.

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit[10:8]</td><td>EASZ[2:0]:<br>External Area Size</td><td>
• These bits are used to define the size of the External area. The setting defines which bits of the corresponding External Area Select register (EAS) are used for the address comparison.
<table>
<tr><th>bit10<br>EASZ2</th><th>bit9<br>EASZ1</th><th>bit8<br>EASZ0</th><th>External Area Size</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>64 KByte, EAS A[7:0] bits are valid</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>128 KByte, EAS A[7:1] bits are valid</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>256 KByte, EAS A[7:2] bits are valid</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>512 KByte, EAS A[7:3] bits are valid</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1 MByte, EAS A[7:4] bits are valid</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>2 MByte, EAS A[7:5] bits are valid</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>4 MByte, EAS A[7:6] bits are valid</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>8 MByte, EAS A[7] bit is valid</td></tr>
</table>
• The initial value is ’110’. This means only the upper two bits of corresponding EAS register are evaluated and the size of the External area is 4 MByte.<br>
• These bits exist only for the External Area Configuration registers EACH[5:2]. The address size of the External areas 0 and 1 is fixed.<br>
• EASZ[2:0] is set to ’111’ (8 MByte) at external Boot Vector fetch in non-multiplexed bus mode on MB96(F)38x.
</td></tr>
<tr><td>bit 11</td><td>CSE:<br>Chip Select output enable</td><td>
• This bit enables the output of the corresponding Chip select signal.<br>
• ’0’ - Chip select output disabled (I/O port function enabled).<br>
• ’1’ - Chip select output enabled.<br>
• The initial value is ’0’. CSE of EACR[5] however is changed to ’1’ by the Boot ROM program in case of an external Boot Vector fetch, except at start up in non-multiplexed bus mode on MB96(F)38x where CSE of EACR[4] is set to ’1’.
</td></tr>
<tr><td>bit 12</td><td>CSL:<br>Chip Select level select</td><td>
This bit selects the active level of the corresponding Chip select signal.<br>
• ’0’ - active level is ’L’, inactive level is ’H’.<br>
• ’1’ - active level is ’H’, inactive level is ’L’.<br>
• The corresponding Chip select signal CSx has active level, if the corresponding address area is enabled and an access to this area occurs. If the address is inside the internal ROM/Flash area and an external bus access shall be initiated, also the external ROM enable bit must be set in the EBM register.<br>
• The initial value is ’0’.
</td></tr>
<tr><td>bit 13</td><td>ATL:<br>Access Type Limitation</td><td>
This bit selects if code read is possible from the External area.<br>
• ’0’ - Code read is permitted.<br>
• ’1’ - Code read is not possible. A read access returns an undefined instruction exception.<br>
• Data reading from the External area is always possible.<br>
• After setting this bit to ’1’, it is not possible to write it back to ’0. It can only be cleared by reset.<br>
• The initial value is ’0’.
</td></tr>
<tr><td>bit 14 - bit 15</td><td>undefined</td><td>
• The read value of these bits is undefined.<br>
• Always write ’0’ to these bits.<br>
• RMW operations are no affected.
</td></tr>
</table>

## 12.2.6 External Area Select register (EAS[5:2])

The external bus interface supports two types of External address areas:

- Areas with fixed address range (External areas 0 and 1)
- Areas with selectable address range (External areas 2 to 5)

### Overview

The external bus interface supports two types of address areas:

- Fixed address range (External areas 0 and 1):
  - External area 0 is fixed to address range $00.00f0_H$...$00.00ff_H$.

- External area 1 is fixed to address range $00.0c00_H$...RAM-Start.

  The RAM-Start depends on device configuration and may be different for various devices of the same series. It cannot be changed by the Software.

  The RAM-Start address on EVA-Devices can be configured and depends on the emulated device.

- Programmable address range (External areas 2 to 5):
  - The External Address Select register EAS for each area defines the upper 8 address bits of the corresponding external area, allowing addresses within 64 KB to be specified.
  - The EASZ[2:0] bits of the corresponding External Area Configuration register EACH[5:2] define how many bits of the EAS register are used for the address comparison. Hence External areas from 64 KByte to 8 MByte can be selected.
  - The address area below $10:0000_H$ however cannot be selected.

### External Area Select Register EAS[5:2]

Figure 12-11. External Area Select registers (EAS[5:2])

External area select register EAS[5:2]

address: EAS2: $0006EC_H$
EAS3: $0006ED_H$
EAS4: $0006EE_H$
EAS5: $0006EF_H$

| 15/7 | 14/6 | 13/5 | 12/4 | 11/3 | 10/2 | 9/1 | 8/0 | EAS[5:2] |
|---|---|---|---|---|---|---|---|---|
| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | Initial value |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | ******** |

These registers are used to set the address decode area for the External areas 2-5. They specify the upper 8 bits of the address value, allowing an address within 64kB to be specified.

Disable the corresponding External area in the External Bus Mode register (EBM) when changing the EAS register to avoid malfunction.

### Initial values of EAS[5:2]

The EAS[5:2] registers have different initial values. The initial values divide the memory area above address $10.0000_H$ in 4 sections. The external area 5 and the corresponding Chip select CS5 is used during external reset vector fetch, except on MB96(F)38x at external Boot Vector fetch in non-multiplexed bus mode, where external area 4 and CS4 are used. The value of EAS[5:2] can be changed by Software at any time.

Table 12-5. Initial values of EAS[5:2]

| External area | Initial value of EAS register | Default Memory area | Chip select signal |
|---|---|---|---|
| External Area 2 | EAS2= $00_H$<br>(EACH2: EASZ = "110") | h'10.0000...h'3f.ffff (3MByte) | CS2 |
| External Area 3 | EAS3= $40_H$<br>(EACH3: EASZ = "110") | h'40.0000...h'7f.ffff (4MByte) | CS3 |
| External Area 4 | MB96(F)38x at external Boot Vector fetch in non-multiplexed bus mode:<br>EAS4= $80_H$<br>(EACH4: EASZ = "111") | h'80.0000...h'ff.ffff (8MByte incl. external Reset vector) | CS4 |
| | all other devices and modes:<br>EAS4= $80_H$<br>(EACH4: EASZ = "110") | h'80.0000...h'bf.ffff (4MByte) | |
| External Area 5 | EAS5= $C0_H$<br>(EACH5: EASZ = "110") | h'c0.0000...h'ff.ffff<br>(=4MByte incl. external Reset vector) | CS5 |

### Operation of the External address area selection

When the CPU accesses program or data, an external bus access is performed if a match between the upper 8 bits of an address and any EAS[5:2] setting is detected (the corresponding External area must be activated in the EBM register). Depending on the setting of the corresponding EASZ[2:0] bits of the EACH register, a certain number of LSB bits of the EAS register is ignored, and decoding becomes possible for an area from 64 KB to 8 MB.

Examples:

EAS2= $F0_H$
EACH2: EASZ = "000"
- not decoded banks: 00...EF, F1-FF
- decoded banks: F0

EAS2= $F0_H$
EACH2: EASZ = "001"
- not decoded banks: 00...EF, F2-FF
- decoded banks: F0, F1

EAS2= $17_H$
EACH2: EASZ = "000"
- not decoded banks: 00...16, 18-FF
- decoded banks: 17

EAS2= $00_H$
EACH2: EASZ = "110"
- not decoded banks: 40...FF
- decoded banks: 00...3F
  00...0F: no access possible because not selectable for external area 2-5.
  10...3F: access possible

If an address inside the possible address range of the external bus is accessed but the address is not within the area of an activated external bus area, then the access has no effect (a read access returns ’0’ and a write access is ignored).

The address ranges of the external areas 2 to 5 should not overlap. If an address is included in the access range of more than one active external area, then the configuration of the external area with the lowest number has priority. The settings of the other EACL registers are not used.

The chip select signals of all areas which match with the current access are output (if enabled).

## 12.3 External Memory Access Control Signal Operation

This chapter describes the control signals during external bus operations.

### 12.3.1 Multiplexed external bus mode

**All timing charts are shown with following settings**

- CSx active level is 'L': EACHx:CSL='0'
- all address outputs are enabled: EBAE:A[23:00]='11...11'
- The inactive level of the external clock is '0': EBCF: CKI = '0'

**The following timing diagrams show the effects of the following settings:**

- Figure 12-12 shows a standard multiplex access in 16 bit external bus mode.
- Figure 12-13 and Figure 12-15 show the effect of the Strobe Time Select bit (EACL:STS).
- Figure 12-14 and Figure 12-15 show the effect of the Address Cycle Extension (EACL:ACE).
- Figure 12-16 shows a standard multiplex access in 8 bit external bus mode.
- Figure 12-17 shows the effect of the automatic ready function (one wait cycle inserted).

**Timing charts for multiplexed bus modes**

Figure 12-12. Multiplexed 16-bit bus: Word and Byte access for EACL:STS='0' and EACL:ACE='0'

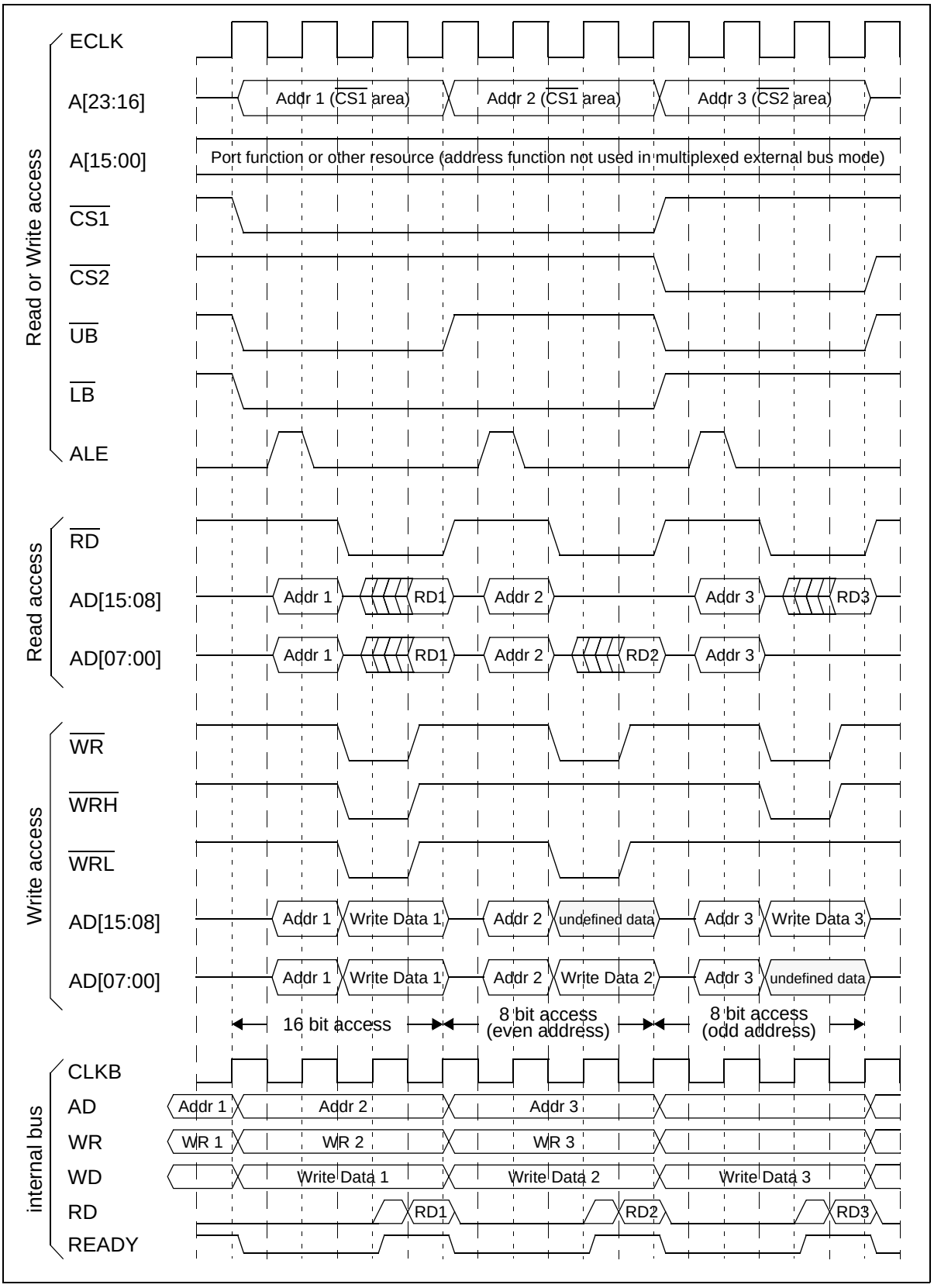


Figure 12-13. Multiplexed 16-bit bus: Word and Byte access for EACL:STS='1' and EACL:ACE='0'

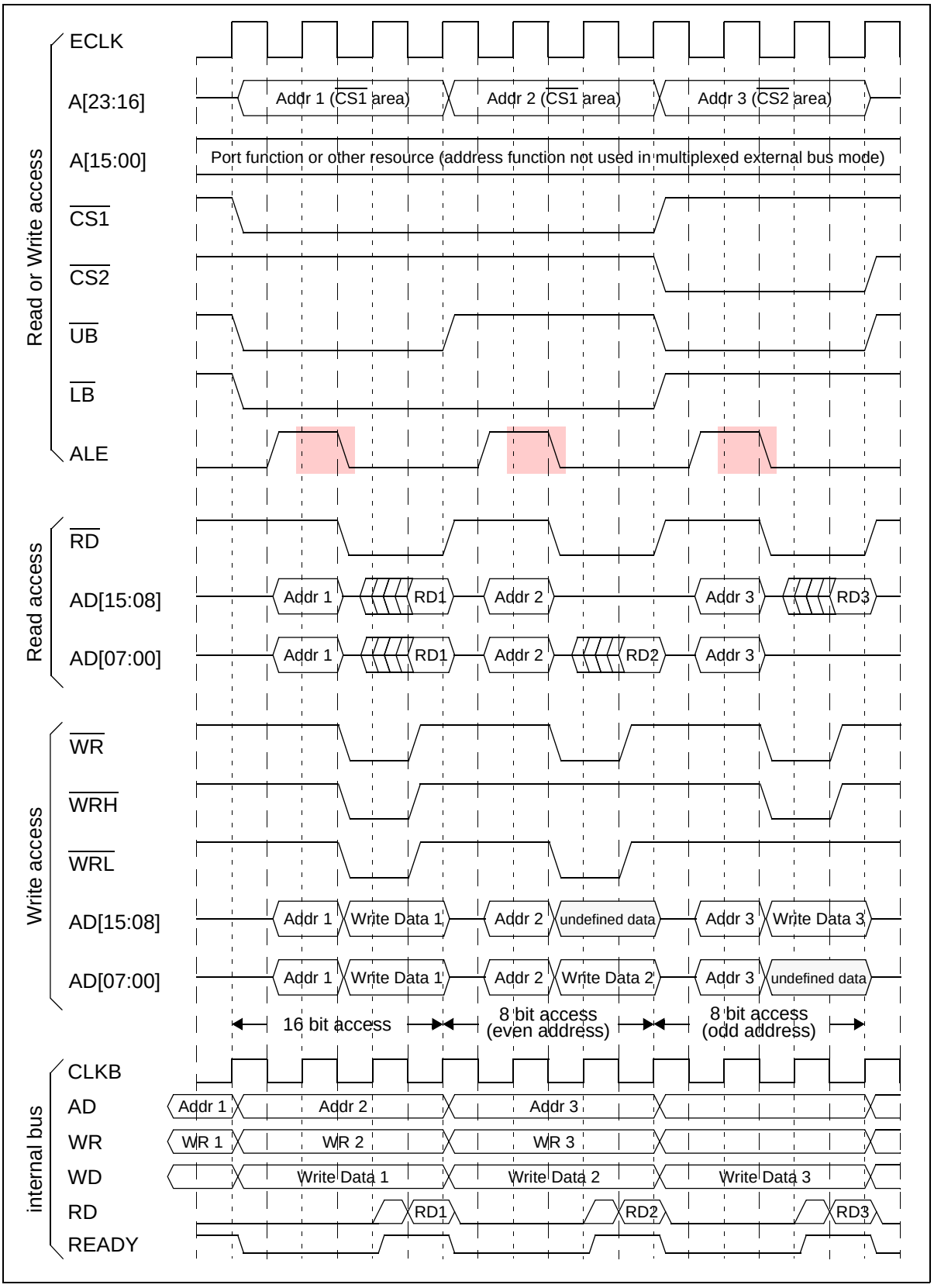


Figure 12-14. Multiplexed 16-bit bus: Word and Byte access for EACL:STS='0' and EACL:ACE='1'

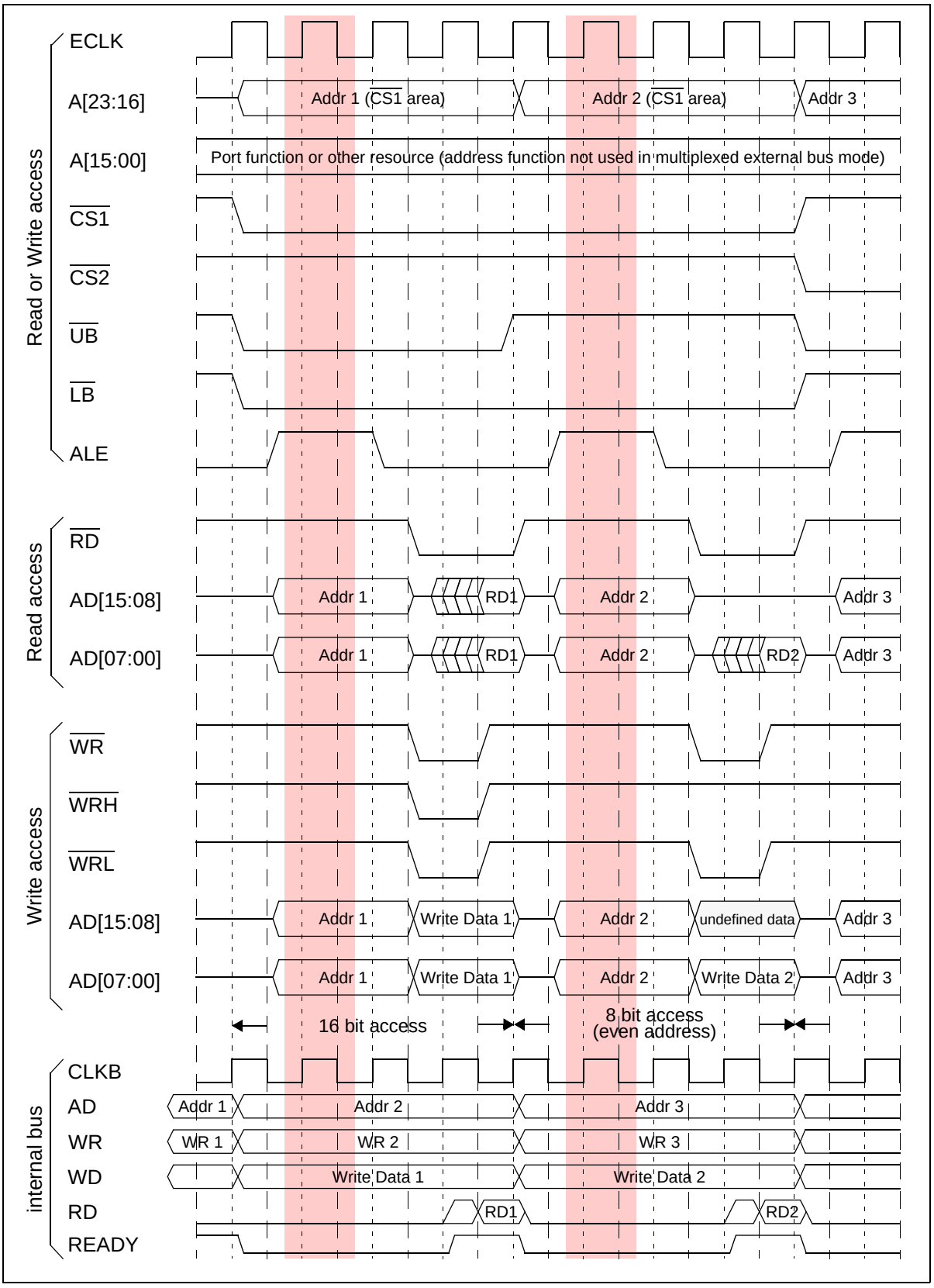


Figure 12-15. Multiplexed 16-bit bus: Word and Byte access for EACL:STS='1' and EACL:ACE='1'

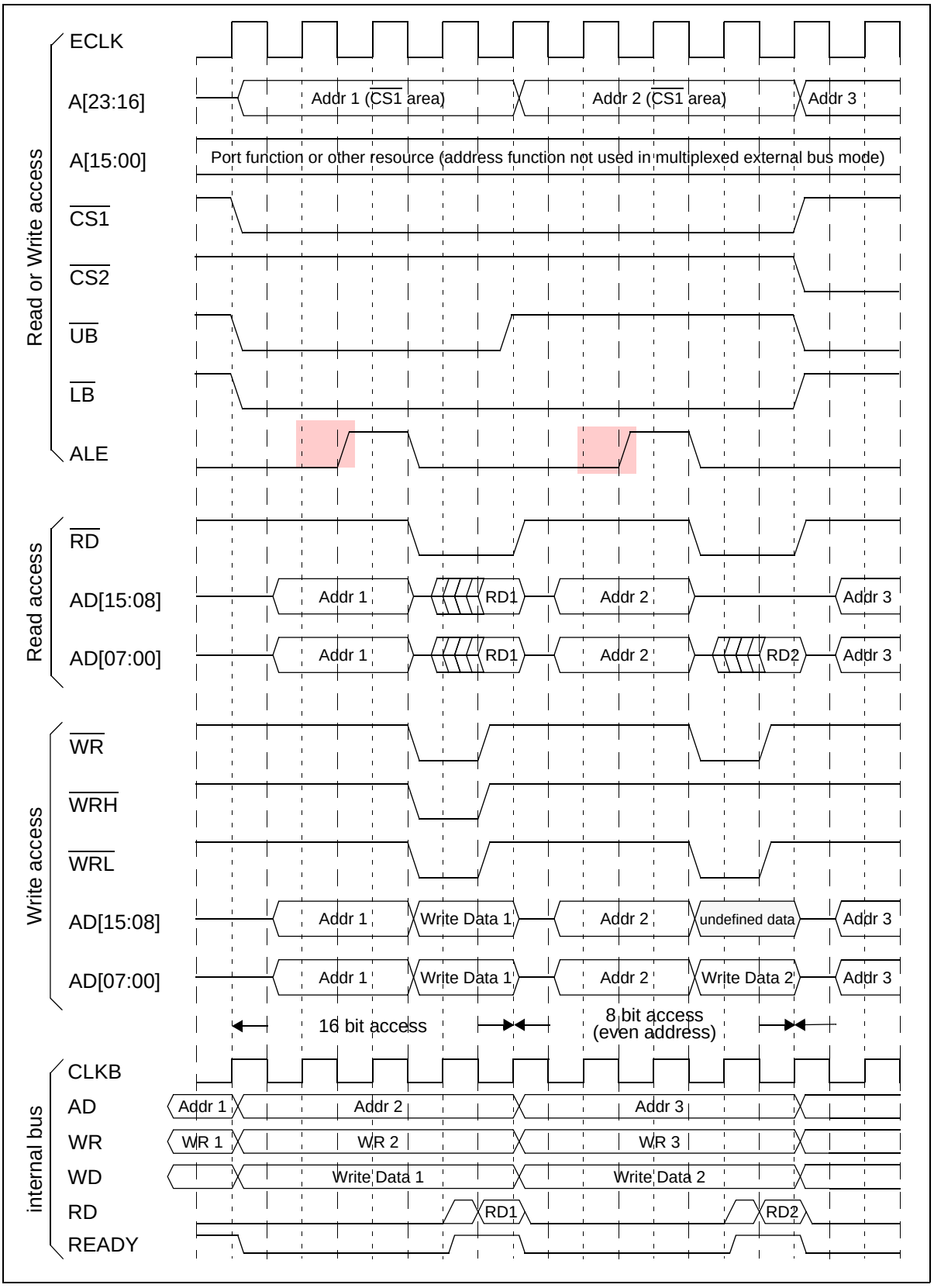
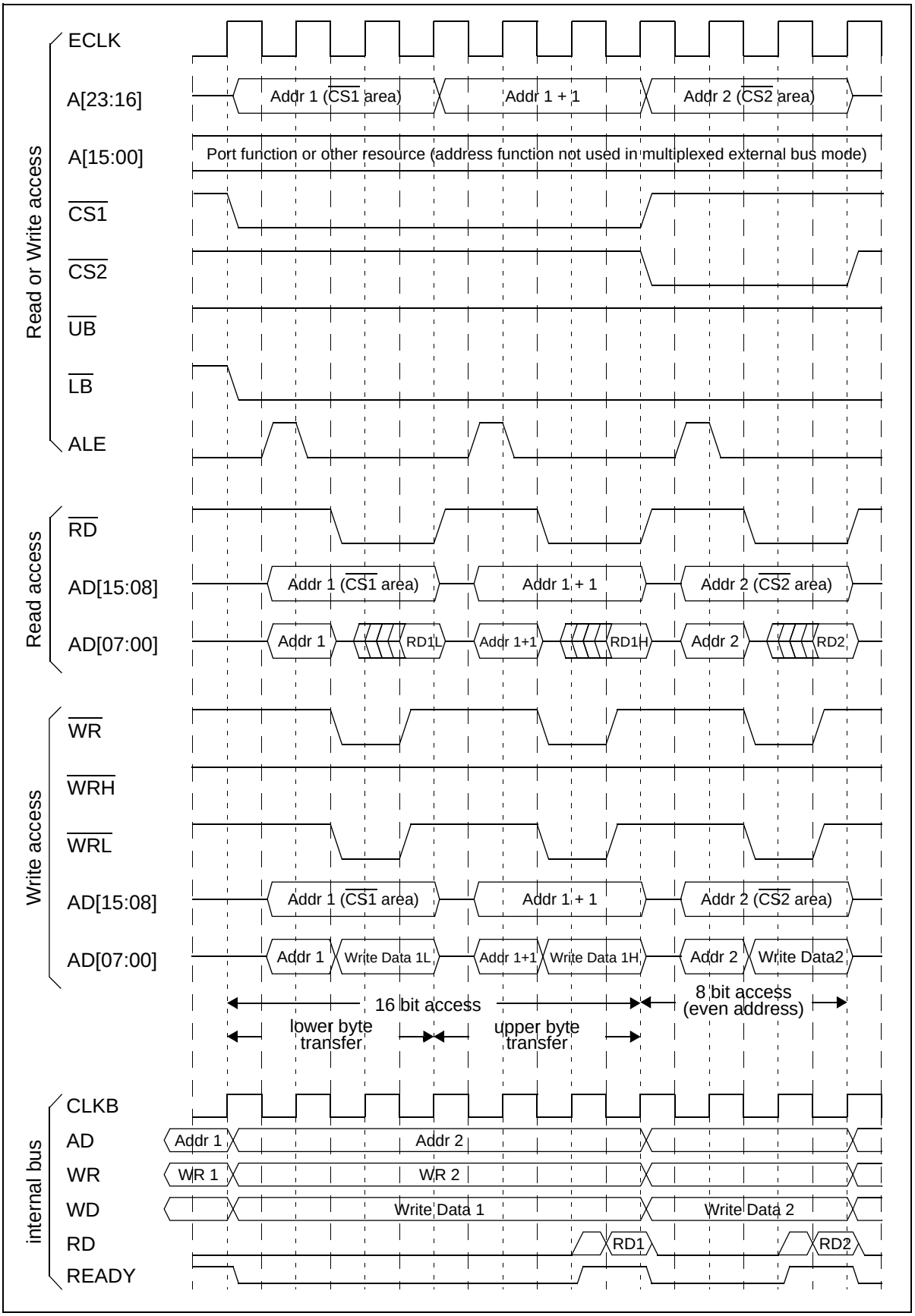
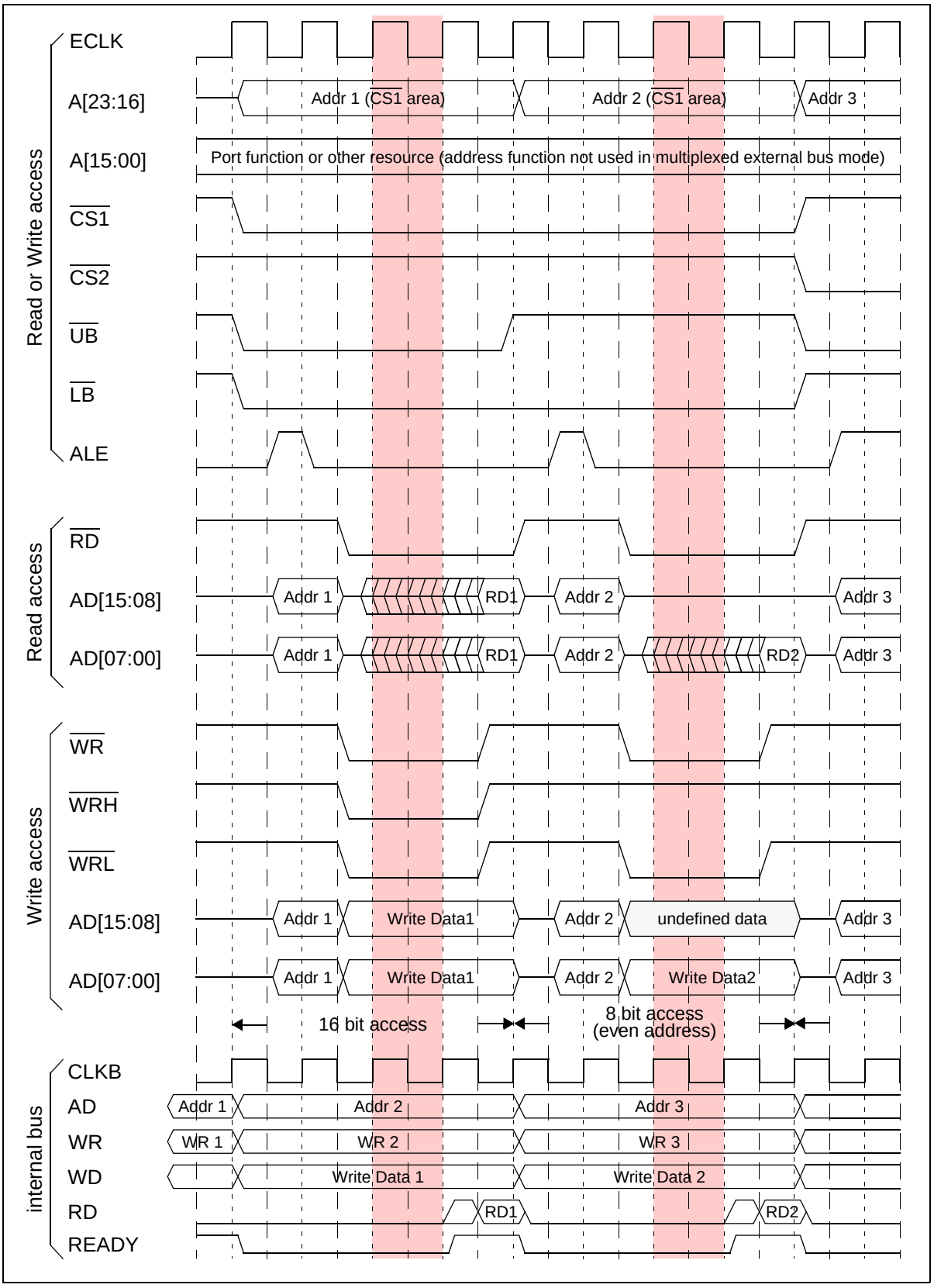


Figure 12-16. Multiplexed 8-bit bus: Word and Byte access for EACL:STS='0' and EACL:ACE='0'

Figure 12-17. Multiplexed 16-bit bus: One automatic ready wait cycle added (EACL:R[2:0]='001', EACL:STS='0', EACL:ACE='0)'


ECLK
A[23:16]
A[15:00]
CS1
CS2
UB
LB
ALE
Read or Write access
Addr 1 (CS1 area)
Addr 1 + 1
Addr 2 (CS2 area)
Port function or other resource (address function not used in multiplexed external bus mode)
RD
AD[15:08]
AD[07:00]
Read access
Addr 1
RD1L
Addr 1+1
RD1H
Addr 2
RD2
WR
WRH
WRL
AD[15:08]
AD[07:00]
Write access
Write Data 1L
Write Data 1H
Write Data2
16 bit access
8 bit access
(even address)
lower byte
transfer
upper byte
transfer
CLKB
AD
WR
WD
RD
READY
internal bus
WR 1
WR 2
Write Data 1
Write Data 2
RD1
RD2


Read or Write access
ECLK
A[23:16]
A[15:00]
CS1
CS2
UB
LB
ALE
Addr 1 (CS1 area)
Addr 2 (CS1 area)
Addr 3
Port function or other resource (address function not used in multiplexed external bus mode)
Read access
RD
AD[15:08]
AD[07:00]
Addr 1
RD1
Addr 2
Addr 3
RD2
Write access
WR
WRH
WRL
AD[15:08]
AD[07:00]
Write Data1
undefined data
Write Data2
16 bit access
8 bit access
(even address)
internal bus
CLKB
AD
WR
WD
RD
READY
WR 1
WR 2
WR 3
Write Data 1
Write Data 2

### 12.3.2 Non-multiplexed external bus mode

**All timing charts are shown with following settings**

- CSx active level is 'L': EACHx:CSL='0'
- all address outputs are enabled: EBAE:A[23:00]='11...11'
- The inactive level of the external clock is '0': EBCF: CKI = '0'

**The following timing diagrams show the effects of the following settings:**

- Figure 12-18 shows a standard non-multiplex access in 16 bit external bus mode.
- Figure 12-19 shows the effect of the Strobe Time Select bit (EACL:STS).
- Figure 12-20 shows a standard non-multiplex access in 8 bit external bus mode.
- Figure 12-21 shows the effect of the automatic ready function (one wait cycle inserted).

**Timing charts for non-multiplexed bus mode**

Figure 12-18. Non-multiplexed 16-bit bus: Word and Byte access for EACL:STS='0'

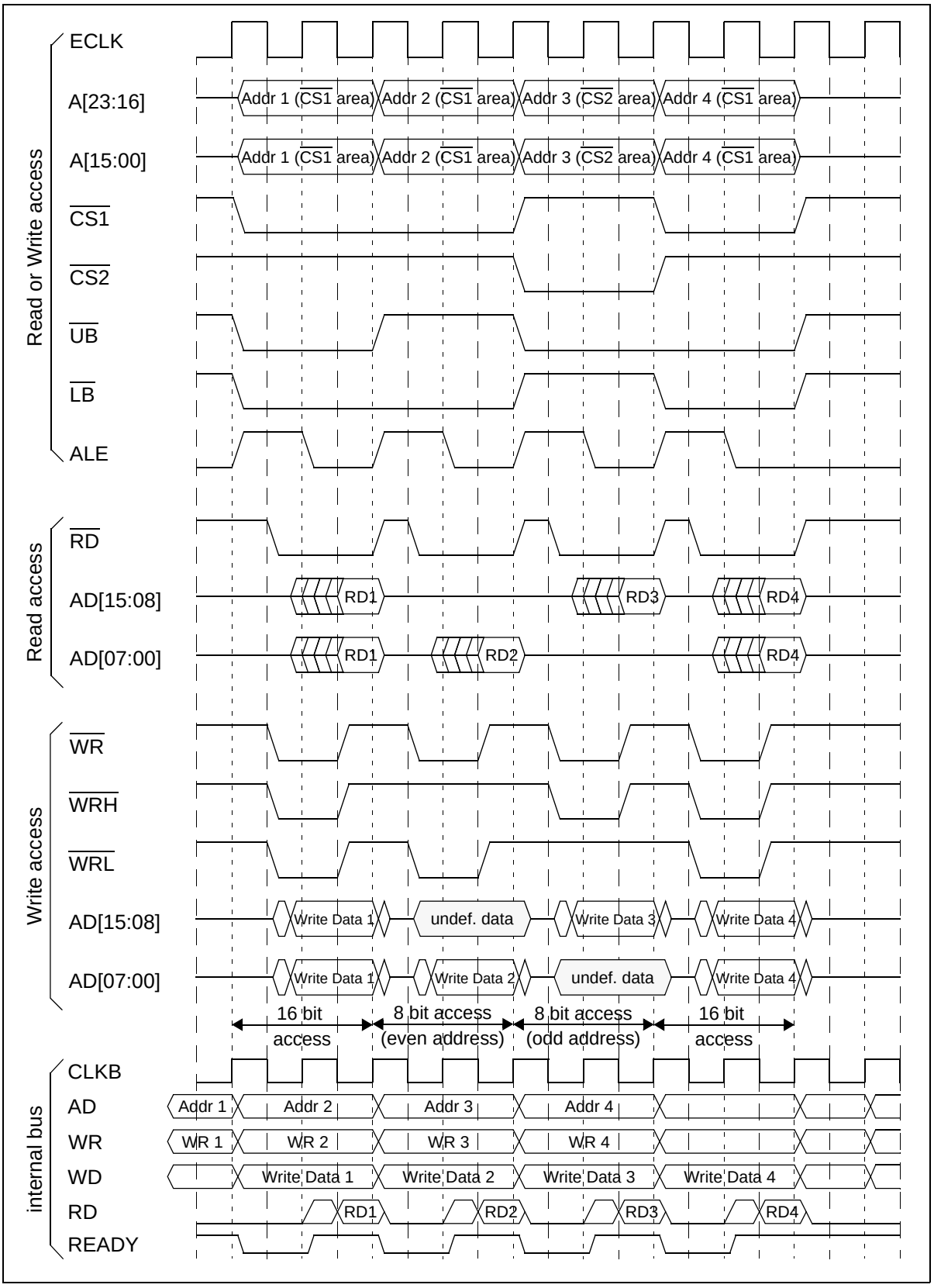


Figure 12-19. Non-multiplexed 16-bit bus: Word and Byte access for EACL:STS='1'

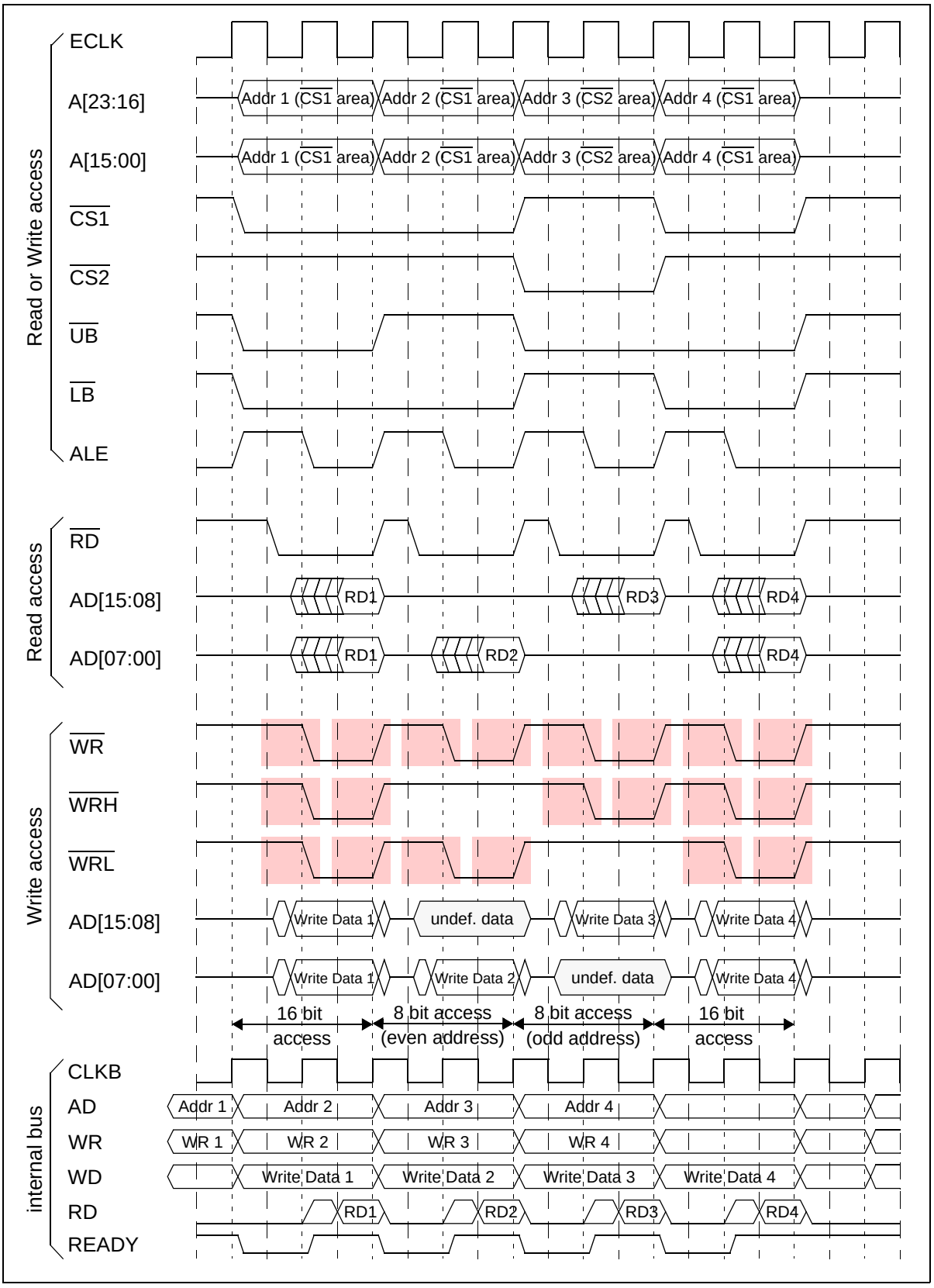


Figure 12-20.  Non-multiplexed 8-bit bus: Word and Byte access for EACL:STS=’0’ and EACL:ACE=’0’

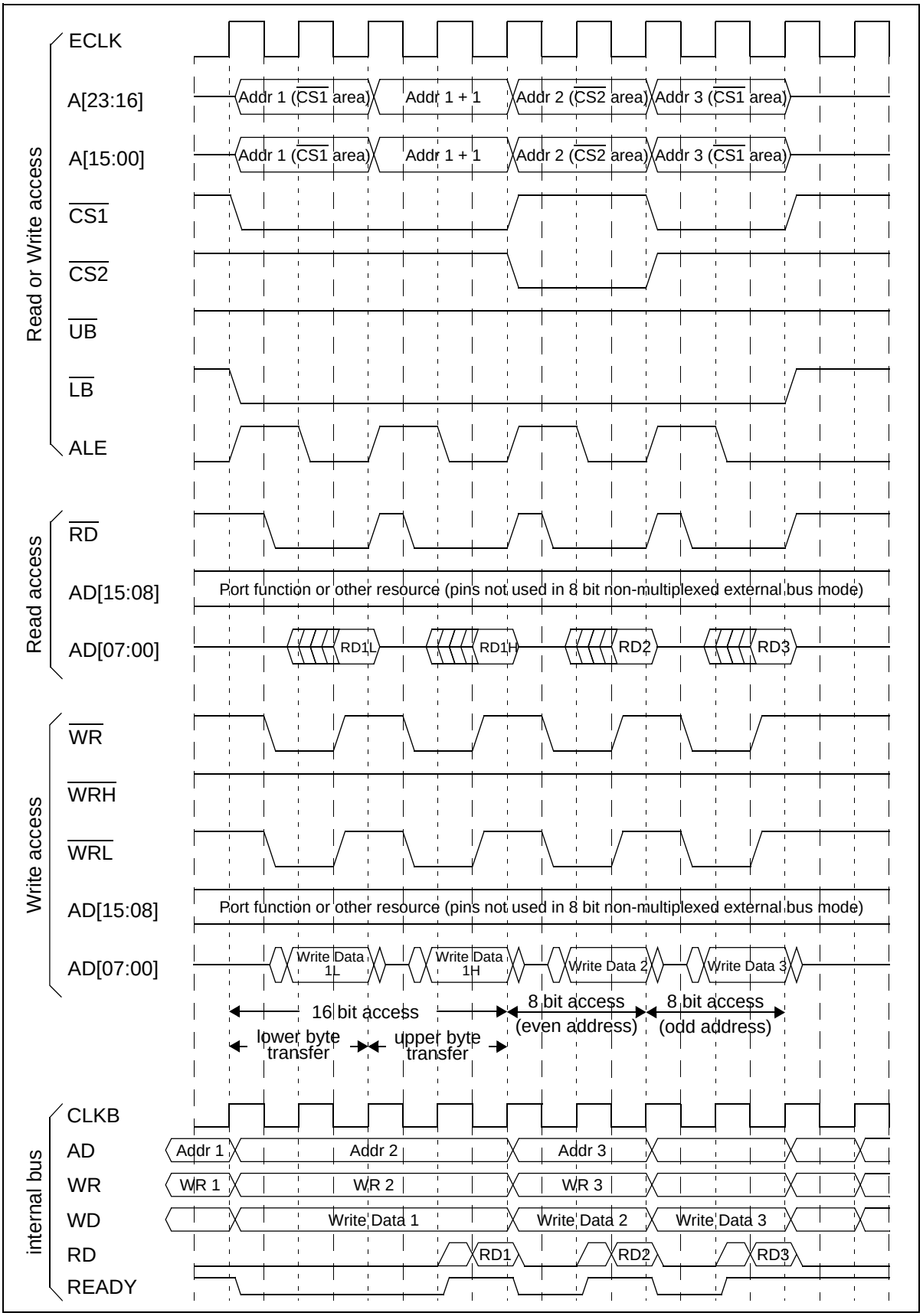
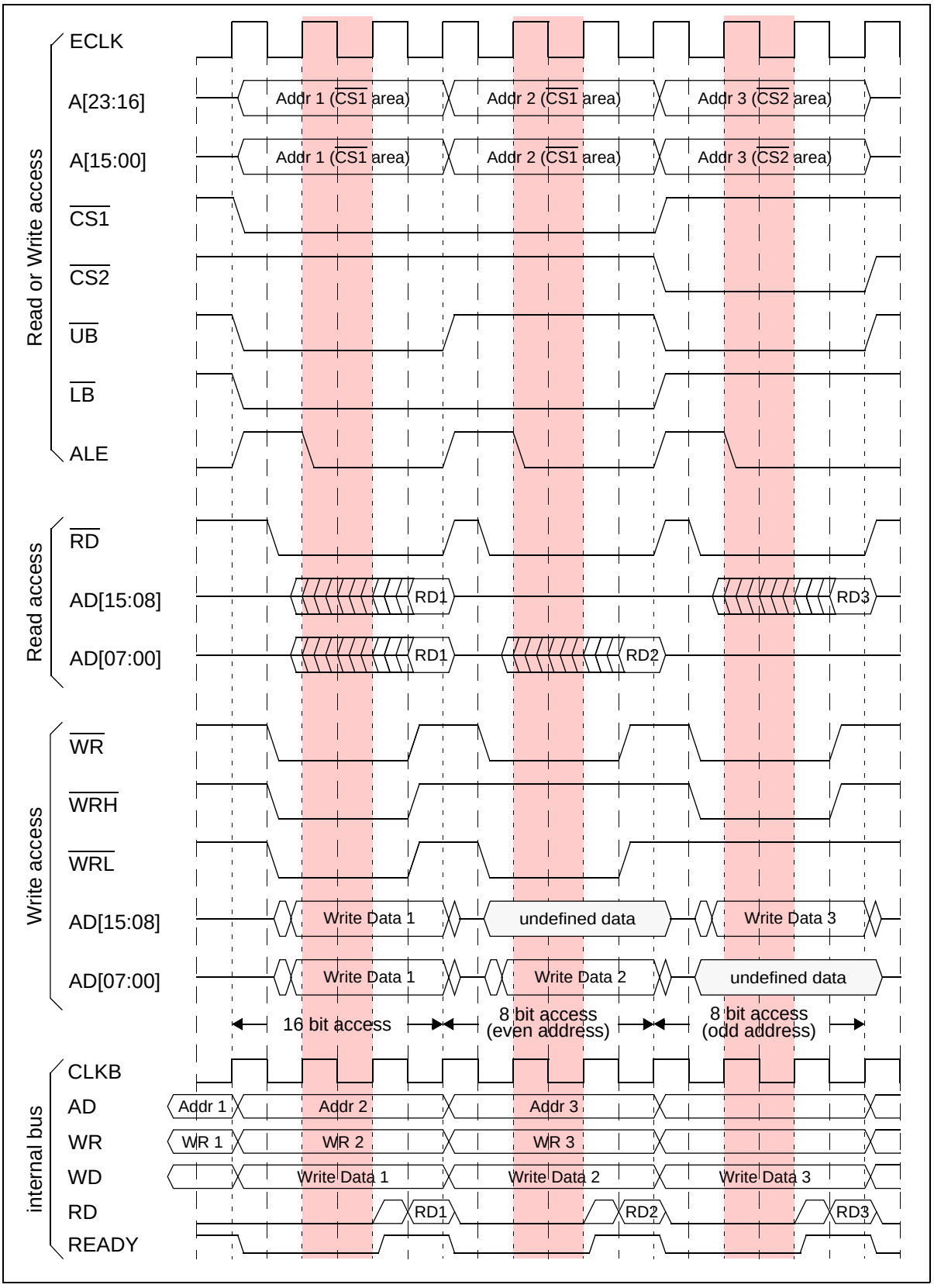

Read or Write access
ECLK
A[23:16]
A[15:00]
CS1
CS2
UB
LB
ALE
Addr 1 (CS1 area)
Addr 1 + 1
Addr 2 (CS2 area)
Addr 3 (CS1 area)
Read access
RD
AD[15:08]
AD[07:00]
Port function or other resource (pins not used in 8 bit non-multiplexed external bus mode)
RD1L
RD1H
RD2
RD3
Write access
WR
WRH
WRL
AD[15:08]
AD[07:00]
Write Data 1L
Write Data 1H
Write Data 2
Write Data 3
16 bit access
lower byte transfer
upper byte transfer
8 bit access (even address)
8 bit access (odd address)
internal bus
CLKB
AD
WR
WD
RD
READY
Addr 1
Addr 2
Addr 3
WR 1
WR 2
WR 3
Write Data 1
Write Data 2
Write Data 3
RD1
RD2
RD3

Figure 12-21. Non-multiplexed 16-bit bus: One automatic ready wait cycle added (EACL:R[2:0]='001', EACL:STS='0', EACL:ACE='0)'


Read or Write access
ECLK
A[23:16]
A[15:00]
CS1
CS2
UB
LB
ALE
Addr 1 (CS1 area)
Addr 2 (CS1 area)
Addr 3 (CS2 area)
Read access
RD
AD[15:08]
AD[07:00]
RD1
RD2
RD3
Write access
WR
WRH
WRL
AD[15:08]
AD[07:00]
Write Data 1
undefined data
Write Data 3
Write Data 2
16 bit access
8 bit access
(even address)
8 bit access
(odd address)
internal bus
CLKB
AD
WR
WD
RD
READY
Addr 1
Addr 2
Addr 3
WR 1
WR 2
WR 3

## 12.3.3 Ready Function

Enabling the external ready function (RDY-pin) and/or the automatic wait function enables access to low-speed memory and peripheral circuits.

### *12.3.3.1 Ready Function*

The read and write access to an external device can be extended with the ready function. The $F^2$MC-16FX MCU offers two possibilities to extend the data cycle of the external access, the automatic ready function and the external ready function.

Stopping the external bus by the ready function also stops the internal system bus until the external bus operation is finished.

#### Automatic ready function

The automatic ready function inserts 1 to 32 wait cycles to extend the data cycle without an external circuit. The number of inserted wait cycles can be selected by the EACL:R[2:0] bits separately for each external area. See Figure 12-17 and Figure 12-21 for timing diagrams showing the effect of the automatic ready function.

#### External ready function

The external ready function can be used after setting the RYE bit of the EBCF register to ’1’ and activating the RDY pin input function with the corresponding input enable register PIER.

The input signal of the RDY input pin is sampled at the beginning of the last cycle of the external bus access. For RDY=’0’, the data cycle is extended until RDY is set back to ’1’.

The external ready function can be used together with the automatic ready function. In this case, the value of the RDY pin is also read at the beginning of the last external bus cycle, means after the automatic wait cycles are expired.

Figure 12-22 and Figure 12-23 show timing diagrams of the external ready function.

Figure 12-22. Multiplexed 16-bit bus: Data cycle extended by external ready function

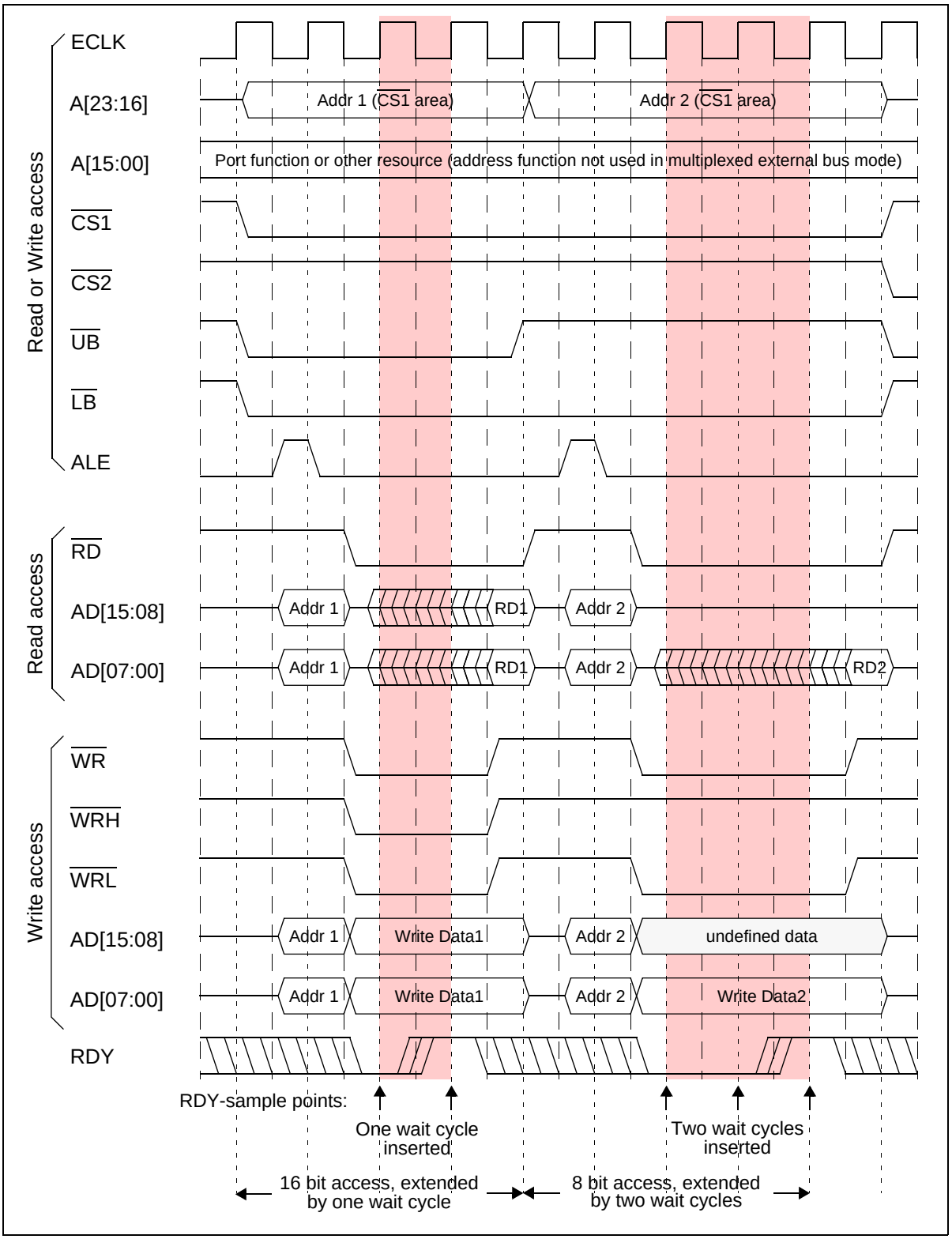
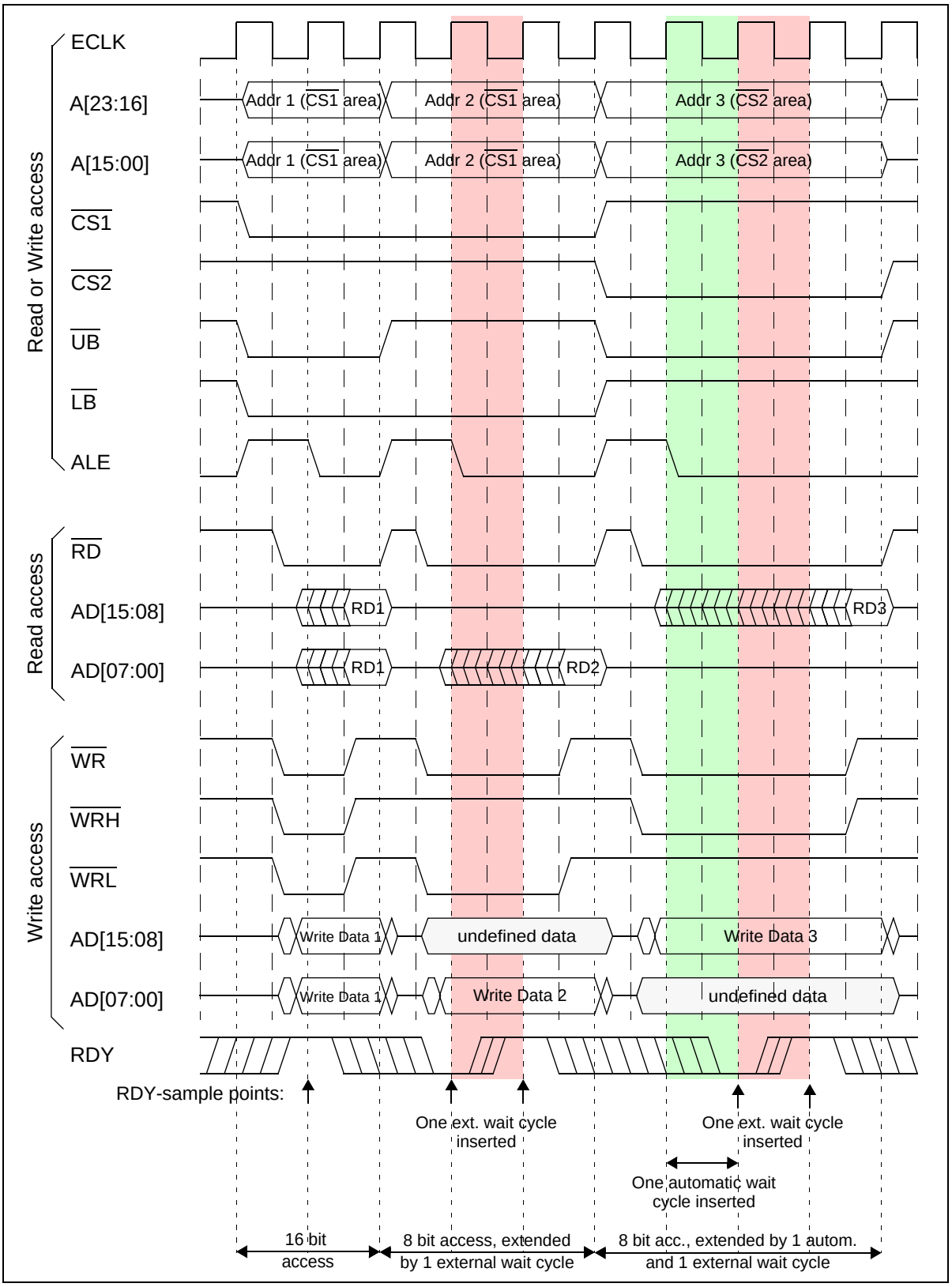


Figure 12-23. Non-multiplexed 16-bit bus: Data cycle extended by automatic and external ready function



## 12.3.4 Hold Function

The Hold function is used if multiple devices are used to access the same external bus. The Hold function is enabled by setting the HDE-bit in the External Bus Clock and Function register (EBCF) to ’1’. Inputting a ’1’ at the HRQ input pin then forces the pins of the external bus to Hi-Z state after a currently performed external bus access is completed. The Hi-Z state allows another device to access the external bus. The Hold state is indicated by the HAK pin.

### *12.3.4.1 Hold Function*

The Hold function is needed if multiple device are connected to the same external bus and more than one of this devices shall access the bus as a master. In this case the outputs of all other master devices must be disabled and the pin state of their external bus pins must be changed to high impedance.

The bus request from another master device is indicated by a high level at the HRQ input pin (Hold request). Then all necessary external bus pins are set to Hi-Z and the HAK-pin is set to ’0’ to allow the other master device to start its external bus access. The Hold function must be enabled by setting the EBCF:HDE bit to ’1’ before.

If a Hold request occurs while an external bus operation is in process, then this operation is completed before the Hold state is entered and the HAK-pin is set to ’0’. In case of a 16 bit access in 8 bit external bus mode (which is split into two accesses), both accesses must be completed.

The internal operation of the MCU can continue in Hold state as long as the MCU does not try to start an external access.

If the MCU tries to access the external bus in Hold state (by CPU or DMA), then the internal system bus is stopped until the HRQ-pin is released (’0’). No internal bus operation can be performed until the Hold state is released.

In Hold state (HAK-pin outputs ’0’), the following pins are placed in high-impedance state (when they are used as External bus output or External bus bidirectional signal):

- Address output:
  - A23 to A16 in multiplexed and non-multiplexed mode.
  - A15 to A00 in non-multiplexed mode (in multiplexed mode, these pins are not used).
- Address/Data I/O:
  - AD15 to AD00 in any mode.
- Bus control signal:
  - Strobe signals: ALE/AS, RD, WRL/WR, WRH
  - Byte select signals: LB, UB
  - Chip select signals: CS[5:0].

The bus clock ECLK is set to Hi-Z in Hold state only for EBCF:CSM=’1’ (clock suspend mode).

The corresponding Data Direction Register (DDR) must be set to ’0’ (input) and all other resources mapped to the same pin must be disabled to achieve the high-impedance state.

Internal or external pull-up or pull-down resistors can be used to avoid floating control signals if some pins are not driven by the other master. It is possible to drive selected pins to ’1’ or ’0’ in Hold state by setting the corresponding DDR register to "output" and writing the required pin level to the PDR register.

The Hold state is left by setting the HRQ-pin back to ’0’. Then the HAK-pin outputs high-level, and all other output pins restore the values which they were driving before the Hold state.

### *12.3.4.2 Hold request during Standby-Mode*

The Hold-Request is accepted in all Stand-by modes (Sleep, Timer and Stop mode). The hold request input signal HRQ asynchronously forces the external bus pins to Hi-Z (input signal is not sampled with ECLK as in Run mode for synchronous operation).

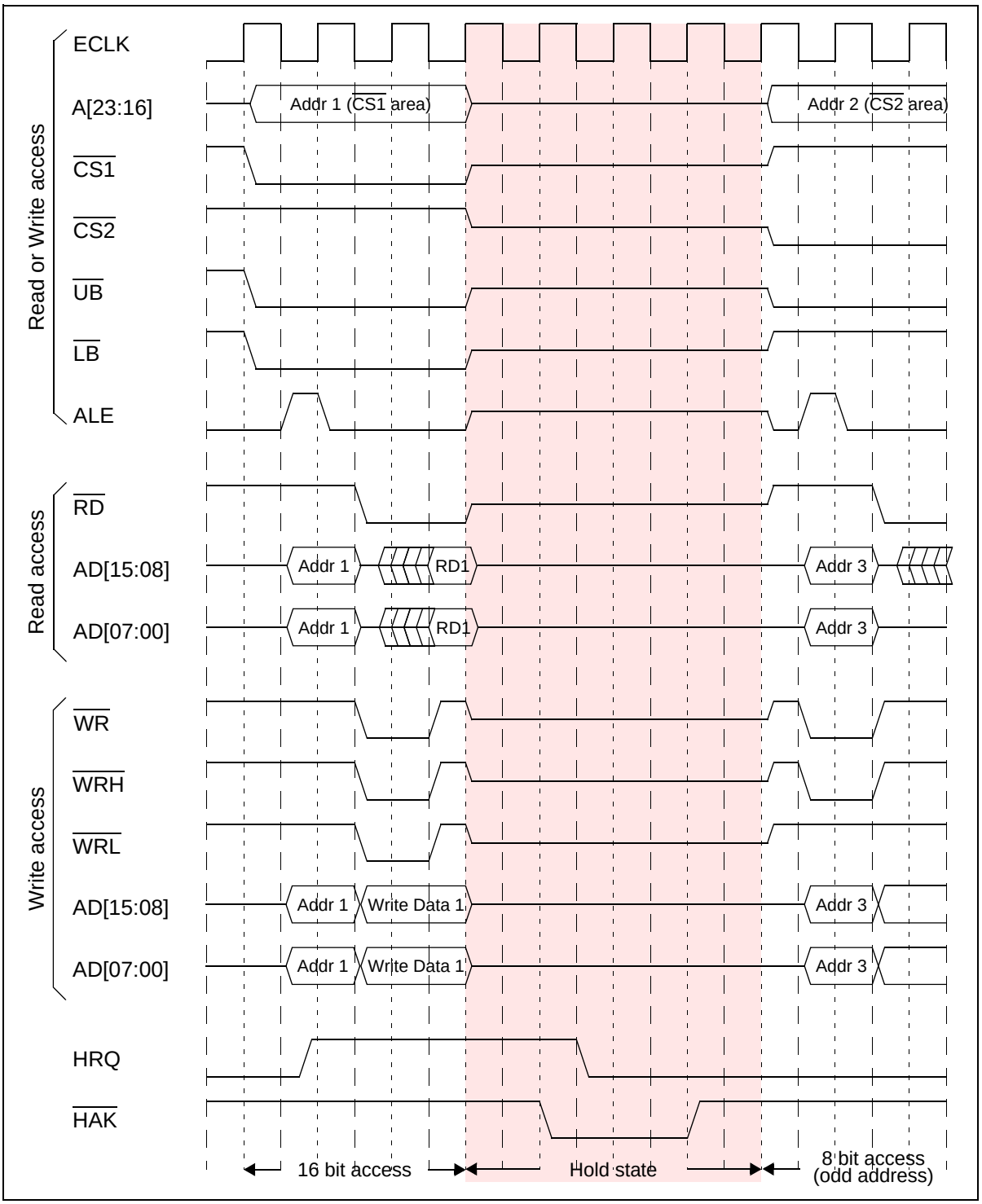
### *12.3.4.3 Watchdog timer*

Note that the Hold function can stop the internal operation of the MCU if the CPU or DMA tries to access the external bus. No software routine for clearing the Watchdog timer can be executed in this situation.

### *12.3.4.4 Timing diagram*

Figure 12-24 shows the timing of the Hold function.

Figure 12-24. Hold request in multiplexed 16-bit bus mode



## 12.4 Notes on using the external bus

The handling of pins for the external bus is the same as for all other resources. This chapter describes how to set up the necessary pin to use the external bus interface.

### Setup of input pins

If the Hold function or/and the external Ready function are used, the corresponding input pins (HRQ and RDY) must be set to input in the corresponding DDR (Data direction register) and PIER (Input enable register). The bits EBCF:HDE and EBCF:RYE only enable the corresponding function in the external bus module but not the input of the corresponding pin.

The output function of other resources connected to these pins must be disabled.

### Setup of output pins

The output function of the data pins (data cycle of the AD[15:08] and AD[07:00] pins) is automatically enabled depending on the selected bus mode (8 or 16 bit data width).

The output of address signals must be enabled in the External Bus Address output Enable registers (EBAE[2:0]) for all address pins in all modes. The address is only driven to the pin during an external bus access. Hence the pin state of the address pins in a pause between two external bus accesses depends on the setting of the corresponding DDR and PDR registers.

All control signals (ALE/$\overline{AS}$, $\overline{RD}$, $\overline{WRL}$/$\overline{WR}$, $\overline{WRH}$, $\overline{LB}$, $\overline{UB}$, CS[5:0] and the clock ECLK are enabled with its dedicated output enable signals. The $\overline{HAK}$ output function is enabled with the EBCF:HDE bit.

If the output of these control signals is enabled in the corresponding bit, then the value of the DDR of these pins is ignored unless the external bus is in Hold-state. For setting the external bus pin to Hi-Z in Hold state, the DDR must be set to ’0’ (input).

Another resource connected to a used external bus output pin must always be disabled.

All output pins not used for the external bus function can be used as port I/O or for another resource function connected to the same pin.

### In-/output pins

The AD[15:00] pins are used for reading and writing data and for address output in multiplexed modes.

The input function of the used pins (depending on 8-bit or 16-bit external bus mode) must be enabled in the corresponding Input enable register (PIER) by setting these bits to ’1’. The corresponding bits of the Data Direction register (DDR) must be set to Input and the output function of other resources connected to these pins must be disabled.

### Summary of all external bus pins and its setting

The following table shows all external bus pins and their state depending on the current operation mode of the external bus interface.

A description of the used terms can be found below the table.

Table 12-6. Recommended state of external bus pins

| Pin name | Status | Operation mode | | | |
|---|---|---|---|---|---|
| | | 8-bit non-multiplexed | 16-bit non-multiplexed | 8-bit multiplexed | 16-bit multiplexed |
| AD[07:00] | during external bus access | used for data | | used for data<br>used for address output (if enabled in EBAE0) | |
| | pause between two external bus accesses | Hi-Z | | | |
| | Hold-state | | | | |
| AD[15:08] | during external bus access | not used | used for data | used for address output (if enabled in EBAE1) | used for data<br>used for address output (if enabled in EBAE1) |
| | pause between two external bus accesses | | Hi-Z | Hi-Z | Hi-Z |
| | Hold-state | | | | |
| A[07:00] | during external bus access | used for address output (if enabled in EBAE0) | | not used | |
| | pause between two external bus accesses | Hi-Z | | | |
| | Hold-state | | | | |
| A[15:08] | during external bus access | used for address output (if enabled in EBAE1) | | not used | |
| | pause between two external bus accesses | Hi-Z | | | |
| | Hold-state | | | | |
| A[23:16] | during external bus access | used for address output (if enabled in EBAE2) | | | |
| | pause between two external bus accesses | Hi-Z | | | |
| | Hold-state | | | | |
| ECLK | during external bus access | Used for external bus clock output (depends on EBCF: CKE and CKI) | | | |
| | pause between two external bus accesses | Used for external bus clock output (depends on EBCF: DIV,CKE, CKI and CSM) | | | |
| | Hold-state | EBCF:CSM='0' and EBCF:CKE='1': Used for external bus clock output<br>EBCF:CSM='1' or EBCF:CKE='0': Hi-Z | | | |
| RDY | during external bus access | RDY input function<br>(if enabled by EBCF:RYE) | | | |
| | pause between two external bus accesses | | | | |
| | Hold-state | | | | |
| $\overline{WRH}$ | during external bus access | not used (outputs '1' if enabled in EBCS:WRHE) | used for $\overline{WRH}$ output (if enabled in EBCS:WRHE) | not used (outputs '1' if enabled in EBCS:WRHE) | used for $\overline{WRH}$ output (if enabled in EBCS:WRHE) |
| | pause between two external bus accesses | outputs '1' if enabled in EBCS:WRHE | | | |
| | Hold-state | Hi-Z | | | |
| $\overline{WRL}/\overline{WR}$ | during external bus access | used for $\overline{WRL}/\overline{WR}$ output (if enabled in EBCS:WRLE) | | | |
| | pause between two external bus accesses | outputs '1' if enabled in EBCS:WRLE | | | |
| | Hold-state | Hi-Z | | | |

Table 12-6. Recommended state of external bus pins

| Pin name | Status | Operation mode | | | |
|---|---|---|---|---|---|
| | | 8-bit non-multiplexed | 16-bit non-multiplexed | 8-bit multiplexed | 16-bit multiplexed |
| $\overline{RD}$ | during external bus access | used for $\overline{RD}$ output (if enabled in EBCS:RDE) | | | |
| | pause between two external bus accesses | outputs '1' if enabled in EBCS:RDE | | | |
| | Hold-state | Hi-Z | | | |
| ALE/$\overline{AS}$ | during external bus access | used for ALE/$\overline{AS}$ output (if enabled in EBCS:ASE) | | | |
| | pause between two external bus accesses | outputs inactive level if enabled in EBCS:ASE | | | |
| | Hold-state | Hi-Z | | | |
| $\overline{UB}$ | during external bus access | not used (outputs '1' if enabled in EBCS:UBE) | used for $\overline{UB}$ output (if enabled in EBCS:UBE) | not used (outputs '1' if enabled in EBCS:UBE) | used for $\overline{UB}$ output (if enabled in EBCS:UBE) |
| | pause between two external bus accesses | outputs '1' if enabled in EBCS:UBE | | | |
| | Hold-state | Hi-Z | | | |
| $\overline{LB}$ | during external bus access | used for $\overline{LB}$ output (if enabled in EBCS:LBE) | | | |
| | pause between two external bus accesses | outputs '1' if enabled in EBCS:LBE | | | |
| | Hold-state | Hi-Z | | | |
| HRQ | during external bus access | HRQ input function (if enabled by EBCF:HDE) | | | |
| | pause between two external bus accesses | | | | |
| | Hold-state | | | | |
| $\overline{HAK}$ | during external bus access | outputs '1' if Hold function is enabled with EBCF:HDE | | | |
| | pause between two external bus accesses | | | | |
| | Hold-state | outputs '0' | | | |
| CS[5:0] | during external bus access | used for CS[5:0] output (if enabled in EACH[5:0]:CSE) | | | |
| | pause between two external bus accesses | outputs inactive level if enabled in EACH[5:0]:CSE | | | |
| | Hold-state | Hi-Z | | | |

Explanation of the table:

| | |
|---|---|
| not used | • The pin is not driven by the External bus module and can be used as general purpose I/O or for another resource located on this pin. |
| used for data | • Corresponding input must be enabled by setting PIER to '1'. Otherwise no data input is possible.<br>• The external bus interface is driving the write data to the output pin during the write cycle.<br>• Another output resource sharing the same pin must be disabled.<br>• The corresponding DDR must be set to '0' (input), otherwise no data input is possible. |

| | |
|---|---|
| used for xxx output (if enabled in Exxx) | • The pin can be used as external bus output signal.<br>• The output of the external bus function can be enabled/disabled with the corresponding output enable bit in the Exxx register.<br>• If the pin is used as external bus output, another output resource sharing the same pin must be disabled.<br>• If the pin is used as external bus output and the Hold function should be used, set the corresponding DDR to input. Otherwise the PDR value will be output in Hold state instead of Hi-Z.<br>• If the pin is not used as external bus output, it can be used as port pin or output of another resource. The Hold function has no effect in this case. |
| xxx input function | • The pin can be used as external bus input pin.<br>• The corresponding input must be enabled by setting PIER to '1'.<br>• Another output resource sharing the same pin must be disabled.<br>• The corresponding DDR must be set to '0' (input). |
| Hi-Z | • The corresponding DDR must be set to 'input' for Hi-Z. As for all used external bus output pins, other resources sharing the same pin must be disabled.<br>• If DDR is set to output, the value defined in the corresponding PDR register is output instead of Hi-Z. |

If another state than Hi-Z should be used for output pins during Hold state or external bus access pause, set the DDR to output and write the required value to the PDR register of the corresponding pin. When the external bus disables its output enable of these pins to change the pin state to Hi-Z, the PDR value is output to the pin instead.

Note: Pull-up resistors can also be used to terminate the pin state when they are not driven by the external bus module.

## 12.5 External Boot Vector fetch

Devices with external bus interface offer the possibility to read the Boot Vector (User program start address) via the external bus interface. This is possible in three different external bus operation modes depending on the MD[2:0] mode pin setting.

The Boot ROM program initializes the external bus interface depending on the mode pin setting. Together with the Boot Vector, the External Bus Mode Byte is read and written to the External Bus Mode register EBM before start of the User program execution.

### Mode pins setting for external Boot Vector fetch

Following mode pin settings can be used for external Boot Vector fetch:

Table 12-7. Mode pin settings for external vector modes

| MD[2:0] | Operation mode |
|---|---|
| 000 | 8-bit data width<br>address/data multiplexed |
| 001 | 16-bit data width<br>address/data multiplexed |
| 110 | 8-bit data width<br>address/data non-multiplexed |

### Location of Boot Vector and External Bus Mode Byte

The startup data (Boot Vector and External Bus Mode Byte) is read from address FF:FFDC$_H$ -FF:FFDF$_H$ from the external bus if an external vector mode is selected by the Mode pins MD[2:0]. Within the address range, this data is located as follows:

Table 12-8. Boot Vector and External Bus Mode Byte

| Address | upper byte bit[15:08] | lower byte bit[07:00] |
|---|---|---|
| FF:FFDD/$DC_H$ | Start address bit[15:08] | Start address bit[07:00] |
| FF:FFDF/$DE_H$ | External bus mode byte | Start address bit[23:16] |

The External Bus Mode Byte is written to the External Bus Mode register EBM before start of the User program execution.

The configuration of the External Bus Mode Byte is the same as for the External Bus Mode register of the external bus interface. See Chapter 12.2.1 External Bus Mode registers (EBM) for details.

### Procedure of External Boot Vector fetch

1. Internal reset is released.
2. MCU wakes up with RC clock and starts the Boot-ROM Start-up program.
3. Boot-ROM program reads the mode pin setting -> external vector mode detected.
4. Boot-ROM program configures the following registers depending on the mode pin setting:

Table 12-9. Registers configured by the Boot ROM program before reading the external start vector ( except of MB96(F)38x in non-moltiplexed bus mode (MD[2:0]='110'))

| Register | configured bits | Explanation |
|---|---|---|
| EBM | EAE[5]='1' | Activate external area 5 |
| | ERE='1' | Enable External ROM |
| | For MD[2:0]='110': NMS='1' | Set external bus to non-multiplexed mode for MD[2:0]='110' |
| EBAE[2:0] | A[23:0]='11...11' | Enable all address output pins |
| EBCS | RDE='1' | Enable Read strobe output |
| | ASE='1' | Enable Address strobe output |
| EACL5 | R[2:0]='011' | Set automatic ready function to 3 wait cycles |
| | For MD[2:0]='000' or '110': BW='1' | Set external bus width to '8 bit' for MD[2:0]='000' or '110' |
| EACH5 | CSE='1' | Enable Chip select CS5 output |
| PIER[AD[15:00]] | PIER[AD[07:00]]='1' | Enable inputs of AD[07:00] data input pins |
| | For MD[2:0]='001': PIER[AD[15:08]]='1' | Enable inputs of AD[15:08] data input pins for 16 bit external start vector fetch mode |

Table 12-10. Registers configured by the Boot ROM program before reading the external start vector on MB96(F)38x in non-multiplexed bus mode (MD[2:0]='110')

| Register | configured bits | Explanation |
|---|---|---|
| EBM | EAE[4]='1' | Activate external area 4 |
| | ERE='1' | Enable External ROM |
| | NMS='1' | Set external bus to non-multiplexed mode |
| EBAE[2:0] | A[23:0]='11...11' | Enable all address output pins |
| EBCS | RDE='1' | Enable Read strobe output |
| | ASE='1' | Enable Address strobe output |
| EACL4 | R[2:0]='011' | Set automatic ready function to 3 wait cycles |
| | BW='1' | Set external bus width to '8 bit' for MD[2:0]='110' |
| EACH4 | CSE='1' | Enable Chip select CS4 output |
| PIER[AD[15:00]] | PIER[AD[07:00]]='1' | Enable inputs of AD[07:00] data input pins |

5. Boot-ROM program reads address FF:FFDCH to FF:FFFH from external bus.

6. Boot-ROM program writes the External Bus Mode Byte to the EBM register. The external bus interface is now configured as defined by the External Bus Mode Byte.
7. Boot-ROM program jumps to the user program start address which was read from the external bus.
8. User program is executed. All registers of the external bus interface can be rewritten by software at any time.

**Note:**

- The external bus clock (ECLK) and the byte select signals $\overline{UB}$/$\overline{LB}$ are not enabled during external Boot Vector fetch.

## 12.6 Pin status in different MCU states

This chapter describes the status of the external bus pins in the following MCU states:

- Reset
- Run mode with external bus access
- Run mode with internal access
- Standby modes
- Hold state

### 12.6.1 Status of external bus pins

The following table describes the status of each external bus pin in different MCU states. The statements are valid only if the external bus is enabled. See Table 12-6 for details about pin usage in external bus modes. To ensure Hi-Z state of the external bus pins, follow the recommendation of Chapter 12.4 Notes on using the external bus.

Table 12-11. Status of each pin of the external bus

<table>
<tr><th rowspan="2">Pin name</th><th rowspan="2">Reset</th><th colspan="2">Run mode</th><th rowspan="2">Standby mode + SMCR:SPL='0'*1</th><th rowspan="2">external bus Hold state</th></tr>
<tr><th>external bus access</th><th>internal access</th></tr>
<tr><td>AD[07:00]</td><td rowspan="16">Input disabled / output Hi-Z</td><td>used for data and/or address</td><td colspan="3">output Hi-Z</td></tr>
<tr><td>AD[15:08]</td><td>can be used for data and/or address (depending on bus mode)</td><td colspan="3">output Hi-Z if used for external bus</td></tr>
<tr><td>A[07:00]</td><td rowspan="2">output if enabled (non multiplexed bus mode only)</td><td colspan="3" rowspan="2">output Hi-Z (non multiplexed bus mode only)</td></tr>
<tr><td>A[15:08]</td></tr>
<tr><td>A[23:16]</td><td rowspan="8">output if enabled</td><td colspan="2">output Hi-Z</td><td rowspan="4">output Hi-Z</td></tr>
<tr><td>ALE/$\overline{AS}$</td><td colspan="2">output inactive level if enabled</td></tr>
<tr><td>$\overline{RD}$</td><td colspan="2" rowspan="3">output 'H' if enabled</td></tr>
<tr><td>$\overline{WRL}$/$\overline{WR}$</td></tr>
<tr><td>$\overline{WRH}$</td></tr>
<tr><td>$\overline{UB}$/$\overline{LB}$</td><td colspan="2">output inactive level if enabled</td></tr>
<tr><td>CS[5:0]</td><td colspan="2">output inactive level if enabled</td><td rowspan="2">output Hi-Z</td></tr>
<tr><td>ECLK</td><td>output if enabled (state and activity depends on clock configuration)</td><td>output 'H' or 'L' if enabled (inactive level for EBCF:CSM='1' and DIV[2:0]≠"000")</td></tr>
<tr><td>RDY</td><td>input if enabled</td><td colspan="3">unused input</td></tr>
<tr><td>HRQ</td><td colspan="3">input if enabled</td><td>input</td></tr>
<tr><td>$\overline{HAK}$</td><td colspan="3">'H' output if enabled</td><td>'L' output</td></tr>
</table>

*1 Standby modes are: Sleep-, Timer- and Stop mode; the external bus is not in Hold state

### Notes about ECLK

When no external bus access is performed, the state of ECLK depends on the setting of EBCF:DIV[2:0], CKI and CSM bits. If the MCU is set to a Standby mode, the clock supplying the external bus interface is stopped and therefore the external bus clock ECLK is stopped too. Hence the state of ECLK in Standby mode depends on the ECLK state at the transition to Standby mode. If the Clock suspend mode is disabled or the ECLK clock divider is disabled (EBCF:DIV[2:0]="000"), this state may be ’H’ or ’L’. If the Clock suspend mode is enabled and the ECLK clock divider is active, this is always the inactive level defined by EBCF:CKI.

# 13. I/O Ports

This chapter explains the functions and operations of the I/O ports.

13.1 I/O Ports

13.2 I/O Port Registers

13.3 Register usage

## 13.1 I/O Ports

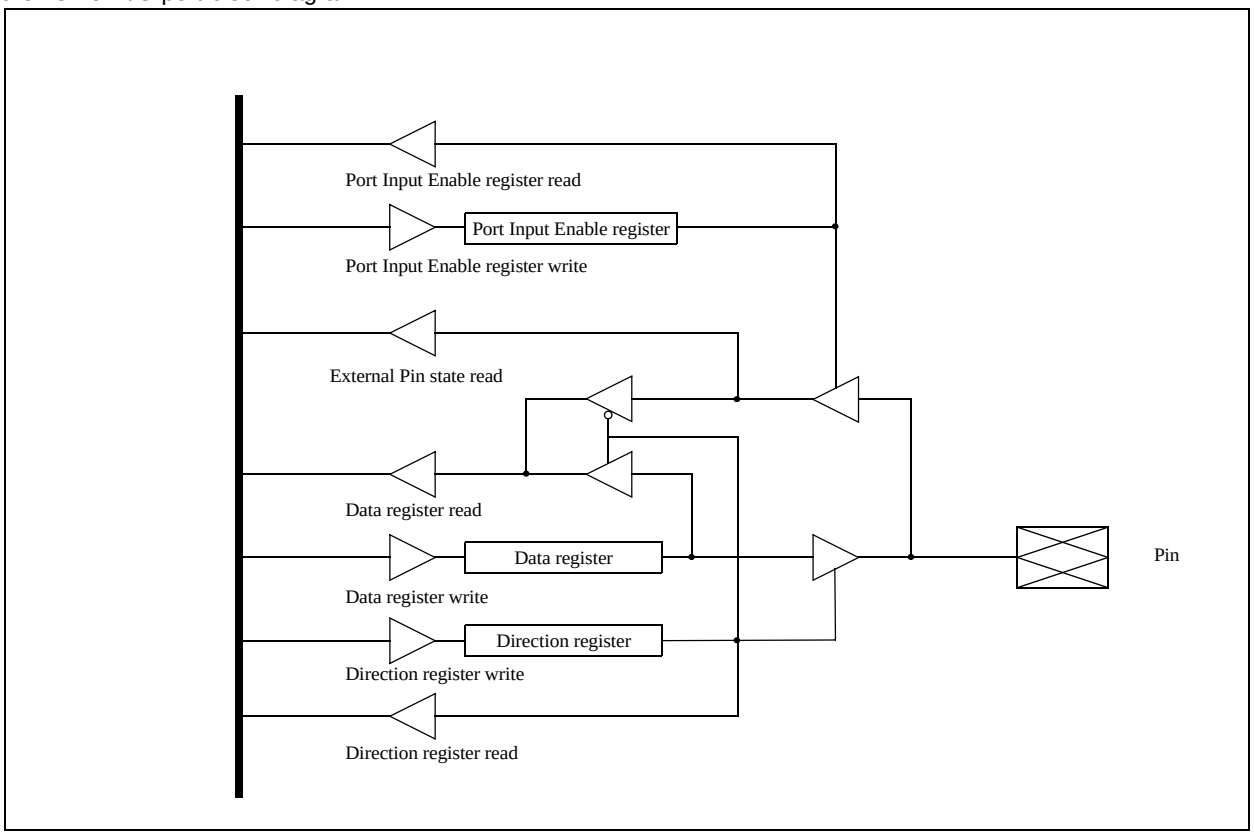
Each pin of the ports can be specified as input or output using the direction register, if the corresponding peripheral does not use the pin. Before using a pin as input, it must be enabled by setting the Port Input Enable register. When a pin is specified as input, the logic level at the pin is read. When a pin is specified as output, the data register value is read. The above also applies to a read operation for the read-modify-write instructions.

### I/O ports

When a pin is used as an output of other peripheral function, the peripheral output value is read regardless of the direction register value.

It is generally recommended that the read-modify-write instructions should not be used for setting the data register prior to setting the port as an output. This is because the read-modify-write instruction in this case results reading the logic level at the port rather than the register value.

Figure 13-25. I/O port block diagram




## 13.2 I/O Port Registers

Each general purpose port pin GPxx is controlled by nine types of registers. These are:

- Port Data Register (PDR00 to PDRnn)
- External Pin State Register (EPSR00 to EPSRnn)
- Port Direction Register (DDR00 to DDRnn)
- Port Input Enable Register (PIER00 to PIERnn)
- Port Input Level Register (PILR00 to PILRnn)
- Extended Port Input Level Register (EPILR00 to EPILRnn)
- Port Output Drive Register (PODR00 to PODRnn)
- Port High Drive Register (PHDR00 to PHDRnn)
- Pull-Up Control Register (PUCR00 to PUCRnn)

### I/O port registers

The general purpose port pin GPxx_y is controlled by bit y of the I/O port register xx. For example, the data direction of port pin GPxx_y is controlled by DDRxx_y.

Figure 13-1. I/O port registers (only shown for port 00)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 | Port 0 Data Register (PDR00) |
| | PS7 | PS6 | PS5 | PS4 | PS3 | PS2 | PS1 | PS0 | Port 0 External Pin State Register (EPSR00) |
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Port 0 Direction Register (DDR00) |
| | IE7 | IE6 | IE5 | IE4 | IE3 | IE2 | IE1 | IE0 | Port 0 Input Enable Register (PIER00) |
| | IL7 | IL6 | IL5 | IL4 | IL3 | IL2 | IL1 | IL0 | Port 0 Input level Register (PILR00) |
| | EIL7 | EIL6 | EIL5 | EIL4 | EIL3 | EIL2 | EIL1 | EIL0 | Port 0 Extended Input Level Register (EPILR00) |
| | OD7 | OD6 | OD5 | OD4 | OD3 | OD2 | OD1 | OD0 | Port 0 Output Drive Register (PODR00) |
| | HD7 | HD6 | HD5 | HD4 | HD3 | HD2 | HD1 | HD0 | Port 0 High Drive Register (PHDR00) |
| | PU7 | PU6 | PU5 | PU4 | PU3 | PU2 | PU1 | PU0 | Port 0 Pull-Up Control Register (PUCR00) |

## 13.2.1 Port Data Register (PDRnn)

Note that R/W for I/O ports differ from R/W for memory in the following points:

- Input mode

Read: The level at the corresponding pin is read.

Write: Data is written to an output latch.

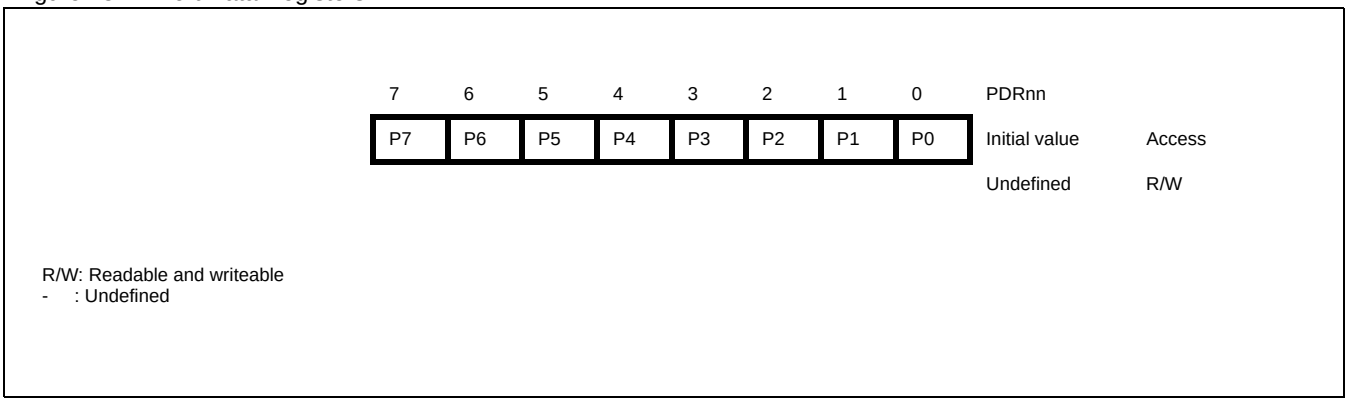
- Output mode

Read: The data register latch value is read.

Write: Data is written to an output latch and output to the corresponding pin.

### 13.2.1.1 Port Data Register

Figure 13-2. Port Data Registers




### 13.2.1.2 Reading the Port Data Register

**Bits Px (PDR00 to PDRnn)**

When a Port Data Register is read, the read value depends on the corresponding bit in the Data Direction Register, on the current status of the resource that is connected to the same pin (if applicable) and the used instruction (RMW or not RMW). The following cases are possible:

Table 13-1. Reading Port Data Register

| DDR value | Resource output | Pin value | Read value of 'Read instructions' | Read value of 'RMW instructions' |
|---|---|---|---|---|
| 0 (input) | enabled | value of resource output | input pin state | value of PDR |
| | disabled | port input | input pin state | value of PDR |
| 1 (output) | enabled | value of resource output | value of PDR | value of PDR |
| | disabled | value of PDR | value of PDR | value of PDR |

Reading PDR value with RMW instructions ensures, that single bits of PDR are not accidentally overwritten. This may happen if resource function of pin is selected and pin value is read instead of PDR value.

Please refer to the following example for explanation of difference between reading PDR with RMW instruction on F2MC-16LX and F2MC-16FX.

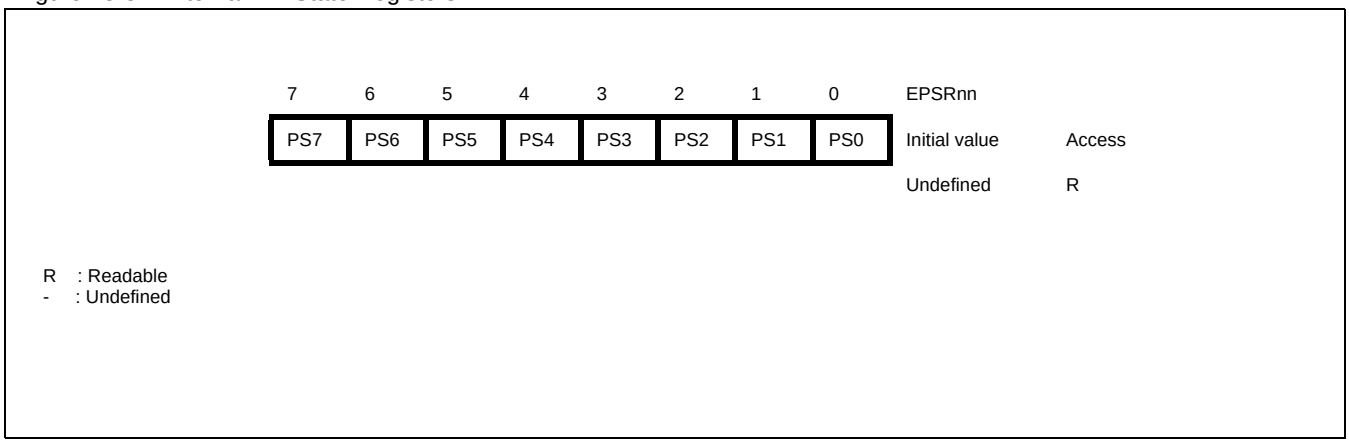
| | |
|---|---|
| Bit # | 7654.3210 |
| DDR | 1111.1111 (= output) |
| PDR (= last value written to PDR) | 0000.0000 |
| OE<br>(resource output enable) | 0010.0000 |
| Resource output | 0010.0000 |
| current pin value | 0010.0000 |
| read by RMW instruction from PDR on 16LX | 0010.0000<br>-> e.g.<br>MOV A, #0<br>OR PDR, A<br>=> writes back #b'0010.0000<br>=> PDR is changed |
| read by RMW instruction from PDR on 16FX | 0000.0000<br>-> e.g.<br>MOV A, #0<br>OR PDR, A<br>=> writes back #b'0000.0000<br>=> PDR is not changed |

## 13.2.2 External Pin State Register (EPSRnn)

With this register the current state of the external pin can be read.

### *13.2.2.1 External Pin State Register*

Figure 13-3. External Pin State Registers



#### *13.2.2.2 Access to the external pin state data register*

**Bits PSx (EPSR00 to EPSRnn)**

These bits mirror the status of the external port pin.

Writing to this register has no effect.

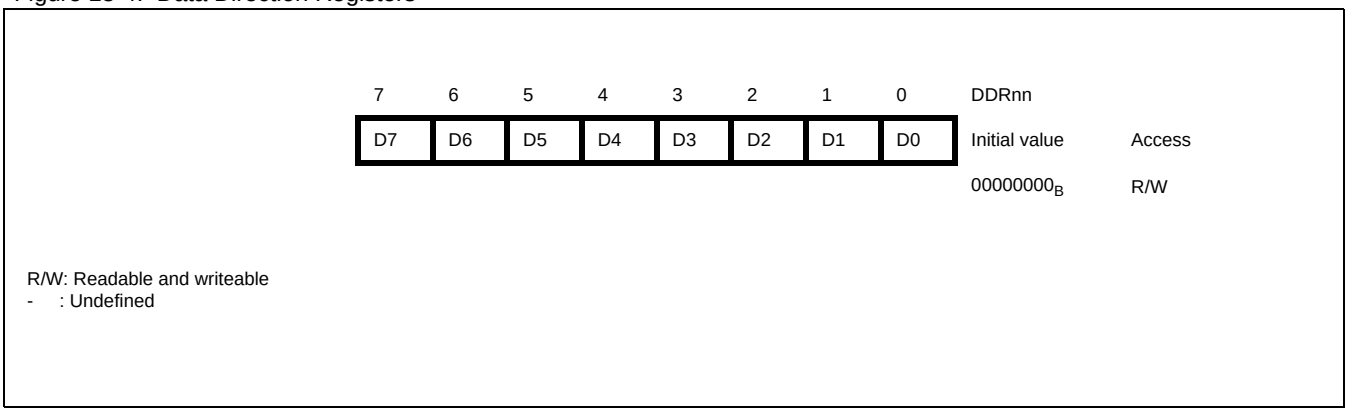
The external pin status register mirrors the status of the external port pin only if the corresponding bit in the Port Input Enable Register is set to "1".

### 13.2.3 Data Direction Register (DDRnn)

When a pin is used as a general purpose port, this register switches the corresponding pin to input mode or output mode.

#### *13.2.3.1 Data direction register*

Figure 13-4. Data Direction Registers




#### *13.2.3.2 Reading the Data Direction Register*

**Bits Dx (DDR00 to DDRnn)**

These bits set the data direction of each port pin.

When a pin is used as a port, the corresponding pin is controlled as described below:

Table 13-2. Mode selection

| | |
|---|---|
| 0 | Input mode |
| 1 | Output mode |

**Note:**

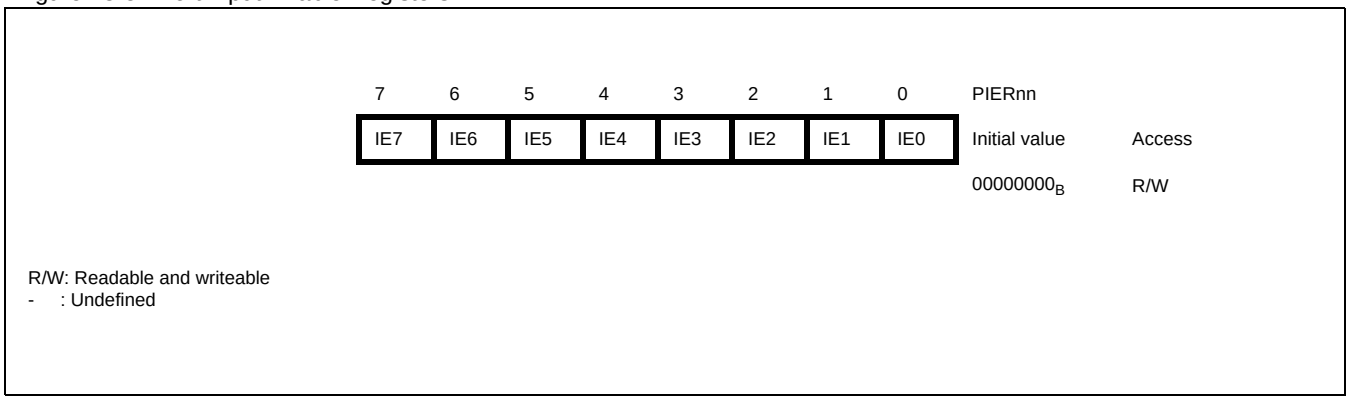
The Data Direction Register can be read independently from the status of the corresponding resource. However, the value of the DDR influences the result of a read access on the Port Data Register.
If pins are used as analog input or output, this value is ignored.

### 13.2.4 Port Input Enable Register (PIERnn)

When a pin is used as general purpose port, the digital input is enabled or disabled with this register. When the digital input is disabled, no transverse current is drawn by the input stage at any input pin potential.

### *13.2.4.1 Port Input Enable Register (PIER)*

Figure 13-5. Port Input Enable Registers




**Bits IEx (PIER00 to PIERnn)**

These bits enable or disable the digital input of each port pin.

When a pin is used as an input port, the corresponding pin is controlled as described below:

Table 13-3. Port enable function

| | |
|---|---|
| 0 | Disable digtal input |
| 1 | Enable digital input |

**Note:**

If pins are used as analog input or output, then the digital input function is disabled.

**Note:**

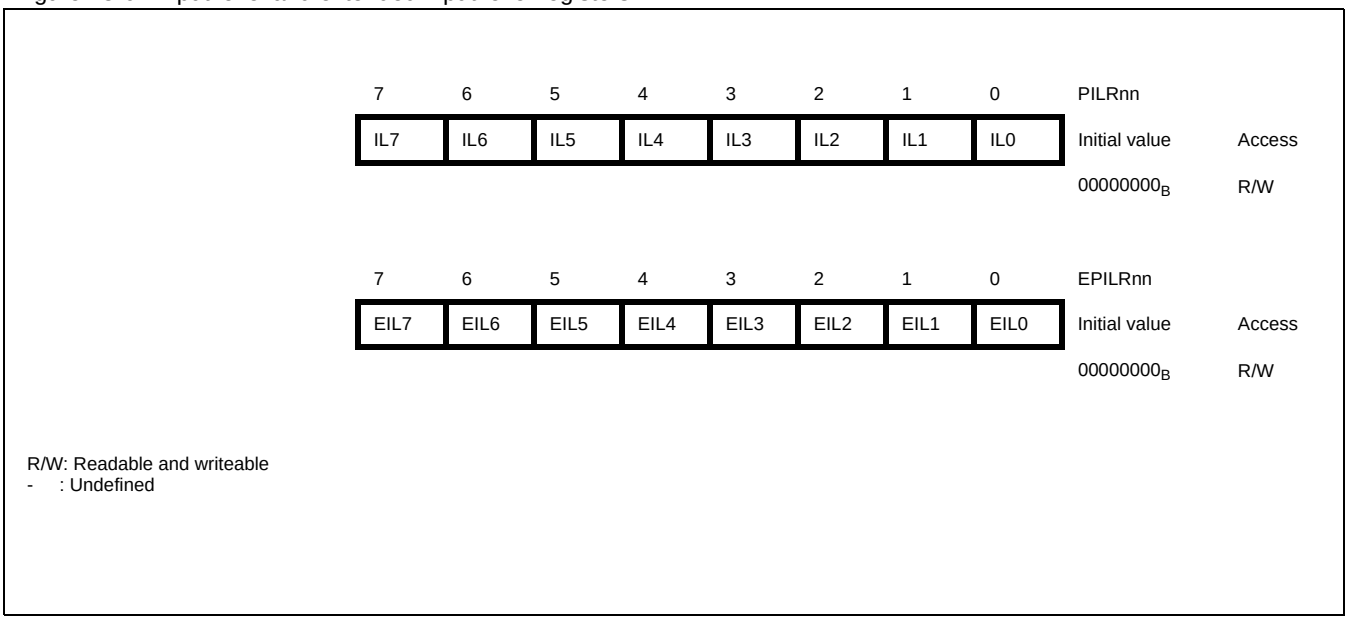
When the MCU enters Stop mode or Timer mode, the input of all ports is disabled irrespective of PIER setting, except for ports, for which an external interrupt is activated.

## 13.2.5 Port Input Level Register (PILRnn) and Extended Port Input Level Register (EPILRnn)

The digital input level of each pin can be programmed between CMOS Hysteresis, Automotive Hysteresis or TTL level with these two registers.

### 13.2.5.1 Port Input Level Register (PILRnn) and Extended Port Input Level Register(EPILRnn)

Figure 13-6. Input level and extended input level registers



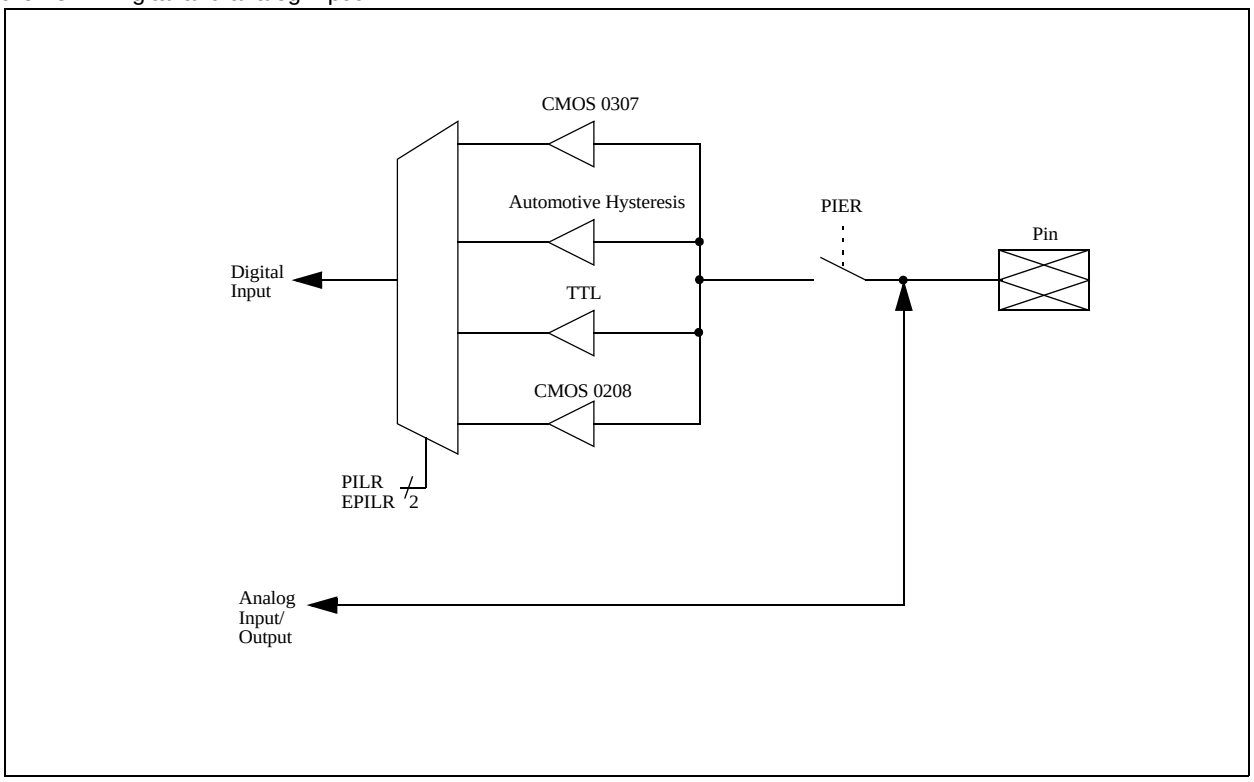

#### Bits ILx (PILR00 to PILRnn) and EILx (EPILR00 to EPILRnn)

Set the digital input level with these bits.

Table 13-4. Input level selection

| ILx | EILx | input level | $V_{IL}$ | $V_{IH}$ |
|---|---|---|---|---|
| 0 | 0 | CMOS (0307) | 0.3 x VDD | 0.7 x VDD |
| 1 | 0 | Automotive Hysteresis | 0.5 x VDD | 0.8 x VDD |
| 0 | 1 | TTL | 0.8V | 2.1V |
| 1 | 1 | CMOS(0208) | 0.2 x VDD | 0.8 x VDD |

Figure 13-7. Digital and analog Input




## 13.2.6 Port Output Drive Register (PODRnn)

The output drive option can be programmed with this register.
Output drive strength is selectable between normal and reduced current.

### *13.2.6.1 Port Output Drive Register (PODRnn)*

Figure 13-8. Port Output Drive Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | PODRnn | |
|---|---|---|---|---|---|---|---|---|---|
| OD7 | OD6 | OD5 | OD4 | OD3 | OD2 | OD1 | OD0 | Initial value | Access |
| | | | | | | | | $00000000_B$ | R/W |

R/W: Readable and writeable
- : Undefined

### Bits ODx (PODR00 to PODRnn)

These bits switch the output current between normal and reduced value of each port pin.
Please refer to the data sheet for electrical characteristics.

Table 13-5. Output drive selection

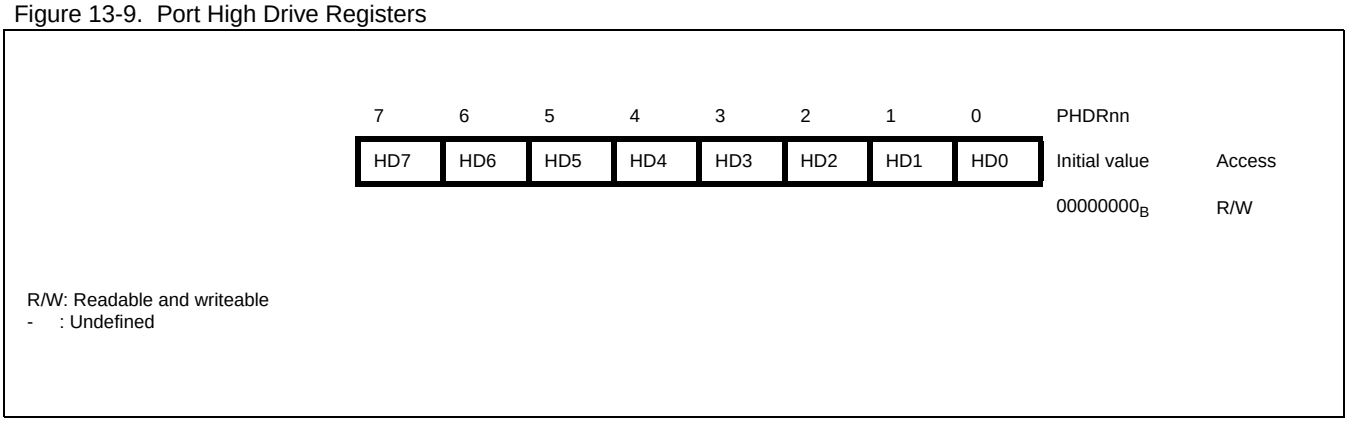
| | |
|---|---|
| 0 | Normal current output drive |
| 1 | Reduced current output drive |

## 13.2.7 Port High Drive Register (PHDR)

The high drive option can be enabled or disabled with this register.

### *13.2.7.1 Port High Drive Register*

Figure 13-9. Port High Drive Registers




### Bits HDx (PHDR00 to PHDRnn)

These bits enable or disable the high drive option for each port pin.
Please refer to the data sheet for electrical characteristics.

Table 13-6. High drive enable function

| | |
|---|---|
| 0 | Disable high drive option (Drive strength is determined by PODR setting) |
| 1 | Enable high drive option (PODR value is ignored) |

**Note:**

This option is not available for all ports. Please refer to the data sheet for details.

## 13.2.8 Pull-Up Control Register (PUCR)

The Pull-up resistor can be enabled or disabled with this register.

### *13.2.8.1 Pull-Up Control Register (PUCR)*

Figure 13-10. Pull-Up Control Register

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | PUCRnn | |
|---|---|---|---|---|---|---|---|---|---|
| PU7 | PU6 | PU5 | PU4 | PU3 | PU2 | PU1 | PU0 | Initial value | Access |
| | | | | | | | | $00000000_B$ | R/W |

R/W: Readable and writeable
- : Undefined

**Bits PUx (PUCR00 to PUCRnn)**

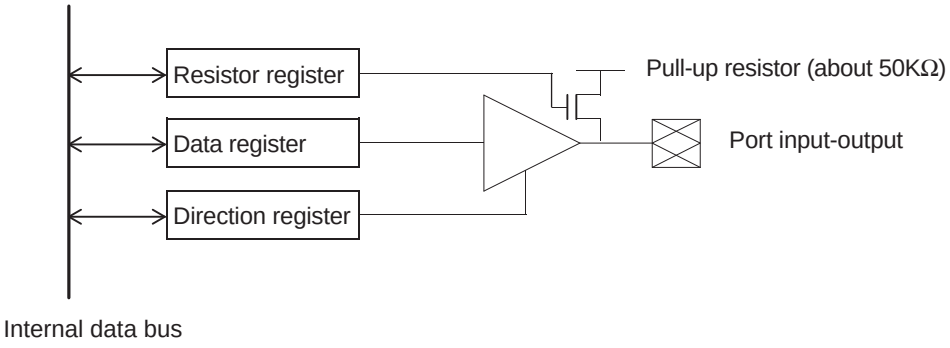
These bits enable or disable the pull-up resistor for each port pin.

Table 13-7. Pull-up function

| | |
|---|---|
| 0 | No Pull-up resistor in input mode |
| 1 | Pull-up resistor in input mode |

### *13.2.8.2 Block Diagram of Pull-Up Control Register (PUCR)*

Figure 13-11. Block Diagram of Pull-Up Control Register (PUCR)




**Note:**

In output mode, this register has no effect (no pull-up resistor). In stop and timer mode (SMCR.SPL=1), the state with no pull-up resistor is entered (high impedance).

## 13.3 Register usage

This chapter gives a summary about usage of IO registers.

### Register settings for different port usage

Table 13-8. Register settings for different port usage

| Register/ Port usage | DDR | PIER | EPSR | PILR | EPILR | PUCR | PODR | PHDR (if avail.) | PDR (read) | PDR (write) |
|---|---|---|---|---|---|---|---|---|---|---|
| initial state | 0 (input) | 0 (off) | undefined | 0 | 0 | 0 (off) | 0 (normal) | 0 (off) | undefined | value stored in PDR |
| General Purpose input | set to 0 (input) | set to 1 (on) | pin state | User setting | User setting | User setting | no effect | no effect | pin state | value is stored in PDR |
| General Purpose output | set to 1 (output) | set to 1 (on) for EPSR | pin state (if PIER = 1) | User setting | User setting | ignored (off) | User setting | User setting | PDR | value is stored in PDR and shown on pin |
| Peripheral Resource input (digital) | set to 0 (input) | set to 1 (on) | pin state | User setting | User setting | User setting | no effect | no effect | pin state | value is stored only in PDR |
| Peripheral Resource output (digital) | ignored | set to 1 (on) for EPSR | pin state (if PIER = 1) | User setting | User setting | ignored (off)[a] | User setting | User setting | PDR | value is stored in PDR |
| Analog I/O | ignored | ignored (input is off) | undefined | ignored | ignored | ignored (off) | ignored | ignored | undefined | value is stored in PDR |

**a. Exception:**
**When the pin is used for I2C bus SDA or SCL line, the user setting of PUCR remains active even when I2C bus controller is enabled.**

### Note:

For pins used in external bus at external vector fetch the initial state is PILR=0, EPILR=1 (TTL).
Peripheral Resource output is activated by the corresponding peripheral resource register.
Analog I/O is activated by register of analogue peripheral resource e.g. by ADER for ADC.
Please refer to HWM chapter of analogue and peripheral resource.

# 14. 16-Bit I/O Timer

This chapter explains the functions and operations of the 16-bit I/O Timer.

14.1 Outline of 16-bit I/O Timer

14.2 16-Bit I/O Timer Registers

14.3 16-bit Free-Running Timer

14.4 Output Compare Unit

14.5 Input Capture Unit

## 14.1 Outline of 16-bit I/O Timer

The 16-bit I/O Timer consists of a 16-bit Free-Running Timer and Output Compare Units and Input Capture Units. It is used to generate pulse sequences and to measure the time duration between external events.

### 16-bit Free-Running Timer

- The 16-bit Free-Running Timer consists of the up counter, the control register, the 16-bit compare clear register, and the prescaler. The output value obtained from this counter is used as the basic timer of the Input Capture Unit.
- Eight counter clocks are available
- An interrupt can be generated upon a counter overflow or a match with compare register 0 and 4.
- The counter value can be initialized to '$0000_H$' upon a reset, software clear, or if a compare match is generated between this timer and the compare clear register values.

### Output Compare Unit (2 channels per one module)

The Output Compare Unit consist of two 16-bit compare registers, two compare output latches, and two control registers.

When a 16-bit Free-Running Timer value matches the corresponding compare register value, the output level is toggled and an interrupt can be issued.

- The two compare registers can be used independently for each Output Compare Unit.
- Output pins can be controlled based on pairs of the two compare registers.

Output pins can be toggled by using the two compare registers.

- Initial values for output pins can be set.
- Interrupts can be generated upon a compare match.

### Input Capture Unit (2 channels per one module)

The Input Capture Unit consists of two 16-bit capture registers and two control registers, each corresponding to two independent external input pins. The 16-bit Free-Running Timer values can be stored in the capture register and an interrupt is issued simultaneously upon detection of an edge of a signal input from an external input pin.

- The detection edge of an external input signal can be specified.

Rising, falling, or both edges can be chosen.

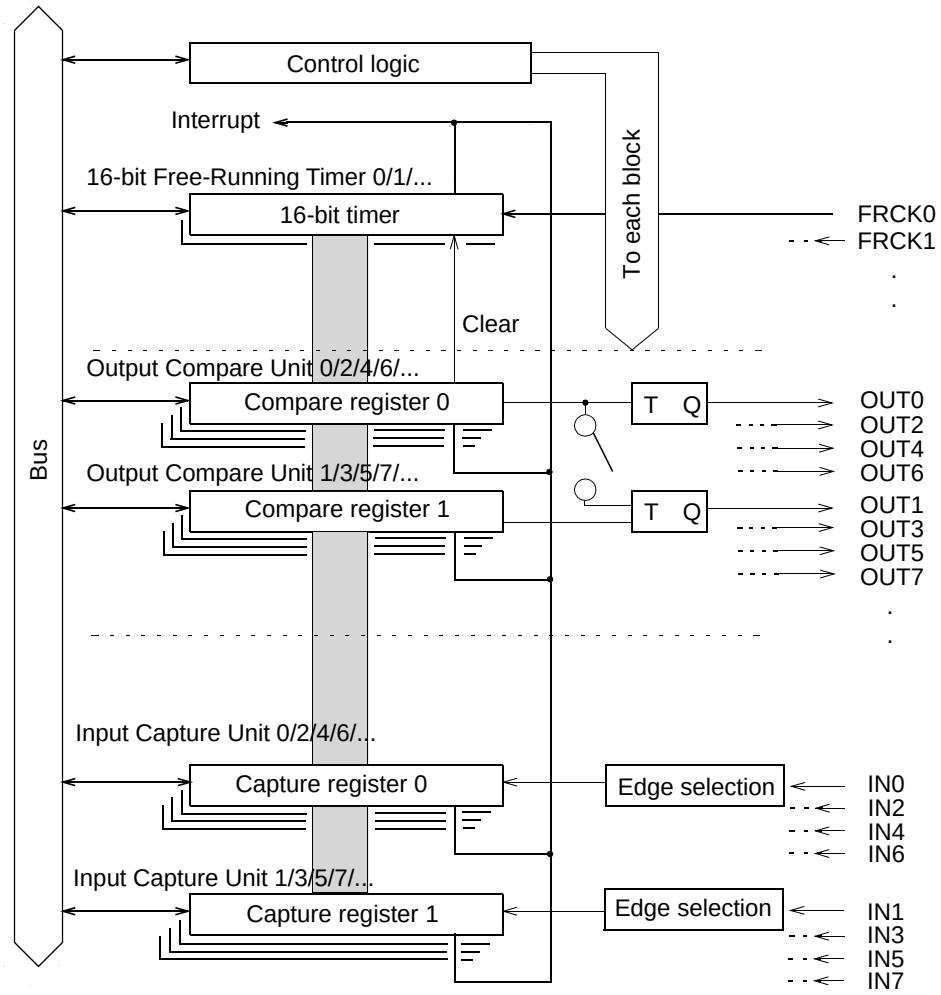
- The two input channels can operate independently.
- An interrupt can be issued upon a valid edge of an external input signal.

The DMA can be activated upon an input capture interrupt.

### Block diagram of the 16-bit I/O Timer

Figure 14-1 shows a block diagram of the 16-bit I/O Timer.

Figure 14-1. Block diagram of 16-bit I/O Timer



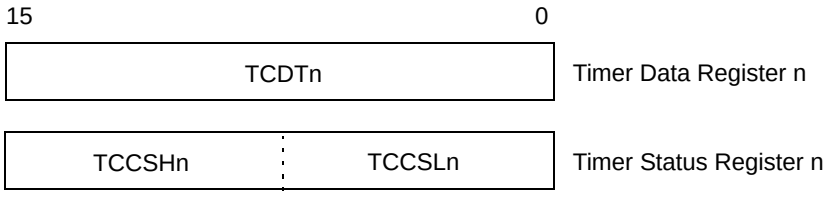

Note: Please refer to the first chapter (Section 16-bit I/O-Timer Configuration on page 26) and the tables therein to check the connections between input/output capture units and free running timers.

## 14.2 16-Bit I/O Timer Registers

The 16-bit I/O timer has the following registers:

- 16-bit Free-Running Timer registers
- 16-bit Output Compare registers
- 16-bit Input Capture registers

### 16-bit Free-Running Timer registers

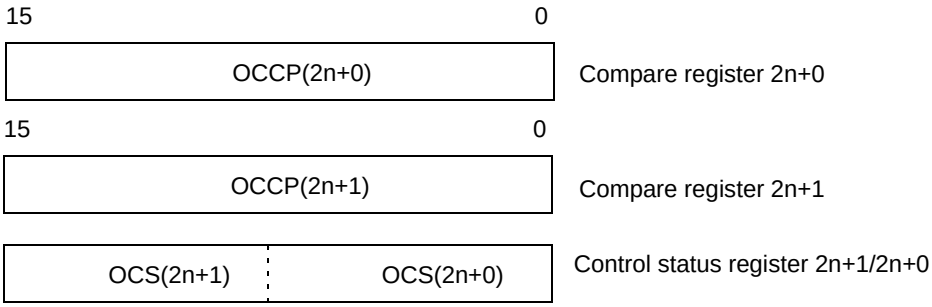



**Remark:**

The suffix "n" denotes the Free-Running Timer number (0, 1, 2, ...). The register name is composed by the register type name and the suffix. Hence, the following registers are available:

- for Free-Running Timer 0: TCDT0, TCCSH0, TCCSL0
- for Free-Running Timer 1: TCDT1, TCCSH1, TCCSL1
- etc.

### 16-bit Output Compare Unit registers

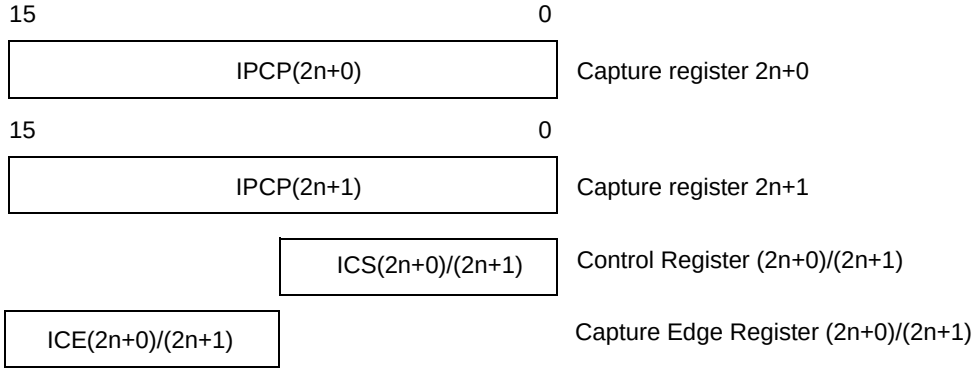



**Remark:**

The suffix "n" denotes the number of the Output Compare Unit (0, 1, 2, ...). The register name is composed by the register type name and the suffix. Hence, the following registers are available:

- for Output Compare Unit 0: OCCP0, OCCP1, OCS0, OCS1
- for Output Compare Unit 1: OCCP2, OCCP3, OCS2, OCS3
- etc.

### 16-bit Input Capture registers




**Remark:**

The suffix "n" denotes the number of the Input Capture Unit (0, 1, 2, ...). The register name is composed by the register type name and the suffix. Hence, the following registers are available:

- for Input Capture Unit 0: IPCP0, IPCP1, ICS01, ICE01
- for Input Capture Unit 1: IPCP2, IPCP3, ICS23, ICE23
- etc.

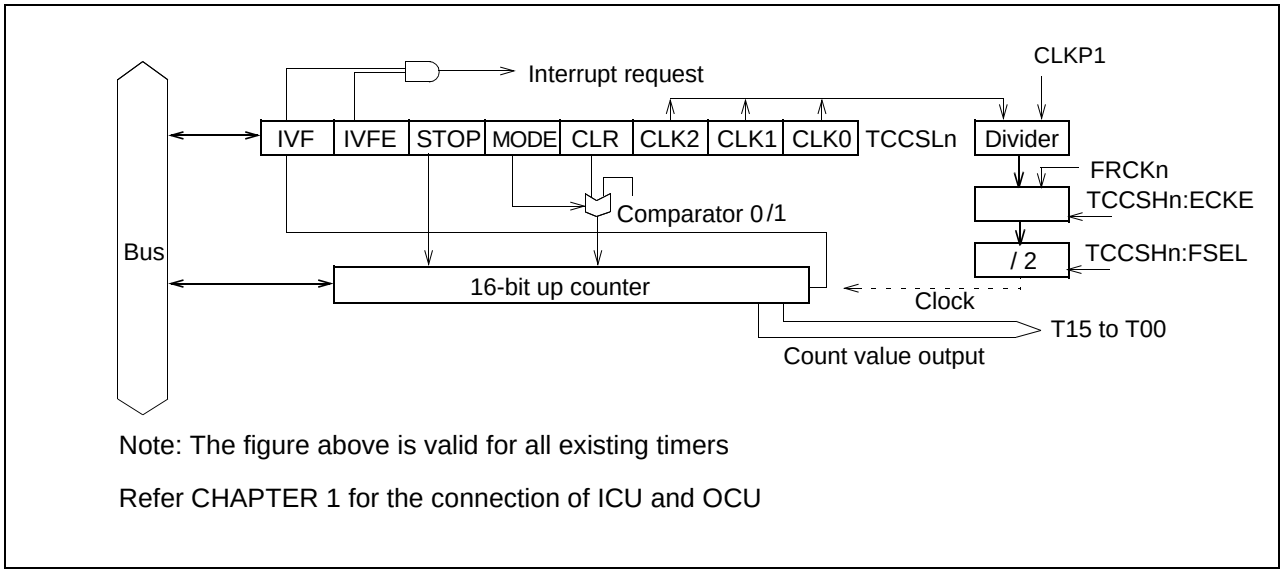
## 14.3 16-bit Free-Running Timer

The 16-bit Free-Running Timer consists of a 16-bit up counter and a control status register. The count values of this timer are used as the base time for the Output Compare Units and Input Capture Units.

- Eight counter clock frequencies are available.
- An interrupt can be generated upon a counter value overflow.
- The counter value can be initialized upon a match with compare register 0 and compare register 4, depending on the mode.

### 16-bit Free-Running Timer block diagram

Figure 14-2. 16-bit Free-Running Timer block diagram




Note: The figure above is valid for all existing timers

Refer CHAPTER 1 for the connection of ICU and OCU

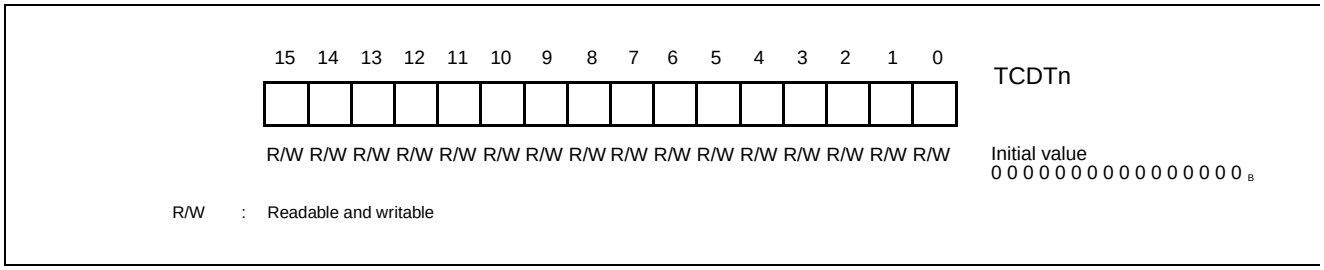
### 14.3.1 Data Register (TCDTn)

The Data Register (TCDTn) can read the count value of the 16-bit Free-Running Timer. The counter value is cleared to "0000" upon a reset. The timer value can be set by writing a value to this register. However, ensure that the value is written while the operation is stopped (STOP=1).

The data register must be accessed by the word access instructions.

**Data register of Free-Running Timer (TCDTn)**

Figure 14-3. Data register of Free-Running Timer (TCDTn)




The 16-bit Free-Running Timer is initialized upon the following factors:

- Reset
- Clear bit (CLR) of control status register
- A match between compare register and the timer counter value.

## 14.3.2 Control Status Register (TCCSLn)

The control status register (TCCSLn) sets the operation mode of the 16-bit Free-Running Timer, starts and stops the 16-bit Free-Running Timer, and controls interrupts.

### Control status register of Free-Running Timer (TCCSLn)

Figure 14-4. Control status register of Free-Running Timer (TCCSLn)

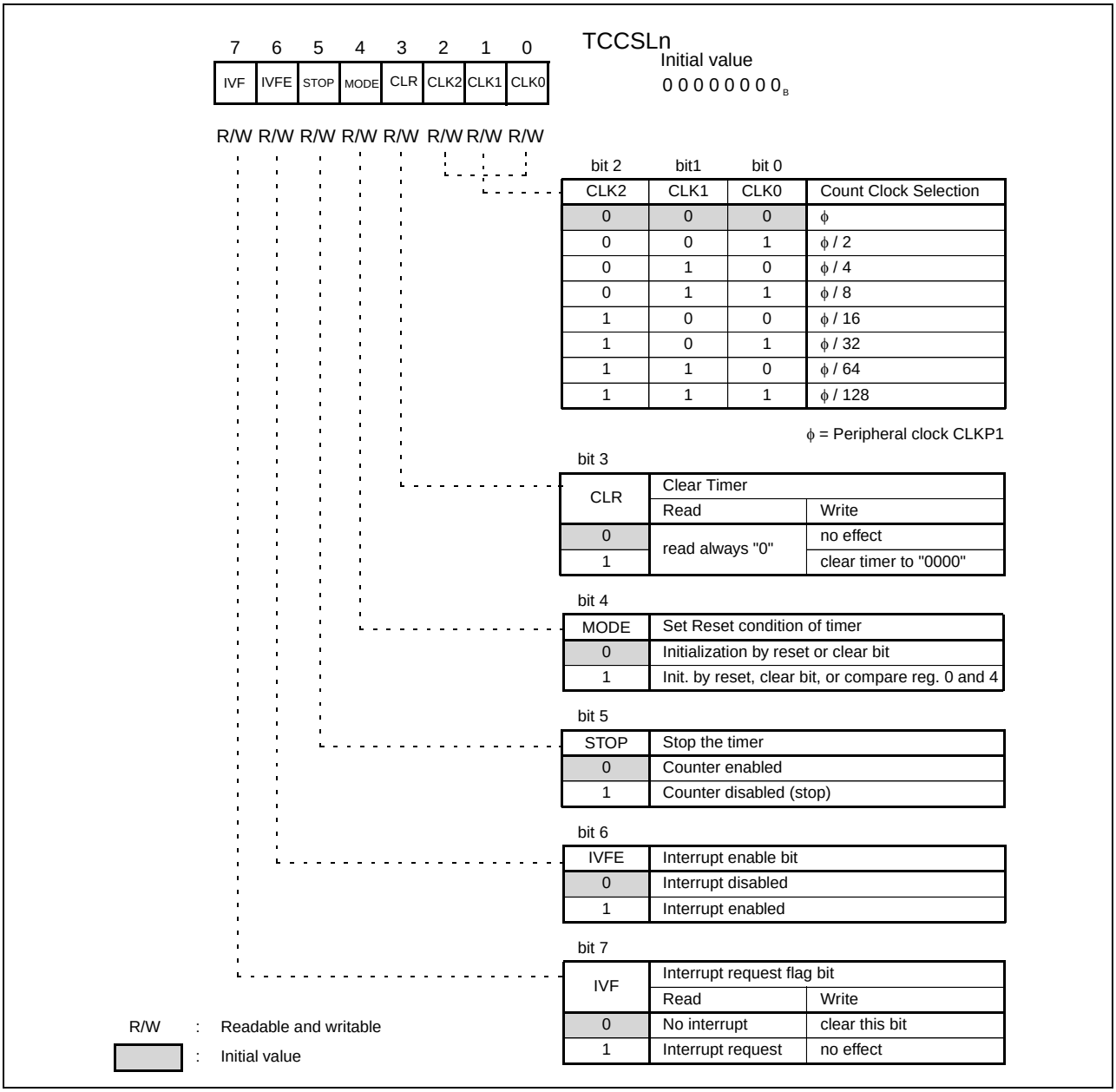


Table 14-1. Control status register of Free-Running-Timer (TCCSLn)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 7</td><td>IVF</td><td>• This bit is the interrupt request flag bit and clear bit<br>• Writing "0": A possible interrupt is cleared.<br>• Writing "1": No effect.<br>• "1" is always read during a read-modify-write cycle.<br>• This bit is set to "1" if the Free Running Timer overflows or if the timer value matches with the compare register 0 or compare register 4 and the MODE bit is set to "1".</td></tr>
<tr><td>bit 6</td><td>IVFE</td><td>• This bit enables the interrupt request<br>• Writing "0": Interrupt disabled.<br>• Writing "1": Interrupt enabled.</td></tr>
<tr><td>bit 5</td><td>STOP</td><td>• The STOP bit is used to stop the timer.<br>• Writing "0": Counter enabled (operation).<br>• Writing "1": Counter disabled (stop).</td></tr>
<tr><td>bit 4</td><td>MODE</td><td>• "0": Initialization by reset or clear bit<br>• "1": Initialization by reset, clear bit, or compare register 0 or compare register 4</td></tr>
<tr><td>bit 3</td><td>CLR</td><td>• The CLR bit initializes the operating Free-Running Timer to the value "0000"<br>• Writing "0": no effect.<br>• Writing "1": Counter is initialized.<br>Note:<br>To initialize the counter value while the timer is stopped (STOP = 1), write "0000" to the data register (TCDT).</td></tr>
<tr><td>bit 2 to 0</td><td>CLK2, CLK1, CLK0</td><td>These bits are used to select the count clock for the 16-bit-Free-Running Timer. The clock is updated immediately after a value is written to these bits. Therefore, ensure that the input capture and output compare operations are stopped before a value is written to these bits.
<table>
<tr><th>CLK2</th><th>CLK1</th><th>CLK0</th><th>Count clock</th><th>Φ = 20 MHz</th><th>Φ = 16 MHz</th><th>Φ = 8 MHz</th><th>Φ = 4 MHz</th><th>Φ = 1 MHz</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Φ</td><td>50 ns</td><td>62.5 ns</td><td>125 ns</td><td>0.25 µs</td><td>1 µs</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Φ / 2</td><td>100 ns</td><td>125 ns</td><td>0.25 µs</td><td>0.5 µs</td><td>2 µs</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Φ / 4</td><td>0.2 µs</td><td>0.25 µs</td><td>0.5 µs</td><td>1 µs</td><td>4 µs</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Φ / 8</td><td>0.4 µs</td><td>0.5 µs</td><td>1 µs</td><td>2 µs</td><td>8 µs</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Φ / 16</td><td>0.8 µs</td><td>1 µs</td><td>2 µs</td><td>4 µs</td><td>16 µs</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Φ / 32</td><td>1.6 µs</td><td>2 µs</td><td>4 µs</td><td>8 µs</td><td>32 µs</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Φ / 64</td><td>3.2 µs</td><td>4 µs</td><td>8 µs</td><td>16 µs</td><td>64 µs</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Φ / 128</td><td>6.4 µs</td><td>8 µs</td><td>16 µs</td><td>32 µs</td><td>128 µs</td></tr>
</table></td></tr>
</table>

## Control status register of Free-Running Timer (TCCSHn)

**Figure 0.0-1 Control status register of Free-Running Timer (TCCSHn)**

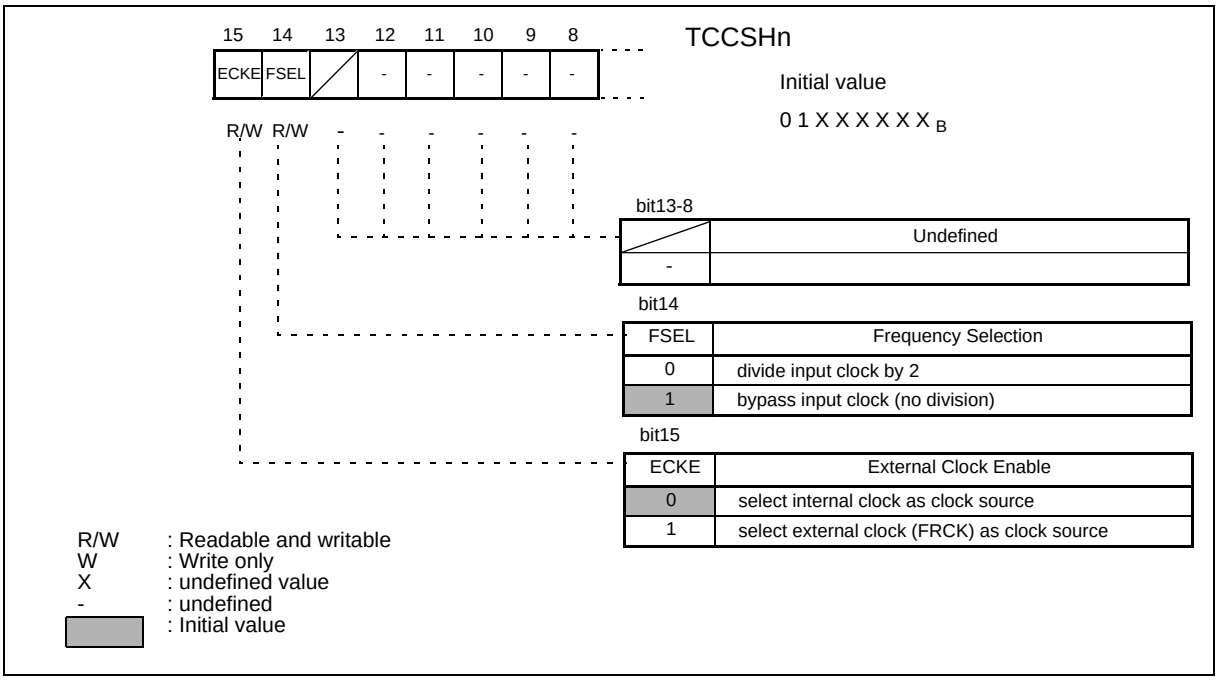



Table 14-2. Function description of each bit of the Control status register of Free-Running timer (TCCSHn)

| Bit name | | Function |
|---|---|---|
| bit 15 | ECKE | This bit is used to select the internal or external clock source. Change this bit status while the output compare or input capture is stopped because the clock may be changed after writing data in the ECKE bit.<br>Note:<br>If the internal clock is selected, set bits CLK2 to CLK0 to the count clock value. This count clock is made the basic clock.<br>If the clock source is input from FRCK, set the corresponding port's data direction register (DDR) to "input" and set the input enable register (IER) to "enabled". |
| bit 14 | FSEL | This bit is used to control division of the counter clock source. Change this bit status while the output compare or input capture is stopped. The FSEL bit is used to specify the count clock division ratio.<br>If FSEL is set to "0", the count clock cycle specified by the count clock selection bit (TCCSL bit:CLK2 to CLK0) is divided by two. |
| bit 13 to bit 8 | undefined | The read value of these bits is undefined.<br>Always write '0' to these bits.<br>RMW instructions are not affected. |

## 14.3.3 16-bit Free-Running Timer Operation

The 16-bit Free-Running Timer starts counting from counter value "0000" after the reset is released. The counter value is used as the base time for the 16-bit Output Compare and 16-bit Input Capture operations.

### *14.3.3.1 16-bit Free-Running Timer operation*

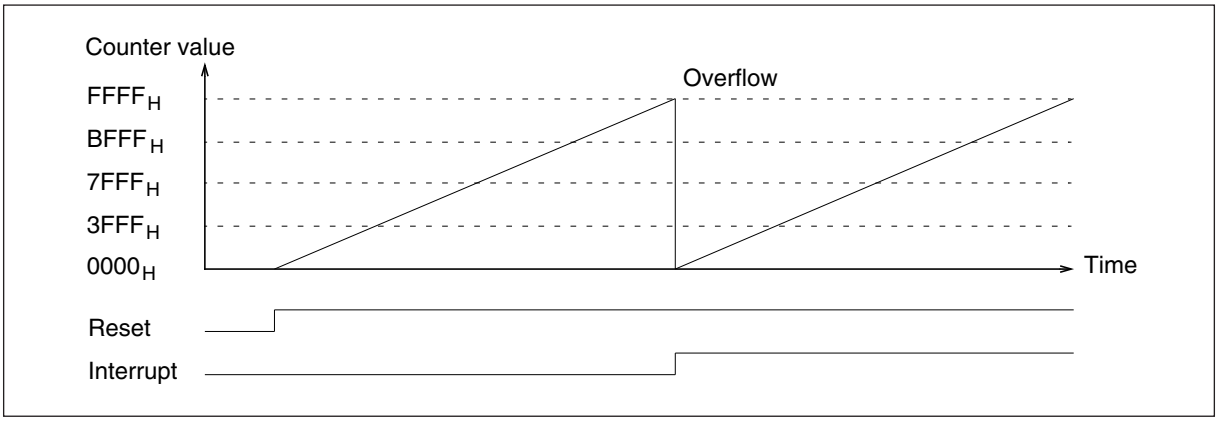
The counter value is cleared in the following conditions:

- When an overflow occurs.
- The compare clear register value matches the 16-bit Free-Running Timer value. (The mode must be set.)
- When a match with the Output Compare register 0 or Output Compare register 4 occurs. (This depends on the mode.)
- When "1" is written to the CLR bit of the TCCSL register during operation.
- When "0000" is written to the TCDT register during stop.
- Reset

An interrupt can be generated when an overflow occurs or when the counter is cleared by a match with the compare register 0 or 4. (Compare match interrupts can be used only in an appropriate mode.)

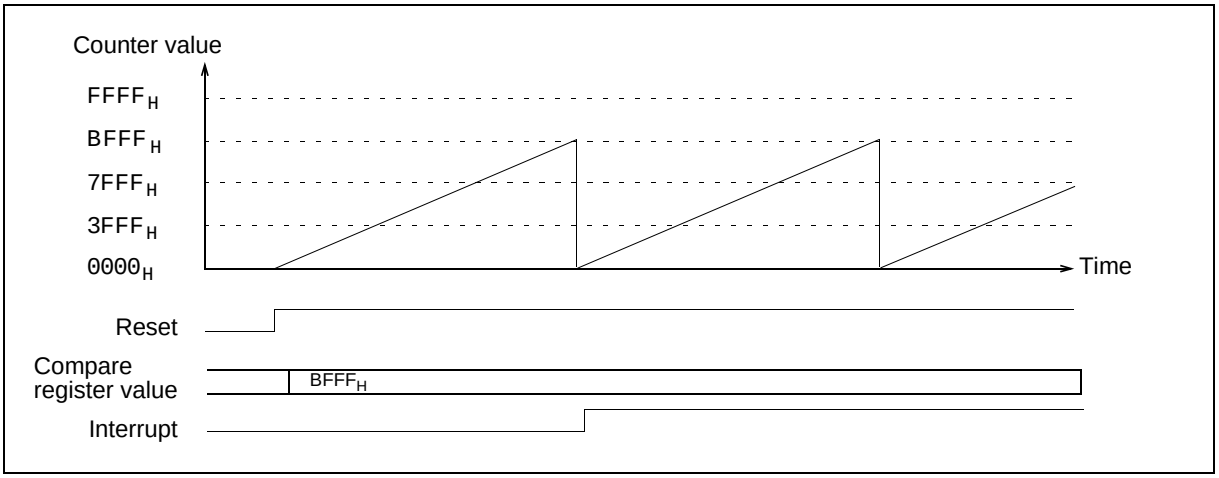
### *14.3.3.2 Clearing the counter by an overflow*

Figure 14-5. Clearing the counter by an overflow




### *14.3.3.3 Clearing the counter at Compare Clear Register value match*

Figure 14-6. Clearing the Counter when the Compare Clear Register Value matches the 16-bit Free-Running Timer value

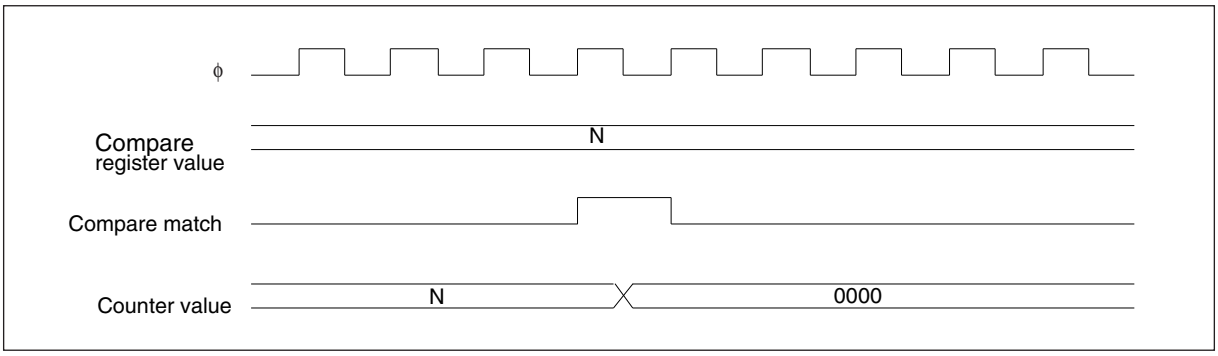


### *14.3.3.4 16-bit Free-Running Timer timing*

#### 16-bit Free-Running Timer clear timing (match with the compare register 0)

The counter can be cleared upon a reset, software clear, or a match with the compare register 0. By a reset, the counter is immediately cleared. By a match with compare register 0 or Compare Clear Register value or by software clear (TCCS:CLR = 1), the counter is cleared in synchronization with the count timing.

Figure 14-7. 16-bit Free-Running Timer clear timing (match with the compare register 0)




## 14.4 Output Compare Unit

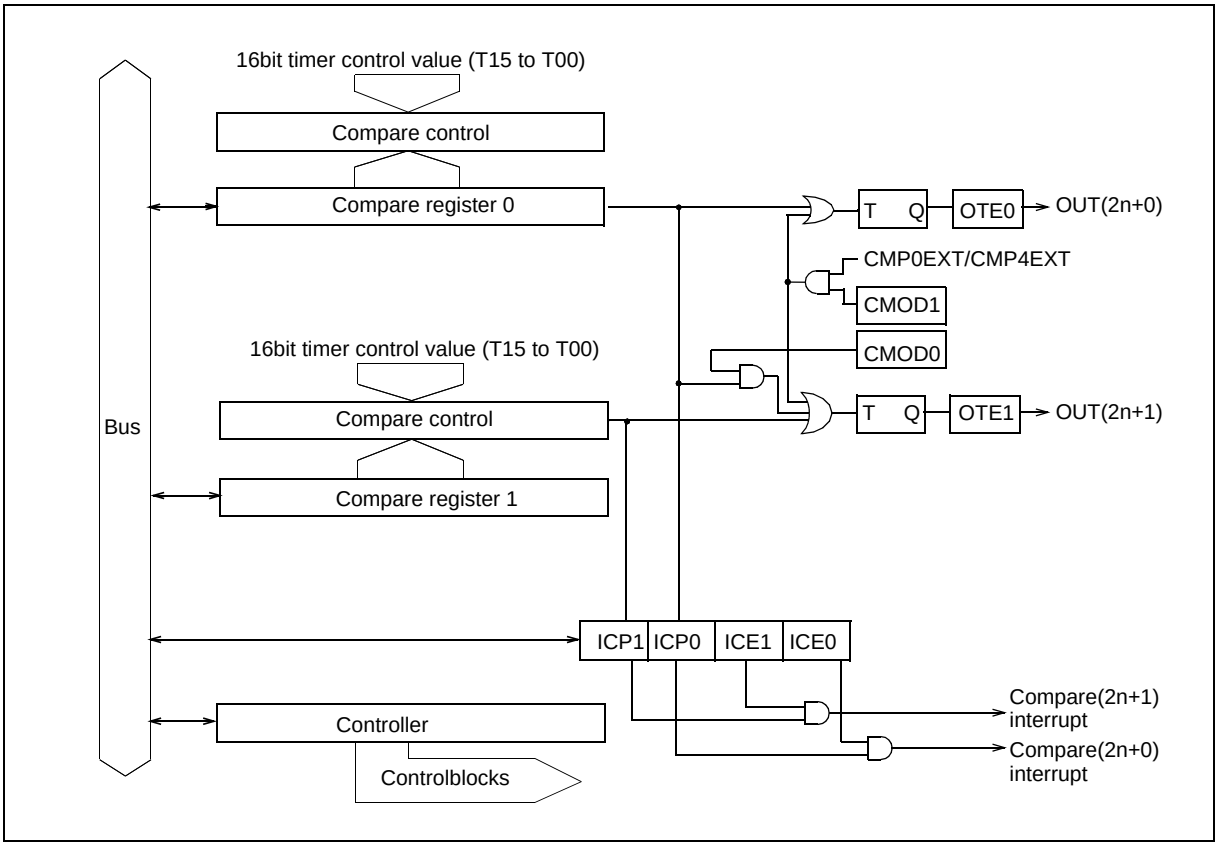
The Output Compare Unit (OCU) consists of two 16-bit compare registers, two compare output pins, and one control register. If the value written to the compare register of this module matches the 16-bit free-running timer value, the output level of the pin can be toggled and an interrupt can be issued.

#### Features of Output Compare Unit (OCU)

- For each Output Compare Unit, two compare registers exist which can be used independently. Depending on the mode setting, the two compare registers can be used to control pin outputs.
- The initial value for each pin output can be specified separately.
- An interrupt can be issued upon a match as a result of comparison.
- Each OCU can generate one PWM signal using two compare registers. (Except for OUT0 and OUT4 pins)
- Two PWM signals can be generated by one output compare module using three compare registers.
- The compare registers can be used to generate one cyclic waveform signal.

### Output Compare Unit block diagram

Figure 14-8. Output Compare Unit block diagram



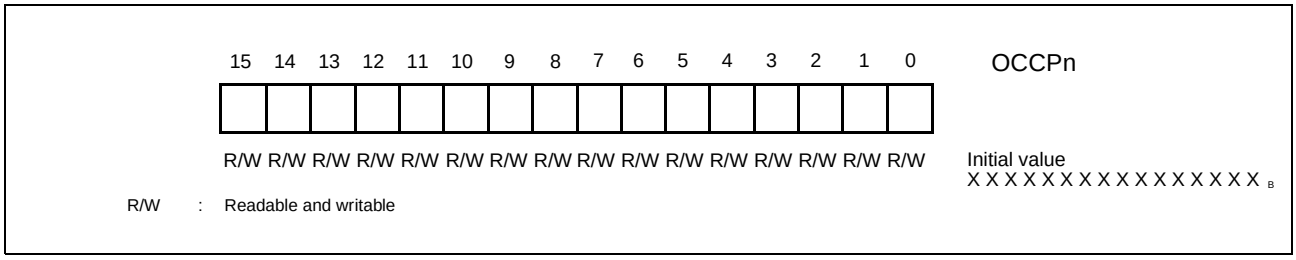

## 14.4.1 Output Compare Register (OCCP(2n) / OCCP(2n+1))

The 16-bit Output Compare registers are compared with the 16-bit Free-Running Timer. Since the initial register values are undefined, set appropriate value before enabling the operation. These registers must be accessed by the word access instructions. When the value of the register matches that of the 16-bit Free-Running Timer, a compare signal is generated and the output compare interrupt flag is set. If output is enabled, the output level corresponding to the compare register is reversed.

If a compare match coincides with write operation to this register, resulting compare operation is not predictable. Therefore make sure to evaluate the counter value of the free running timer or to initialize the timer before writing to this register.

### Output Compare Register (OCCP(2n) / OCCP(2n+1))

Figure 14-9. Output Compare Register (OCCP(2n) / OCCP(2n+1))



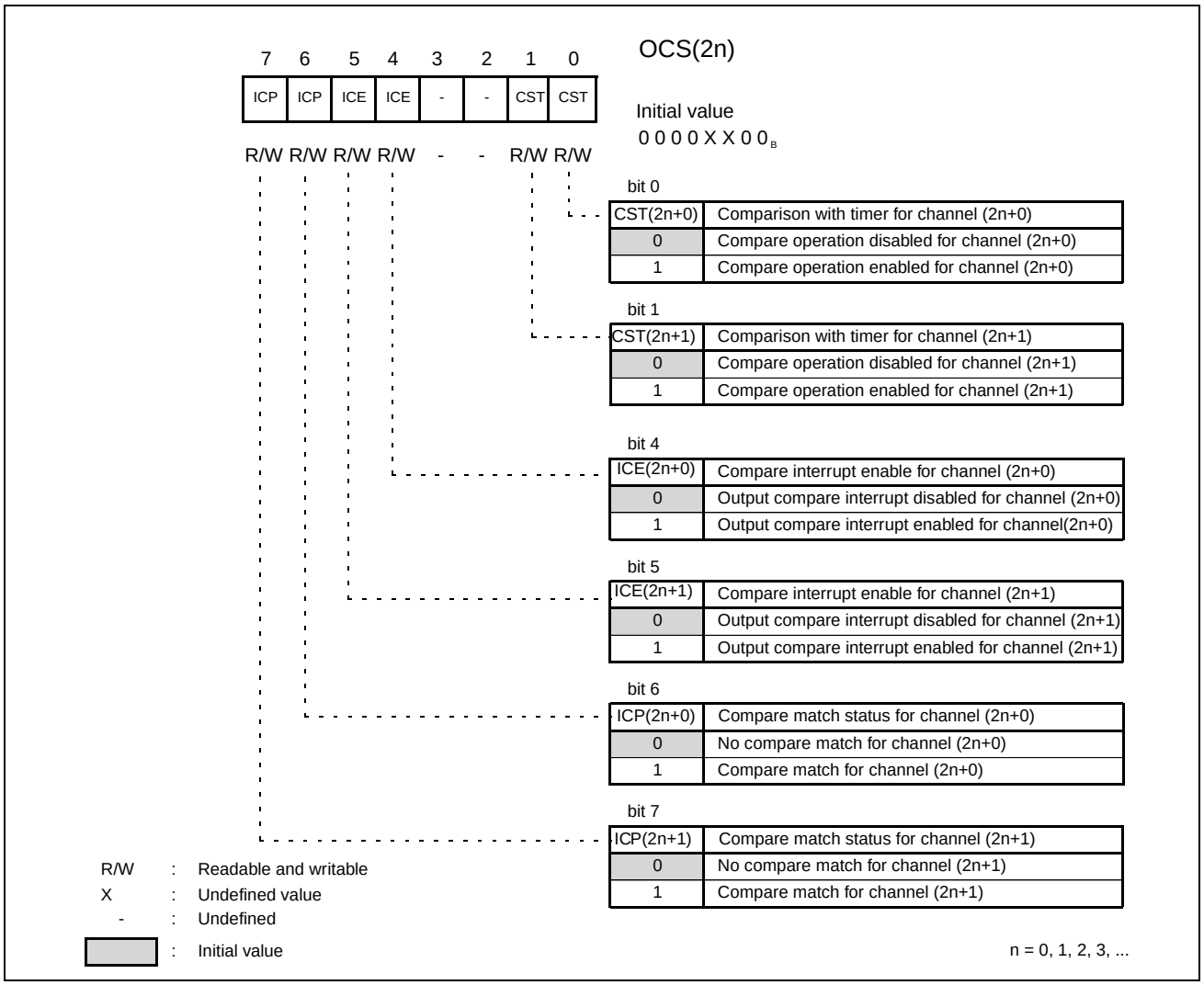

This register can only be accessed as 16-bit word

## 14.4.2 Control Status Registers of Output Compare (OCS(2n) / OCS(2n+1))

The Control Status Register (OCS(2n) / OCS(2n+1)) sets the operation mode of output compare, starts and stops output compare, controls interrupts, and sets the external output pins.

### Output Compare Control Status Register (OCS(2n))

Figure 14-10. Output Compare Control Status Register (OCS(2n))




**Remark:**

The suffix "n" denotes the Output Compare Unit number (0, 1, 2, ...). The register name and the bit names are composed by the register/bit type name and the suffix. For example:

- for Output Compare Unit 0: n = 0, hence OCS0 has bits CST0, CST1, ICE0, ICE1, ICP0, ICP1
- for Output Compare Unit 1: n = 1, hence OCS2 has bits CST2, CST3, ICE2, ICE3, ICP2, ICP3 etc.

Table 14-3. Output Compare Control Status Register (OCS(2n))

| Bit name | | Function |
|---|---|---|
| bit 7 | ICP(2n+1) | • These bits are used as compare match status flags. When the compare register value matches the 16-bit free-run timer value, the bit is set to "1".<br>• When a compare match status bit ICP(2n+1) or ICP(2n+0) bit is set and the corresponding interrupt enable bit ICE(2n+1) or ICE(2n+0) is enabled, an output compare interrupt occurs.<br>• These bits are cleared by writing "0".<br>• "0": No compare match.<br>• "1": Compare match.<br>• Writing "1" has no effect.<br>• "1" is always read by a read-modify-write instruction. |
| bit 6 | ICP(2n+0) | |
| bit 5 | ICE(2n+1) | • These bits are used as output compare interrupt enable flags.<br>• When a compare match status bit ICP(2n+1) or ICP(2n+0) bit is set and the corresponding interrupt enable bit ICE(2n+1) or ICE(2n+0) is enabled, an output compare interrupt occurs.<br>• Writing "0": Output compare interrupt disabled.<br>• Writing "1": Output compare interrupt enabled. |
| bit 4 | ICE(2n+0) | |
| bit 3 to 2 | Undefined | • Always write "0" to these bits<br>• The read value is undefined<br>• Read-modify-write is not affected. |
| bit 1 | CST(2n+1) | • These bits are used to enable the compare operation<br>• Writing "0": Compare operation disabled.<br>• Writing "1": Compare operation enabled.<br>**Note:**<br>Ensure that a value is written to the compare register before the compare operation is enabled.<br>Since output compare is synchronized with the 16-bit Free-Running Timer clock, stopping the 16-bit Free-Running Timer stops compare operation. |
| bit 0 | CST(2n+0) | |

n = 0, 1, 2, 3, ... is the Output Compare Unit number. The bit names are composed by the bit type name and the suffix, e. g. for n = 0: OCS0 has CST(2n+0) = CST0, CST(2n+1) = CST1.

for n = 1: OCS2 has CST(2n+0) = CST2, CST(2n+1) = CST3

### Output Compare Control Status Register (OCS(2n+1))

Figure 14-11. Output Compare Control Status Register (OCS(2n+1))

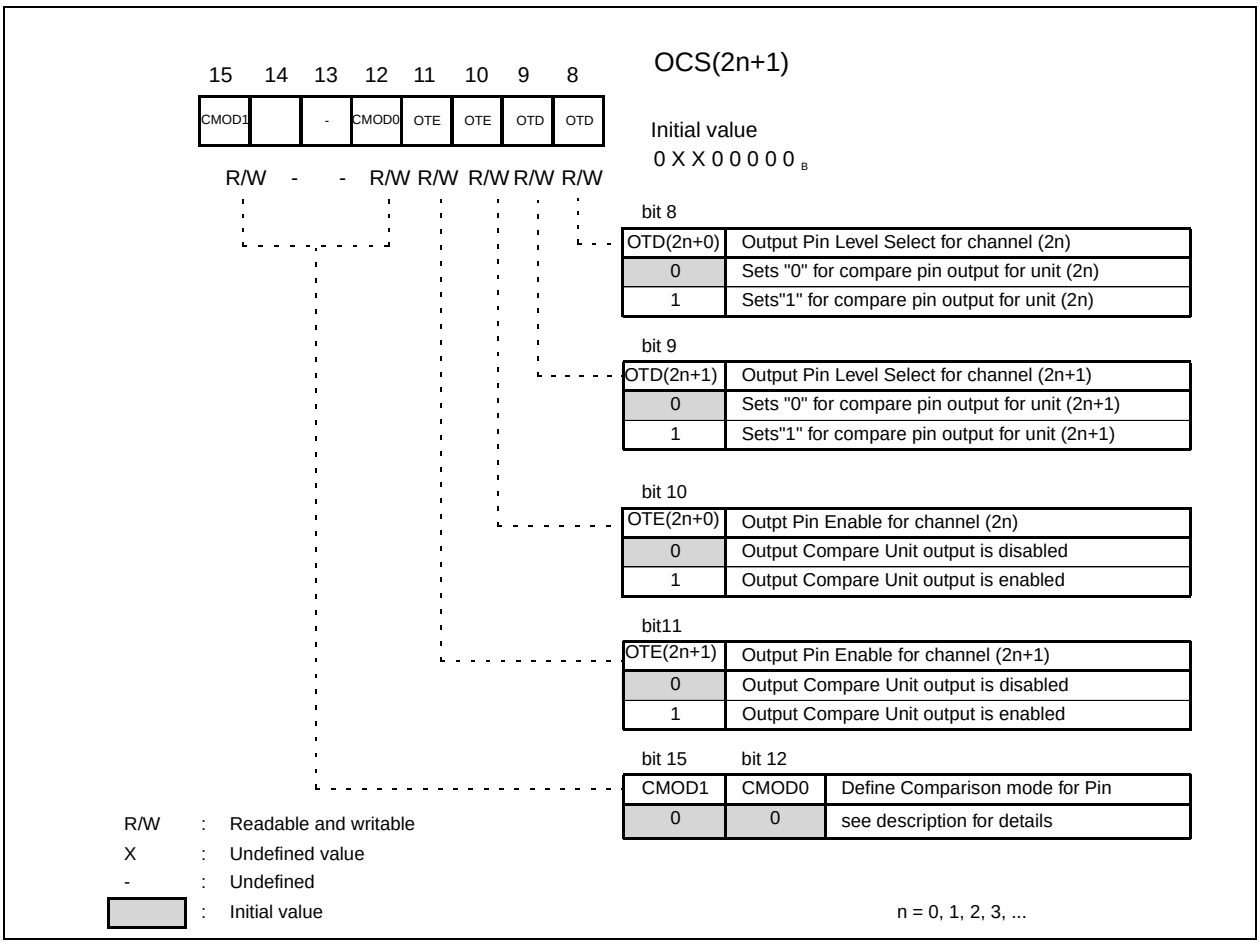



Table 14-4. Output Compare Control Status Register (OCS(2n+1))

| Bit name | | Function |
|---|---|---|
| bit 15 and 12 | CMOD0, CMOD1 | These bits define the operation mode for the pin output signals. Depending on the defined mode, the level is toggled upon a match with different compare registers. See Table 14-5 and Section 14.4.3 16-bit Output Compare Operation. |
| bit 14, 13 | Undefined | • Always write "0" to these bits<br>• The read value is undefined<br>• Read-modify-write is not affected. |
| bit 11 | OTE(2n+1) | These bits are used to enable the Output Compare Unit output pins. The initial value for these bits is "0".<br>• "0": Output Compare Unit output is disabled. The corresponding pin can be used as general-purpose port or for other peripheral function.<br>• "1": Output Compare Unit output is enabled. |
| bit 10 | OTE(2n+0) | |
| bit 9 | OTD(2n+1) | These bits are used to change the pin output level when the Output Compare Unit output pin is enabled. The initial value of the Output Compare Unit output pin is "0". These bits can only be changed if the compare operation for the corresponding channel is disabled (OCS(2n):CST(2n+0) or OCS(2n):CST(2n+1)). When read, these bits indicate the Output Compare Unit output pin value.<br>• Writing "0": Sets "0" for Output Compare Unit output pin.<br>• Writing "1": Sets "1" for Output Compare Unit output pin. |
| bit 8 | OTD(2n+0) | |

n = 0, 1, 2, 3, ... is the Output Compare Unit number. The bit names are composed by the bit type name and the suffix, e. g. for n=0: OCS(2n+1) = OCS1 has bits OTD(2n+0) = OTD0, OTD(2n+1) = OTD1

for n=1: OCS(2n+1) = OCS3 has bits OTD(2n+0) = OTD2, OTD(2n+1) = OTD3.

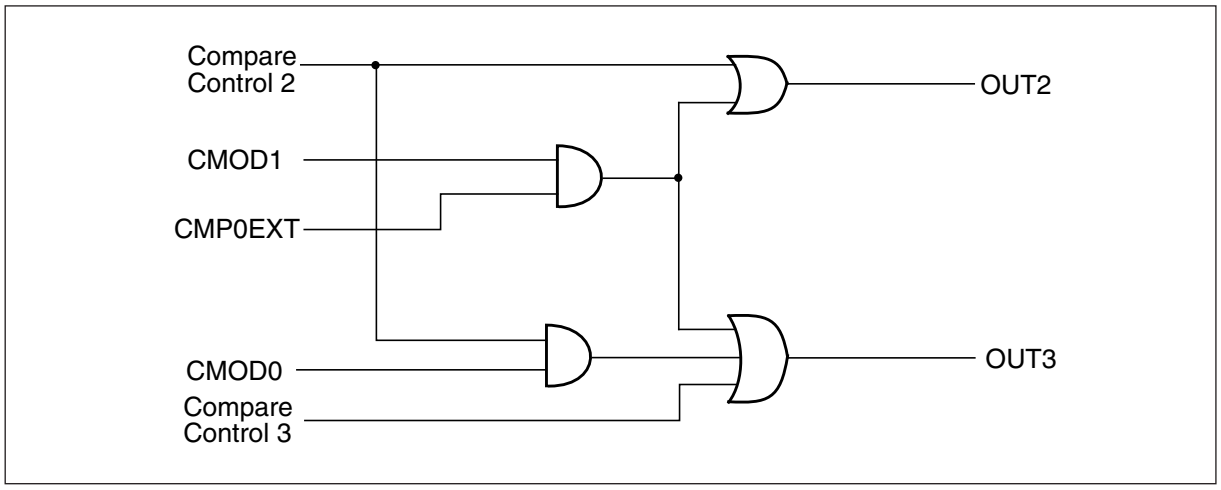
Table 14-5. Function of CMOD1 and CMOD0 bits

Pin output value reversed upon match with compare register number.
How to read the table:

| OCS(2n+1) | | Register OCCPx | |
|---|---|---|---|
| CMOD1 | CMOD0 | output pin OUT(2n+0) | output pin OUT(2n+1) |
| 0/1/x | 0/1/x | OUT(2n+0) toggles at match with the listed OCCP numbers | OUT(2n+1) toggles at match with the listed OCCP numbers |
| 0/1/x | 0/1/x | OUT(2n+0) toggles at match with the listed OCCP numbers | OUT(2n+1) toggles at match with the listed OCCP numbers |

| OCS1 | | Toggle upon match of OCCPx | |
|---|---|---|---|
| CMOD1 | CMOD0 | OUT0 | OUT1 |
| x | 0 | x = 0 | x = 1 |
| x | 1 | x = 0 | x = 0, x = 1 |
| **OCS3** | | **Toggle upon match of OCCPx** | |
| CMOD1 | CMOD0 | OUT2 | OUT3 |
| 0 | 0 | x = 2 | x = 3 |
| 0 | 1 | x = 2 | x = 2, x = 3 |
| 1 | 0 | x = 0, x = 2 | x = 0, x = 3 |
| 1 | 1 | x = 0, x = 2 | x = 0, x = 2, x = 3 |
| **OCS5** | | **Toggle upon match of OCCPx** | |
| CMOD1 | CMOD0 | OUT4 | OUT5 |
| x | 0 | x = 4 | x = 5 |
| x | 1 | x = 4 | x = 4, x = 5 |
| **OCS7** | | **Toggle upon match of OCCPx** | |
| CMOD1 | CMOD0 | OUT6 | OUT7 |
| 0 | 0 | x = 6 | x = 7 |
| 0 | 1 | x = 6 | x = 6, x = 7 |
| 1 | 0 | x = 4, x = 6 | x = 4,x = 7 |
| 1 | 1 | x = 4, x = 6 | x = 4, x = 6, x = 7 |
| **OCS9** | | **Toggle upon match of OCCPx** | |
| CMOD1 | CMOD0 | OUT8 | OUT9 |
| x | 0 | x = 8 | x = 9 |
| x | 1 | x = 8 | x = 8, x = 9 |
| **OCS11** | | **Toggle upon match of OCCPx** | |
| CMOD1 | CMOD0 | OUT10 | OUT11 |
| x | 0 | x = 10 | x = 11 |
| x | 1 | x = 10 | x = 10, x = 11 |

Figure 14-12. Block diagram of output selection (OCU module 1)



For OCU module 1, which requires a match with Output Compare Register 0 if CMOD[1:0] = “$10_B$” the comparison result from module 0 is carried inside by the CMP0EXT signal. Of course, this does not apply to module 0 itself. Here, no other register can be used but OCCP0 and OCCP1.

The equivalent situation applies to OCU module 3, where the result from module 2 is needed as CMP4EXT.

## 14.4.3 16-bit Output Compare Operation

In the 16-bit Output Compare operation, an interrupt request flag can be set and the output level can be toggled when the specified compare register value matches the 16-bit free-run timer value. The CMOD0 and CMOD1 bits can be used to define the corresponding compare registers for each pin.

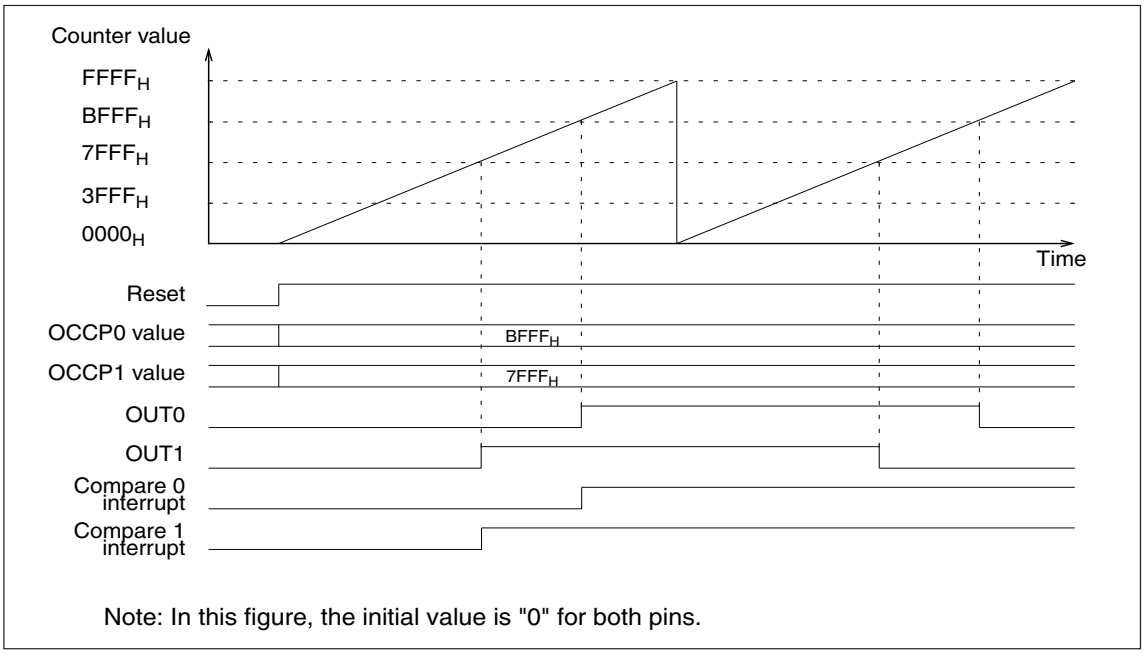
### *14.4.3.1 Sample output waveform when CMOD[1:0] = "$00_B$"*

When CMOD[1:0] = “$00_B$”, the output level of the pin corresponding to the compare register is reversed on every match with the register value. Each output value is controlled by one compare register.

OUT0: The level is only reversed by a match with compare register 0.

OUT1: The level is only reversed by a match with compare register 1.

Figure 14-13. Sample of output waveform when CMOD[1:0] = "$00_B$"




### *14.4.3.2 Sample output waveform with two compare registers when CMOD[1:0] = "$01_B$"*

When CMOD[1:0] = “$01_B$”, the output level of the pin corresponding to compare register 0 (2) is reversed upon every match with the register value. This is identical to the behavior for CMOD[1:0] = “$00_B$”. However, the output level of the second pin is reversed upon a match with either compare register 0 or compare register 1 (3). This allows to define a pulsed signal with one edge defined by the value in compare register 0 and the other edge defined by compare register 1 (3) or vice versa. If both compare registers have the same value, the operation is identical to the case for CMOD[1:0] = “$00_B$”.

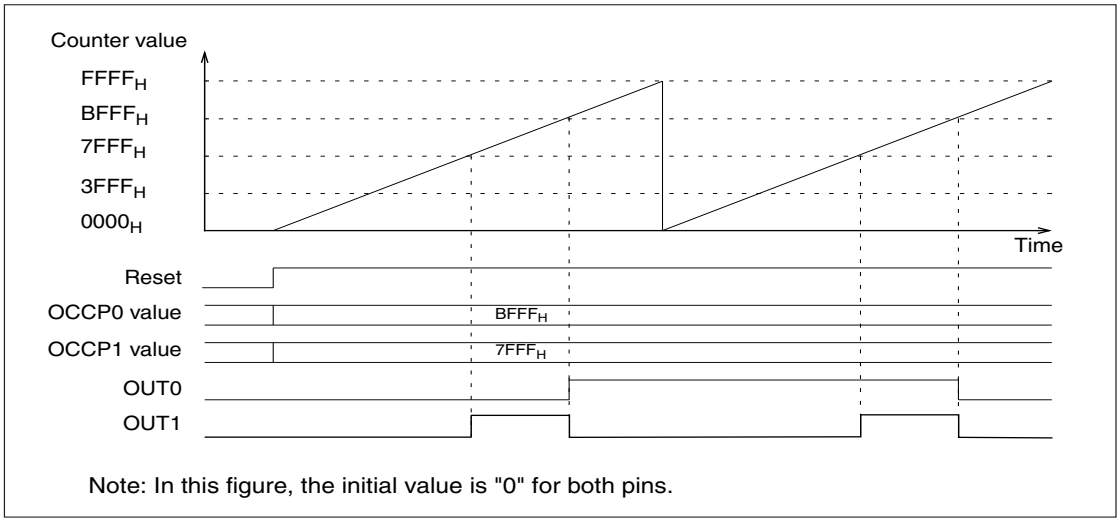
A pulse width modulated signal with differing frequency can be defined by using this mode together with the reset option by compare register match for the Free-Running Timer (MODE-bit in TCCSLx registers).

OUT0 (2): The level is only reversed by a match with compare register 0 (2).

OUT1 (3): The level is reversed by a match with compare register 0 (2) or with compare register 1 (3).

For OUT4, OUT5, OUT6 and OUT7, compare register 4 plays the same role as compare register 0 above.

Figure 14-14. Sample of a output waveform when CMOD[1:0] = "$01_B$" (no timer reset by match)




### *14.4.3.3 Sample output waveform when CMOD[1:0] = "$10_B$"*

The operation mode defined by CMOD[1:0] = “$10_B$” is intended for the use of three pulse width modulated signals for each Free-Running Timer instead of two. If this mode is set to OCU module 1, a match of the timer value with compare register 0 reverses both OUT2 and OUT3. For the third pulsed signal, the CMOD[1:0] bits of OCU module 0 should be set to “$01_B$”.

In register OCS1: CMOD[1:0] = “$01_B$”

OUT0: The level is only reversed by a match with compare register 0.

OUT1: The level is reversed by a match with compare register 0 or with compare register 1.

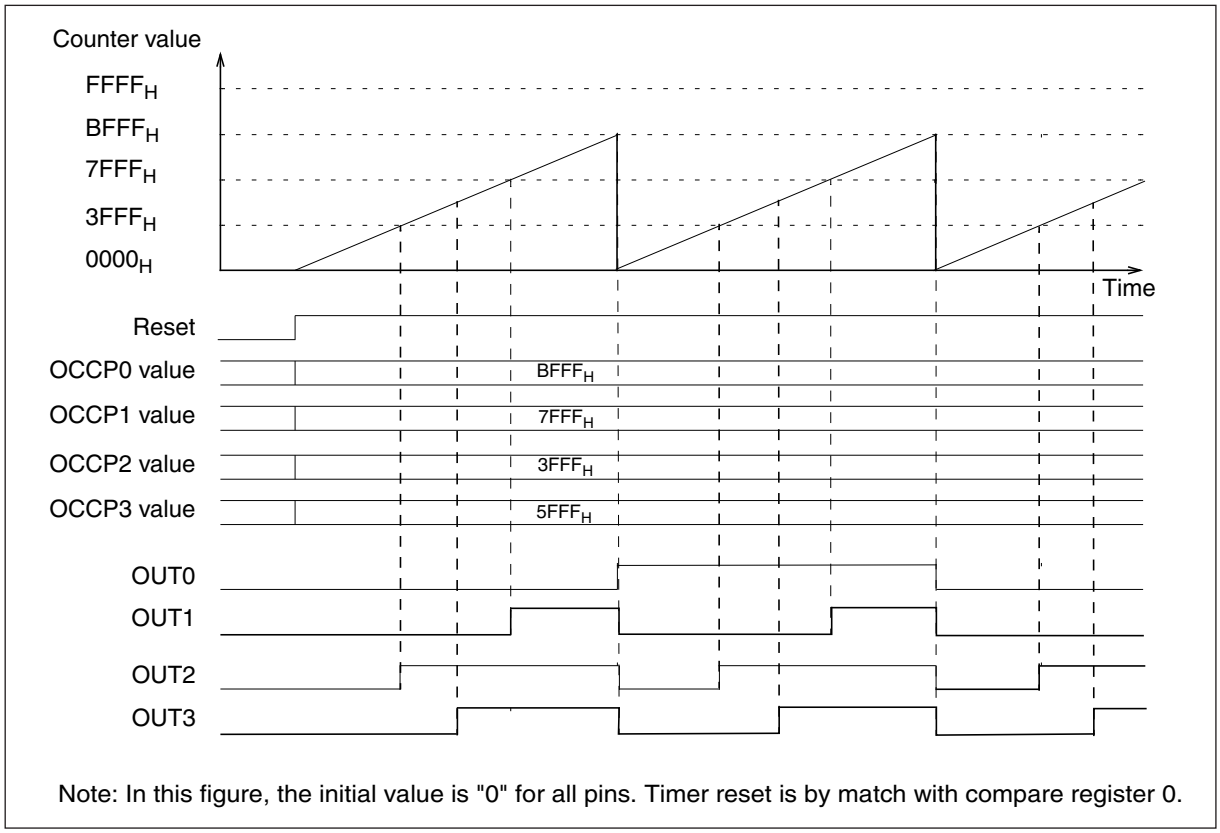
In register OCS3: CMOD[1:0] = “$10_B$”

OUT2: The level is reversed by a match with compare register 0 or with compare register 2.

OUT3: The level is reversed by a match with compare register 0 or with compare register 3.

For OUT4, OUT5, OUT6 and OUT7, compare register 4 plays the same role as compare register 0 above.

Figure 14-15. Output waveform when OCS1:CMOD[1:0] = "$01_B$" and OCS3:CMOD[1:0] = "$10_B$"




Note: In this figure, the initial value is "0" for all pins. Timer reset is by match with compare register 0.

### *14.4.3.4 Sample output waveform when CMOD[1:0] = "$11_B$"*

When CMOD[1:0] = “$11_B$”, the output level of the OUT3 (OUT7) pin is reversed by the compare registers 0, 2 or 3 (4, 6 or 7). For the pin OUT1 (OUT5), this setting is identical to CMOD[1:0] = “$01_B$” (see also Table 14-5)

OUT0: The level is only reversed by a match with compare register 0.

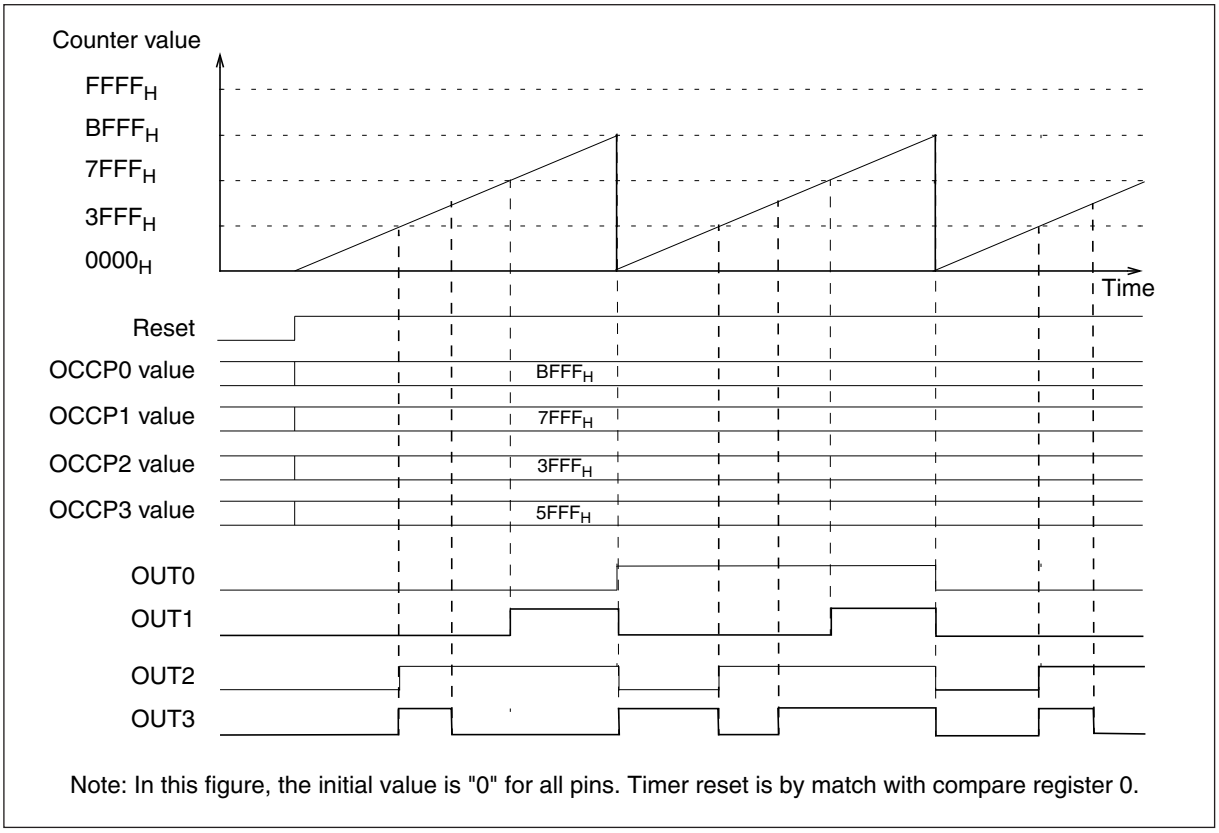
OUT1: The level is reversed by a match with compare register 0 or with compare register 1.

OUT2: The level is reversed by a match with compare register 0 or with compare register 2.

OUT3: The level is reversed by a match with compare register 0, compare register 2 or with compare register 3.

For OUT4, OUT5, OUT6 and OUT7, compare register 4 plays the same role as compare register 0 above.

Figure 14-16. Output waveform when OCS1:CMOD[1:0] = "$11_B$" and OCS3:CMOD[1:0] = "$11_B$"




Note: In this figure, the initial value is "0" for all pins. Timer reset is by match with compare register 0.

### *14.4.3.5 Output compare timing*

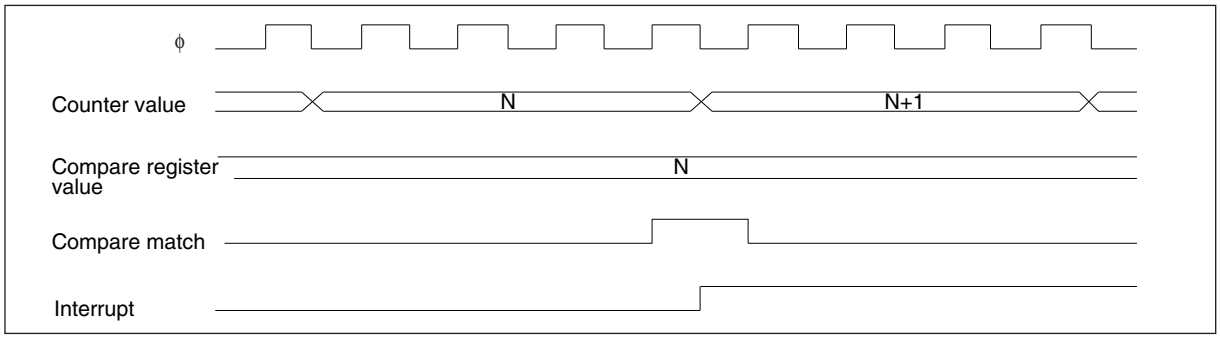
In output compare operation, a compare match signal is generated when the Free-Running Timer value matches the specified compare register value. The output value can be reversed and an interrupt can be issued. The output reverse timing upon a compare match is synchronized with the counter timing.

#### Compare operation upon update of compare register

When the compare register is updated, comparison with the counter value is not performed.

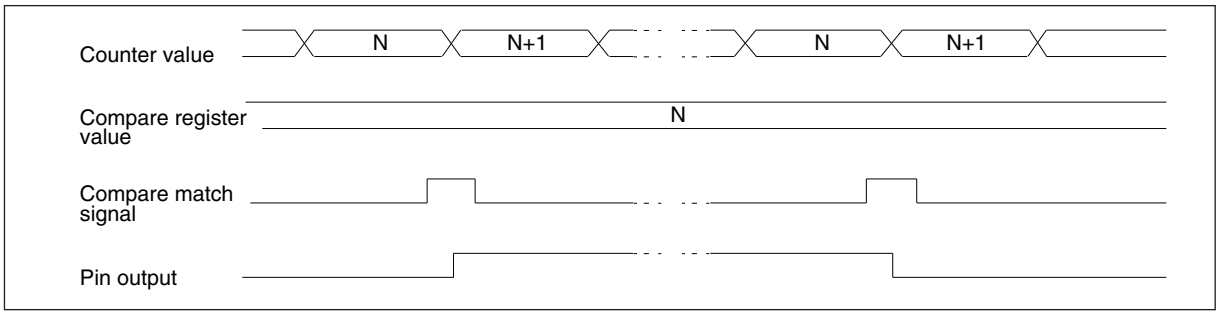
#### Interrupt timing

Figure 14-17. Interrupt timing



### Output pin change timing

Figure 14-18. Output pin change timing




## 14.5 Input Capture Unit

The Input Capture Unit (ICU) detects a rising or falling edge or both edges of an external input signal and stores a 16-bit Free-Running Timer value at that time in a register. In addition, the Input Capture Unit can generate an interrupt upon detection of an edge. One Input Capture Unit consists of two input capture registers and one control register.

### Features of the Input Capture Unit

- The valid edge of an external input can be selected from the following three types:

Table 14-6. Types of external input edges

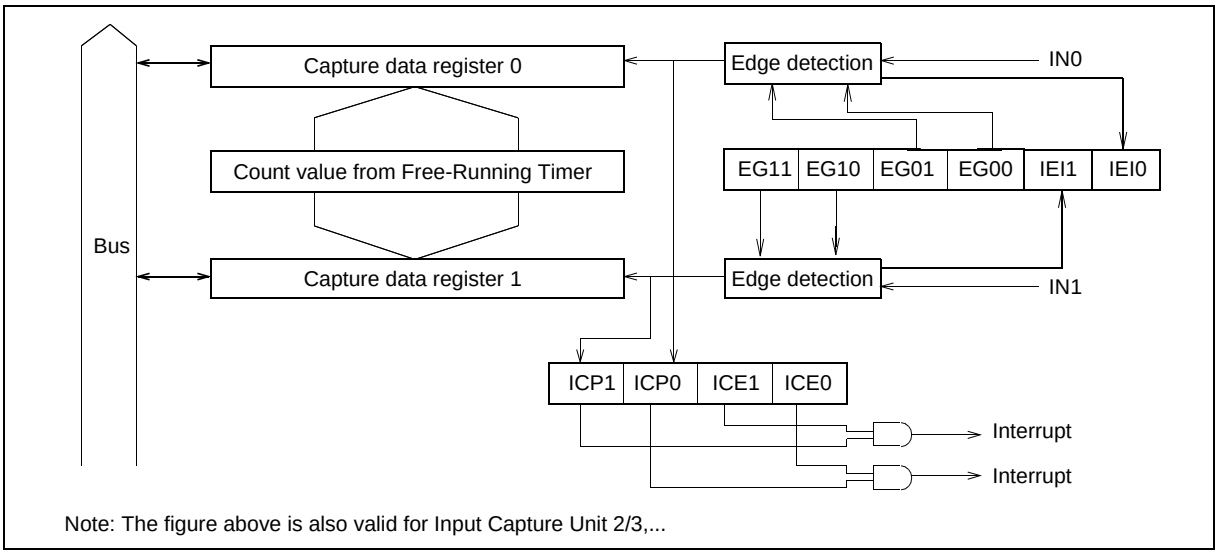
| | |
|---|---|
| Rising edge | ↑ |
| Falling edge | ↓ |
| Both edges | ↑↓ |

- An interrupt can be generated upon detection of a valid edge of an external input.

### Input Capture Unit block diagram

Figure 14-19 shows a block diagram of the Input Capture Unit.

Figure 14-19. Input Capture Unit block diagram



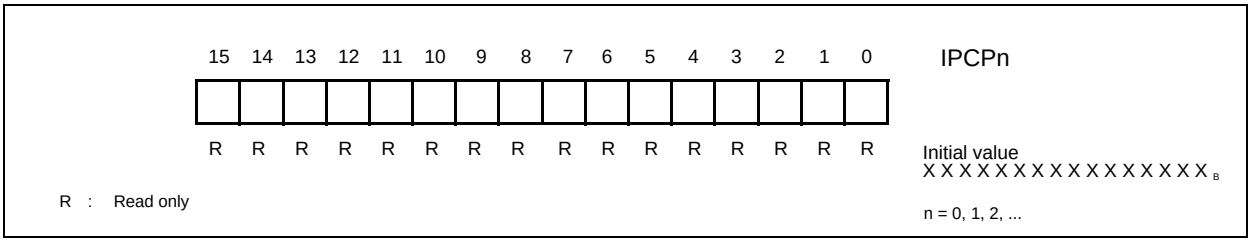

## 14.5.1 Input Capture Unit Register Details

The Input Capture Unit is configured by the Control Status Register (ICS(2n)(2n+1)) and the Edge Register (ICE(2n)(2n+1)).

The Input Capture Unit has a 16bit data register (IPCPn). This register stores a value from the 16-bit Free-Running Timer when a valid edge of the corresponding external pin input waveform is detected. For data consistency, it should be accessed only in word mode.

### Input Capture Data Register (IPCPn)

Figure 14-20. Input Capture Data Register (IPCPn)

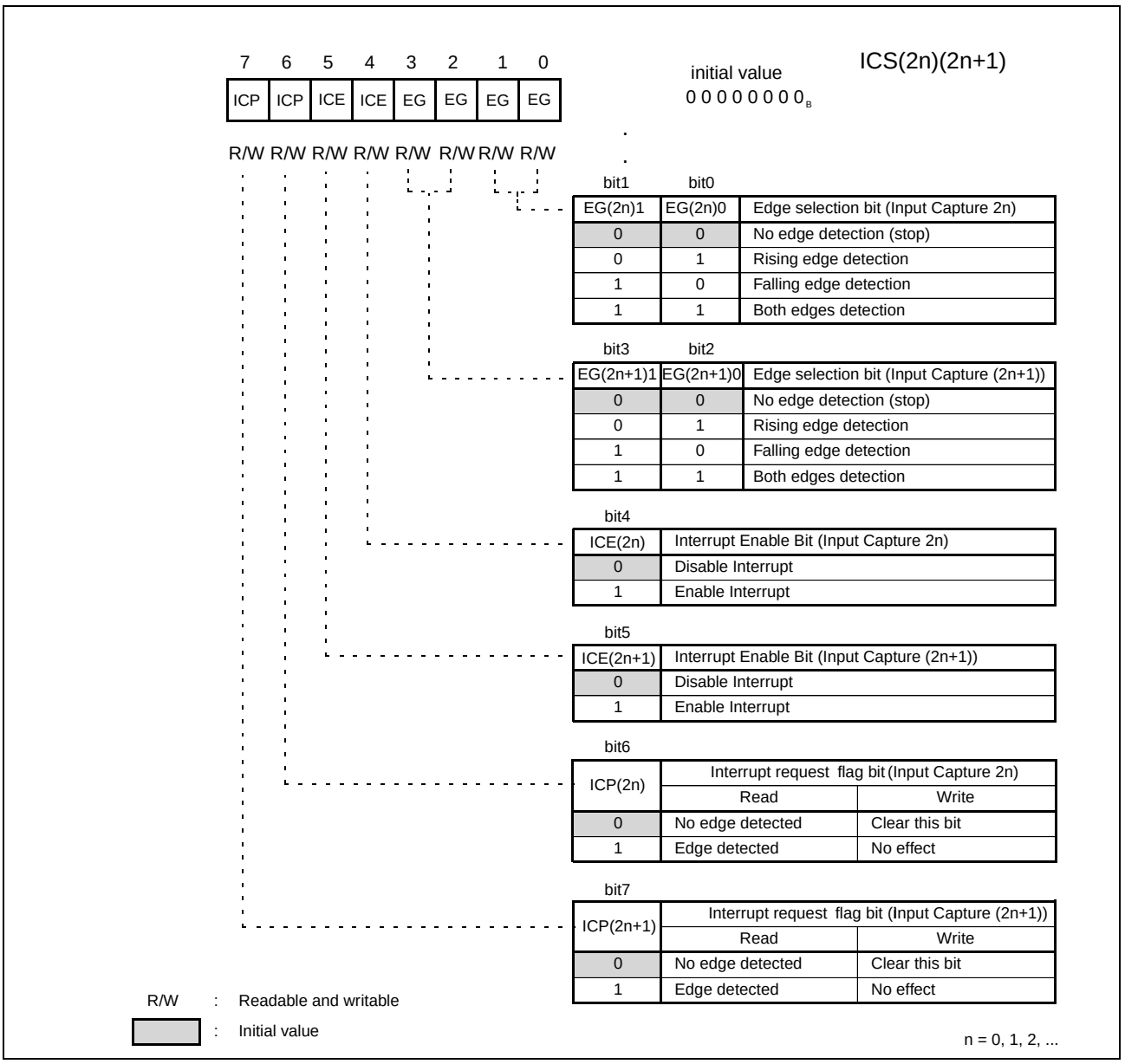



### Remark:

For consistency of data in the upper and lower byte, always read this register in word-mode.

### Input Capture Control Status Register (ICS(2n)(2n+1))

Figure 14-21. Input Capture Control Status Register (ICS(2n)(2n+1))




### Remark:

The suffix "n" denotes the Input Capture Unit number (0, 1, 2, ...). The register name and the bit names are composed by the register/bit type name and the suffix. For example:

- for Input Capture Unit 0: n = 0, hence ICS01 has bits EG01, EG00, EG11, EG10, ICE0, ICE1, ICP0, ICP1
- for Input Capture Unit 1: n = 1, hence ICS23 has bits EG21, EG20, EG21, EG20, ICE2, ICE3, ICP2, ICP3
- etc.

Table 14-7. Input Capture Unit Control Status Register bits

| Bit name | | Function |
|---|---|---|
| bit7 | ICP(2n+1):<br>Interrupt request flag bit (Input capture 2n+1) | • This bit is used as interrupt request flag for Input Capture Unit n, second channel.<br>• "1" is set to this bit upon detection of a valid edge of an external input pin.<br>• While the interrupt enable bit (ICE(2n+1)) is set, an interrupt can be generated upon detection of a valid edge.<br>• Writing "0" will clear this bit.<br>• Writing "1" has no effect.<br>• In read-modify-write operation, "1" is always read. |
| bit6 | ICP(2n):<br>Interrupt request flag bit (Input capture 2n) | • This bit is used as interrupt request flag for Input Capture Unit n, first channel.<br>• "1" is set to this bit upon detection of a valid edge of an external input pin.<br>• While the interrupt enable bit (ICE(2n)) is set, an interrupt can be generated upon detection of a valid edge.<br>• Writing "0" will clear this bit.<br>• Writing "1" has no effect.<br>• In read-modify-write operation, "1" is always read. |
| bit5 | ICE(2n+1):<br>Interrupt request enable bit (Input capture 2n+1) | • This bit is used to enable input capture interrupt request for Input Capture Unit n, second channel.<br>• While “1” is written to this bit, an input capture interrupt is generated when the interrupt flag (ICP(2n+1)) is set. |
| bit4 | ICE(2n):<br>Interrupt request enable bit (Input capture 2n) | • This bit is used to enable input capture interrupt request for Input Capture Unit n, first channel.<br>• While “1” is written to this bit, an input capture interrupt is generated when the interrupt flag (ICP(2n)) is set. |
| bit3/2 | EG(2n+1)1, EG(2n+1)0 | • These bits are used to specify the valid edge polarity of an external input for Input Capture Unit n, second channel<br>• These bits are also used to enable input capture operation |
| bit1/0 | EG(2n)1, EG(2n)0 | • These bits are used to specify the valid edge polarity of an external input for Input Capture Unit n, first channel.<br>• These bits are also used to enable input capture operation |

The suffix n = 0, 1, 2, 3, ... denotes the Input Capture Unit number. The bit names are composed by their type name and the suffix. Hence, for

n = 0: ICP1, ICP0, ICE1, ICE0, EG11, EG10, EG01, EG00

n = 1: ICP3, ICP2, ICE3, ICE2, EG31, EG30, EG21, EG20 etc.

### Input Capture Unit Edge Register (ICE(2n)(2n+1))

The Input Capture Unit Edge Register (ICE(2n)(2n+1)) contain device dependent configuration bits. These registers are described in Chapter Input Capture Unit Source Select for LIN-USART on page 27.

Figure 14-22. Input Capture Unit Edge Register (ICE(2n)(2n+1))

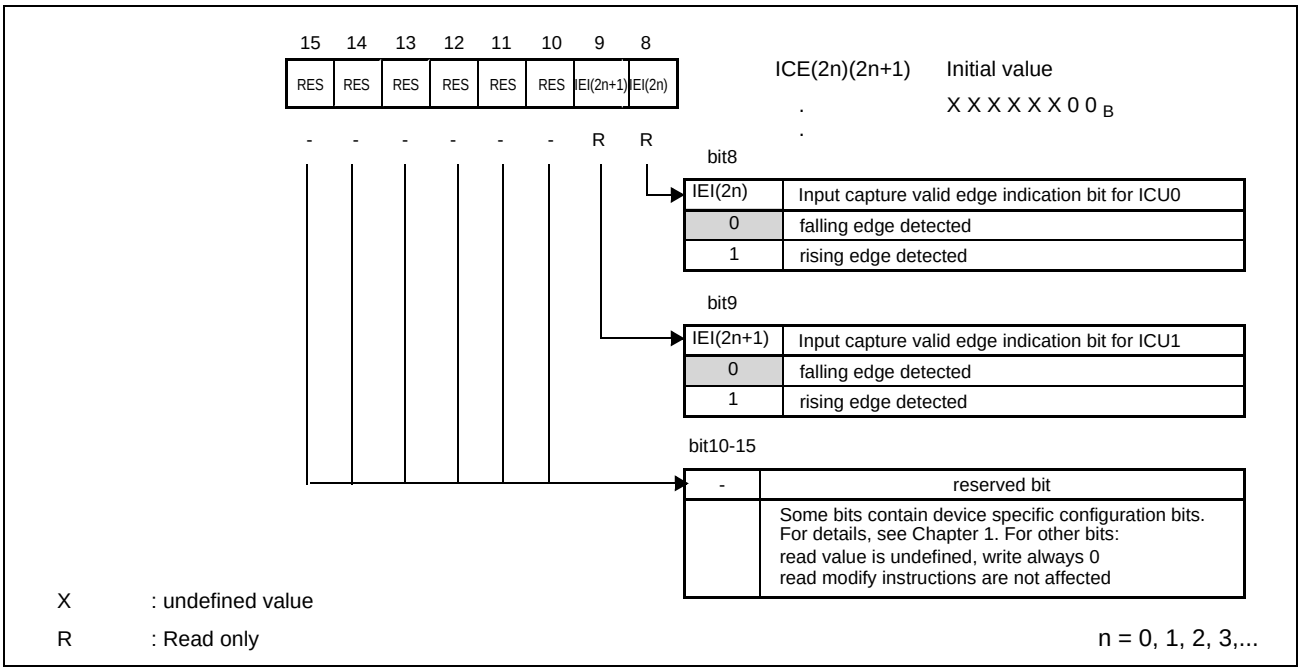



Table 14-8. Input Capture Unit Edge Register bits (upper and lower)

| Bit name | | Function |
|---|---|---|
| bit15 to 10 | RES | Some bits contain device specific configuration bits. For details, see Input Capture Unit Source Select for LIN-USART on page 27 |
| bit9 | IEI(2n+1):<br>Valid edge indication bit | • This bit is the edge indication bit for capture register IPCP(2n+1) to indicate that a rising or falling edge is detected.<br>• "0": falling edge detected.<br>• "1": rising edge detected.<br>• This bit is read only.<br>**Note:** The read value is meaningless, if EG(2n+1)1, EG(2n+1)0 = “00”.<br>The read value is independent from edge selected by EG(2n+1)1 and EG(2n+1)0 |
| bit8 | IEI(2n):<br>Valid edge indication bit | • This bit is the edge indication bit for capture register IPCP0, IPCP2, IPCP4 and IPCP6 to indicate that a rising or falling edge is detected<br>• "0": falling edge detected.<br>• "1": rising edge detected.<br>• This bit is read only.<br>**Note**: The read value is meaningless, if EG(2n)1, EG(2n)0 = “00”.<br>The read value is independent from edge selected by EG(2n)1 and EG(2n)0 |

The suffix n = 0, 1, 2, 3, ... denotes the Input Capture Unit number. The bit names are composed by their type name and the suffix. Hence, for
n = 0: IEI1, IEI0
n = 1: IEI3, IEI2 etc.

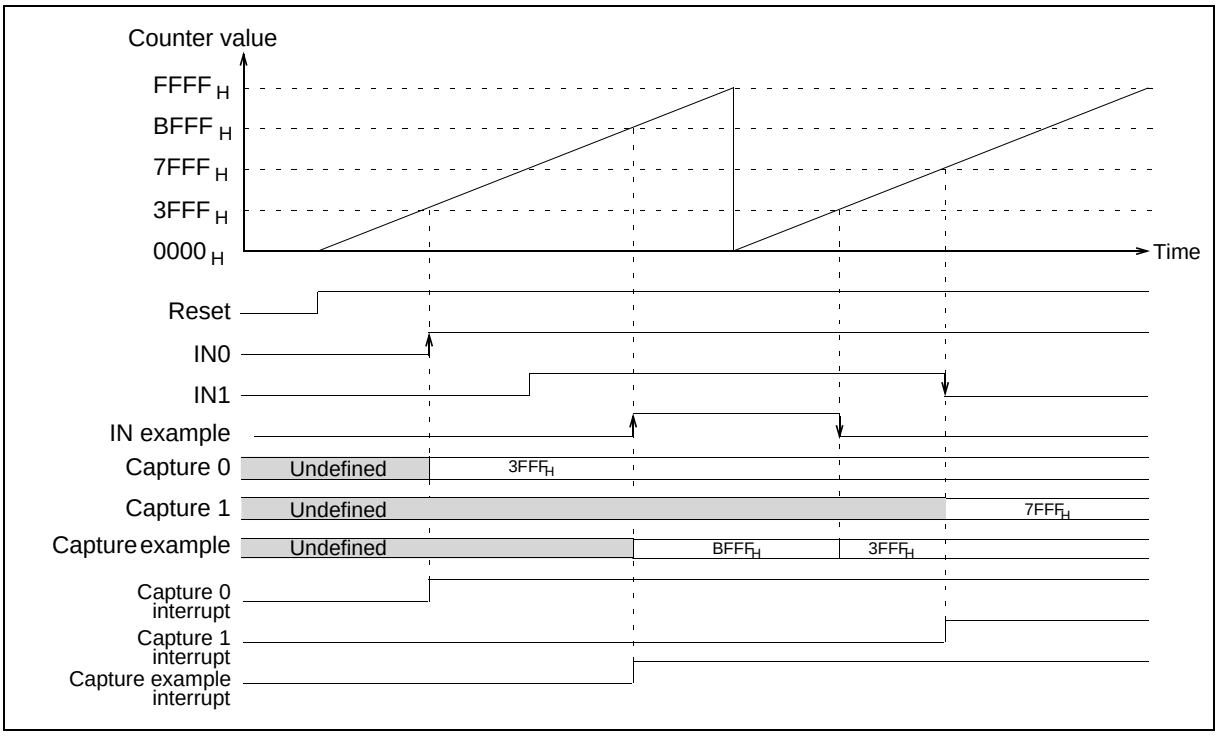
## 14.5.2 16-bit Input Capture Operation

In 16-bit Input Capture operation, an interrupt can be generated upon detection of the specified edge, fetching the 16-bit Free-Running Timer value and writing it to the capture data register.

### *14.5.2.1 Example of input capture fetch timing*

- Capture 0: Rising edge
- Capture 1: Falling edge
- Capture example: Both edges

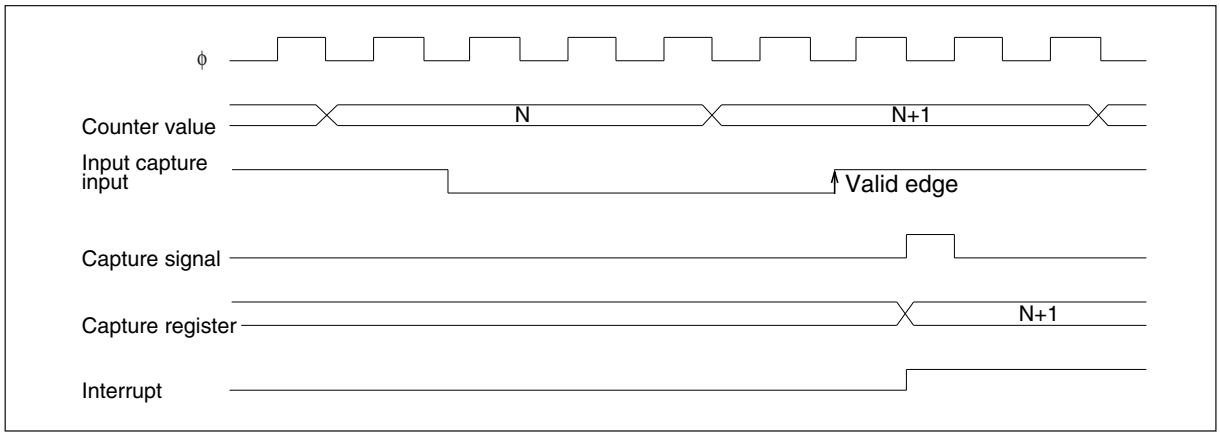
Figure 14-23. Example of Input Capture fetch timing



### *14.5.2.2 Input Capture input timing*

**Capture timing for input signals**

Figure 14-24. Capture timing for input signals



# 15. 16-Bit Reload Timer (With Event Count Function)

This chapter explains the functions and operations of the 16-bit Reload Timer (with the Event Count Function).

15.1 Outline of 16-Bit Reload Timer (with Event Count Function)

15.2 16-Bit Reload Timer (with Event Count Function)

15.3 Internal Clock and External Event Counter Operations of 16-bit Reload Timer

15.4 Underflow Operation of 16-bit Reload Timer

15.5 Output Pin Functions of 16-bit Reload Timer

15.6 Counter Operation State

15.7 Cascading of 16-bit Reload Timers

## 15.1 Outline of 16-Bit Reload Timer (with Event Count Function)

The 16-bit reload timer consists of a 16-bit down-counter, a 16-bit reload register, one input pin (TINn) and one output pin (TOTn), and control registers.

### Outline of 16-bit reload timer (with event count function)

The 16-bit reload timer has the following features

- External and internal clock/event source
- Trigger signal programmable as rising/falling edge or both
- Gated count function
- One-shot or reload counter mode
- Counter state can be made visible at external pin
- Prescaler with 6 different settings for the internal clock and 2 settings for the external clock
- Several Reload Timers can be cascaded to form a longer Reload Timer

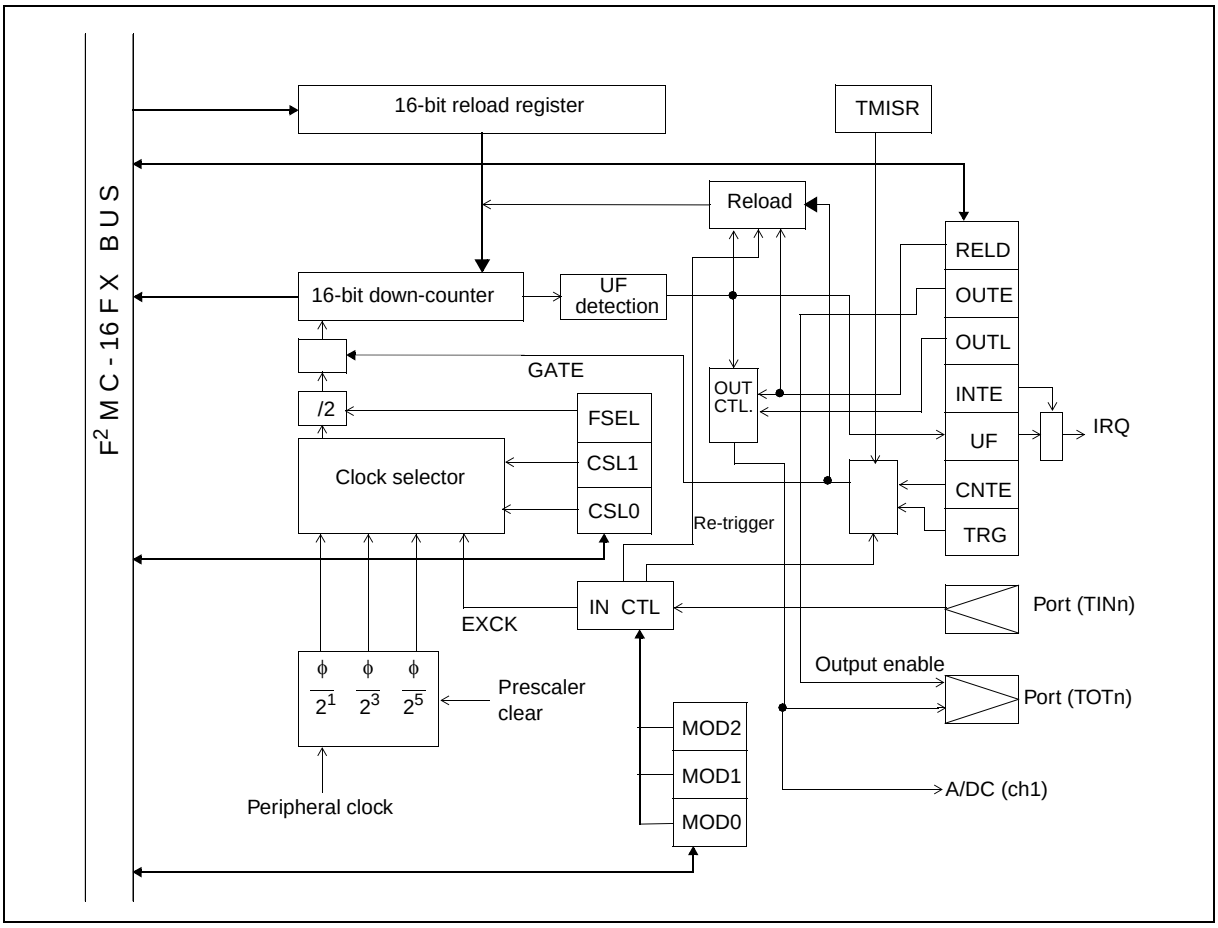
### DMA and interrupts

The timer can generate interrupts which can be used to start DMA transfers.

### Block diagram of 16-bit reload timer

Figure 15-1 shows a block diagram of the 16-bit reload timer.

Figure 15-1. Block diagram of 16-bit reload timer




**Note:**

The suffix "n" denotes the Reload Timer number.

The RLT1 output can start conversion of the A/D converter.

RLT6 timer can be used as PPG clock source. It does not have TIN/TOT pins but, besides that restriction, it can be used as a normal reload timer if not required by the PPG operation.

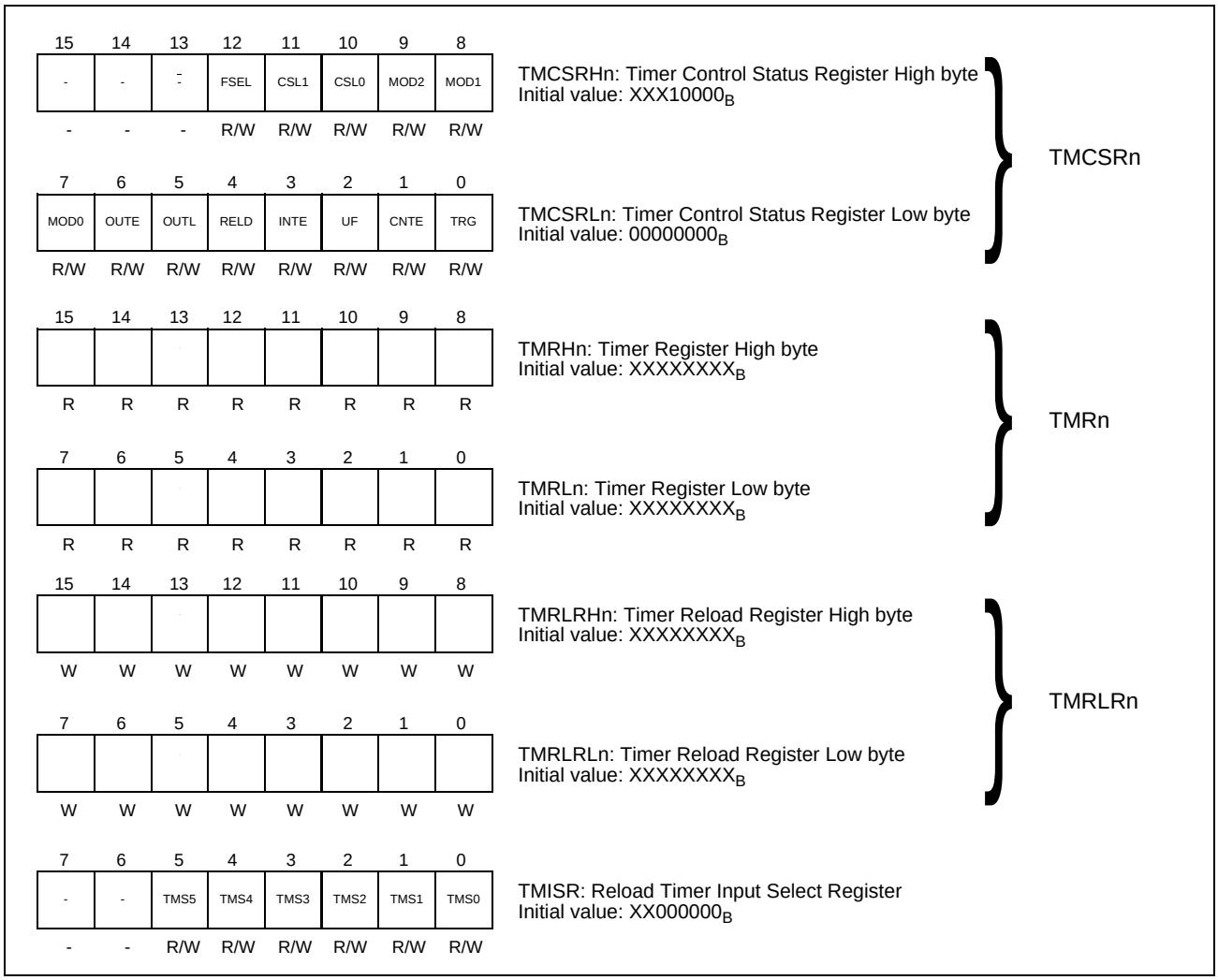
## 15.2 16-Bit Reload Timer (with Event Count Function)

The 16-bit Reload Timer has the following registers:

- Timer Control Status Register (TMCSRn)
- 16-bit Timer Register (TMRn) / 16-bit Reload Register (TMRLRn)

### 16-bit Reload Timer register

Figure 15-2. 16-bit Reload Timer register




The 8-bit registers TMCSRHn, TMCSRLn can be accessed as 16-bit register TMCSRn.

The 8-bit registers TMRHn, TMRLn can be accessed as 16-bit register TMRn.

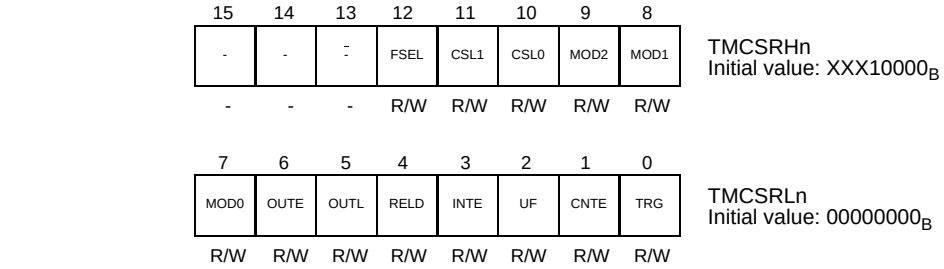
The 8-bit registers TMRLRHn, TMRLRLn can be accessed as 16-bit register TMRLRn.

## 15.2.1 Timer Control Status Register (TMCSRn)

The Timer Control Status Register controls the operation mode and interrupts for the 16-bit Reload Timer.

### *15.2.1.1 Register layout of Timer Control Status Register (TMCSRn)*

Figure 15-3. Register layout of Timer Control Status Register (TMCSRn)




- : irrelevant
X : undefined value
R/W : Readable and writable

### *15.2.1.2 Register contents of Timer Control Status Register (TMCSRn)*

#### [Bit 12] FSEL (Count clock division control)

Specifies the count clock division ratio.

If the FSEL bit is set to "0", the count clock specified by the count clock selection bits (CSL1 and CSL0) is divided by two.

If the FSEL bit is set to "1", the count clock specified by the count clock selection bits (CSL1 and CSL0) is not divided (default).

#### [Bits 11, 10] CSL1, CSL0 (Clock select 1, 0)

Specifies the clock/event source and the clock division ratio.

Table 15-1 lists the selected clock sources for different FSEL and CSL0/1 settings.

Table 15-1. Clock sources for CSL0/1 and FSEL bit settings

| FSEL | CSL1 | CSL0 | Clock Source (Time for peripheral clock CLKP1 = 24 MHz) |
|---|---|---|---|
| 1 | 0 | 0 | CLKP1 / $2^1$ (0.083 µs) |
| 0 | 0 | 0 | CLKP1 / $2^2$ (0.167 µs) |
| 1 | 0 | 1 | CLKP1 / $2^3$ (0.333µs) |
| 0 | 0 | 1 | CLKP1 / $2^4$ (0.667µs) |
| 1 | 1 | 0 | CLKP1 / $2^5$ (1.3 µs) |
| 0 | 1 | 0 | CLKP1 / $2^6$ (2.6 µs) |
| 1 | 1 | 1 | External event count mode |
| 0 | 1 | 1 | External event count mode / 2 |

#### [Bits 9, 8, 7] MOD2, MOD1, MOD0 (Operation mode and TINn function)

These bits set the operation mode and input pin (TINn) functions.

The MOD2 bit selects the input pin (TINn) function. When MOD2 = "0", the pin TINn is used as a trigger input. In this case, the reload register content is loaded to the counter when an active edge is input to the pin TINn and count operation proceeds.

When MOD2 = "1", the timer operates in gated counter mode and the pin TINn is used as a gate input. In this mode, the counter only counts while an active level is input to the pin TINn.

Table 15-2 and Table 15-3 list the MOD2/1/0 bit settings.

Table 15-2. MOD2/1/0 bit settings for internal clock mode (CSL0/1 = "$00_B$", "$01_B$", or "$10_B$")

| MOD2 | MOD1 | MOD0 | Input Pin Function | Active Edge or Level |
|---|---|---|---|---|
| 0 | 0 | 0 | Trigger disabled | - |
| 0 | 0 | 1 | Trigger input | Rising edge |
| 0 | 1 | 0 | | Falling edge |
| 0 | 1 | 1 | | Both edges |
| 1 | x | 0 | Gate input | "L" level |
| 1 | x | 1 | | "H" level |

Table 15-3. MOD2/1/0 bit settings for event counter mode (CSL0/1 = "$11_B$")

| MOD2 | MOD1 | MOD0 | Input Pin Function | Active Edge or Level |
|---|---|---|---|---|
| x | 0 | 0 | - | - |
| | 0 | 1 | Event input | Rising edge |
| | 1 | 0 | | Falling edge |
| | 1 | 1 | | Both edges |

Bits marked as x in the table can be set to any value.

### [Bit 6] OUTE (Output enable)

If this bit is set to "1", the pin TOTn is used as Reload Timer output. If this bit is set to "0" the timer output TOTn is disabled.

### [Bit 5] OUTL (Output level)

This bit sets the output level for the TOTn pin.

### [Bit 4] RELD (Reload)

This bit enables reload operations.

When RELD is "1", the timer operates in reload mode. In this mode, the timer loads the reload register contents into the counter and continues counting whenever an underflow occurs (when the counter value changes from $0000_H$ to $FFFF_H$).

When RELD is "0", the timer operates in one-shot mode. In this mode, the count operation stops when an underflow occurs due to the counter value changing from $0000_H$ to $FFFF_H$.

Table 15-4. OUTE, OUTL, and RELD settings

| OUTE | OUTL | RELD | Output Waveform |
|---|---|---|---|
| 0 | x | X | Timer output disabled |
| 1 | 0 | 0 | Output an "H" level pulse during counting. |
| 1 | 1 | 0 | Output an "L" level pulse during counting. |
| 1 | 0 | 1 | Toggle output. Starts with "L" level output. Changes level on timer reload. |
| 1 | 1 | 1 | Toggle output. Starts with "H" level output. Changes level on timer reload. |

### [Bit 3] INTE (Interrupt enable)

Timer interrupt request enable bit.
When INTE is "1", an interrupt request is generated when the UF bit changes to "1".
When INTE is "0", no interrupt request is generated, even when the UF bit changes to "1".

### [Bit 2] UF (Underflow)

Timer interrupt request flag. UF is set to "1" when an underflow occurs (when the counter value changes from $0000_H$ to $FFFF_H$). Cleared by writing "0" or by DMA Controller. Writing "1" to this bit has no effect. Read as "1" by read-modify-write instructions.

### [Bit 1] CNTE (Count enable)

Timer count enable bit. Writing "1" to CNTE sets the timer to wait for a trigger. Writing "0" stops count operation.

### [Bit 0] TRG (Trigger)

Software trigger bit. Writing "1" to TRG applies a software trigger, causing the timer to load the reload register content to the counter and starts counting.
Writing "0" has no effect. Reading always returns "0". Applying a trigger using this register is only valid when CNTE = "1".
Writing "1" has no effect if CNTE = "0".

Set this bit in 'Gate Input mode' to load the reload register content before counting starts.

## 15.2.2 Register Layout of 16-bit Timer Register (TMRn)/16-bit Reload Register (TMRLRn)

- TMRn contents

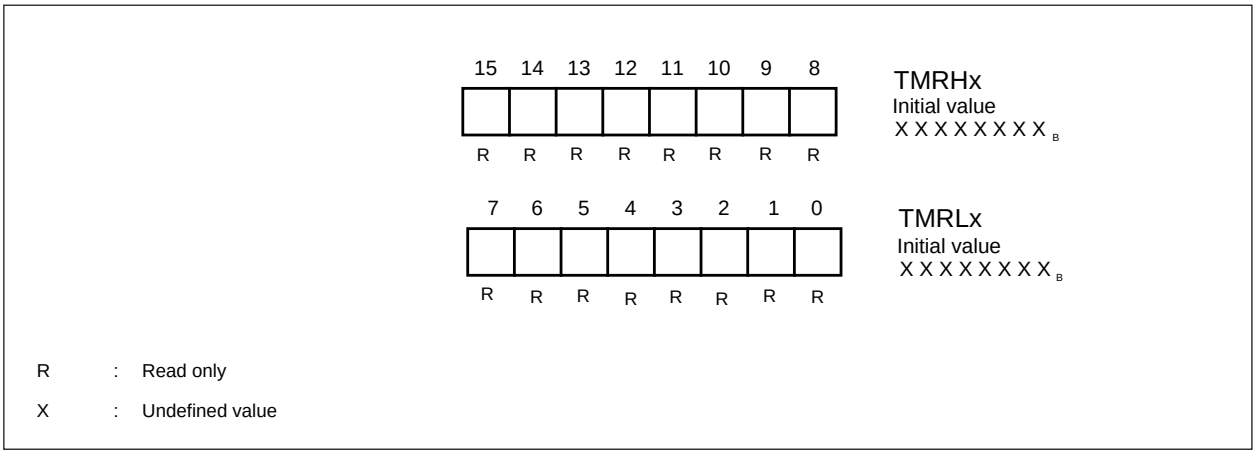
Reading this register returns the count value of the 16-bit Reload Timer. The initial value is undefined.

- TMRLRn contents

The 16-bit reload register holds the reload value. The initial value is undefined.

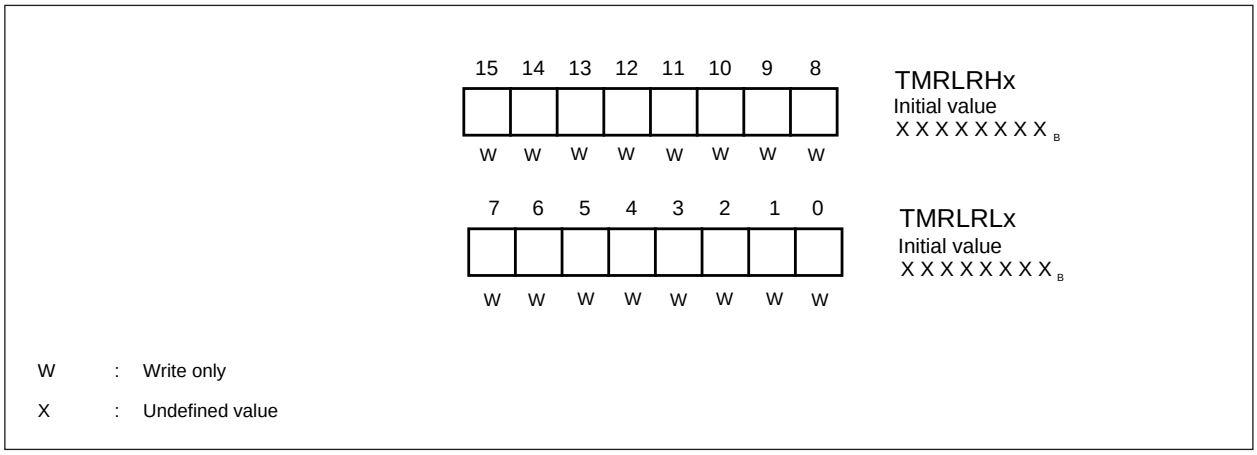
### Register layout of 16-bit timer register (TMRn)

Figure 15-4. Register layout of 16-bit timer register (TMRn)



### Register layout of 16-bit reload register (TMRLRn)

Figure 15-5. Register layout of 16-bit reload register (TMRLRn)




## 15.3 Internal Clock and External Event Counter Operations of 16-bit Reload Timer

In internal clock mode, the peripheral clock CLKP1 with different divider settings can be selected as the clock source for operating the Reload Timer. The external input pin TINn can be selected as either a trigger input or gate input by a register setting.

In event counter mode, the TINn pin is used as an external event input pin. Each active edge on this pin (rising, falling or both) decrements the counter.

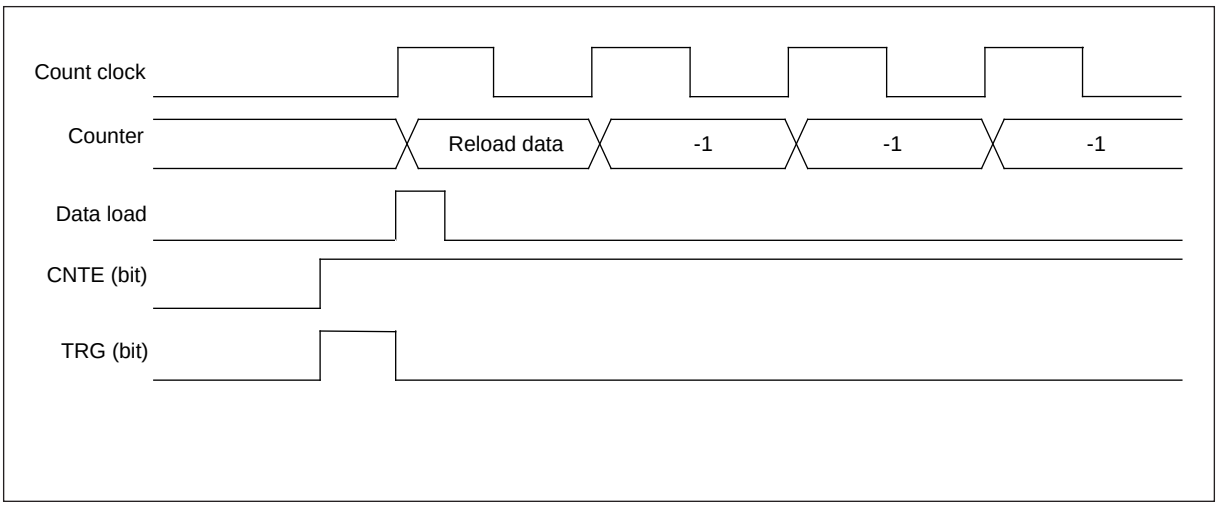
When FSEL = 0, then each second event is counted.

### Internal clock operation of 16-bit Reload Timer

Writing "1" to both the CNTE and TRG bits in the control status register enables and starts counting at the same time. Using the TRG bit as a trigger input is always available when the timer is enabled (CNTE = "1"), regardless of the operation mode.

Figure 15-6 shows counter activation and counter operation.

Figure 15-6. Activation and operation of 16-bit Reload Timer counter

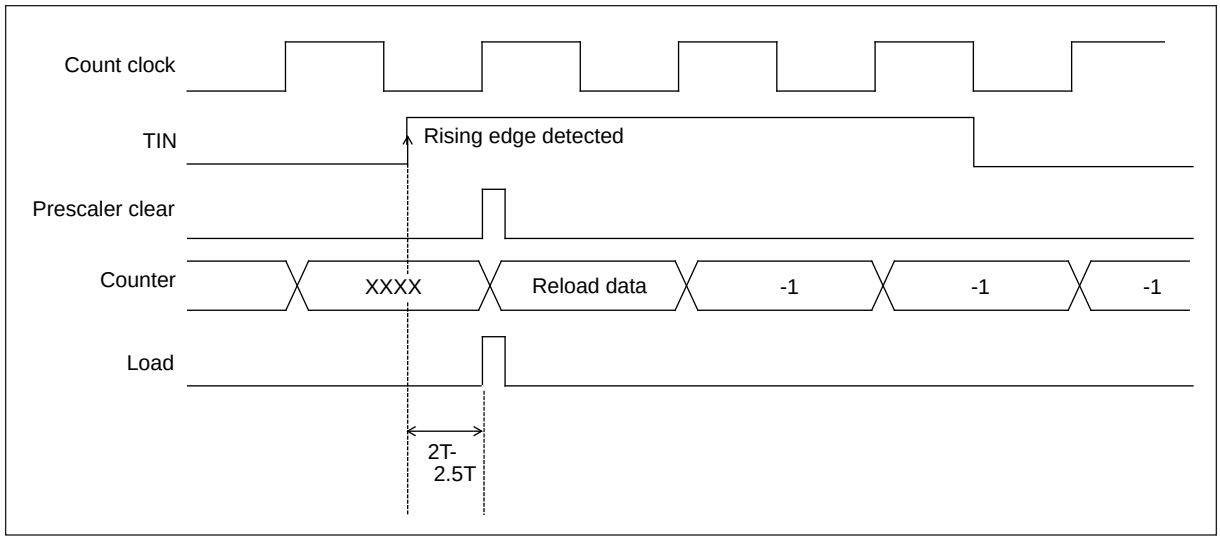


### Input pin functions of 16-bit Reload Timer (in internal clock mode)

The TINn pin can be used as either a trigger input or a gate input when an internal clock is selected as the clock source. When used as a trigger input, an active edge causes the timer to load the reload register contents and resets the internal prescaler. Then, count operation starts. Minimum required pulse width of TIN is 2T + 200ns (T: cycle of peripheral clock CLKP1, 200ns is the minimum pulse length non filtered by the input noise filter).

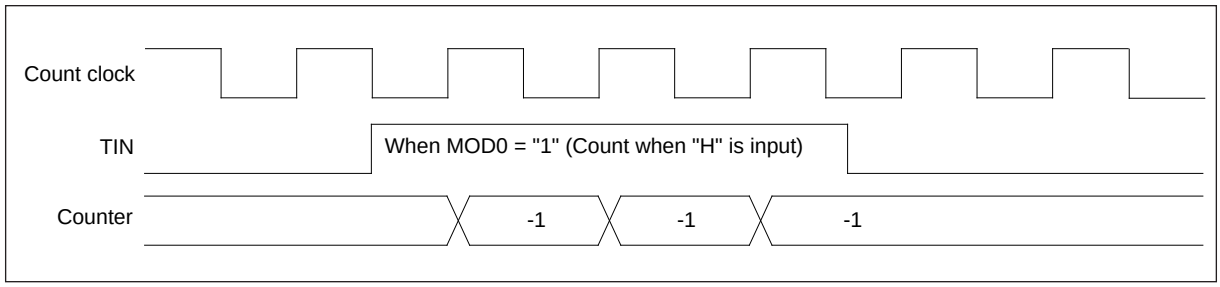
Figure 15-7 shows the operation of trigger input.

Figure 15-7. Trigger input operation of 16-bit Reload Timer




When used as a gate input, the counter only counts while the active level specified by the MOD0 bit of the control register is input to the TINn pin. In this case, the count clock continues to operate unless stopped. The software trigger can be used in gate mode, regardless of the gate level. Input a pulse width of at least 2T + 200ns to the TINn pin. Figure 15-8 shows the operation of gate input.

Figure 15-8. Gate input operation of 16-bit Reload Timer




### External event counter

When external event count mode is selected, the TIN pin is used as an external event input. The counter counts on the active edge specified in the TMCSRn. Input a pulse width of at least 4T + 200ns (T: cycle of peripheral clock CLKP1) to the TINn pin.

## 15.4 Underflow Operation of 16-bit Reload Timer

An underflow is defined for this timer as the time when the counter value changes from $0000_H$ to $FFFF_H$. Therefore, an underflow occurs after (reload register setting + 1) counts.

### Underflow operation of 16-bit Reload Timer

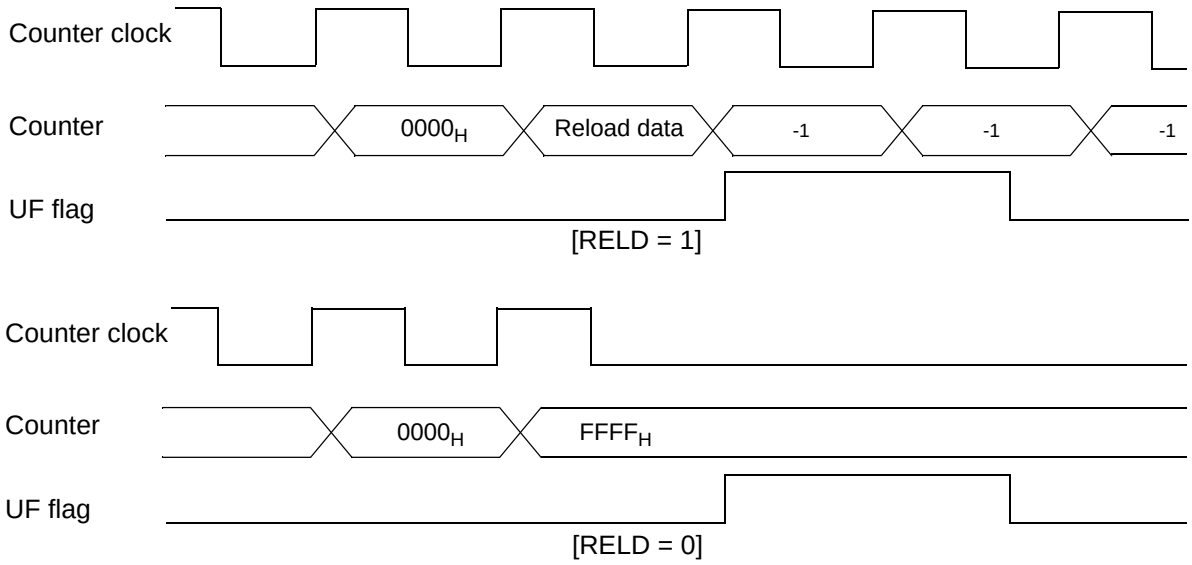
If the RELD bit in the control status register is "1" and an underflow occurs, the contents of the reload register is loaded into the counter and counting continues.

If the RELD bit in the control status register is "0", counting stops when counter reaches $FFFF_H$.

The UF bit in the control status register is set when the underflow occurs. If the INTE bit is "1" at this time, an interrupt request is generated.

Figure 15-9 shows the operation when an underflow occurs.

Figure 15-9. Underflow operation of 16-bit reload timer




## 15.5 Output Pin Functions of 16-bit Reload Timer

In reload mode, the TOTn pin performs toggle output (inverts at each underflow). In one-shot mode, the TOTn pin is used as a pulse output that shows the configured level while the counting is in progress.

### Output pin functions of 16-bit Reload Timer

The OUTL bit of the control status register sets the output polarity.

When OUTL = "0", the initial value for toggle output is "L" and the one-shot pulse output is "H" while the count is in progress.

When OUTL = "1", the output waveforms are opposite.

Figure 15-10 and Figure 15-11 show the output pin functions.

Figure 15-10. Output pin function of 16-bit Reload Timer in reload mode

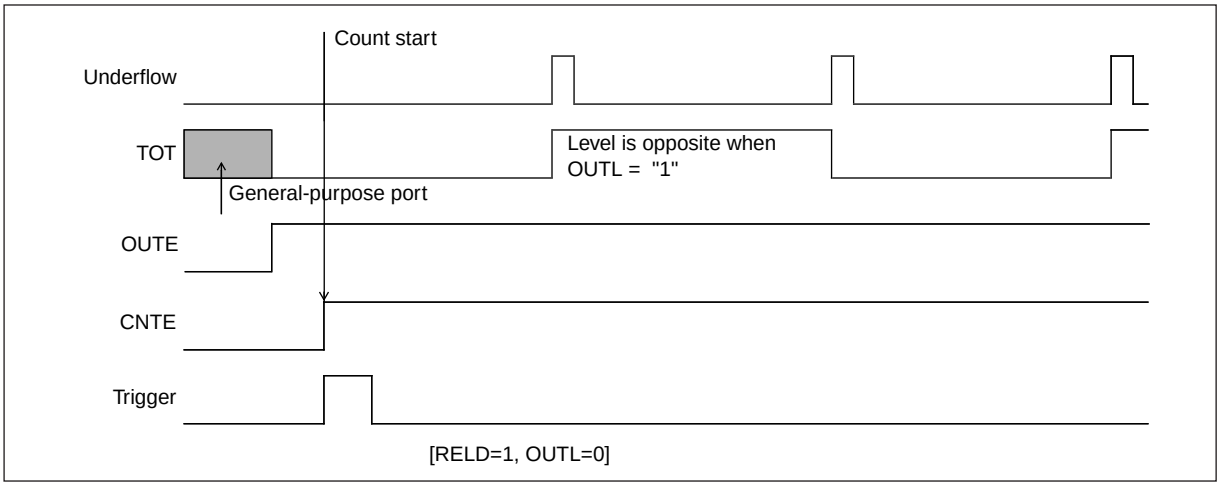
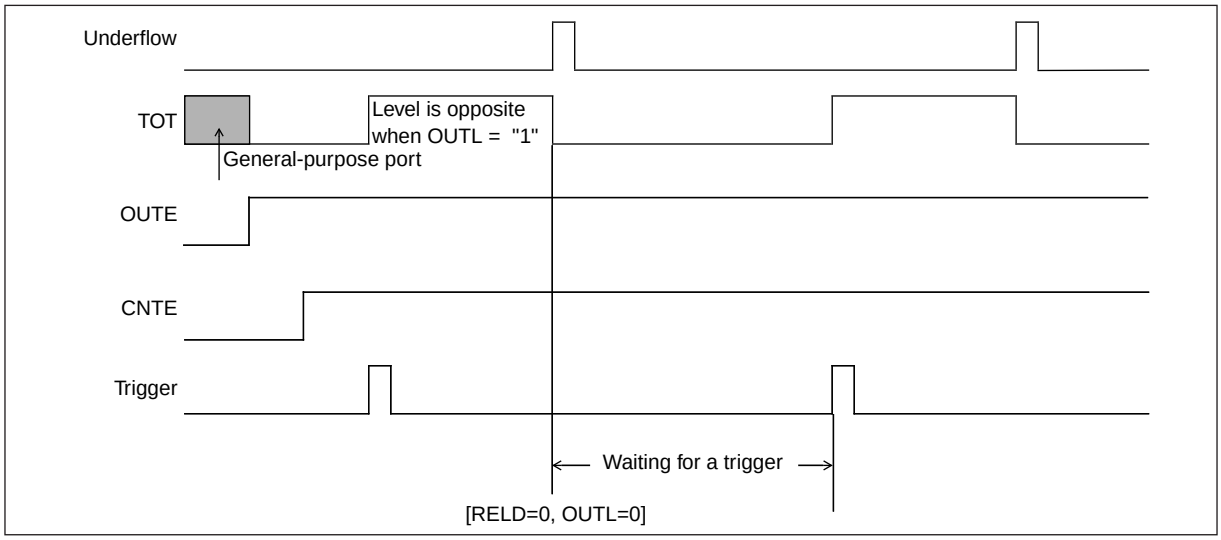



Figure 15-11. Output pin function of 16-bit Reload Timer in one-shot mode



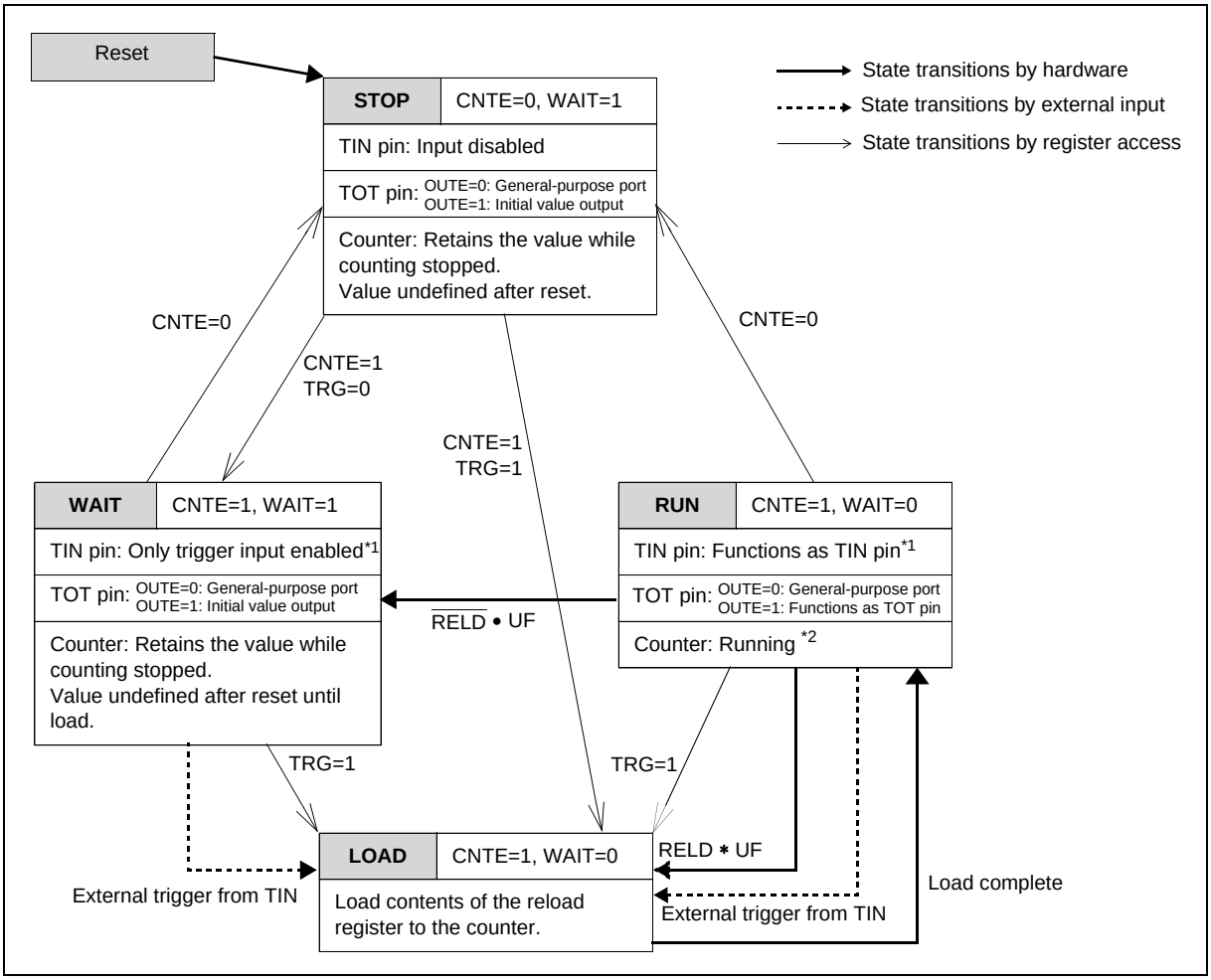
## 15.6 Counter Operation State

The counter state is determined by the CNTE bit in the control status register and the internal WAIT signal. Available states are: CNTE = "0" and WAIT = "1" (STOP state), CNTE = "1" and WAIT = "1" (WAIT state for trigger), and CNTE = "1" and WAIT = "0" (RUN state).

### Counter operation states

Figure 15-12 shows the transitions between each state.

Figure 15-12. Counter state transitions



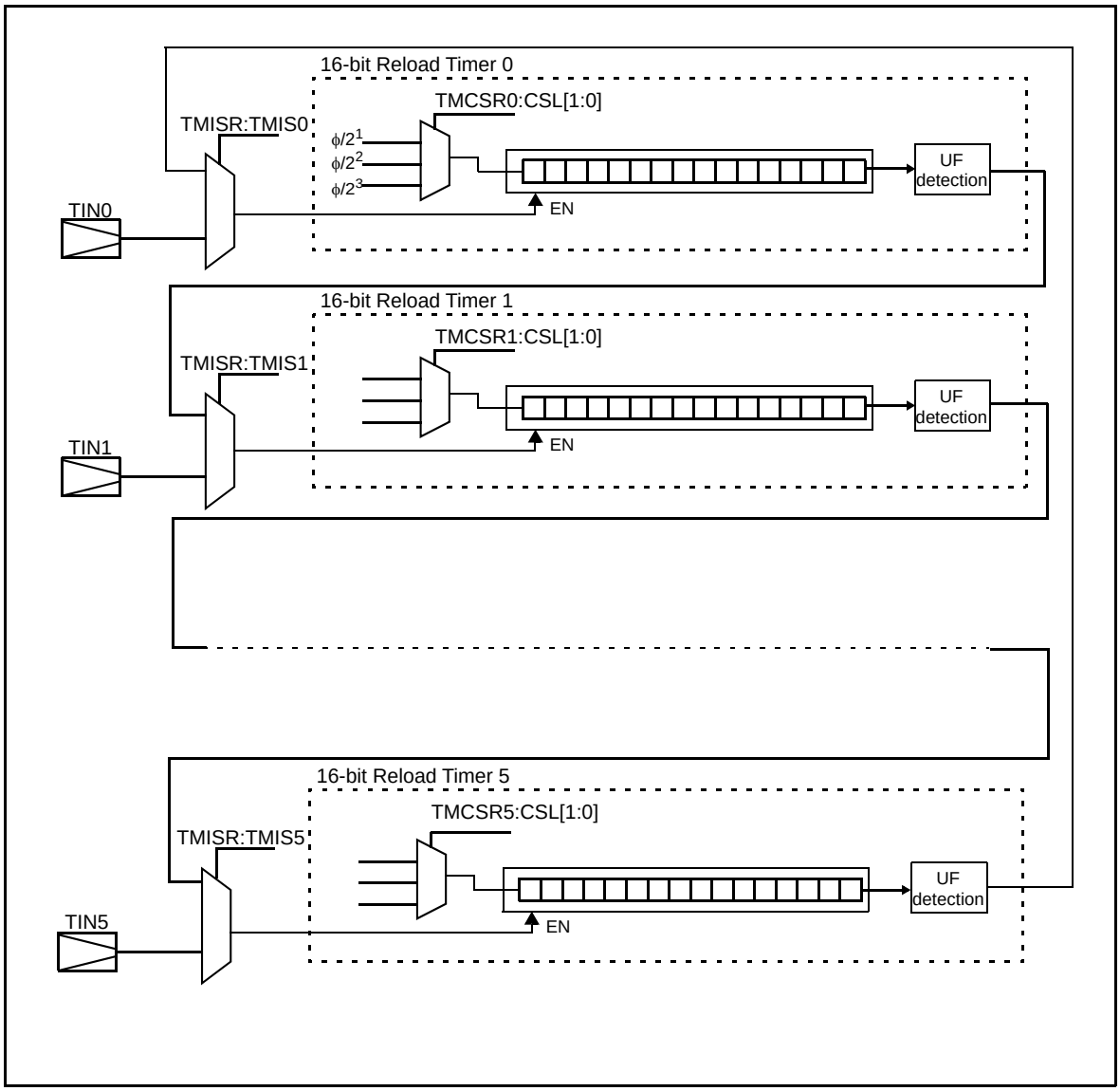

*1: Before using TIN pin, the corresponding bits of the DDR must be set to '0' and PIER register to '1'.
*2: In 'Gate Input mode': Counting can be influenced by TIN.

## 15.7 Cascading of 16-bit Reload Timers

The cascading of multiple adjacent 16-bit Reload Timers allows the user to create its own n * 16-bit-Reload Timer (Example: 3 adjacent Reload Timers available on device).

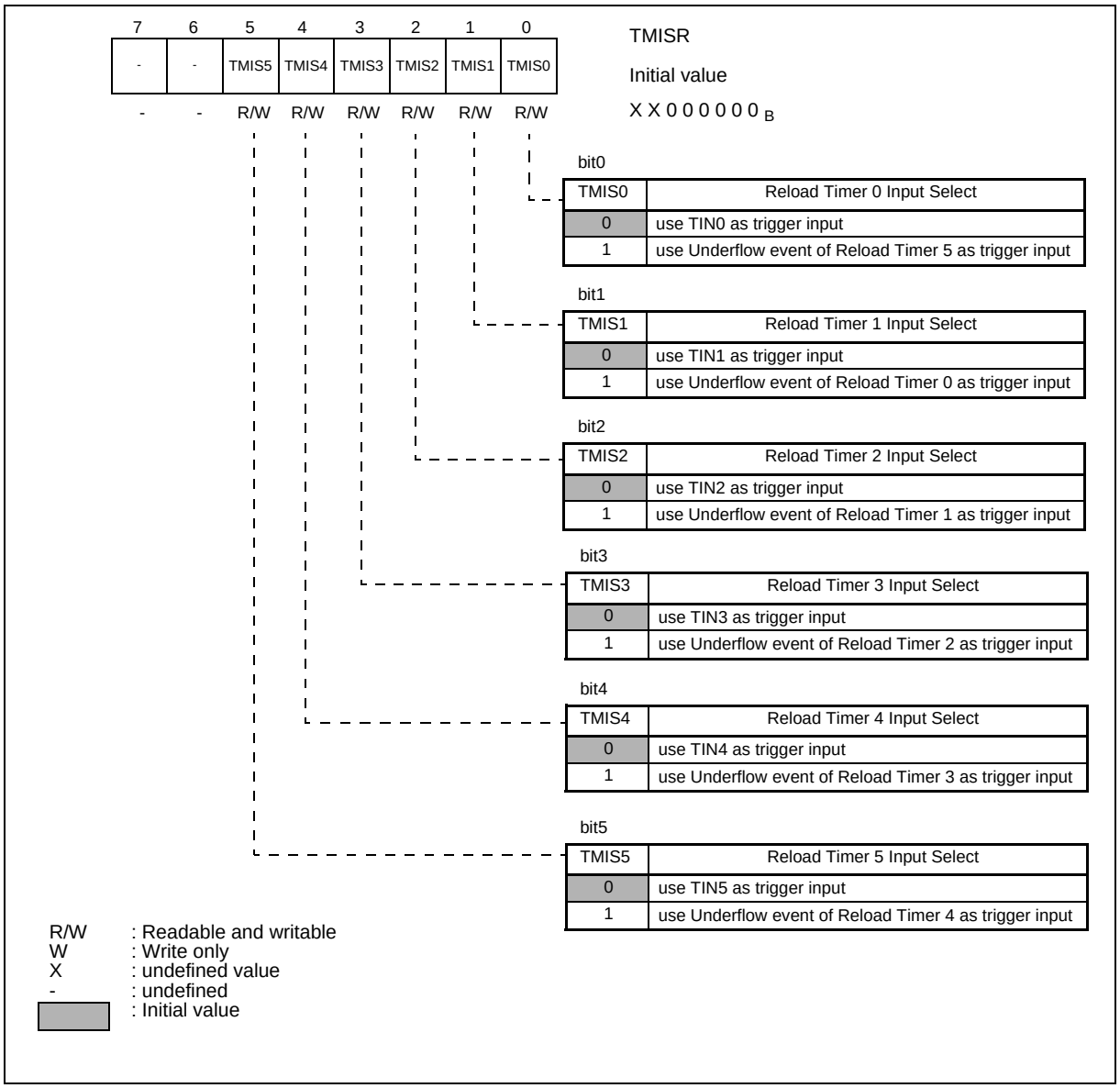
### Cascading overview

Figure 15-13. Block diagram for cascading of 16-bit Reload Timers (Example for 3 Reload Timers available on device)



### Reload Timer Input Select Register (TMISR)

Figure 15-14. Reload Timer Input Select Register (TMISR)




### Description of operation

The first Reload Timer in the chain can either have an internal clock or an external event as counter clock. It can be configured freely.

The next Reload Timer and all following in the chain must be set as follows:

- TMCSRn:MOD[2:0] = "101" : Gate count mode "H" level
- TMCSRn:CSL[1:0] = "00" and TMCSRn:FSEL = "1" : CLKP1 is divided by 2 (cf table 16.2-1) but all RLT, except the first RLT, are clocked with CLKP1 divided by 2 to make sure that the UF signal, which is CLKP1 period long, is sampled.
- TMISR:TMISx = "1" : Use underflow signal of preceding timer as count enable signal

All Reload Timers must be triggered before counting starts to load the reload value and to change the Reload timers to RUN-State (see Figure 15-12).

# 16. Programmable Pulse Generator

This chapter explains the functions and operations of the Programmable Pulse Generator.

16.1 Outline of Programmable Pulse Generator

16.2 Registers

16.3 Operation of Programmable Pulse Generator

16.4 Cautions

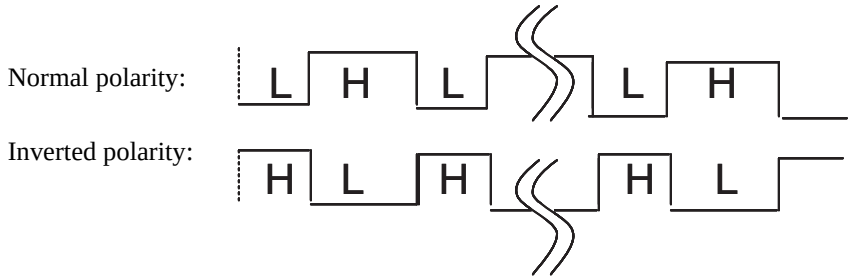
## 16.1 Outline of Programmable Pulse Generator

Programmable Pulse Generators (PPGs) are used to obtain one-shot (rectangular wave) output or pulse width modulation (PWM) output. With their software-programmable cycle and duty capability, the PPGs comfortably fit into a broad range of applications.

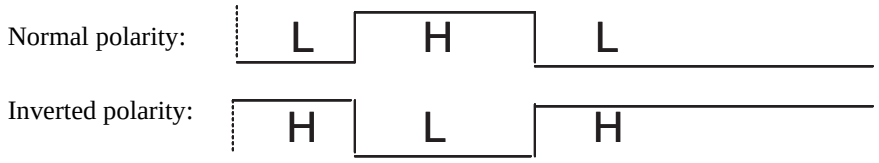
### Features of the Programmable Pulse Generator

Output waveforms: The PPGs can generate the following kinds of waveforms

PWM waveform




One-shot waveform (Rectangular wave)




Clamped output

- Normal polarity: “L” Clamped output
- Inverted polarity: “H” Clamped output

Count clock: Choose from 8 choices:

- 1, 1/4, 1/16, 1/64 of the peripheral clock (CLKP1) or of Reload Timer 6 underflow (refer to 16-Bit Reload Timer (With Event Count Function) on page 325)

Period: Setting range = Duty value ~ 65535 (specified with a 16-bit register)

- Period = Count clock x (PCSR register value + 1)

- (Example) Count clock = 32MHz(31.25ns), PCSR value = 63999
- Period = 31.25ns x (63999+1) = 2ms

Duty: Setting range = 0 ~ Period value (specified with a 16-bit register)

- Duty = Count clock (PDUT register value + 1)

Interrupt: Choose from four choices

- Software trigger or external trigger (TTGx pin)
- Counter borrow (cycle match)
- Duty match
- Counter borrow (cycle match) or duty match

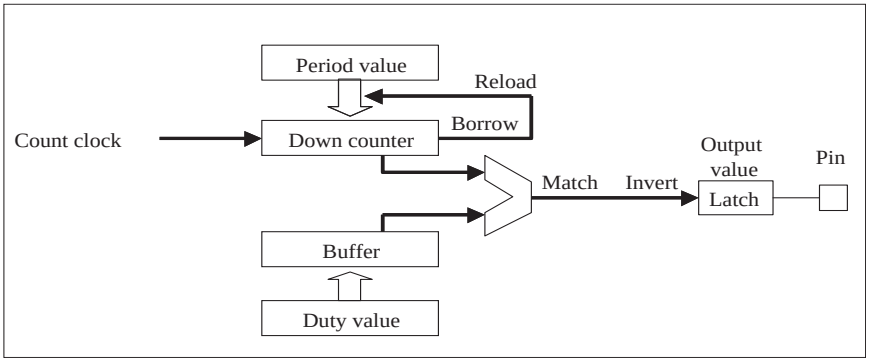
Activation trigger:

- Software trigger
- Internal trigger
- External trigger (TTG pins)

Freely configurable Reload Timer as additional prescalar input

### Simplified block diagram of Programmable Pulse Generator

Figure 16-1. Simplified block diagram of Programmable Pulse Generator

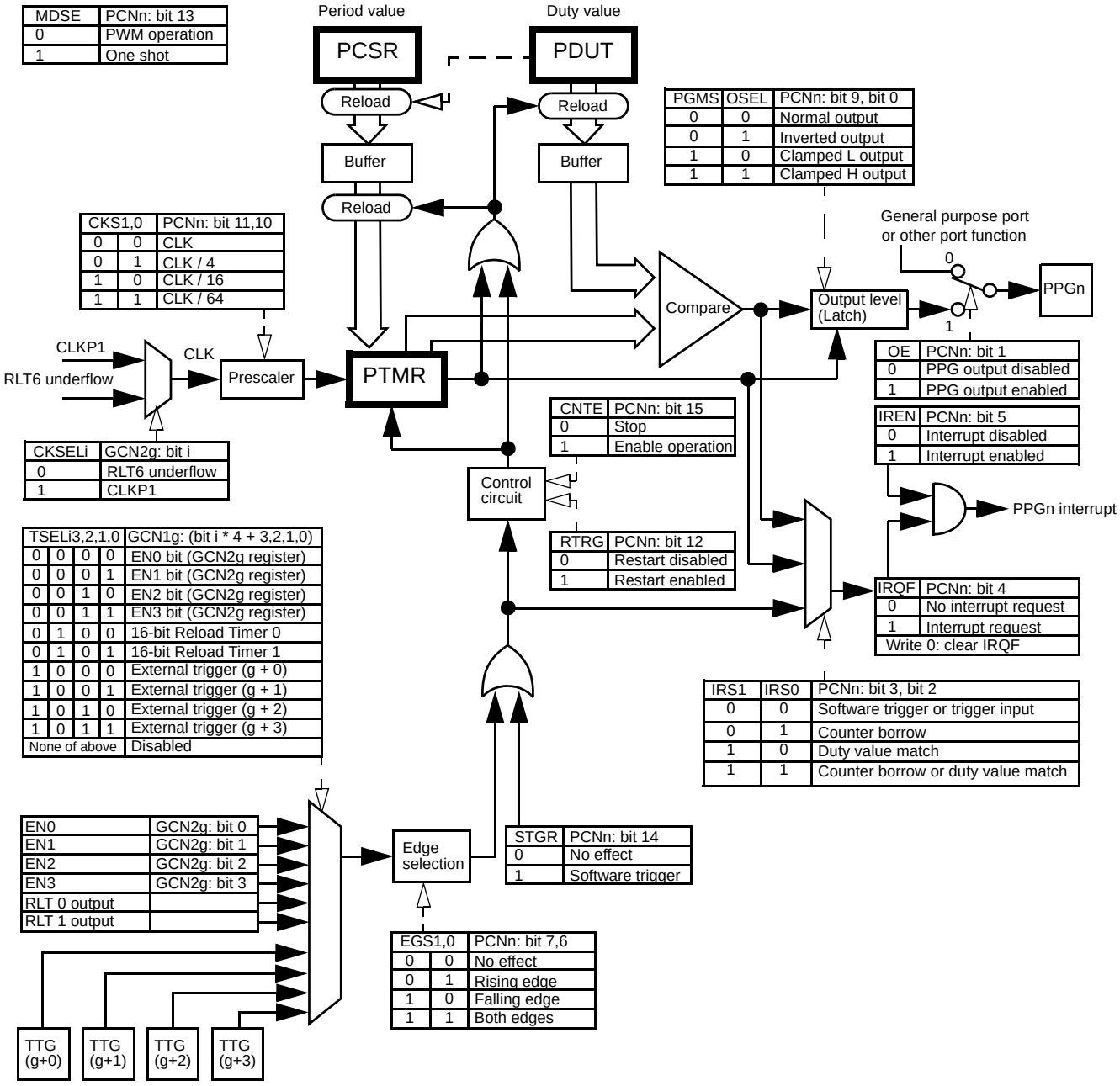



### Configuration of Programmable Pulse Generator

Four Programmable Pulse Generators form a group with common GCN registers. In Figure 16-2 the following notation is used:

- n = Number of programmable pulse generator
- g = INT(n / 4) group number of programmable pulse generator
- i = (n MODULUS 4) index of programmable pulse generator within the group.

Figure 16-2. Configuration diagram of Programmable Pulse Generator




| MDSE | PCNn: bit 13 |
|---|---|
| 0 | PWM operation |
| 1 | One shot |

| PGMS | OSEL | PCNn: bit 9, bit 0 |
|---|---|---|
| 0 | 0 | Normal output |
| 0 | 1 | Inverted output |
| 1 | 0 | Clamped L output |
| 1 | 1 | Clamped H output |

| CKS1,0 | | PCNn: bit 11,10 |
|---|---|---|
| 0 | 0 | CLK |
| 0 | 1 | CLK / 4 |
| 1 | 0 | CLK / 16 |
| 1 | 1 | CLK / 64 |

| OE | PCNn: bit 1 |
|---|---|
| 0 | PPG output disabled |
| 1 | PPG output enabled |

| IREN | PCNn: bit 5 |
|---|---|
| 0 | Interrupt disabled |
| 1 | Interrupt enabled |

| CNTE | PCNn: bit 15 |
|---|---|
| 0 | Stop |
| 1 | Enable operation |

| CKSELi | GCN2g: bit i |
|---|---|
| 0 | RLT6 underflow |
| 1 | CLKP1 |

| RTRG | PCNn: bit 12 |
|---|---|
| 0 | Restart disabled |
| 1 | Restart enabled |

| IRQF | PCNn: bit 4 |
|---|---|
| 0 | No interrupt request |
| 1 | Interrupt request |
| Write 0: clear IRQF | |

| IRS1 | IRS0 | PCNn: bit 3, bit 2 |
|---|---|---|
| 0 | 0 | Software trigger or trigger input |
| 0 | 1 | Counter borrow |
| 1 | 0 | Duty value match |
| 1 | 1 | Counter borrow or duty value match |

| TSELi3,2,1,0 | | | | GCN1g: (bit i * 4 + 3,2,1,0) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | EN0 bit (GCN2g register) |
| 0 | 0 | 0 | 1 | EN1 bit (GCN2g register) |
| 0 | 0 | 1 | 0 | EN2 bit (GCN2g register) |
| 0 | 0 | 1 | 1 | EN3 bit (GCN2g register) |
| 0 | 1 | 0 | 0 | 16-bit Reload Timer 0 |
| 0 | 1 | 0 | 1 | 16-bit Reload Timer 1 |
| 1 | 0 | 0 | 0 | External trigger (g + 0) |
| 1 | 0 | 0 | 1 | External trigger (g + 1) |
| 1 | 0 | 1 | 0 | External trigger (g + 2) |
| 1 | 0 | 1 | 1 | External trigger (g + 3) |
| None of above | | | | Disabled |

| EN0 | GCN2g: bit 0 |
|---|---|
| EN1 | GCN2g: bit 1 |
| EN2 | GCN2g: bit 2 |
| EN3 | GCN2g: bit 3 |
| RLT 0 output | |
| RLT 1 output | |

| STGR | PCNn: bit 14 |
|---|---|
| 0 | No effect |
| 1 | Software trigger |

| EGS1,0 | | PCNn: bit 7,6 |
|---|---|---|
| 0 | 0 | No effect |
| 0 | 1 | Rising edge |
| 1 | 0 | Falling edge |
| 1 | 1 | Both edges |

**Note:**

This diagram is also valid for the other available PPGs. For configuring RLT6 see 16-Bit Reload Timer (With Event Count Function) on page 325.

RLT6 can be used as a normal RLT if it is not required for the PPG operation.

## 16.2 Registers

The programmable pulse generator has the following registers:

- PPG Control Status register (PCNn)
- General Control register 1 (GCN1g), one per group of 4 PPGs
- General Control register 2 (GCN2g), one per group of 4 PPGs
- PPG Cycle Setting Register (PCSRn)
- PPG Duty Setting Register (PDUTn)
- PPG Timer Register (PTMRn)

### 16.2.1 Register list

Figure 16-3. Register list

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PCNn | CNTE | STGR | MDSE | RTRG | CKS1 | CKS0 | PGMS | - | EGS1 | EGS0 | IREN | IRQF | IRS1 | IRS0 | OE | OSEL |
| GCN1g | TSEL33 | TSEL32 | TSEL31 | TSEL30 | TSEL23 | TSEL22 | TSEL21 | TSEL20 | TSEL13 | TSEL12 | TSEL11 | TSEL10 | TSEL03 | TSEL02 | TSEL01 | TSEL00 |
| GCN2g | - | - | - | - | CKSEL3 | CKSEL2 | CKSEL1 | CKSEL0 | - | - | - | - | EN3 | EN2 | EN1 | EN0 |
| PCSRn<br>Access: Write only | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PDUTn<br>Access: Write only | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PTMRn<br>Access: Read only | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

Note: n denotes an individual PPG, g denotes the corresponding groupof 4 PPGs.

## 16.2.2 PPG Control Status register (PCNn)

The PPG Control Status register (PCNn) controls the operations and status of the PPG.

Figure 16-4. PPG Control Status register (PCNn)

PCNHn

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | CNTE | STGR | MDSE | RTRG | CKS1 | CKS0 | PGMS | - |
| Access: | R/W | R0/W | R/W | R/W | R/W | R/W | R/W | Rx/Wx |
| Initial value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x |
| Rewrite during operation: | O | O | X | X | X | X | X | - |

PCNLn

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | EGS1 | EGS0 | IREN | IRQF | IRS1 | IRS0 | OE | OSEL |
| Access: | R/W | R/W | R/W | R(RM1)/W | R/W | R/W | R/W | R/W |
| Initial value: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Rewrite during operation: | X | X | O | O | X | X | O | X |

R/W : Readable and writable
x : Undefined
O : Rewritable
X : Not writable

### Bit 15: Timer enable operation

| CNTE | Operation |
|---|---|
| 0 | Stop |
| 1 | Operation |

This bit enables the operation of the PPG.

### Bit 14: Software trigger

| STGR | Operation |
|---|---|
| 0 | The operation is unaffected by writing (The read value is always “0”). |
| 1 | Software trigger activation |

When the Software Trigger bit is set to “1”, a software trigger is generated to activate the PPG, separately from the generation of an internal or external trigger (EN bit, reload timer output, TTG input). This trigger is independent from setting of the edge selection bits EGS1 and EGS0.

### Bit 13: Mode selection

| MDSE | Mode |
|---|---|
| 0 | PWM operation |
| 1 | One-shot operation |

When the Mode Selection bit is set to “0”, a PWM operation is enabled to generate pulses in sequence.

When the Mode Selection bit is set to “1”, pulse output takes place only once.

### Bit 12: Restart enable

| RTRG | Operation |
|---|---|
| 0 | Disable restart. |
| 1 | Enable restart. |

When the Enable Restart bit is set to “1”, a trigger (software/internal/external) will restart the PPG operation (depending on the configuration of the triggers).

### Bits 11-10: Counter clock selection

| CKS1 | CKS0 | Down Counter Count Clock Selection |
|---|---|---|
| 0 | 0 | Clock selected by CKSEL |
| 0 | 1 | Clock selected by CKSEL divided by 4 |
| 1 | 0 | Clock selected by CKSEL divided by 16 |
| 1 | 1 | Clock selected by CKSEL divided by 64 |

### Bit 9: PPG output mask selection

| PGMS | Operation |
|---|---|
| 0 | No output mask |
| 1 | Output mask (Output “L” level latched:OSEL=“0”) |

When the PPG Output Mask Selection bit is set to “1”, the PPG output can be clamped at “L” or “H” regardless of the mode, cycle, and duty settings.

The output level can be specified using the Output Polarity Specification bit (PCN:OSEL).

### Bit 8: Undefined

Write always 0. The read value is undefined. Read-modify write is not affected.

### Bits 7-6: Trigger input edge selection

| EGS1 | EGS0 | Selected Edge |
|---|---|---|
| 0 | 0 | no edge selected, triggering of PPG only possible by PCNHn:STRG |
| 0 | 1 | Rising edge |
| 1 | 0 | Falling edge |
| 1 | 1 | Both edges (rising edge, or, falling edge) |

When EGS1=0 and EGS0=0, the PPG can only be triggered by PCNn:STRG. The triggers EN[3:0], Reload timer and external triggers are disabled for PPGn.

The other settings of EGS1 and EGS0 effect only the triggers EN[3:0], Reload timer and external triggers. Writing ’1’ to PCNn:STRG triggers the corresponding PPG independent of the setting of PCNn:EGS1 and PCNn:EGS0.

### Bit 5: Interrupt request enable.

| IREN | Operation |
|---|---|
| 0 | Disable interrupt requests. |
| 1 | Enable interrupt requests. |

### Bit 4: interrupt request flag

| IRQF | Read Operation | Write Operation |
|---|---|---|
| 0 | No interrupt request | Clear the Interrupt Request flag. |
| 1 | Interrupt request | Writing "1" has no effect. |

If the Interrupt Request flag (IRQF) equals “1” and writing “0” to the flag take place at the same time, the setting of the Interrupt Request flag (IRQF=“1”) by hardware has higher priority.

### Bit 3-2: Interrupt cause selection

| IRS1 | IRS0 | Selection |
|---|---|---|
| 0 | 0 | Software trigger or external trigger input |
| 0 | 1 | Counter borrow |
| 1 | 0 | The counter matches the duty value. |
| 1 | 1 | Counter borrow or the counter equals the duty value. |

Select the operation in which to generate an interrupt request.

### Bit 1: PPG output enable

| OE | Operation |
|---|---|
| 0 | Output disabled |
| 1 | Output enabled |

### Bit 0: PPG output polarity specification

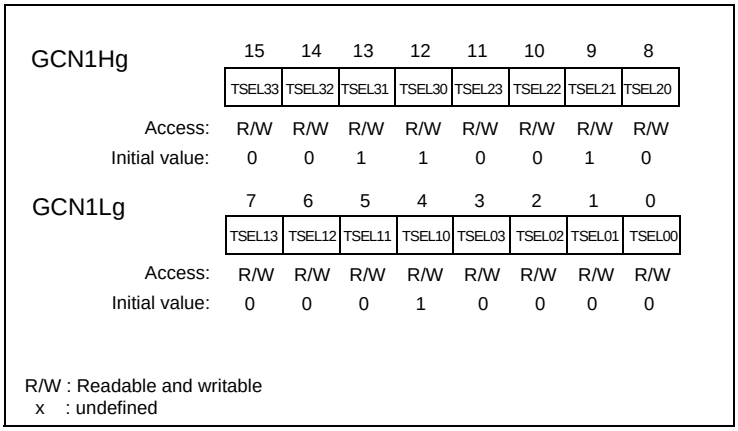
| OSEL | Operation |
|---|---|
| 0 | Normal polarity |
| 1 | Inverted polarity |

When the PPG Output Mask Selection bit (PCN:PGMS) has been set to “1”, if the Output Polarity Specification bit (OSEL) is set to “0”, the output is clamped at “L”; if the Output Polarity Specification bit is set to “1”, the output is clamped at “H”.

## 16.2.3 General Control Register 1 (GCN1g)

The General Control Register 1 (GCN1g) selects a trigger input to a PPG group of 4 PPGs.

Figure 16-5. General Control Register 1 (GCN1g)



In the following table, the index "i" denotes the PPG channel number within the group of PPGs (i = 0, 1, 2, or 3).

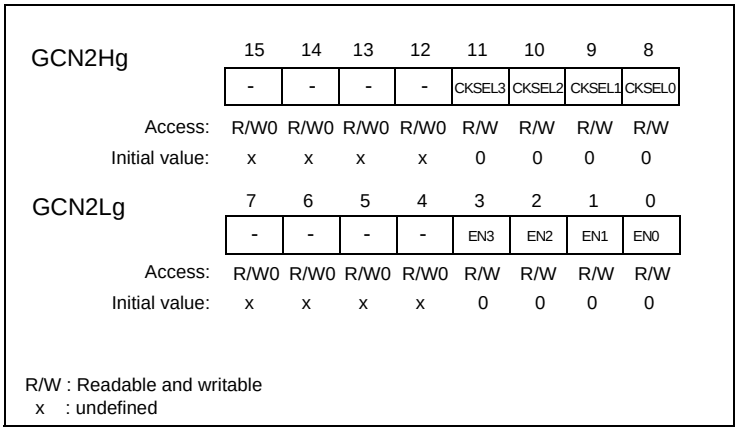
| TSELi3 | TSELi2 | TSELi1 | TSELi0 | Activation trigger specification |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | EN0 bit (GCN2 register) |
| 0 | 0 | 0 | 1 | EN1 bit (GCN2 register) |
| 0 | 0 | 1 | 0 | EN2 bit (GCN2 register) |
| 0 | 0 | 1 | 1 | EN3 bit (GCN2 register) |
| 0 | 1 | 0 | 0 | 16-bit reload timer 0 output |
| 0 | 1 | 0 | 1 | 16-bit reload timer 1 output |
| 1 | 0 | 0 | 0 | External trigger (g + 0) |
| 1 | 0 | 0 | 1 | External trigger (g + 1) |
| 1 | 0 | 1 | 0 | External trigger (g + 2) |
| 1 | 0 | 1 | 1 | External trigger (g + 3) |
| None of the above | | | | Disabled |

PPG0 to PPGn as selected are activated when the edge specified by the Trigger Input Edge Selection bits (PCNn:EGS[1:0]) are detected by the specified activation trigger.

## 16.2.4 General Control Register 2 (GCN2g)

The General Control Register 2 (GCN2x) generates internal trigger levels using software for a PPG block of 4 PPGs.

Figure 16-6. General Control Register 2 (GCN2g)




### Bits 15-12: Undefined

Always write “0”. The read value is undefined. Read-modify-write is not affected.

### Bits 11-8: Prescalar input selection

| CKSEL0..3 | Operation |
|---|---|
| 0 | Use CLKP1 as prescaler input |
| 1 | Use RLT6 as prescaler input |

### Bits 7-4: Undefined

Always write “0”. The read value is undefined. Read-modify-write is not affected.

**Bit 3-0: EN trigger input**

| EN0..3 | Internal Triggers EN0, EN1, EN2, and EN3 |
|---|---|
| 0 | Set the level to “L”. |
| 1 | Set the level to “H”. |

Set the levels of internal triggers EN0, EN1, EN2 and EN3.

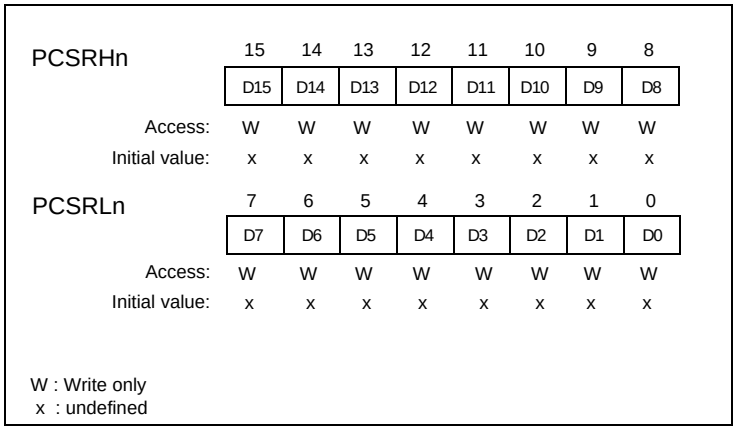
If any of the EN trigger inputs (EN0, EN1, EN2, EN3) is selected with the trigger specification bits (GCN1g:TSEL0[3:0], GCN1g:TSEL1[3:0], GCN1g:TSEL2[3:0], and GCN1g:TSEL3[3:0]) of PPGn, then the selected EN serves as a PPG trigger input bit.

If the state selected with the trigger input edge selection bit (EGS[1:0]) is generated by software using the trigger input bit (selected EN0, EN1, EN2, or EN3), the choice serves as an activation trigger to activate the PPG.

## 16.2.5 PPG Cycle Setting Register (PCSRn)

The PPG Cycle Setting Register (PCSRn) controls the cycle of the PPG.

Figure 16-7. PPG Cycle Setting Register (PCSRn)




The PPG Period Setting registers come with buffers. Transfers from the buffers to the counter take place automatically upon counter borrow.

After the PPG Period Setting registers have been written, be sure to set PPG Duty Setting registers PDUT.

## 16.2.6 PPG Duty Setting Register (PDUTn)

The PPG Duty Setting Register (PDUTn) sets the duty of the PPG output waveform.

Figure 16-8. PPG Duty Setting Register (PDUTn)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| PDUTHn | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| Access: | W | W | W | W | W | W | W | W |
| Initial value: | x | x | x | x | x | x | x | x |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PDUTLn | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Access: | W | W | W | W | W | W | W | W |
| Initial value: | x | x | x | x | x | x | x | x |

W : Write only
x : undefined

The PPG Duty Setting Registers are buffered. Transfers from the buffers to the counter take place automatically upon counter borrow.

Set a value smaller than the setting of PPG Period Setting register PCSR in a PPG Duty Setting register.

If the same value is set in a PPG Duty Setting register as is set in PPG Period Setting register PCSR, then

- “H” is always output to (OSEL=”0”) at normal polarity time.
- “L” is always output to (OSEL=”1”) at inverted polarity time.
- (The OSEL bit is an output polarity specification bit of the PPG control register PCN.)

## 16.2.7 PPG Timer Register (PTMRn)

The PPG Timer Register (PTMRn) reads the counts of PPGn.

Figure 16-9. PPG Timer Register (PTMRn)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| PTMRHn | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| Access: | R | R | R | R | R | R | R | R |
| Initial value: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PTMRLn | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Access: | R | R | R | R | R | R | R | R |
| Initial value: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

R : Read only, write has no effect
x : undefined

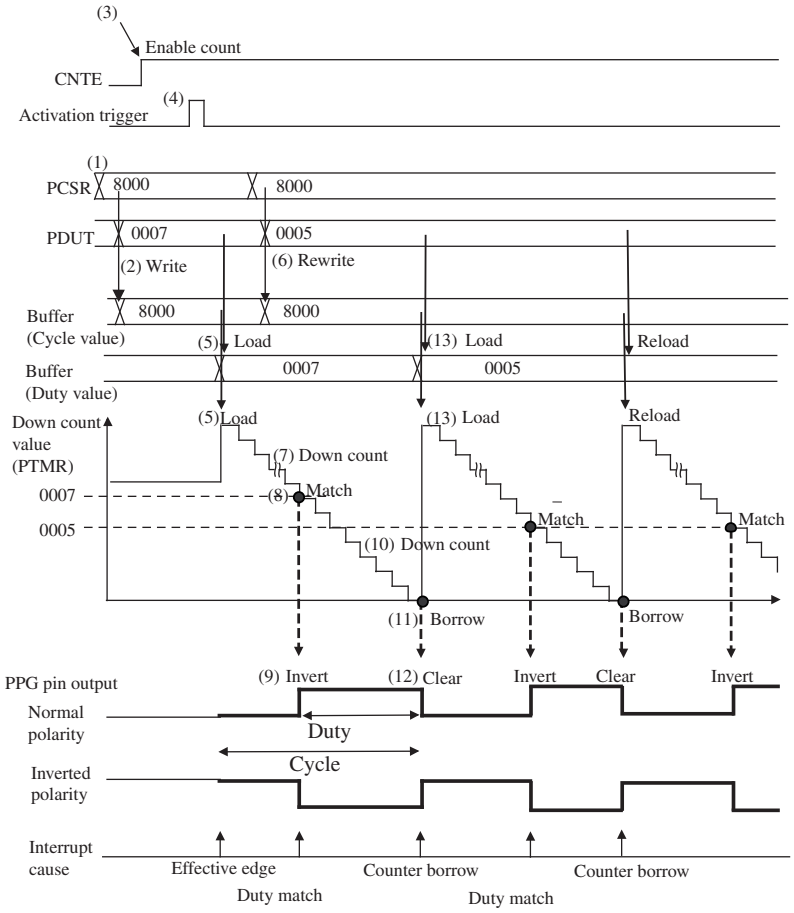
The count of the 16-bit down counter can be read.

## 16.3 Operation of Programmable Pulse Generator

The Programmable Pulse generators (PPGs) provide programmable pulse output independently or jointly. The individual modes of operation are described below

### 16.3.1 PWM Operation

In PWM operation, variable-duty pulses are generated from the PPG pin.

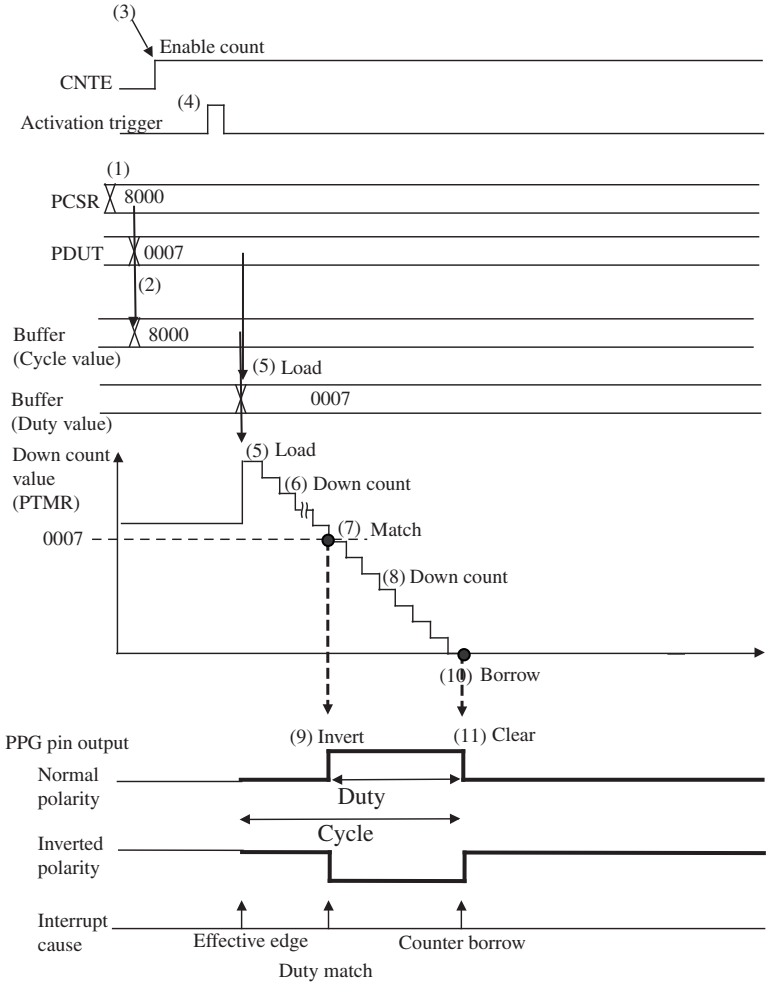



- (1) Write a cycle value.
- (2) Write a duty value and transfer the cycle value to buffers.
- (3) Enable PPG operation.
- (4) Generate an activation trigger.
- (5) Load the cycle and duty values.
- (6) Rewrite the duty value and transfer the cycle value to buffers.
- (7) Counter down count.
- (8) The down counter equals the duty value.
- (9) Inverses the PPG pin output level.
- (10) Counter down count.
- (11) Counter borrow.
- (12) Clear the PPG pin output level (return to normal).
- (13) Reload the cycle value.
- (14) Reload the duty value.

- (15) Steps from (7) to (14) are iterated.
- Equation:
  - Period = {Period value (PCSR) + 1} x Count clock
  - Duty = {Duty value (PDUT) + 1} x Count clock
  - Width up to pulse output = {Period value (PCSR) – Duty value (PDUT)} x Count clock

## 16.3.2 One-Shot Operation

In one-shot operation, one-shot pulses are generated from the PPG pin.



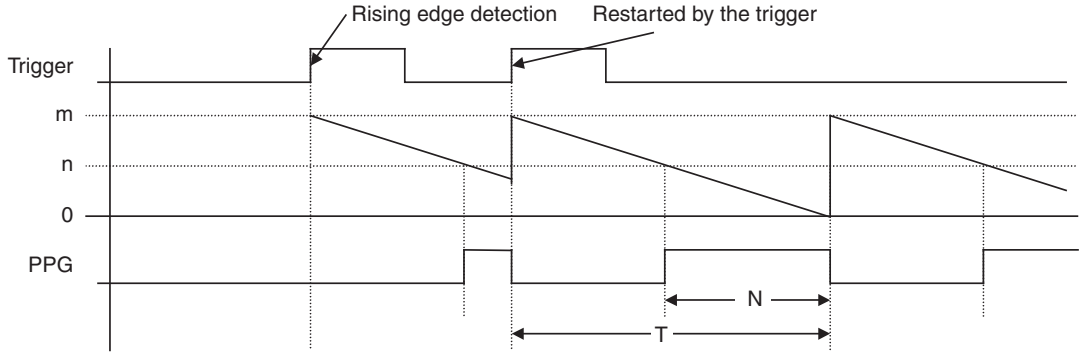

- (1) Write a cycle value.
- (2) Write a duty value and transfer the cycle value to buffers.
- (3) Enable PPG operation.
- (4) Generate an activation trigger.
- (5) Load the cycle and duty values.
- (6) Counter down count.
- (7) The down counter equals the duty value.
- (8) Inverses the PPG pin output level.
- (9) Counter down count.
- (10) Counter borrow.

- (11) Clear the PPG pin output level (return to normal).
- (12) The operating sequence is now completed

## 16.3.3 Restart Operation

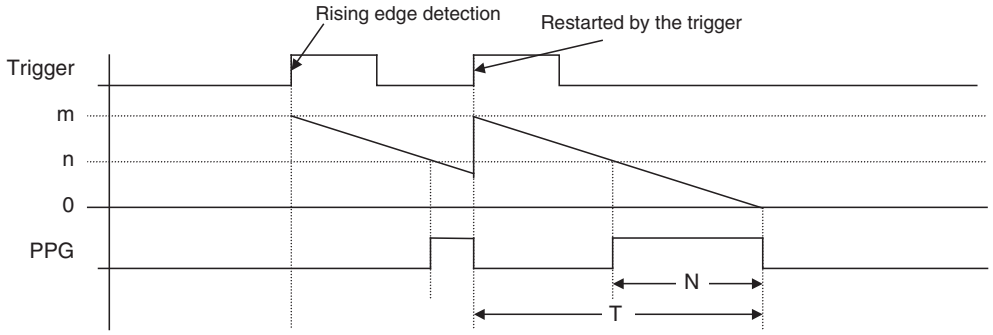
The restart operation is described below:

**Restart available in PWM operation:**




N = duty, T = cycle

**Restart available in one-shot operation:**



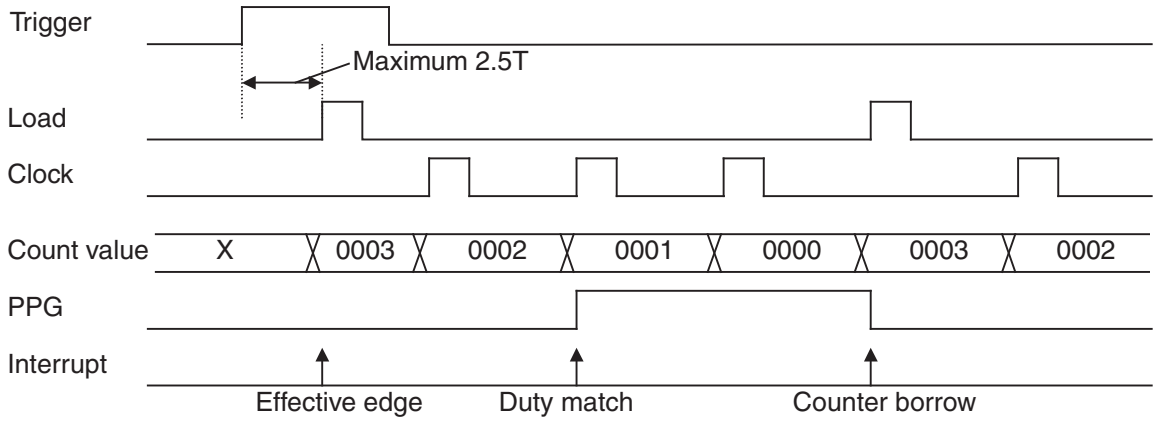

If a restart is not available, the second and subsequent triggers have no effect in both PWM and one-shot operations. (The second and subsequent triggers following a shutdown of the down counter are functional).

## 16.4 Cautions

This section describes the cautions to be considered while using the programmable pulse generator

### Cautions

- If the Interrupt Request flag (PCN:IRQF) equals “1” and the Interrupt Request flag is set to “0” at the same timing, the setting of the Interrupt Request flag to “1” overrides the flag clear request.
- The first load has a maximum delay of 2.5T after the activation trigger. (T: Count clock)
  If the down counter is loaded and counts at the same time, the load operation overrides.




- Be sure to write duty value PDUT after cycle PCSR has been initialized and rewritten.
  (Always write in the order of (1) PCSR and (2) PDUT).
  Only the PDUT can be written for rewriting the duty.
- The duty value PDUT must be equal or smaller than the cycle value PCSR. If any larger value has been set, disable the operation of the PPG before replacing the duty value with a smaller value.
- Always access PPG Period Setting registers PCSR and PPG Duty Setting registers PDUT in a word (16-bit) format. If these registers are byte-accessed, no values would be written to their upper and lower bit positions.
- To activate a PPG, it is necessary to set the Timer Operation Enable bits (PCN:CNTE) to “1” before or concurrently with the activation to enable the PPG operation.
- The values of mode (MDSE), restart enable (RTRG), count clock (CKS[1:0]), trigger input edge (EGS[1:0]), interrupt cause (IRS), internal trigger (TSEL), and output polarity specification (OSEL) may not be changed while the PPG is operating.
  If any of these values has been changed while the PPG was operating, disable the operation of the PPG before reloading the register.
- Whenever writing a value to GCN2, be sure to write “0” to any undefined part of the upper 4 bits.
  If “1” is written, disable the operation of the PPG before reloading the register.
- If Activation Trigger Specification bits (TSEL0[3:0]), (TSEL1[3:0]), (TSEL2[3:0]), (TSEL3[3:0]) have been set to any value outside the specified range (0110, 0111, 1000 - 1111), disable the operation of the PPG and then write the specified value to let the register return to normal.
- If the Timer Operation Enable bit (PCN:CNTE) is set to "0" to disable PPGn while it is operating, the PPG stops and PPG output level is set back to the default value (low level if PCN:OSEL bit is set to "0" or high level if PCN:OSEL bit is set to "1").
  To activate PPG operation again, PCN:CNTE bit has to be set to "1" and after providing the selected PPG trigger the PPG counter is loaded with PPG cycle value and PPG pulse generation is re-started.

# 17. External Interrupts

This chapter explains the functions and operations of the External Interrupts.

17.1 Outline of External Interrupts

17.2 External Interrupt Registers

17.3 Notes on using the External Interrupt functions

## 17.1 Outline of External Interrupts

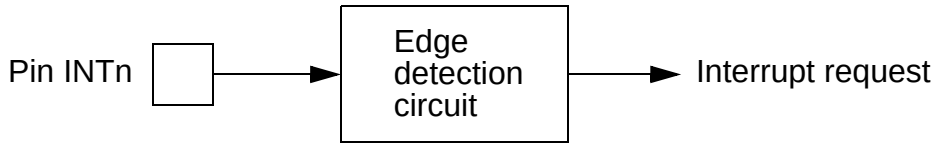
The External Interrupt detects a signal input to an external interrupt pin and generates an interrupt request.

### External interrupts

For an external interrupt request, four request levels are available: "H", "L", rising edge, and falling edge are available.

### Block diagram of External Interrupts

Figure 17-1. Block diagram of External interrupts



### External interrupts registers

| | bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|
| ENIR0 | | EN7 | EN6 | EN5 | EN4 | EN3 | EN2 | EN1 | EN0 | 00000000 B |
| ENIR1 | bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | EN15 | EN14 | EN13 | EN12 | EN11 | EN10 | EN9 | EN8 | 00000000 B |
| EIRR0 | bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | | ER7 | ER6 | ER5 | ER4 | ER3 | ER2 | ER1 | ER0 | 00000000 B |
| EIRR1 | bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | | ER15 | ER14 | ER13 | ER12 | ER11 | ER10 | ER9 | ER8 | 00000000 B |
| ELVR0 | bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | LB3 | LA3 | LB2 | LA2 | LB1 | LA1 | LB0 | LA0 | 00000000 B |
| ELVR0 | bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | | LB7 | LA7 | LB6 | LA6 | LB5 | LA5 | LB4 | LA4 | 00000000 B |
| ELVR1 | bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | LB11 | LA11 | LB10 | LA10 | LB9 | LA9 | LB8 | LA8 | 00000000 B |
| ELVR1 | bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | | LB15 | LA15 | LB14 | LA14 | LB13 | LA13 | LB12 | LA12 | 00000000 B |

## 17.2 External Interrupt Registers

The External Interrupt mdule has the following registers:

- Interrupt Enable register (ENIRn: External Interrupt Request Enable Register)
- Interrupt flag (EIRRn: External Interrupt Request Register)
- Request level setting register (ELVRn: External Level Register)

### Interrupt Enable Register (ENIRn: External Interrupt Request Enable Register)

Figure 17-2. External Interrupt Request Enable Register (ENIRn)

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| ENIR0 | EN7 | EN6 | EN5 | EN4 | EN3 | EN2 | EN1 | EN0 | $00000000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| ENIR1 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | EN15 | EN14 | EN13 | EN12 | EN11 | EN10 | EN9 | EN8 | $00000000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W : Readable and writable

The ENIRn register contains the interrupt enable bits of the external interrupt module. When an ENx bit is enabled, an interrupt request is signaled if the specified interrupt event is detected at the corresponding pin. Writing "1" to these bits enables and writing "0" disables interrupt requests.

**Note:**

When an interrupt should be enabled, please also set the corresponding port's Port Input Enable Register PIER (please refer to section Port Input Enable Register (PIERnn) on page 292).

### Interrupt flags (EIRRn: External Interrupt Request Register)

Figure 17-3. External Interrupt Request Register (EIRRn)

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| EIRR0 | ER7 | ER6 | ER5 | ER4 | ER3 | ER2 | ER1 | ER0 | $00000000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| EIRR1 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
| | ER15 | ER14 | ER13 | ER12 | ER11 | ER10 | ER9 | ER8 | $00000000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W : Readable and writable

The EIRRn register contains the interrupt flags.

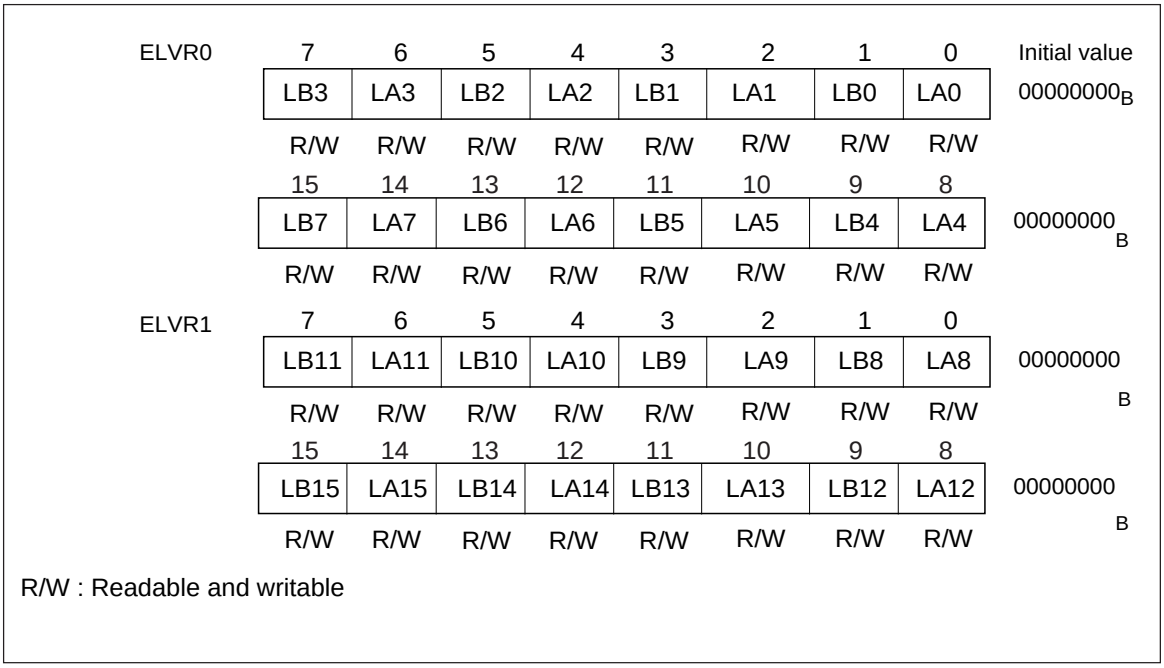
If the specified interrupt event is detected at a pin, the corresponding ERx bit is set and if the ENx bit is "1", then an interrupt request is signaled to the interrupt controller. Writing "1" to these bits has no effect and writing "0" clears the flags. "1" is always read from these bits by the read-modify-write instructions.

**Note:**

When clearing the interrupt flag, make sure to clear the flag which caused the interrupt alone but not others.

### Request level setting register (ELVRn: External Interrupt Level Register)

Figure 17-4. External Interrupt Level Register (ELVRn)




ELVRn defines the request event at the external pin. Each pin is assigned with two bits as described in Table 17-1. If a request is detected by the input level, the interrupt flag is set as long as the input is at the specified level even after the flag is reset by software.

Table 17-1. Interrupt request detection factor for external pins

| LBx | LAx | Interrupt request detection factor |
|---|---|---|
| 0 | 0 | L level pin input |
| 0 | 1 | H level pin input |
| 1 | 0 | Rising edge pin input |
| 1 | 1 | Falling edge pin input |

## 17.3 Notes on using the External Interrupt functions

The following points must be considered to use the External Interrupt function.

- Conditions on the behavior of external circuit for use of DMA
- Clearing interrupt flag
- External interrupt request level

### Conditions on the behavior of external circuit for use of DMA

- An external circuit which uses DMA must be able to inactivate the request signal when the requested DMA transfer is performed. For the correct usage of external interrupts to start DMA transfers please refer to application note mcu-an-300203-e.

### Clearing interrupt flag

- When it is used as an external interrupt the interrupt flag must be cleared within the interrupt service routine. Otherwise the same service is performed again after the completion of the first interrupt service.

- When level detection is specified as the event input, the interrupt flag is set again even after it is cleared as long as the active level is kept at the input pin. In this case, the external cause of the request should be cleared or the interrupt enable bit should be cleared.

### External interrupt request level

- When edge detection is specified as the event input, the pulse of the input signal must have a minimum width to be recognized as an input edge and not be filtered by the noise filter. See datasheet of the specific device for the minimum pulse width length.
- When level detection is specified as the event input, the interrupt flag keeps active status once the specified level is input even after the input signal changes to the inactive level as shown in Figure 17-6. In order to clear the request, the interrupt flag must be cleared.

Figure 17-5. Clearing interrupt cause register upon level set

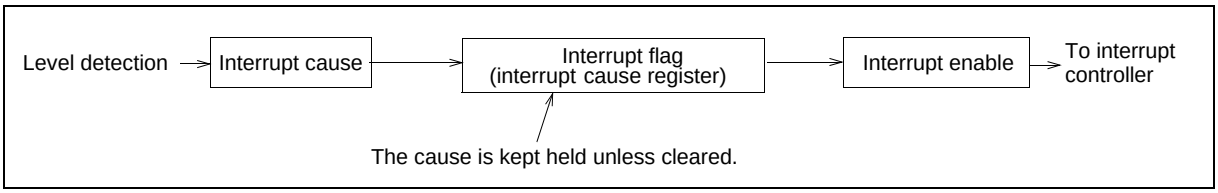
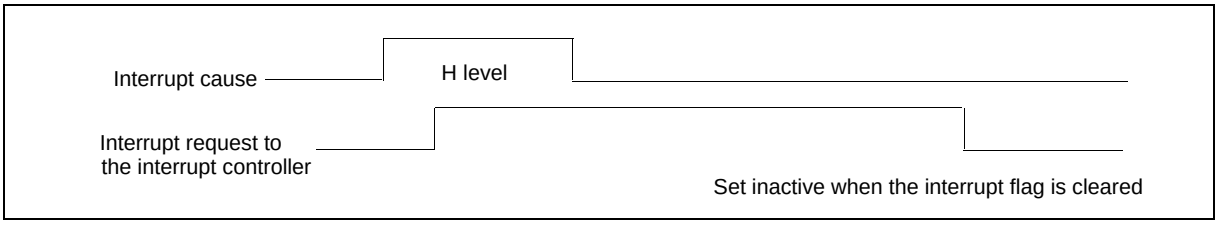



Figure 17-6. Interrupt cause and interrupt request to the interrupt controller while interrupts are enabled



# 18. A/D Converter

This chapter explains the function and operation of the A/D converter.

18.1 Outline of A/D Converter

18.2 Registers for A/D Converter

18.3 Operation of A/D Converter

18.4 Conversion using DMA

18.5 Conversion Data Protection Function

## 18.1 Outline of A/D Converter

The A/D converter converts analog input voltages to digital values.

### Outline of A/D converter

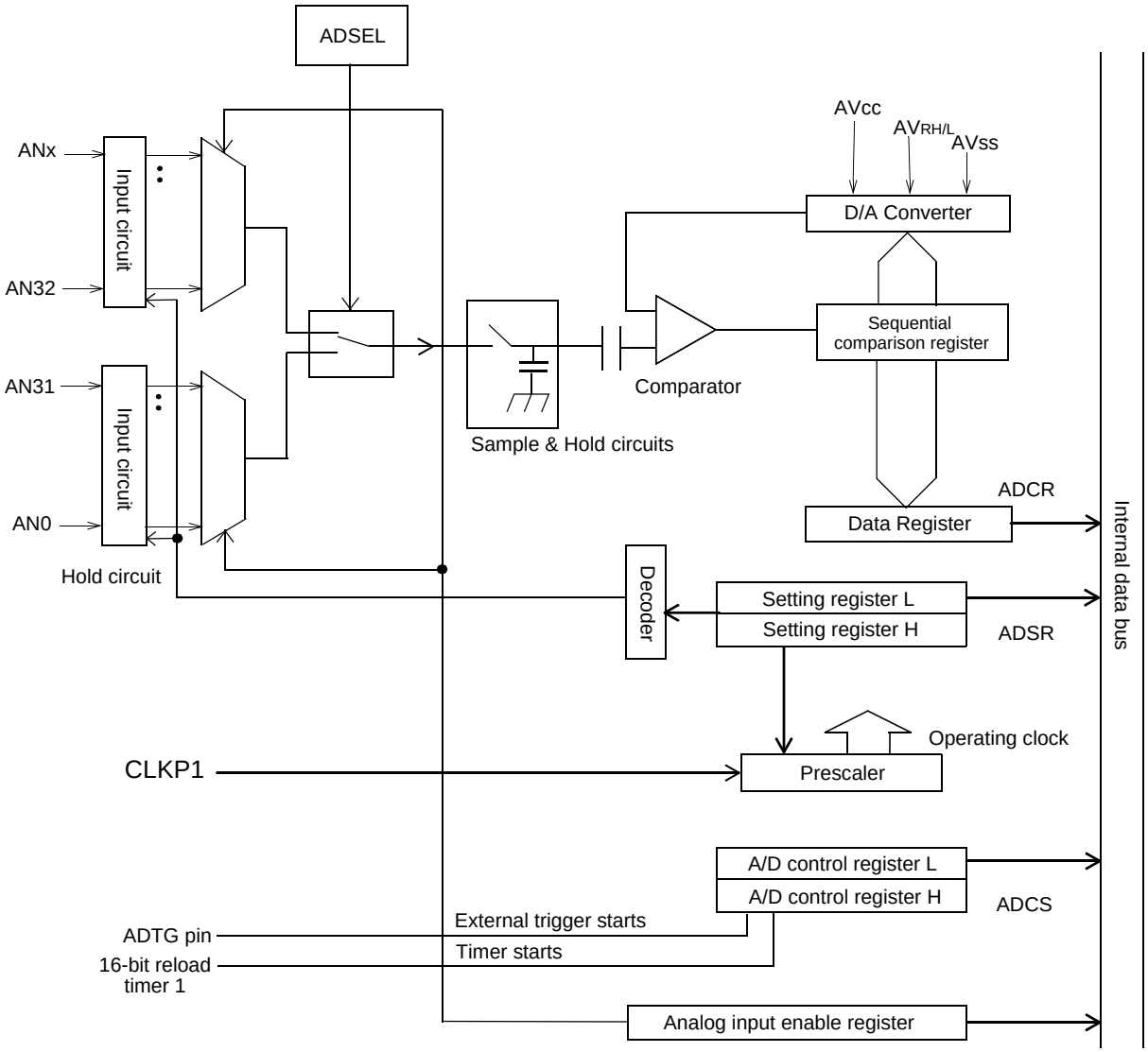
- Conversion time: 1.9 μs min. per channel (with 24-MHz machine clock)
- RC successive approximation conversion with sample and hold circuit
- Resolution: 10 bits or 8 bits
- Analog input pin (channel) selected by program
  - Single-channel conversion: 1 channel is selected and converted.
  - Scan conversion: Sequential channels are converted.
- Mode setting
  - Single mode: Converts the specified channels once.
  - Continuous mode: Converts the specified channels repeatedly.
  - Stop mode: Converts one channel, then the converter stops and waits for the next activation. (starting of conversion can be synchronized)
- Interrupt requests

  At the end of A/D conversion, the interrupt request for an A/D conversion can be signaled to the interrupt controller. This interrupt can start DMA, which can transfer A/D conversion result data to the memory.
- Selectable activation cause

  The activation can be done by software, Reload Timer 1, or external trigger (falling edge).

### Block diagram of A/D converter

Figure 18-1. Block diagram of A/D converter




**Remark:**

Set pins to be used as analog input to “analog input enable” by setting “1” to corresponding bit of Analog Input Enable Register (ADER).

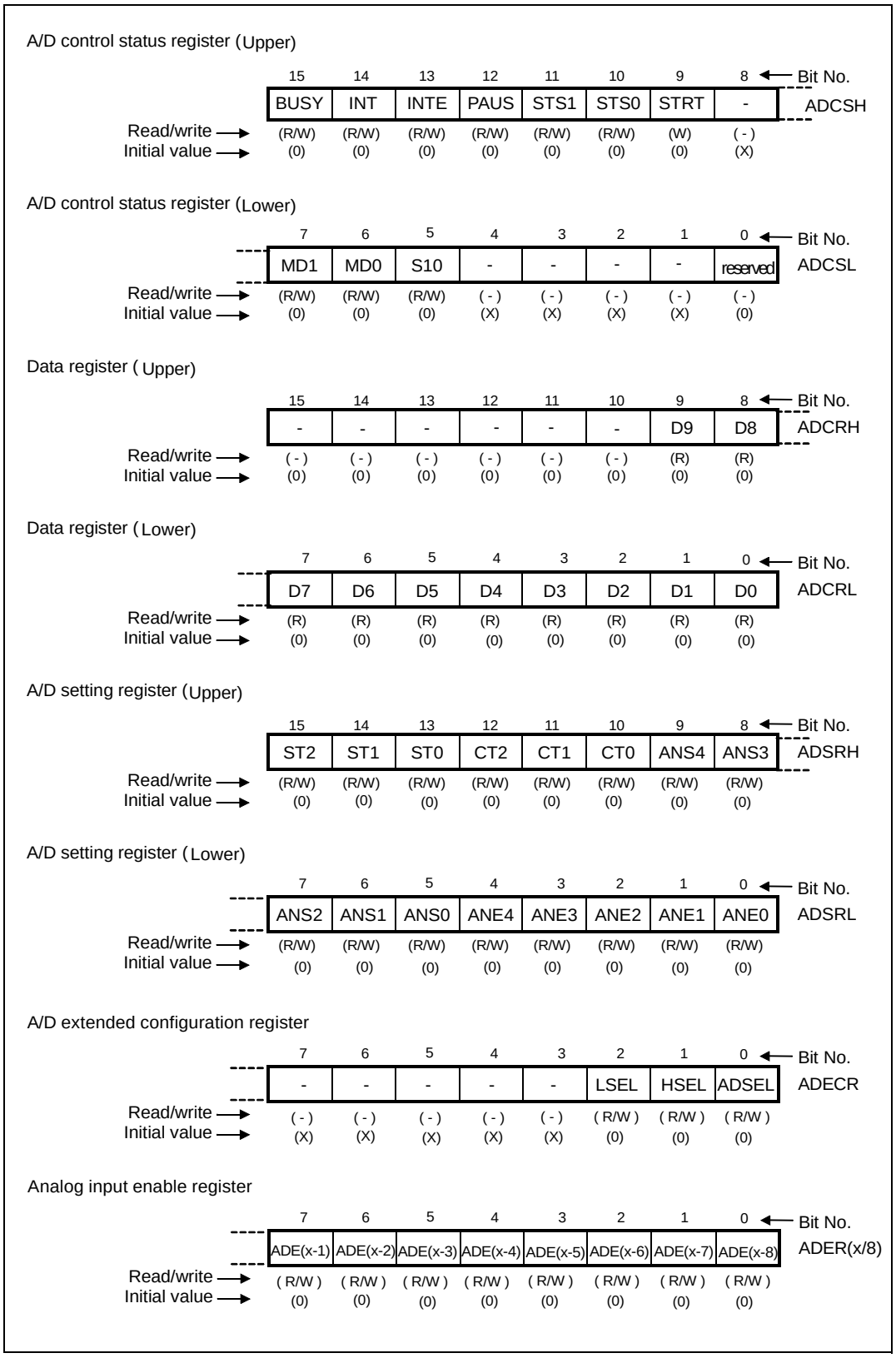
## 18.2 Registers for A/D Converter

The A/D converter has the following registers:

- Control status register: ADCS
- Data register: ADCR
- Setting register: ADSR
- Extended configuration register: ADECR
- Analog input enable register: ADERx

## Registers for A/D converter

Figure 18-2. Registers of the A/D Converter

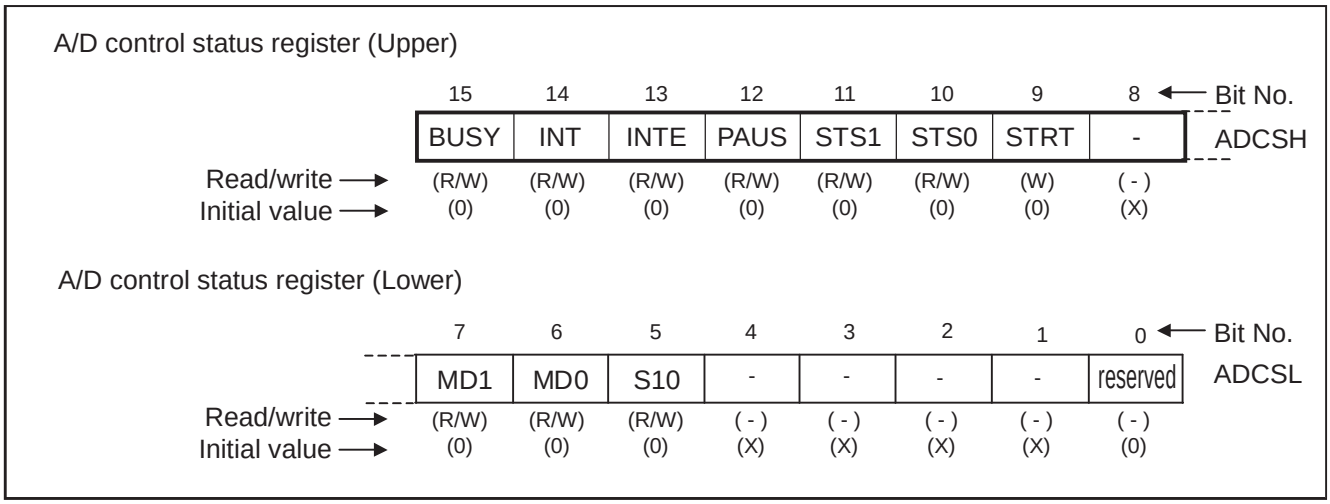


## 18.2.1 Control status register (ADCS)

The control status register (ADCS) controls the A/D converter and indicates its status. Do not rewrite ADCS during A/D conversion.

### *18.2.1.1 Control Status Register (ADCS)*

Figure 18-3. A/D control status register (ADCS)




### *18.2.1.2 Control Status Register (ADCSH)*

**[bit 15] BUSY (busy flag and stop)**

- Read operation:
  - This bit indicates an operation of the A/D converter.
  - The bit is set by starting of A/D conversion, and cleared by end of A/D conversion.
- Write operation:
  - When “0” is written to this bit during the A/D conversion, the conversion is forced to stop.
  - Writing “1” to this bit has no effect.

**Notes:**

- “1” is read from this bit when an read-modify-write instruction is used.
- In the single mode, this bit is cleared when A/D conversion ends.
- In the continuous or stop mode, the A/D conversion does not stop until writing “0” to this bit.
- This bit is initialized to 0 by reset.
- Do not perform the forced stop and the activation concurrently (using software (BUSY = 0, STRT = 1), external trigger, or timer).

**[bit 14] INT (interrupt)**

This bit is set when converted data is written to ADCR. When this bit is set with bit 13 (INTE) set to “1”, an interrupt request is generated. If DMA is configured and enabled, DMA is activated instead. Writing “1” to this bit has no effect. This bit is cleared by writing “0” or by the interrupt clear signal from the DMA.

**Notes:**

- To clear this bit by writing “0”, ensure that A/D conversion is not in progress.
- This bit is initialized to “0” at reset.

- Using an read-modify-write instruction, “1” is read from this bit.

### [bit 13] INTE (Interrupt enable)

This bit enables and disables the interrupt.

- 0: Interrupt disabled
- 1: Interrupt enabled

When using DMA, set this bit (DMA is started by an interrupt request).

This bit is initialized to 0 at reset.

### [bit 12] PAUS (A/D converter pause)

PAUS bit indicates that the A/D conversion data protection function is activated. PAUS bit is valid only if A/D converter interrupt is enabled (ADCS:INTE=1).

When A/D conversion data protection function was activated this bit is set to "1"

This bit can only be cleared by writing "0" to it.

Writing "1" to this bit sets it.

When A/D conversion is performed with enabled interrupt (ADCS:INTE=1), an interrupt request is generated at the same time as the interrupt request flag bit (ADCS:INT) is set after the A/D conversion ends once. If the next A/D conversion ends without clearing the interrupt request flag bit (ADCS:INT), the A/D conversion operation will be paused to prevent the previous data from being overwritten and destroyed (A/D conversion data protection function). When an A/D conversion operation is paused, PAUS bit is set to "1".

When the interrupt request flag bit (ADCS:INT) is cleared, the A/D converter cancels the pause state and restarts the A/D conversion operation.

The interrupt request flag bit (ADCS:INT) is cleared by writing "0" to it. In addition, if an A/D conversion result is transferred from the A/D data register using DMA, the interrupt request flag bit (ADCS:INT) is cleared by the DMA controller once the transfer of the A/D conversion result is completed.

### Notes:

For the A/D conversion data protection function, see 18.5 Conversion Data Protection Function.

ADCS:PAUS bit is not cleared automatically even when the pause state is canceled. Write "0" to clear PAUS bit.

### [bit 11 and bit 10] STS1 and STS0 (Start source select)

These bits are initialized to “$00_B$” by reset.
Select an A/D conversion activation cause by setting these bits.

Table 18-1. Function Settings

| STS1 | STS0 | Function |
|---|---|---|
| 0 | 0 | Activation A/D conversion by software |
| 0 | 1 | Activation A/D conversion by external pin trigger and by software |
| 1 | 0 | Activation A/D conversion by timer and by software |
| 1 | 1 | Activation A/D conversion by external pin trigger, timer, and by software |

In a mode allowing two or more activation causes, A/D conversion is activated by the source that occurs first. When changing the setting of these bits during A/D conversion, the result is immediately reflected. Therefore it is not a recommended practice.

### Notes:

- An external pin trigger is detected by the falling edge.
  When the external trigger input level is Low, setting external pin triggers starting may start A/D converter.

- When the timer is selected, the 16-bit reload timer 1 is selected and the conversion is activated when the output from the timer becomes “1”.

### [bit 9] STRT (Start)

A/D conversion is started by writing “1” to this bit.
To reactivate A/D conversion, write “1” to this bit again.
At reset, this bit is initialized to “0”.
Reading this bit always returns “0”. Reactivation during the operation is only available in the single mode 1, but not supported in the single mode2, continuous mode and the stop mode. Check the BUSY bit before writing “1” to this bit in the latter three modes.
Do not force to stop A/D conversion concurrently with activation of A/D conversion by software (BUSY=0, STRT=1).

### [bit 8] Unused bit

Always write "0" to this bit.

Reading this bit returns an undefined value.

Read-modify-write is not affected.

## *18.2.1.3 Control Status Register (ADCSL)*

### [bit 7 and bit 6] MD1 and MD0 (A/D converter mode set)

Table 18-2. Operation Mode Settings

| MD1 | MD0 | Operation mode |
|---|---|---|
| 0 | 0 | Single mode 1 (Reactivation during A/D conversion is allowed.) |
| 0 | 1 | Single mode 2 (Reactivation during A/D conversion is not allowed.) |
| 1 | 0 | Continuous mode (Reactivation during A/D conversion is not allowed.) |
| 1 | 1 | Stop mode (Reactivation during A/D conversion is not allowed.) |

- Single mode
  A/D conversion is continuously performed from the channel specified with ANS4 to ANS0 to the channel specified with ANE4 to ANE0. The conversion stops once it has been done for all these channels.
- Continuous mode
  A/D conversion is repeatedly performed from the channel specified with ANS4 to ANS0 to the channel specified with ANE4 to ANE0 in a row.
- Stop mode
  A/D conversion is performed from the channel specified with ANS4 to ANS0 to the channel specified with ANE4 to ANE0, pausing for each channel. The A/D conversion is resumed upon an activation.

### Note:

- The A/D conversion in the continuous or stop mode continues until it is stopped by the BUSY bit.
- Write 0 to the BUSY bit to stop the A/D conversion.
- Reactivation is disabled in single mode2, continuous mode, and stop mode. This applies to all kinds of activation.

### [bit 5] S10

This bit specifies the resolution of conversion. When “0” is written to this bit, 10-bit A/D conversion is performed. When “1” is written to this bit, 8-bit A/D conversion is performed and the conversion result is stored in D7 to D0.

### [bits 4 to bit 1] Unused bits

Always write "0" to these bits. The read value of these bits is undefined.

**[bit 0] reserved (reserved bit)**

This bit is a reserved bit. Always write “0” to this bit.

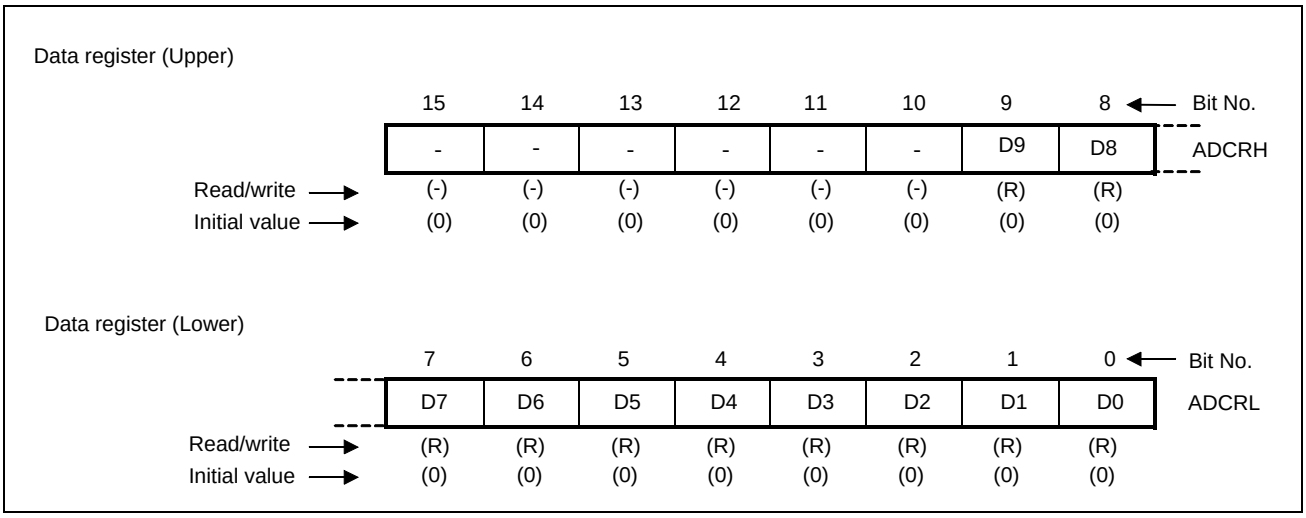
Reading this bit always returns “0”.

## 18.2.2 Data Register (ADCR)

The data register (ADCR) is used to store digital value generated as a result of conversion. The register value is rewritten every time the conversion ends. Normally, the last converted value is stored in these register's bits.

### *18.2.2.1 Data register (ADCR)*

Figure 18-4. Data register (ADCR)




Reading bit 10 to bit 15 of ADCRH always returns “0”.

When S10 bit of ADCSL is “1”, the 8-bit mode is established, storing converted data in bit7 to bit0. In this case, Reading bit9 to bit8 always returns “0”.

For the use of the conversion data protection function, see section 18.5 Conversion Data Protection Function

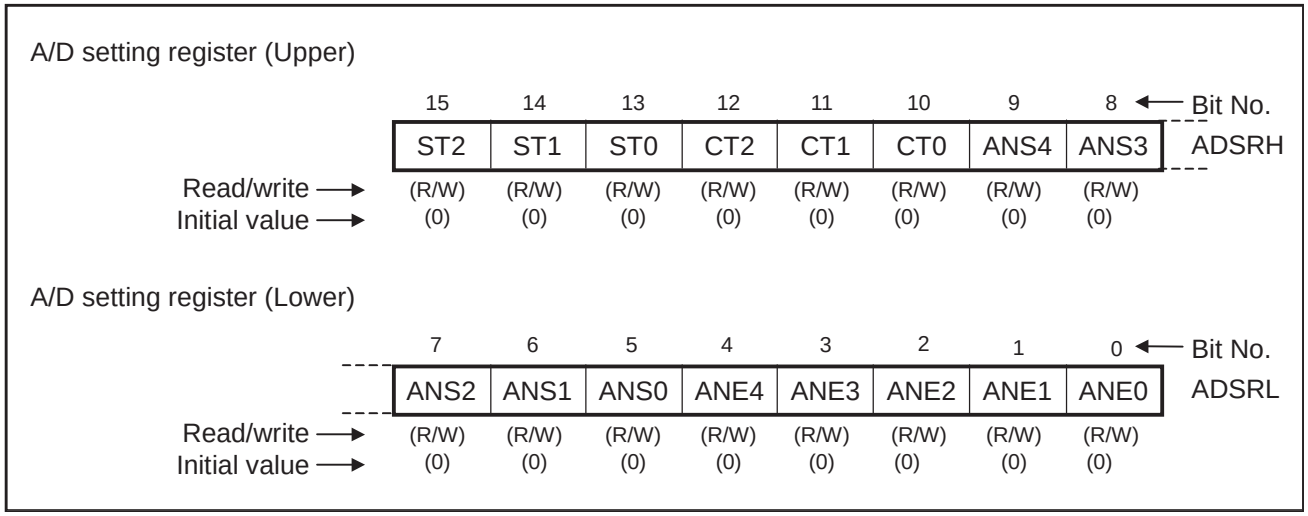
Do not write to this register.

## 18.2.3 Setting Register (ADSR)

The setting register (ADSR) is used to set the A/D conversion time and the sampling channels and to indicate the current sampling channel.

### 18.2.3.1 Setting Register (ADSR)

Figure 18-5. Setting register (ADSR)




**[bit 15 to bit 13] ST2, ST1, ST0 (Sampling time)**

Table 18-3. Function Settings

| ST2 | ST1 | ST0 | Function |
|---|---|---|---|
| 0 | 0 | 0 | 4-peripheral clock CLKP1 cycle (8 MHz, 0.5 µs) |
| 0 | 0 | 1 | 6-peripheral clock CLKP1 cycle (8 MHz, 0.75 µs) |
| 0 | 1 | 0 | 8-peripheral clock CLKP1 cycle (16 MHz, 0.5 µs) |
| 0 | 1 | 1 | 12-peripheral clock CLKP1 cycle (24 MHz, 0.5 µs) |
| 1 | 0 | 0 | 24-peripheral clock CLKP1 cycle (8 MHz, 3 µs) |
| 1 | 0 | 1 | 36-peripheral clock CLKP1 cycle (16 MHz, 2.25 µs) |
| 1 | 1 | 0 | 48-peripheral clock CLKP1 cycle (16 MHz, 3.0 µs) |
| 1 | 1 | 1 | 128-peripheral clock CLKP1 cycle (24 MHz, 5.3 µs) |

These bits define the sampling time for the analog input signal. Setting for these bits must be carefully evaluated. Otherwise, the result may not provide the guaranteed conversion accuracy.

The required sampling time depends on the external driving impedance, the board capacitance at the A/D converter input pin and the Avcc voltage level. See the datasheet for details.

**[bit 12 and bit 10] CT2, CT1, CT0 (Compare time)**

Table 18-4. Function Settings

| CT2 | CT1 | CT0 | Function |
|---|---|---|---|
| 0 | 0 | 0 | 22-peripheral clock CLKP1 cycle (8 MHz, 2.8 µs) |
| 0 | 0 | 1 | 33-peripheral clock CLKP1 cycle (16 MHz, 2.1 µs) |
| 0 | 1 | 0 | 44-peripheral clock CLKP1 cycle (20 MHz, 2.2 µs) |
| 0 | 1 | 1 | 66-peripheral clock CLKP1 cycle (24 MHz, 2.8 µs) |
| 1 | 0 | 0 | 88-peripheral clock CLKP1 cycle (8 MHz, 11.0 µs) |
| 1 | 0 | 1 | 132-peripheral clock CLKP1 cycle (16 MHz, 8.3 µs) |
| 1 | 1 | 0 | 176-peripheral clock CLKP1 cycle (20 MHz, 8.8 µs) |
| 1 | 1 | 1 | 264-peripheral clock CLKP1 cycle (24 MHz, 11.0 µs) |

These bits define the conversion time of the successive approximation. Setting for these bits must meet the following conditions. Otherwise, the result may not provide the guaranteed conversion accuracy.

$\geqq$ 1.0µs; For 4.5V $\leqq$ Avcc $\leqq$ 5.5V

$\geqq$ 2.0µs; For 3.0V $\leqq$ Avcc < 4.5V

### [bit 9 to bit 5] ANS4, ANS3, ANS2, ANS1, ANS0 (Analog start channel set)

These bits are used to set the starting channel for A/D conversion, and to indicate the current analog input channel. When the A/D converter is activated, A/D conversion is started from the channel selected by these bits.

Table 18-5. Starting Channel Settings

| ANS4 | ANS3 | ANS2 | ANS1 | ANS0 | Starting channel when ADECR:ADSEL = 0 | Starting channel when ADECR:ADSEL = 1 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | AN0 | AN32 |
| 0 | 0 | 0 | 0 | 1 | AN1 | AN33 |
| 0 | 0 | 0 | 1 | 0 | AN2 | AN34 |
| 0 | 0 | 0 | 1 | 1 | AN3 | AN35 |
| | | | | | • • • | |
| 1 | 1 | 1 | 0 | 0 | AN28* | AN60* |
| 1 | 1 | 1 | 0 | 1 | AN29* | AN61* |
| 1 | 1 | 1 | 1 | 0 | AN30* | AN62* |
| 1 | 1 | 1 | 1 | 1 | AN31* | AN63* |

* For the highest ADC channel number available on the device, please refer to the corresponding data sheet.

■ Read operation

These bits indicates the current analog input channel for the conversion operation. When the conversion operation is stopped, the last channel is indicated.
Before A/D conversion starts, the previous conversion channel will be read even if these bits have already been set to the new value.

■ At reset: These bits are initialized to "$00000_B$".

### [bit 4 to bit 0] ANE4, ANE3, ANE2, ANE1, ANE0 (Analog end channel set)

These bits are used to set the ending channel for A/D conversion.

Table 18-6. Ending Channel Settings (Sheet 1 of 2)

| ANE4 | ANE3 | ANE2 | ANE1 | ANE0 | Ending channel when ADECR:ADSEL = 0 | Ending channel when ADECR:ADSEL = 1 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | AN0 | AN32 |
| 0 | 0 | 0 | 0 | 1 | AN1 | AN33 |
| 0 | 0 | 0 | 1 | 0 | AN2 | AN34 |
| 0 | 0 | 0 | 1 | 1 | AN3 | AN35 |
| | | | | | • • • | |
| 1 | 1 | 1 | 0 | 0 | AN28* | AN60* |

Table 18-6. Ending Channel Settings<Italic> (continued) (Sheet 2 of 2)

| ANE4 | ANE3 | ANE2 | ANE1 | ANE0 | Ending channel when ADECR:ADSEL = 0 | Ending channel when ADECR:ADSEL = 1 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | AN29* | AN61* |
| 1 | 1 | 1 | 1 | 0 | AN30* | AN62* |
| 1 | 1 | 1 | 1 | 1 | AN31* | AN63* |

* For the highest ADC channel number available on the device, please refer to CHAPTER 1 "OVERVIEW"

- At reset, these bits are initialized to "$00000_B$".

**Note:**

- When writing to this register, always use word access. When byte write or read-modify-write is performed for this register, A/D conversion may be started from an unintended channel.
- When the same channel is written to the ANE4 to ANE0 bits and to the ANS4 to ANS0 bits, conversion is performed for only 1 channel (single channel conversion).
- When conversion of the Analog end channel set by ANE4 to ANE0 is ended, with the continuous mode or the stop mode, control returns to the Analog start channel set by the ANS4 to ANS0 bits.
- When the ANS > ANE is set for channel, conversion starts from the Analog start channel up to channel 31, and then returns to AN0 and ends in the Analog end channel. For products with less than 32 A/D converter channels, the conversion results can be incorrect. To prevent this, do not set ANS larger than ANE.
- It is not possible to have a start channel < 32 and an end channel ≥ 32. If this is needed, please use the following sequence:
  - Set ADECR:ADSEL = 0 to convert channels from required start channel to channel 31.
  - Set ADECR:ADSEL = 1 and convert channel 32 to the required end channel.
- Do not set number ANEx bits or ANSx bits to channel numbers that do not exist on the device. Please refer to the data sheet of the product to determine available A/D converter channels.
- The conversion is done for all channels between ANSx and ANEx independent of whether a channel is available on a device or not. Please refer to the data sheet of the product to dertermine available A/D converter channels.

### 18.2.4 Extended Configuration Register (ADECR)

The extended configuration register (ADECR) is used to enable the A/D Converter input channels 32 and higher.

It also enables to switch the A/D converters high and low reference voltage between AVRH + AVRL, AVRH + AVSS and AVRH2 + AVSS. Switching the reference voltages enables adjusting the dynamic range of the A/D converter to the signal source.

### *18.2.4.1 ADC Extended Configuration Register (ADECR)*

Figure 18-6. ADC Extended configuration register (ADECR)

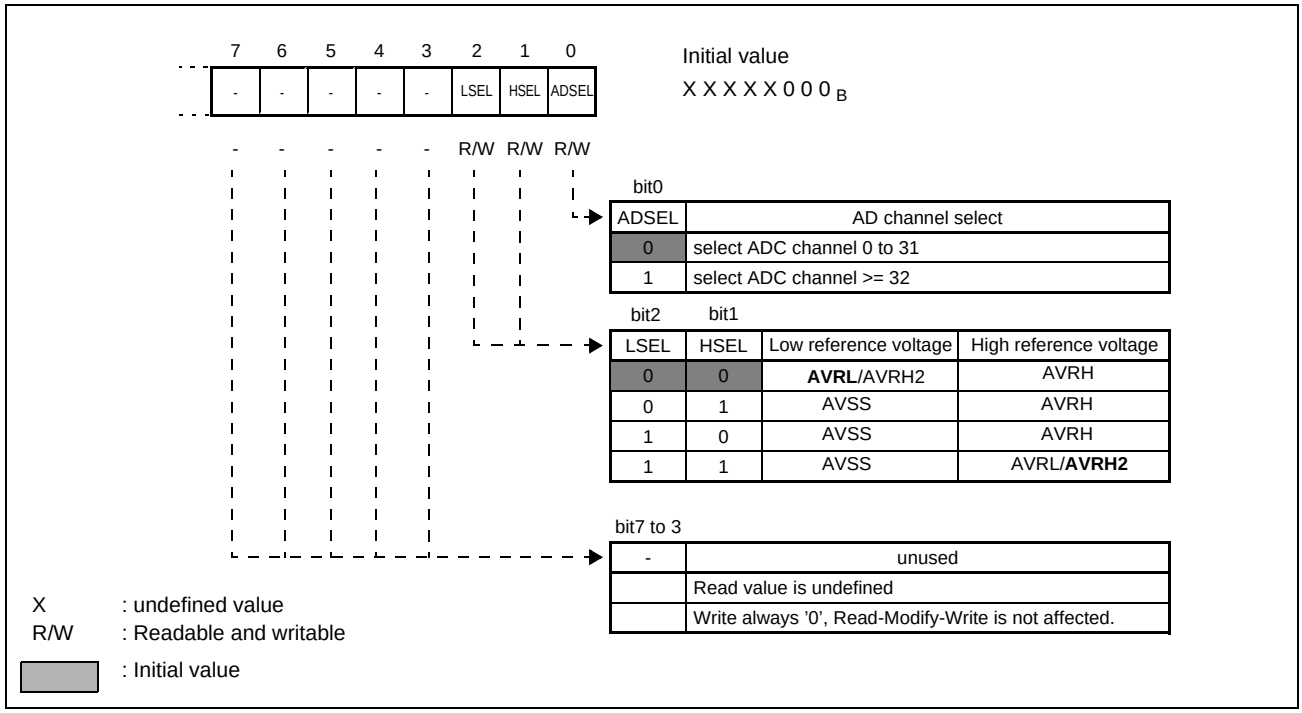
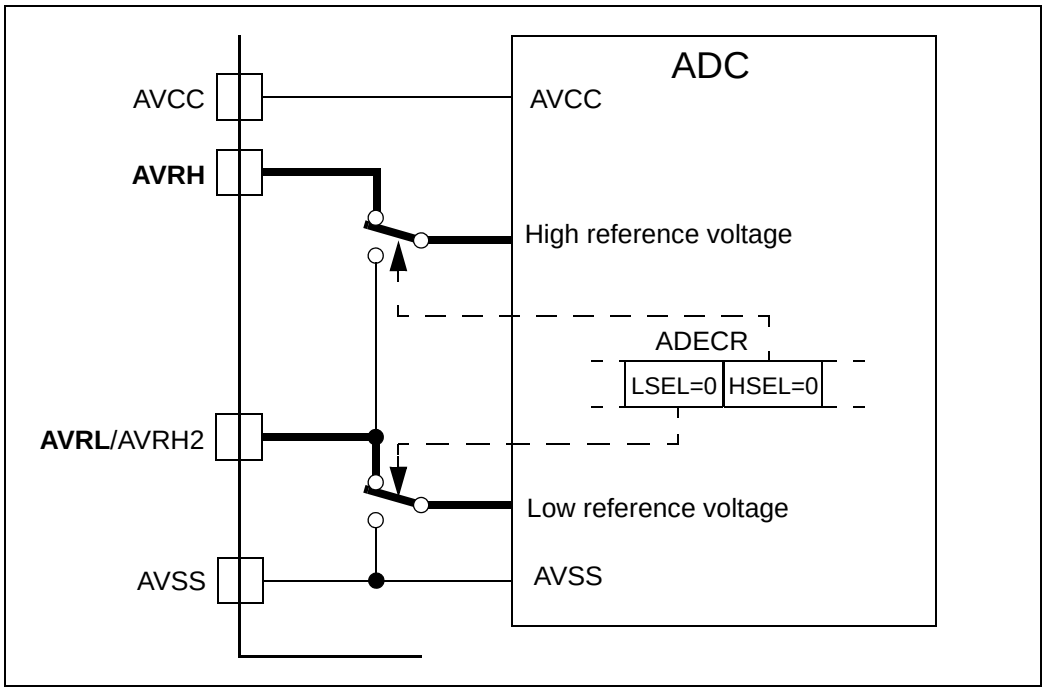


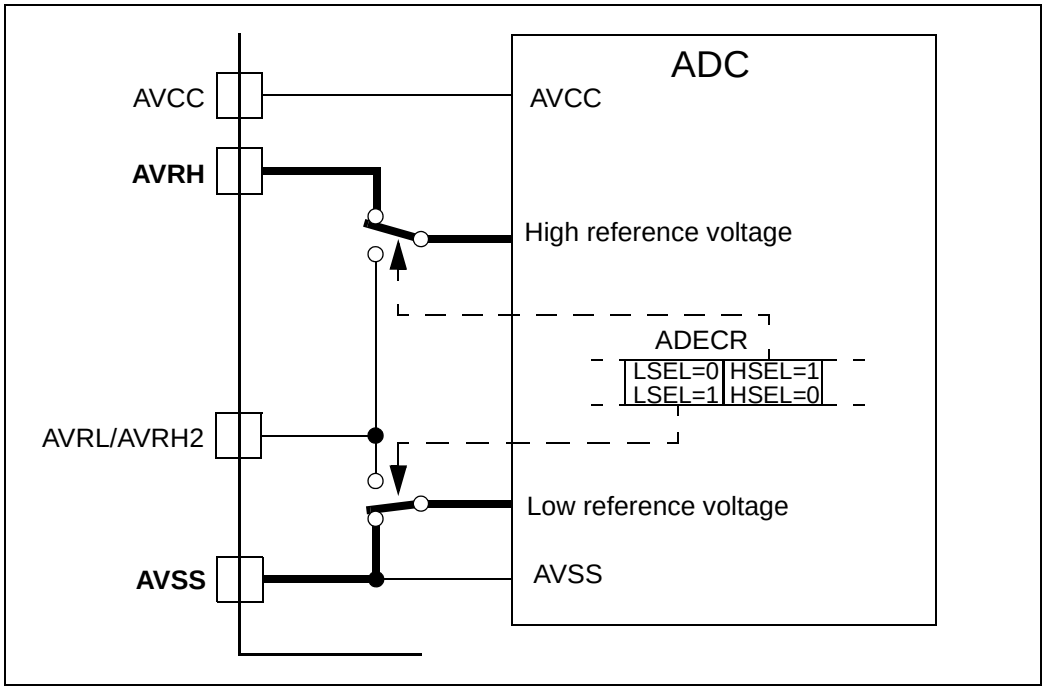

Figure 18-7. Switching of ADC Reference Voltage (LSEL="0", HSEL="0").



**Note:**

This setting is compatible to16LX configuration of A/D converter. It enables to have the low reference voltage of the A/D converter different from AVSS potential.

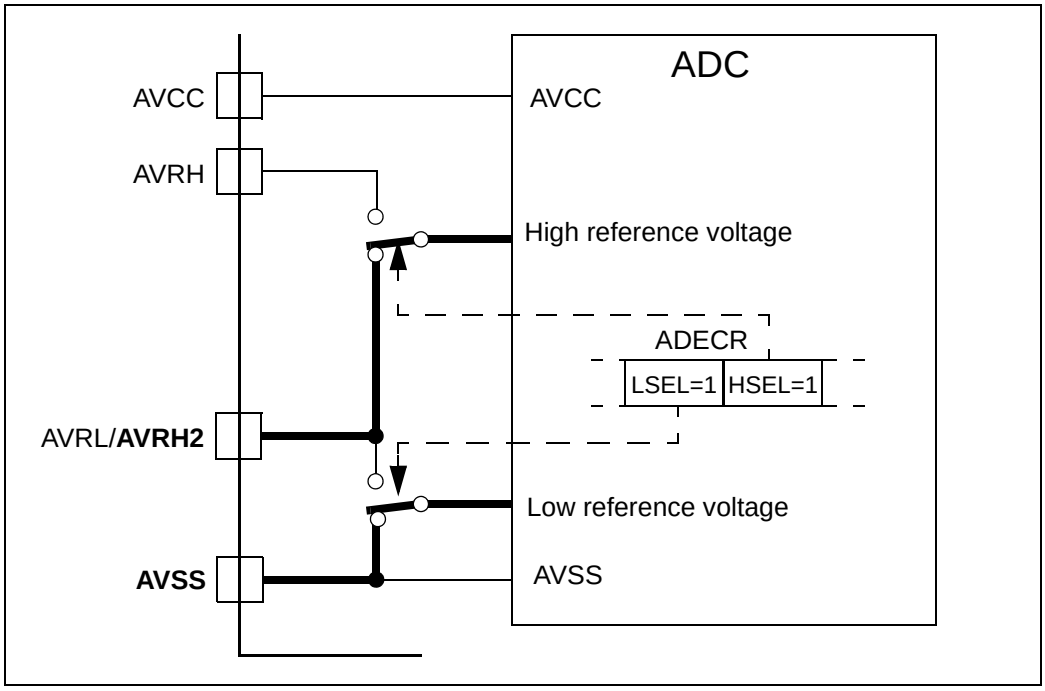
Figure 18-8. Switching of ADC Reference Voltage (LSEL="0", HSEL="1" or LSEL="1", HSEL="0")




**Note:**

This setting can be used in the case the low reference voltage of the A/D converter is equal to AVSS potential (e.g. system ground potential).

Figure 18-9. Switching of ADC Reference Voltage (LSEL="1", HSEL="1"




**Note:**

This setting can be used in the case the low reference voltage of the A/D converter is equal to AVSS potential (e.g. system ground potential). In addition, the pin AVRL/AVRH2 can be used to supply a high reference voltage different from the AVRH potential.

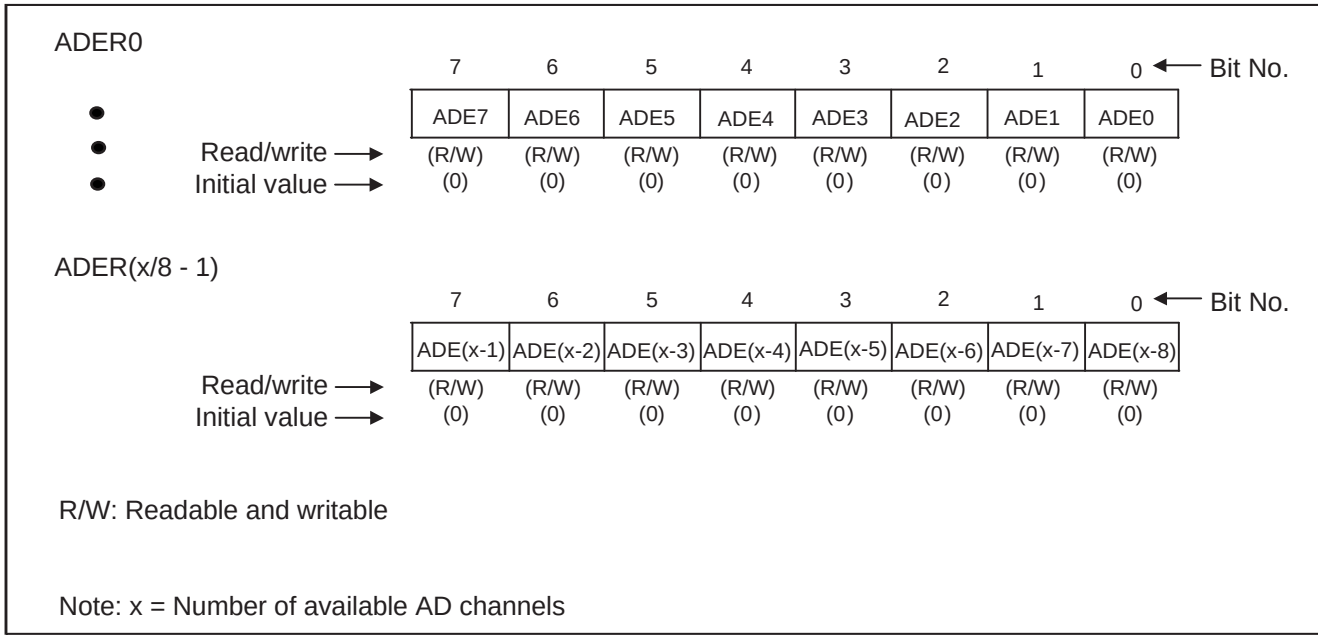
Please refer to the datasheet for the stabilization time needed when switching the high reference voltage of the A/D converter between AVRH and AVRH2.

### 18.2.5 Analog Input Enable Register (ADERx)

This register enables the analog input functions for the AD converter

#### *18.2.5.1 Analog input enable register (ADERx)*

Figure 18-10. Analog input enable register (ADERx)




**[bit 7 to bit 0] ADEx (Analog input enable bit)**

If an external pin is used as an analog input for the A/D converter, the corresponding bit should be set to "1".

Table 18-7. Mode selection

| | |
|---|---|
| 0 | Port input/output mode |
| 1 | Analog input mode |

## 18.3 Operation of A/D Converter

The A/D converter operates using the sequential-comparison converter system; and 10 bits or 8 bits can be selected for the A/D converter's resolution.

Since this A/D converter has only one register (10 bits; conversion result data register ADCR) for storing conversion results, this register is rewritten every time conversion is ended. So, this A/D converter alone is not suitable for continuous conversion; and therefore conversion be performed with transferring converted data to the memory using the DMA is recommended.

### 18.3.1 Single mode 1 / 2

In the single mode, analog inputs from the starting channel set by the ANS bits to the ending channel set by the ANE bits are sequentially converted; and when conversion of the ending channel is ended, A/D conversion is stopped. When the starting channel and the ending channel are the same (ANS = ANE), conversion is performed for only the channel set by the ANS bits.

Example:

ANS = $00000_B$, ANE = $00011_B$:

Start --> AN0 --> AN1 --> AN2 --> AN3 --> End

ANS = 00010 $_B$, ANE = 00010 $_B$:

Start --> AN2 --> End

### 18.3.2 Continuous mode

In the continuous mode, A/D conversion is repeated until “0” is written to the BUSY bit (when “0” is written to the BUSY bit, operation of the A/D converter is forced to stop).
Note that when operation of the A/D converter is forced to stop, A/D conversion may stop halfway. In this case, the conversion result register contains the previous data for which conversion completed.

### 18.3.3 Stop mode

In the stop mode, analog inputs from the starting channel set by the ANS bits to the ending channel set by the ANE bits are sequentially converted, but every time one channel is converted, conversion stops.

When conversion of the ending channel is ended, control returns to the starting channel to continue A/D conversion. When the starting channel and the ending channel are the same (ANS = ANE), conversion is performed for only one channel.

Example:

ANS = 00000$_B$, ANE = 00011 $_B$:

Start --> AN0 --> Stop --> Start --> AN1 --> Stop --> Start --> AN2 --> Stop --> Start --> AN3 --> Stop --> Start --> AN0 -->-->--> Repetition of sequence

ANS = 00010 $_B$, ANE = 00010 $_B$:

Start --> AN2 --> Stop --> Start --> AN2 --> Stop --> Start --> AN2 -->-->--> Repetition of sequence

In the above sequences, only A/D activation causes set using the STS1 and STS0 bits are valid.

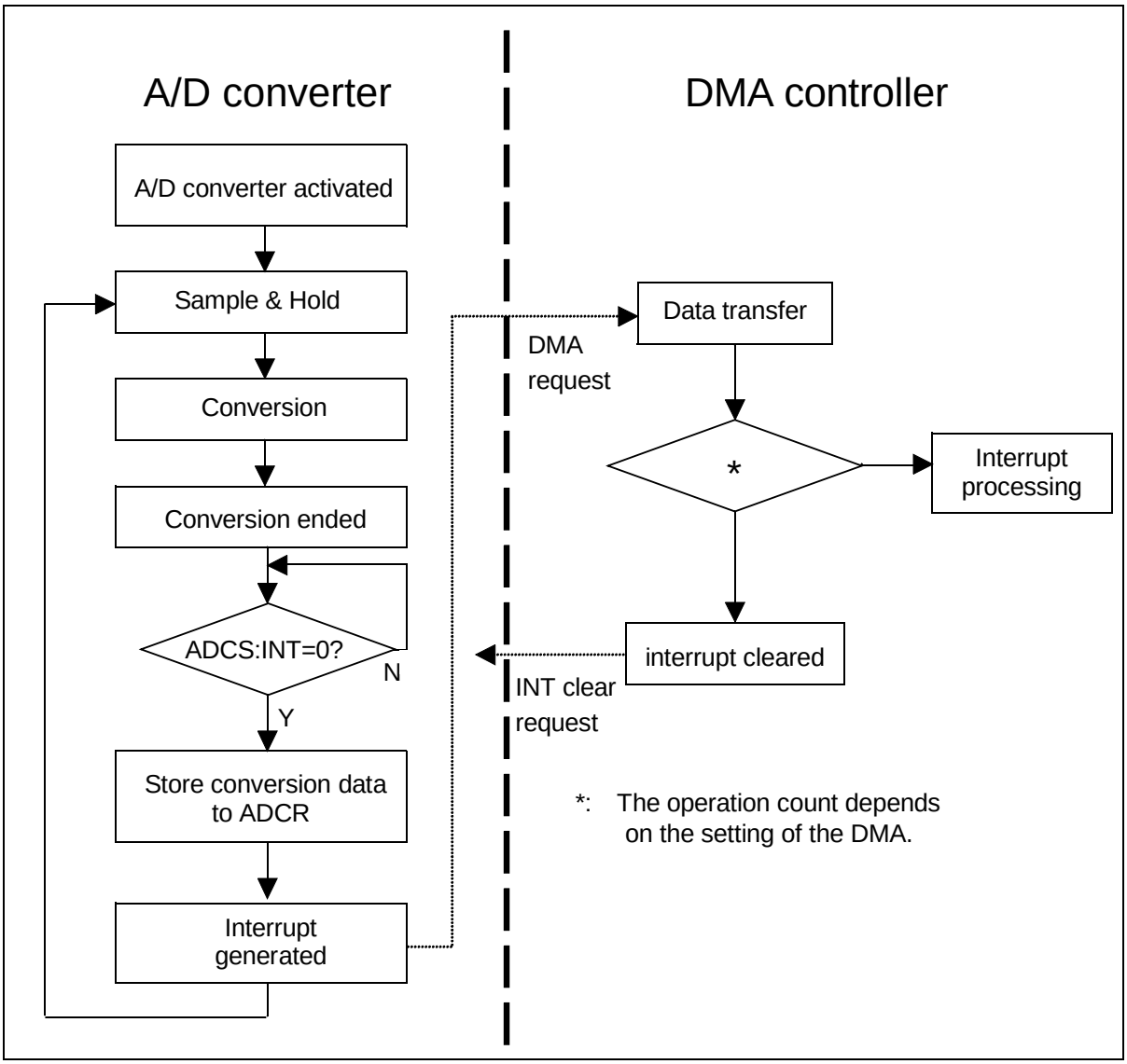
The start of conversion can be synchronized by this mode.

## 18.4 Conversion using DMA

Figure 18-11 gives an example of flow for the activation of A/D conversion to the transfer of converted data in continous mode.

### 18.4.1 Conversion using DMA

Figure 18-11. Example of flow for the activation of A/D conversion to the transfer of converted data in continous mode.



## 18.5 Conversion Data Protection Function

The data protection function is triggered when A/D conversion is performed while interrupt request output is enabled.

### 18.5.1 Explanation of A/D Conversion Data Protection Function

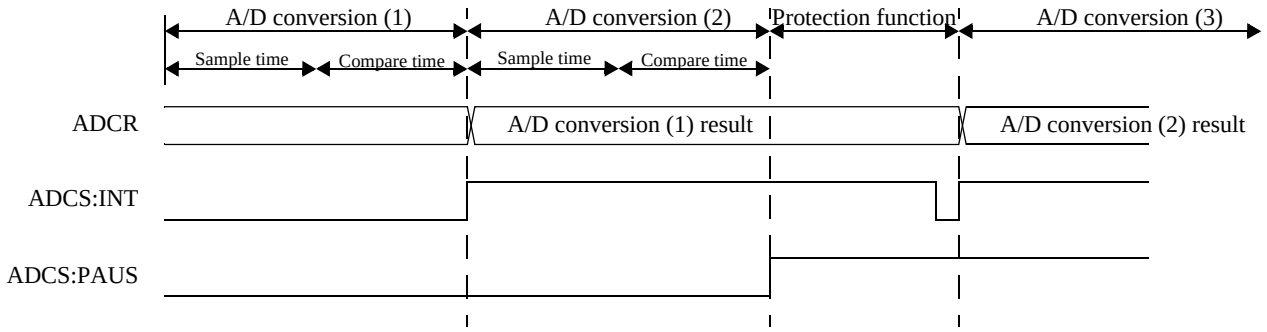
The A/D conversion data protection function is to prevent unexpectedly failing to take some A/D conversion data.

The A/D converter has one Data Register (ADCR) for storing converted data and one successive approximation circuit for storing the data currently being A/D converted. During A/D conversion execution, the A/D converter stores conversion data by one bit in the successive approximation circuit, and when the A/D conversion is completed, it stores the A/D conversion result in the Data Register.

The A/D converter operations with/without using the A/D conversion data protection function are shown below.

- When the interrupt enable bit (ADCS:INTE) is not set, the data protection function will be disabled. In this case, if A/D conversion is performed continuously, the A/D converter stores the conversion result in the Data register (ADCR) every time the converter ends the conversion. This means that the latest conversion data is always stored.
- When the interrupt enable bit (ADCS:INTE) is set, the data protection function will be valid. If A/D conversion is performed continuously in this state, the interrupt request flag bit is set to ADCS:INT=1 when the first conversion is completed. Then, the next A/D conversion is performed. If the conversion ends with ADCS:INT=1 state, the A/D converter enters "pause state" immediately before it transfers the conversion result from the successive approximation circuit to the Data register (ADCR) to prevent the conversion data from being overwritten. The pause flag bit (ADCS:PAUS) in the Control status register is set to "1" at this time. If the interrupt request flag bit (ADCS:INT) is cleared to "0" during a pause state, the data stored in the successive approximation circuit will be transferred to the Data register (see Figure 18-12).

Figure 18-12. Operation of A/D conversion data protection function




### 18.5.2 Data protection function for the case A/D conversion result is read by CPU

- After an analog input is A/D converted, the interrupt request flag bit (ADCS:INT) in the Control status register is set to "1" when the A/D conversion result is stored in the Data register (ADCR).
- If the interrupt request flag bit (ADCS:INT) that was set at the completion of the previous A/D conversion remains set at the end of the next A/D conversion, the A/D conversion operation pauses to protect the previous data from beeing immediately overwritten by new data to the Data register, if interrupt request is enabled (ADCS:INTE=1).
- Since interrupt request in the Control status register is enabled (ADCS:INTE=1), an interrupt request is generated when the INT bit is set. When the INT bit is cleared, pause state of the A/D conversion operation will be cancelled.
- If A/D conversion is performed continuously, the A/D converter starts the next A/D conversion operation. At this time, the pause flag bit (ADCS:PAUS) is not cleared to "0" automatically. In order to clear, write "0" to this bit.

**Note:**

If interrupt request output is disabled (ADCS:INTE=0) during a pause state, this may start A/D conversion and rewrite the data in the Data register.
If A/D conversion is performed continuously for more than once, read the data stored in the Data register before clearing the interrupt request flag bit (ADCS:INT). If the interrupt request flag bit (ADCS:INT) is cleared before reading the data stored in the Data register when A/D conversion is in a pause state, the first stored conversion data will be overwritten by the next conversion data and destroyed.

### 18.5.3 Data protection function for the case A/D conversion result is transferred by DMA

After an A/D conversion, if the next A/D conversion ends while the A/D conversion result is being transferred from the Data register using the DMA, the A/D conversion operation pauses to protect data immediately before it overwrites new data to the Data register. When an A/D conversion operation pauses, the pause flag bit (ADCS: PAUS) in the Control status register is set to "1".

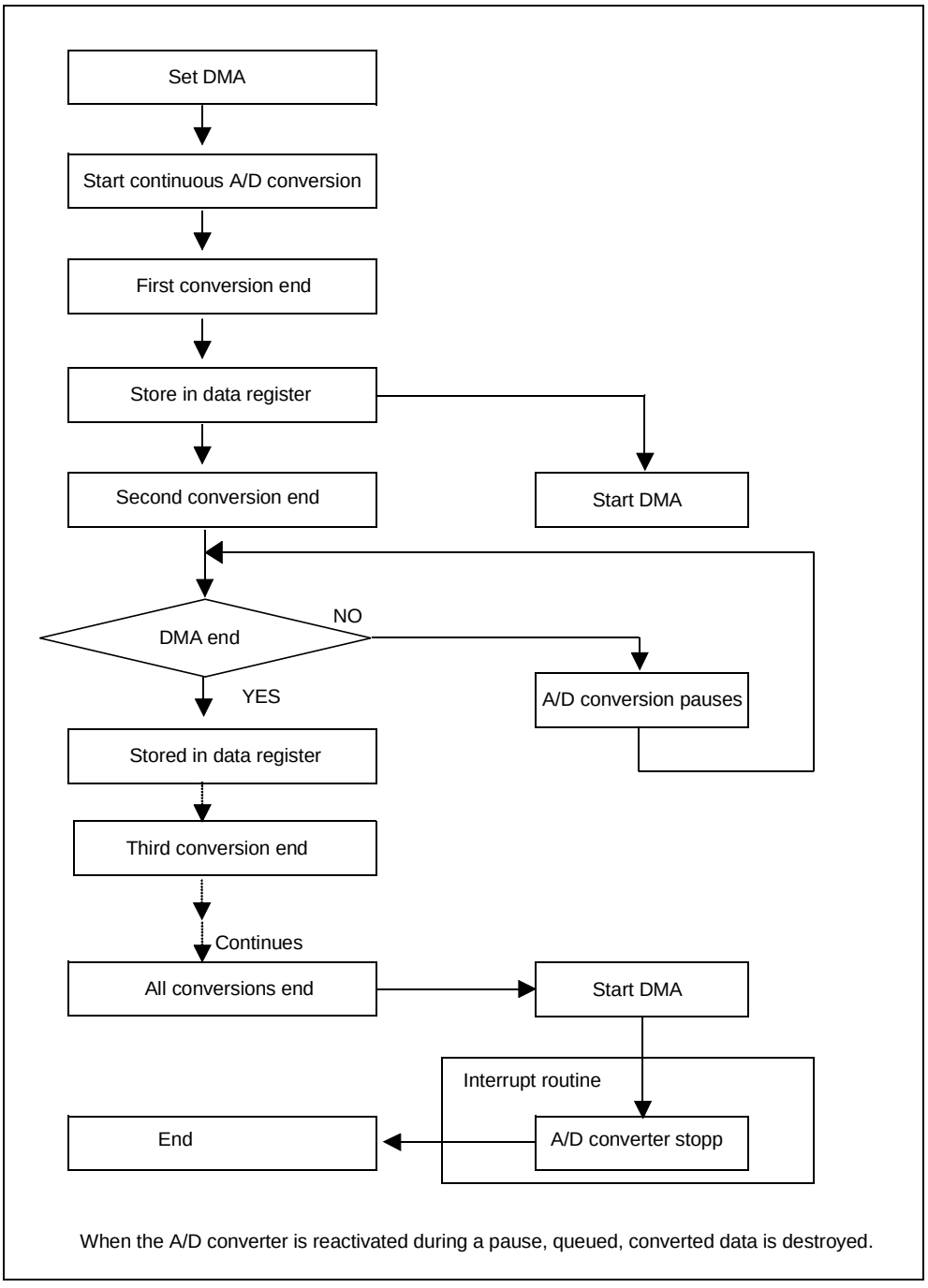
When a transfer of an A/D conversion result is completed by using the DMA, a pause state of A/D conversion will be cancelled. If A/D conversion is performed continuously, the A/D conversion operation is restarted. At this time, the pause flag bit (ADCS:PAUS) is not cleared to "0" automatically. In order to clear, write "0" to this bit.

**Note:**

Do not clear the interrupt request flag bit (ADCS:INT=0) from CPU when an A/D conversion result is transferred by using DMA. Otherwise the data in the Data register in transfer process may be rewritten.
Do not disable interrupt request output when an A/D conversion result is transferred using the DMA. If interrupt request output is disabled (ADCS:INTE=0) during a pause state, this may start A/D conversion and rewrite the data in the A/D data register.
Do not restart when an A/D conversion result is transferred using the DMA. If the converter is restarted during a conversion pause state, the conversion result may be destroyed.

## 18.5.4 Example of flow of Conversion Data Protection Function (when DMA is used)

Figure 18-13. Example of flow of Conversion Data Protection Function (when DMA is used)




## 18.5.5 Cautions

To select the external trigger or the internal timer as the activation causes of A/D converter, the A/D start source select bits (STS1 and STS0 bits) of ADCSH register are used.In this case, ensure that the input value of an external trigger or of the internal timer are “inactive”. If the values are “active”, A/D conversion may start operating immediately.

When setting the STS1 bit and STS0 bit, always set the ADTG pin to “1” (input) and set the internal timer (reload timer 1) to “0” (output).

# 19. Alarm Comparator

This chapter explains the functions and operations of the Alarm Comparator.

19.1 Outline of Alarm Comparator

19.2 Alarm Comparator Registers

19.3 Alarm Comparator operating modes

## 19.1 Outline of Alarm Comparator

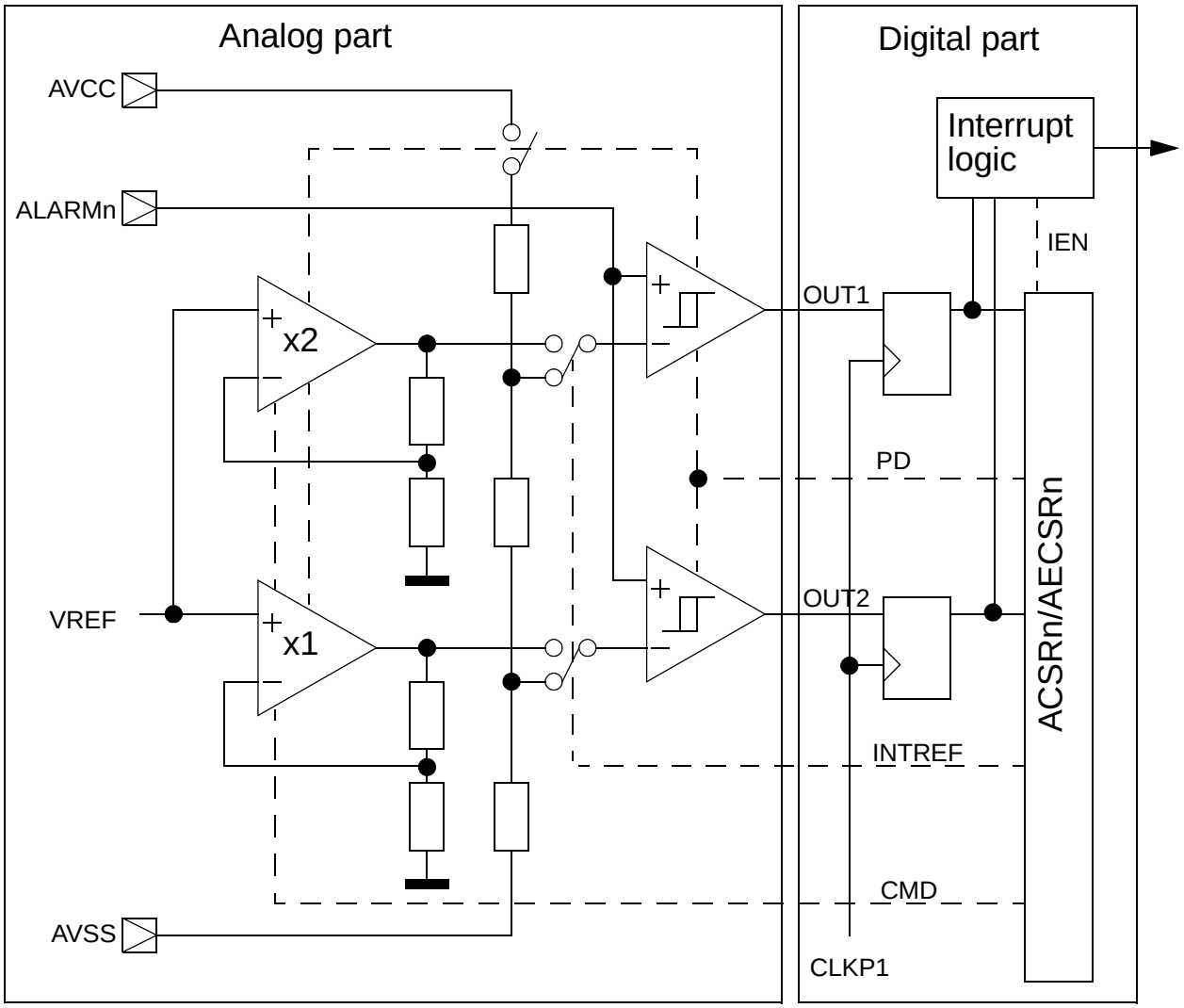
This chapter provides an overview of the Alarm Comparator (also called Under/Overflow voltage Detection), describes the register structure, modes and the operation of the Alarm Comparator

### Outline of Alarm Comparator

- The Alarm Comparator consists of an analog and a digital part.
- The analog part compares the input voltage with AVCC, AVSS or an internal reference voltage VREF and generates digital output signals OUT1 and OUT2 for the digital part.
- If the voltage level at the input pin is higher than the lower threshold, OUT2 switches to high state.
- If the voltage level at the input pin is higher than the upper threshold, OUT1 switches to high state.
- The digital part uses OUT1 and OUT2 and generates an interrupt to CPU if the voltage is higher than the upper threshold or lower than the lower threshold.
- The Alarm Comparator can be configured and controlled by the ACSRn and AECSRn registers.
- The Alarm Comparator can be switched between fast and slow comparison mode to reduce power consumption
- The Alarm Comparator can be switched to power down mode.

### Block diagram of Alarm Comparator

Figure 19-1. Block diagram of Alarm Comparator




**Note:**

- The suffix ’n’ denotes the number of the Alarm Comparator module.

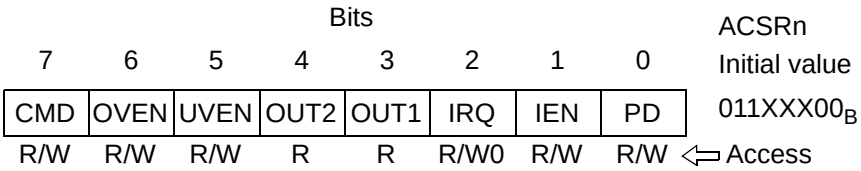
## 19.2 Alarm Comparator Registers

The Alarm Comparator has the following two registers:

- Alarm Comparator Control/Status register (ACSRn)
- Alarm Comparator Extended Control/Status register (AECSRn)

## 19.2.1 Alarm Comparator Control/Status register (ACSRn)

Figure 19-2. Structure of Alarm Comparator control/status register




**Note:**

- The suffix 'n' denotes the number of the Alarm Comparator module.

### Bit 7: CMD (Comparison Mode)

| | |
|---|---|
| 0 | Slow comparison mode (less power consumption). [Initial value] |
| 1 | Fast comparison mode (higher power consumption). |

See Datasheet for Power consumption and comparison time of the Alarm Comparator.

### Bit 6: OVEN (Overvoltage Enable)

| | |
|---|---|
| 0 | No interrupt in case of overvoltage |
| 1 | Interrupt enabled in case of overvoltage [Initial value] |

### Bit 5: UVEN (Undervoltage Enable)

| | |
|---|---|
| 0 | No interrupt in case of undervoltage |
| 1 | Interrupt enabled in case of undervoltage [Initial value] |

### Bit 4: OUT2 (Synchronized output of Alarm Comparator UV output)

| | |
|---|---|
| 0 | Analog input voltage < $V_{EVTL}$ when AECSRn:INTREF = 0 or<br>analog input voltage < $V_{IVTL}$ when AECSRn:INTREF = 1 |
| 1 | Analog input voltage > $V_{EVTL}$ when AECSRn:INTREF = 0 or<br>analog input voltage > $V_{IVTL}$ when AECSRn:INTREF = 1 |

### Bit 3: OUT1 (Synchronized output of Alarm Comparator OV output)

| | |
|---|---|
| 0 | Analog input voltage < $V_{EVTH}$ when AECSRn:INTREF = 0 or<br>analog input voltage < $V_{IVTH}$ when AECSRn:INTREF = 1 |
| 1 | Analog input voltage > $V_{EVTH}$ when AECSRn:INTREF = 0 or<br>analog input voltage > $V_{IVTH}$ when AECSRn:INTREF = 1 |

### Bit 2: IRQ (Interrupt request)

| | |
|---|---|
| 0 | No under- or overvoltage condition detected |
| 1 | Under- or overvoltage condition detected |

Write "0" to IRQ to clear an interrupt request. Writing "1" to IRQ has no effect. "1" is read from this bit when a read-modify-write instruction is used.

### Remark:

When IRQ is cleared at the same time, the Alarm comparator requests an interrupt, the interrupt request has higher priority and IRQ is set to "1". To clear an interrupt caused by a persistent out of voltage range condition, first disable the undervoltage and/or overvoltage interrupt generation (OVEN = 0 and/or UVEN = 0), then clear the IRQ bit separately.

### Bit 1: IEN (Interrupt enable)

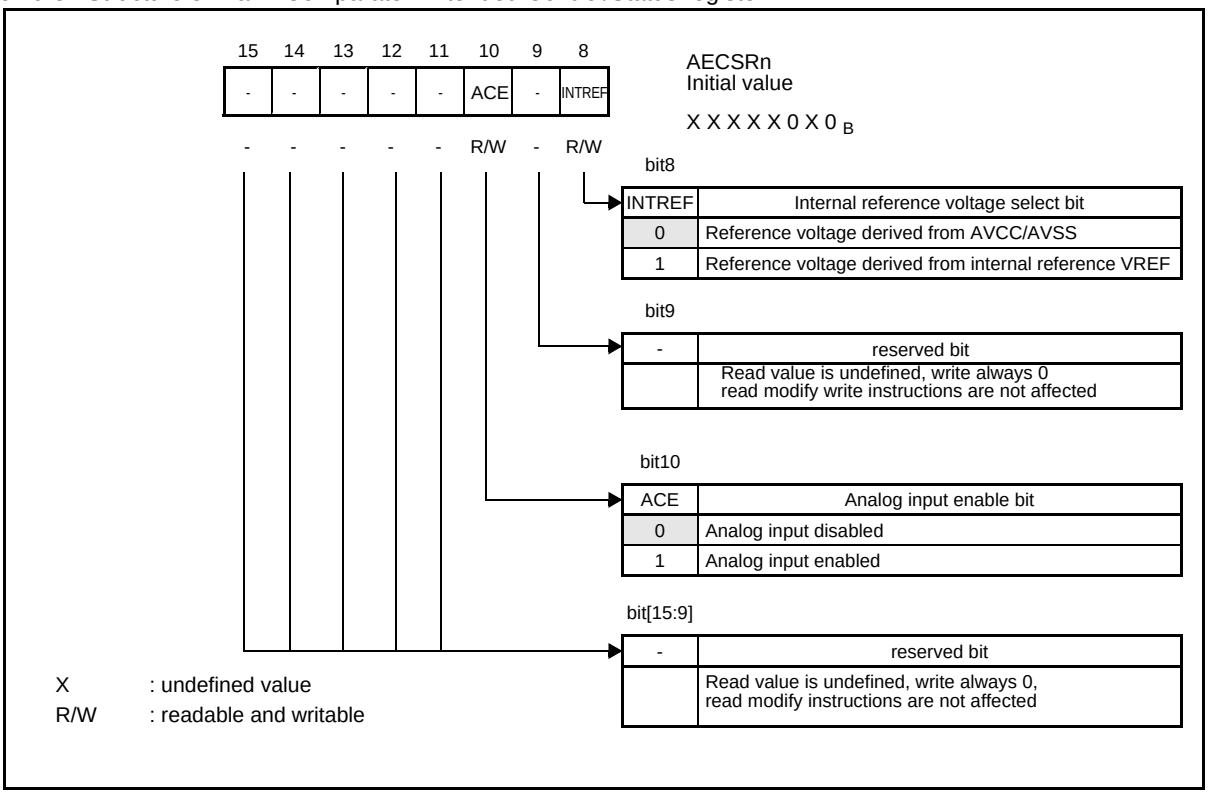
| | |
|---|---|
| 0 | Interrupt assertion disabled [Initial value] |
| 1 | Interrupt assertion enabled |

### Bit 0: PD (Power down)

| | |
|---|---|
| 0 | Run mode (analog part) [Initial value] |
| 1 | Power down mode (analog part) |

## 19.2.2 Alarm Comparator Extended Control/Status Register (AECSRn)

Figure 19-3. Structure of Alarm Comparator Extended Control/Status register



### Bit 10: ACE (Analog input enable)

| | |
|---|---|
| 0 | Analog input disabled |
| 1 | Analog input enabled |

### Bit 8: INTREF (Internal Reference voltage select)

| | |
|---|---|
| 0 | AVCC/AVSS is selected as Reference voltage. [Initial value] |
| 1 | Internal reference voltage VREF is selected. |

## 19.3 Alarm Comparator operating modes

The Alarm Comparator circuit can operate in interrupt or polling mode. The internal interrupt logic will detect each interrupt event independent of setting of the IEN bit.

### Interrupt Mode (IEN=1)

The following truth table describes the valid interrupt events.

Table 19-1. Valid interrupt events

| INTREF | OUT2 | OUT1 | IRQ | Analog input voltage range |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | Vin > $V_{EVTH}$ (overvoltage) |
| 0 | 1 | 0 | 0 | $V_{EVTL}$ < Vin < $V_{EVTH}$ (normal operation) |
| 0 | 0 | 0 | 1 | Vin < $V_{EVTL}$ (undervoltage) |
| 1 | 1 | 1 | 1 | Vin > $V_{IVTH}$ (overvoltage) |
| 1 | 1 | 0 | 0 | $V_{IVTL}$ < Vin < $V_{IVTH}$ (normal operation) |
| 1 | 0 | 0 | 1 | Vin < $V_{IVTL}$ (undervoltage) |

The interrupt Bit IRQ will be set with the next positive transition of CLKP1 after detecting an interrupt event. If IEN=1 this will create an interrupt request to the CPU. In order to determine the reason for the asserted interrupt - if both interrupts are enabled - it is necessary to read the ACSR register immediately inside the interrupt service routine. OUT2 and OUT1 always contain the actual status of the voltage comparator outputs, i.e. the interrupt trigger event will not be stored.

### Polling Mode (IEN=0)

The IRQ register bit will be set by an active interrupt event and can be reset by writing to the ACSR register. The ACSR can be polled continuously in order to monitor the input voltage which is fed to the Alarm Comparator's voltage comparator inputs.

### Setting and Resetting of IRQ-Flagbit

The IRQ bit of the ACSR can be reset to zero by writing a "0" to it. Writing an "1" to the IRQ bit of ACSR has no effect. IRQ can only be set to "1" by hardware, i.e. by the outputs of the comparator circuits. IRQ will remain active as long as an active interrupt status is detected, even if a "0" is written to it.

A bitset command performed on the ACSR will result in a RMW access. Every read access during performing a RMW command will return a "1" for the IRQ flag to the CPU. That avoids any loss in detecting interrupt events due to software setting of IRQ-Flag Bit.

### Power Down Modes and Comparison modes of the Alarm Comparator

The different operation modes of the Alarm Comparator circuit are described in the following table:

Table 19-2. Alarm Comparator operation modes

| STOP | TIMER | SLEEP | Digital part |
|---|---|---|---|
| 0 | 0 | 0 | operating |
| 0 | 0 | 1 | operating |
| 0 | 1 | 0 | stopped |
| 1 | 0 | 0 | stopped |

| ACSR:PD | Analog part |
|---|---|
| 0 | run mode |
| 1 | power down |

Precaution: The outputs of the Alarm Comparator (analog parts) will remain undefined for at least 3 us after power on and also after reentering the run mode. Disable interrupt generation for the time in which the analog part of the Alarm Comparator is not stable.

If the Alarm Comparator shall be disabled in any MCU standby mode (STOP, TIMER, SLEEP), the ACSR:PD bit must be set before the MCU standby mode is set in the SMCR register.

Use the following sequence for switching the Alarm Comparator to power down mode:

1. Disable interrupt generation (ACSRn:IEN = 0)
2. Power down the Alarm Comparator (ACSRn:PD = 1)

Use the following sequence for switching the Alarm Comparator from power down mode to run mode:

1. Switch Alarm Comparator from power down mode to run mode (ACSRn:PD = 0)
2. Wait 3μs
3. Enable interrupts (ACSRn:IEN = 1) if required or use Alarm Comparator in polling mode.

# 20. USART

This chapter explains the functions and operation of the LIN USART.

20.1 Overview of USART

20.2 Configuration of USART

20.3 USART Pins

20.4 USART Registers

20.5 USART Interrupts

20.6 USART Baud Rates

20.7 Operation of USART

20.8 Notes on Using USART

## 20.1 Overview of USART

The USART with LIN (Local Interconnect Network) - Function is a general-purpose serial data communication interface for performing synchronous or asynchronous communication with external devices. The USART provides bidirectional communication function (normal mode), master-slave communication function (multiprocessor mode in master/slave systems), and special features for LIN-bus systems (working both as master or as slave device).

### USART functions

The USART is a general-purpose serial data communication interface for transmitting serial data to and receiving data from another CPU and peripheral devices. It has the functions listed in Table 20-1.

Table 20-1. USART functions (Sheet 1 of 2)

| Item | Function |
|---|---|
| Data buffer | Full-duplex |
| Serial Input | 5 times oversampling in asynchronous mode |
| Transfer mode | • Clock synchronous (start-stop synchronization and start-stop-bit-option)<br>• Clock asynchronous (using start-, stop-bits) |
| Baud rate | • A dedicated baud rate generator is provided, which consists of a 16-bit-reload counter<br>• An external clock can be input and also be adjusted by the reload counter |
| Data length | • 7 bits (not in synchronous or LIN mode)<br>• 8 bits |
| Signal mode | Non-return to zero (NRZ) |
| Start bit timing | Clock synchronization to the falling edge of the start bit in asynchronous mode |

Table 20-1. USART functions (Sheet 2 of 2)

| Item | Function |
|---|---|
| Reception error detection | • Framing error<br>• Overrun error<br>• Parity error (Not supported in Mode 1) |
| Interrupt request | • receive interrupt (reception complete, reception error detect, LIN-Synch-break detect)<br>• Transmission interrupt (transmission data empty)<br>• Interrupt request to ICU (LIN synch field detection) transmission and reception support for DMA function. |
| Master-slave communication function (multiprocessor mode) | One-to-n communication (one master to n slaves)<br>(This function is supported both for master and slave system). |
| Synchronous mode | Function as Master- or Slave-USART |
| Transceiving pins | Direct access possible |
| LIN bus options | • Operation as master device<br>• Operation as slave device<br>• Generation of LIN-Sync-break<br>• Detection of LIN-Sync-break<br>• ICUs can be used to measure the bit time of detected start/stop edges in the LIN-Sync-field. See section Input Capture Unit Source Select for LIN-USART on page 27 for the assignement of ICU to the Sync field signal of the LIN-USART. |
| Synchronous serial clock | The synchronous serial clock can be output continuously on the SCK pin for synchronous communication with start & stop bits |
| Clock delay option | Special synchronous Clock Mode for delaying clock (useful for SPI) |

## USART operation modes

The USART operates in four different modes, which are determined by the MD0- and the MD1-bit of the Serial mode register (SMRn). Mode 0 and 2 are used for bidirectional serial communication, mode 1 for master/slave communication and mode 3 for LIN master/slave communication.

Table 20-2. USART operation modes

| Operation mode | | Data length | | Synchronization of mode | Length of stop bit | data bit direction *1 |
|---|---|---|---|---|---|---|
| | | parity disabled | parity enabled | | | |
| 0 | normal mode | 7 or 8 | | asynchronous | 1 or 2 | L/M |
| 1 | multiprocessor | 7 or 8 + 1 *2 | -- | asynchronous | 1 or 2 | L/M |
| 2 | normal mode | 8 | | synchronous | 0, 1 or 2 | L/M |
| 3 | LIN mode | 8 | -- | asynchronous | 1 | L |

*1: means the data bit transfer format: LSB or MSB first.

*2: "+1" means the indicator bit of the address/data selection in the multiprocessor mode, instead of parity.

### Note:

Mode 1 operation is supported both for master or slave operation of the USART in a master-slave connection system. In Mode 3 the USART function is locked to 8N1-Format, LSB first.

If the mode is changed, USART cuts off all possible transmission or reception and awaits then new action.

The MD1 and MD0 bit of the Serial Mode Register (SMRn) determine the operation mode of the USART as shown in the following table:

Table 20-3. Mode bit setting

| MD1 | MD0 | Mode | Description |
|---|---|---|---|
| 0 | 0 | 0 | Asynchronous (normal mode) |
| 0 | 1 | 1 | Asynchronous (multiprocessor mode) |
| 1 | 0 | 2 | Synchronous (normal mode) |
| 1 | 1 | 3 | Asynchronous (LIN mode) |

## 20.2 Configuration of USART

This section provides a short overview on the building blocks of USART.

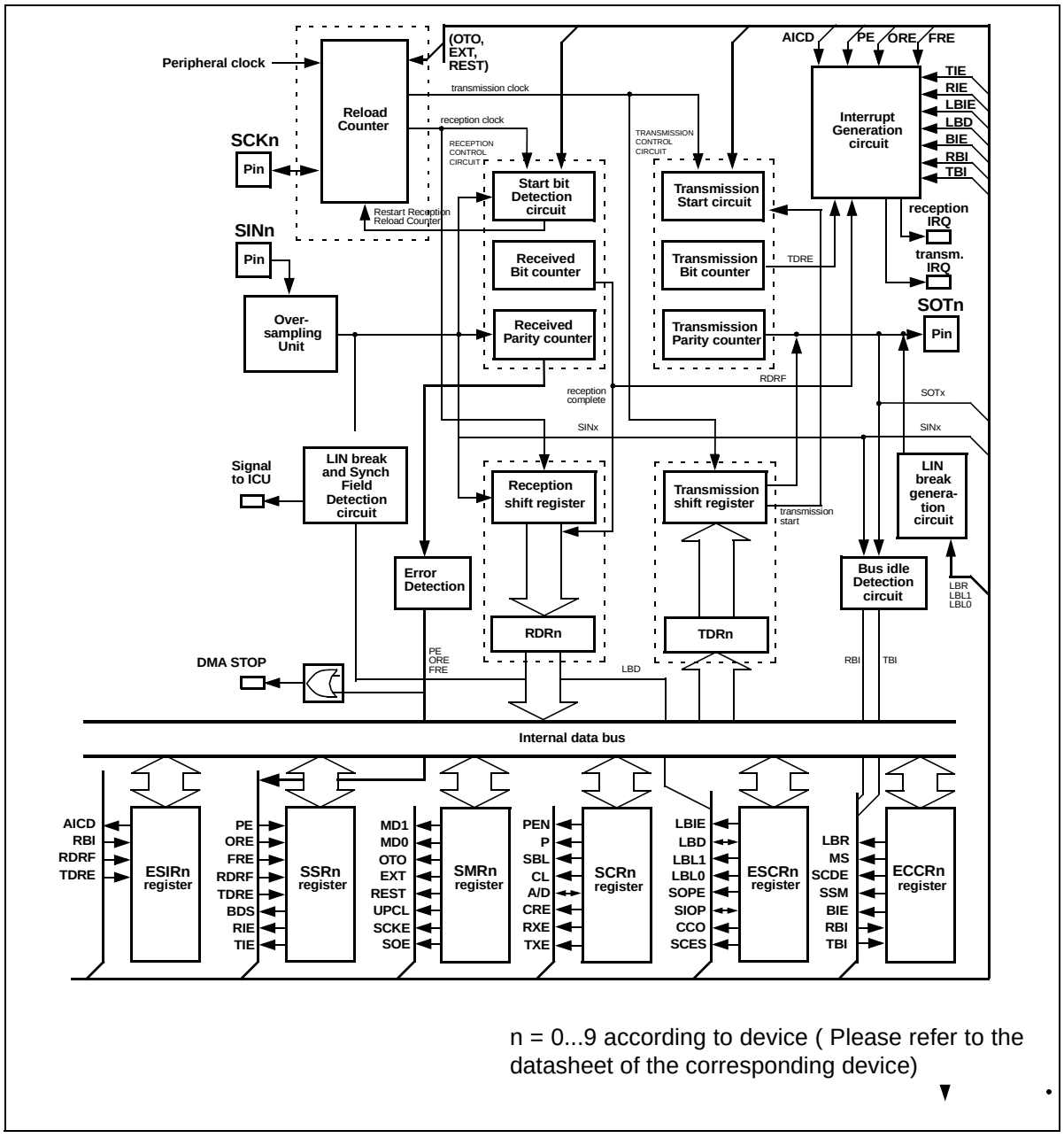
### 20.2.1 Block diagram of USART

USART consists of the following blocks:

- Reload Counter
- Reception Control Circuit
- Reception Shift Register
- Reception Data Register (RDRn)
- Transmission Control Circuit
- Transmission Shift Register
- Transmission Data Register (TDRn)
- Error Detection Circuit
- Oversampling Unit
- Interrupt Generation Circuit
- LIN Synch Break/Synch Field Detection
- Bus Idle Detection Circuit
- LIN-USART Serial Mode Register (SMRn)
- Serial Control Register (SCRn)
- Serial Status Register (SSRn)
- Extended Com. Contr. Reg. (ECCRn)
- Extended Status/Contr. Reg. (ESCRn)
- Extended Serial Interrupt Register (ESIRn)

The suffix "n" denotes the USART number.

Figure 20-1. Block diagram of USART




## Reload Counter

The reload counter functions as the dedicated baud rate generator. It can select external input clock or internal clock for the transmitting and receiving clocks. The reload counter has a 16-bit register for the reload value. The actual count of the transmission reload counter can be read via the BGRn.

## Reception Control Circuit

The reception control circuit consists of a received bit counter, start bit detection circuit, and received parity counter. The received bit counter counts reception data bits. When reception of one data item for the specified data length is complete, the received bit counter sets the reception data register full flag. The start bit detection circuit detects start bits from the serial

input signal and sends a signal to the reload counter to synchronize it to the falling edge of these start bits. The reception parity counter calculates the parity of the reception data.

### Reception Shift Register

The reception shift register fetches reception data input from the SINn pin, shifting the data bit by bit. When reception is complete, the reception shift register transfers receive data to the RDRn register.

### Reception Data Register (RDRn)

This register retains reception data. Serial input data is converted and stored in this register.

### Transmission Control Circuit

The transmission control circuit consists of a transmission bit counter, transmission start circuit, and transmission parity counter. The transmission bit counter counts transmission data bits. When the transmission of one data item of the specified data length is complete, the transmission bit counter sets the Transmission data register full flag. The transmission start circuit starts transmission when data is written to TDRn. The transmission parity counter generates a parity bit for data to be transmitted if parity is enabled.

### Transmission Shift Register

The transmission shift register transfers data written to the TDRn register to itself and outputs the data to the SOTn pin, shifting the data bit by bit.

### Transmission Data Register (TDRn)

This register sets transmission data. Data written to this register is converted to serial data and output.

### Error Detection Circuit

The error detection circuit checks if there was any error during the last reception. If an error has occurred it sets the corresponding error flags.

### Oversampling Unit

The oversampling unit oversamples the incoming data at the SINn pin for five times. It is switched off in synchronous operation mode.

### Interrupt Generation Circuit

The interrupt generation circuit administers all cases of generating a reception or transmission interrupt. If a corresponding enable flag is set and an interrupt case occurs the interrupt will be generated immediately.

### LIN synch Break and Synchronization Field Detection Circuit

The LIN break and LIN synchronization field detection circuit detects a LIN break, if a LIN master node is sending a message header. If a LIN break is detected a special flag bit is generated. The first and the fifth falling edge of the synchronization field is recognized by this circuit by generating an internal signal for the Input Capture Unit to measure the actual serial clock time of the transmitting master node.

### LIN Synch Break Generation Circuit

The LIN break generation circuit generates a LIN break of a determined length.

### Bus Idle Detection circuit

The bus idle detection circuit recognizes if neither reception nor transmission is going on. In this case, the circuit generates the special flag bits TBI and RBI.

### LIN-USART Serial Mode Register (SMRn)

This register performs the following operations:

- Selecting the LIN-USART operation mode
- Selecting a clock input source
- Selecting if an external clock is connected “one-to-one” or connected to the reload counter
- Resetting dedicated reload timer
- Resetting the LIN-USART (preserving the settings of the registers)
- Specifying whether to enable serial data output to the corresponding pin
- Specifying whether to enable clock output to the corresponding pin

### Serial Control Register (SCRn)

This register performs the following operations:

- Specifying whether to provide parity bits
- Selecting parity bits
- Specifying a stop bit length
- Specifying a data length
- Selecting a frame data format in mode 1
- Clearing the error flags
- Specifying whether to enable transmission
- Specifying whether to enable reception

### Serial Status Register (SSRn)

This register performs the following functions:

- Indicating status of receive/transmit operations and errors
- Specifying LSB first or MSB first
- Receive interrupt enable/disable
- Transmit interrupt enable/disable

### Extended Status/Control Register (ESCRn)

This register performs the following functions:

- LIN synch break interrupt enable/disable
- Indicating LIN synch break detection
- Specifying LIN synch break length
- Directly accessing SINn and SOTn pins
- Specifying continuous clock output operation
- Specifying sampling clock edge

### Extended Communication Control Register (ECCRn)

This register performs the following functions:

- Indicating bus idle state
- Specifying synchronous clock
- Specifying LIN synch break generation

### Extended Serial Interrupt Register (ESIRn)

This register performs the following function:

- Change handling of interrupts to enable usage together with DMA

## 20.3 USART Pins

This section describes the USART pins and provides a pin block diagram.

### USART pins

The USART pins are shared with general purpose ports. Table 20-4 lists the pin functions, I/O formats, and settings required to use the USART.

Table 20-4. USART pins

| Pin name | Pin function | I/O format | Pull-up | Standby control | Setting required to use pin |
|---|---|---|---|---|---|
| Pxx_i/SINn | Port I/O or serial data input | Please refer to I/O Ports on page 287for how to select available formats | Programmable pull-up resistor.<br>(refer to forI/O Ports on page 287 how to activate the pull-up resistor) | Provided | Set corresponding general purpose port to "input":<br>DDRxx:Di = 0<br>PIERxx:IEi = 1 |
| Pyy_j/SOTn | Port I/O or serial data output | | | | Set output enable mode:<br>(SMRn: SOE = 1) |
| Pzz_k/SCKn | Port I/O or serial clock input/output | | | | When a clock is input, then set corresponding general purpose port to "input":<br>DDRzz:Dk = 0<br>PIERzz:IEk = 1 |
| | | | | | When a clock is output, then set output enable mode:<br>SMRn:SCKE = 1 |

## 20.4 USART Registers

The following figure shows the USART registers.

### USART registers

Figure 20-2. USART registers

bit15 bit8 bit7 bit0

| | |
|---|---|
| SCRn (Serial Control Register) | SMRn (Serial Mode Register) |
| SSRn (Serial Status Register) | RDRn/TDRx (RX, TX Data Register) |
| ESCRn (Extended Status/Control Reg.) | ECCRn (Extended Comm. Contr. Reg.) |
| BGRHn (Baud Rate Generator Reg.) | BGRLn (Baud Rate Generatpr Reg.) |
| | ESIRn (Extended Serial Interrupt Reg.) |

n = 0...9 according to device ( Please refer to the datasheet of the correspond-

### 20.4.1 Serial Control Register (SCRn)

This register specifies parity bits, selects the stop bit and data lengths, selects a frame data format in mode 1, clears the reception error flag, and specifies whether to enable transmission and reception.

### Serial control register (SCRn)

Figure 20-3. Configuration of the serial control register (SCRn)

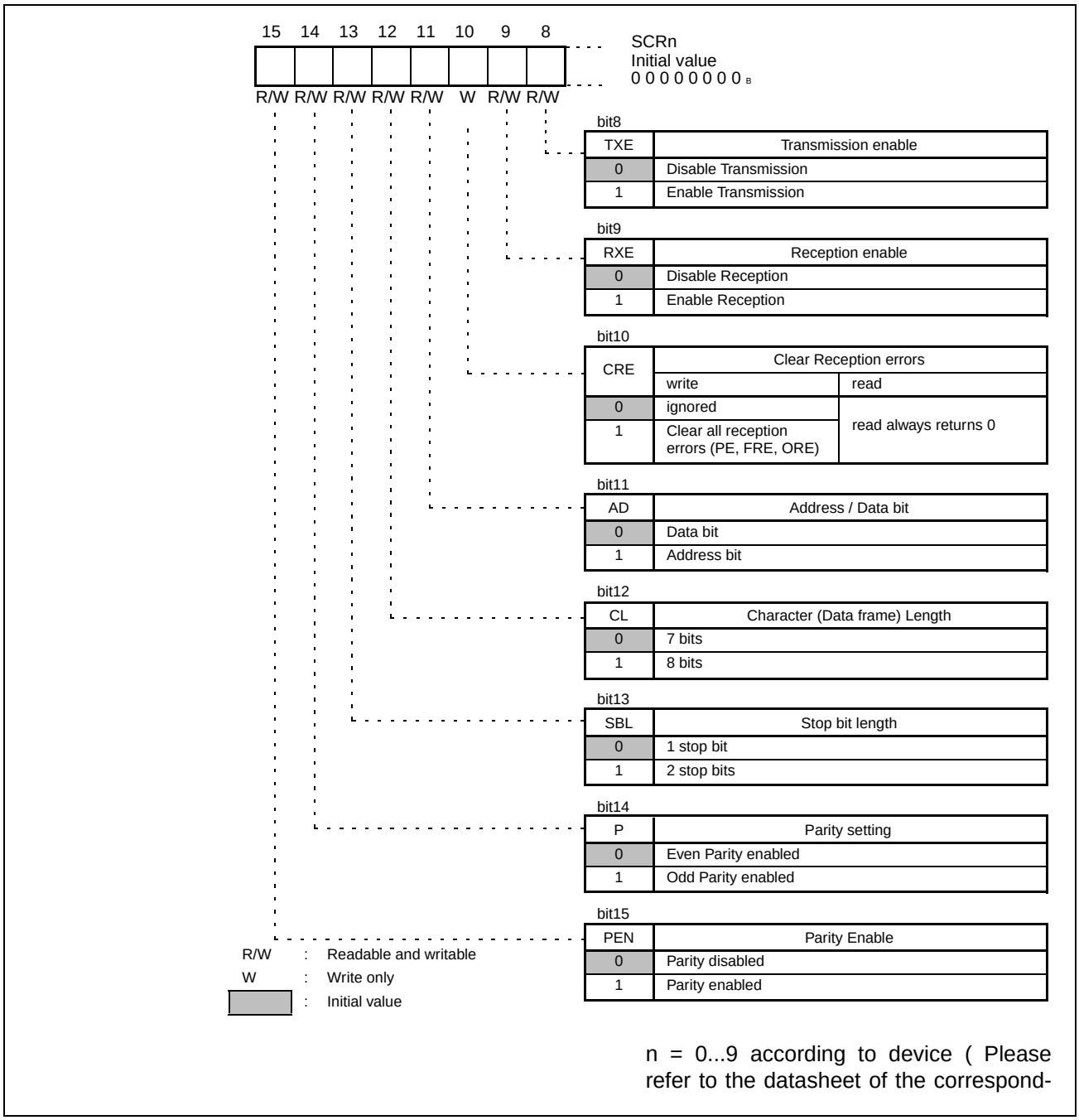


Table 20-5. Functions of each bit of control register (SCRn)

| Bit name | | Function |
|---|---|---|
| bit15 | PEN:<br>Parity enable bit | This bit selects whether to add a parity bit during transmission or detect it during reception.<br>Parity is only provided in mode 0 and in mode 2 if SSM of the ECCRn is selected. This bit is fixed to 0 (no parity) in mode 1 (multiprocessor) and in mode 3 (LIN). |
| bit14 | P:<br>Parity selection bit | When parity is provided and enabled this bit selects even (0) or odd (1) parity. |
| bit13 | SBL:<br>Stop bit length selection bit | This bit selects the length of the stop bit of an asynchronous data frame or a synchronous frame if SSM of the ECCRn is selected. This bit is fixed to 0 (1 stop bit) in mode 3 (LIN). |
| bit12 | CL:<br>Data length selection bit | This bit specifies the length of transmission or reception data. This bit is fixed to 1 (8 bits) in mode 2 and 3. |
| bit11 | AD:<br>Address/Data selection bit | This bit specifies the data format in multiprocessor mode 1. Writing to this bit is provided for a master CPU, reading from it for slave CPU. A 1 indicates an address frame, a 0 indicates a usual data frame.<br>**Note:**<br>Please read the hints about using this bit in Section 20.8 Notes on Using USART. |
| bit10 | CRE:<br>Clear reception error flags bit | This bit clears the FRE, ORE, and PE flag of the Serial Status Register (SSRn) and resets any ongoing reception.<br>Writing a 1 to it clears the error flag.<br>Writing a 0 has no effect.<br>Reading from it always returns 0.<br>**Note:**<br>Clear reception error flags after the receive operation. |
| bit9 | RXE:<br>Reception enable bit | This bit enables/disables LIN-USART reception.<br>If this bit is set to 0, USART disables the reception of data frames.<br>If this bit is set to 1, USART enables the reception of data frames.<br>The LIN synch break detection in mode 3 remains unaffected.<br>**Note:**<br>If reception is disabled (RXE=0) during receiving, it is stopped immediately. In this case, data is not guaranteed. |
| bit8 | TXE:<br>Transmission enable bit | This bit enables/disables LIN-USART transmission.<br>If this bit is set to 0, USART disables the transmission of data frames.<br>If this bit is set to 1, USART enables the transmission of data frames.<br>**Note:**<br>If transmission is disabled (TXE=0) during transmitting, it is stopped immediately. In this case, data is not guaranteed. |

## 20.4.2 Serial mode register (SMRn)

This register selects an operation mode and baud rate clock and specifies whether to enable output of serial data and clocks to the corresponding pin.

### Serial mode register (SMRn)

Figure 20-4. Configuration of the serial mode register (SMRn)

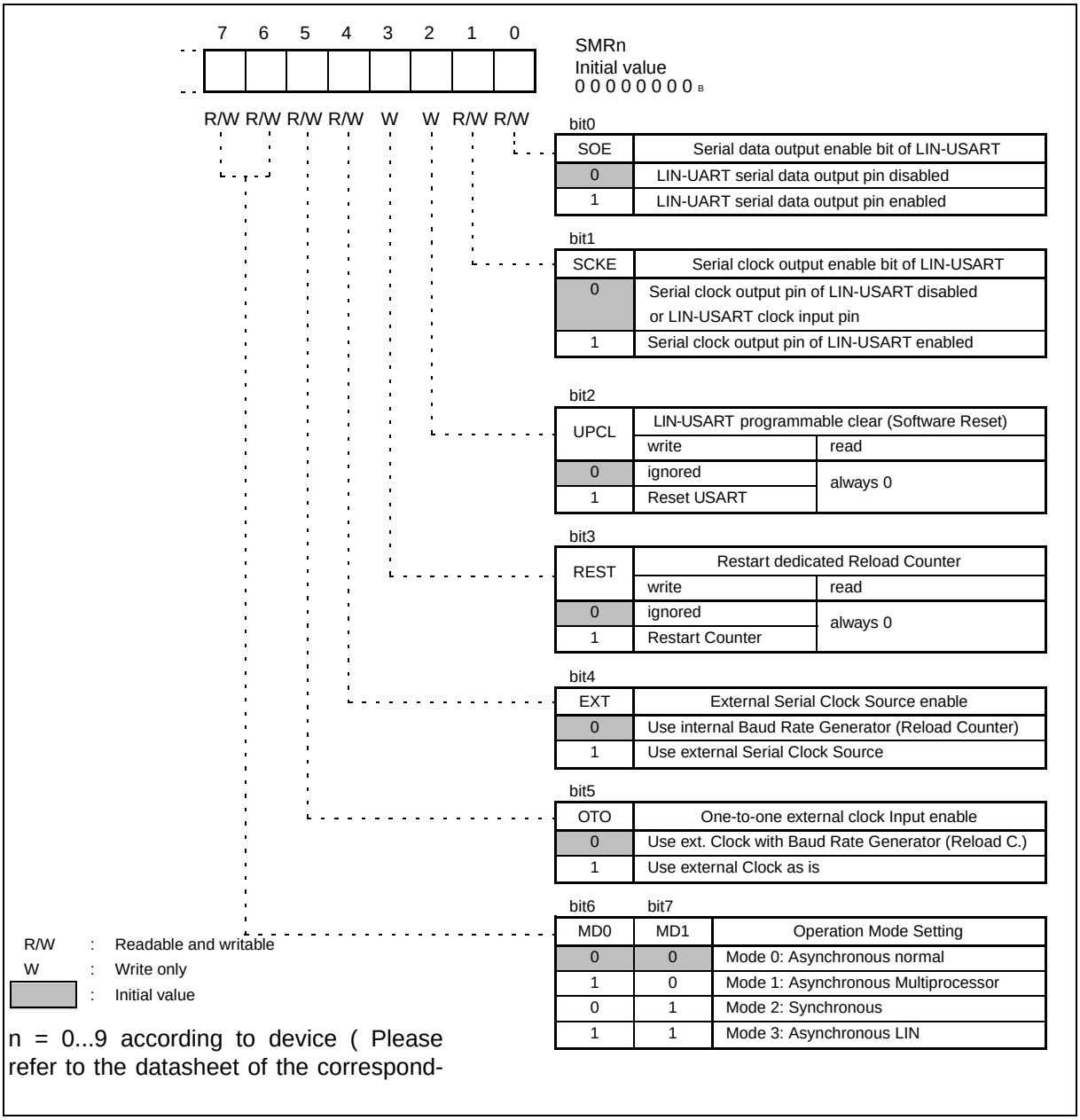


Table 20-6. Bit function of the serial mode register (SMRn)

| Bit name | | Function |
|---|---|---|
| bit7<br>bit6 | MD1 and MD0:<br>Operation mode selection bits | These two bits sets the USART operation mode.<br>**Note:**<br>Changing the mode bit MD1, MD0 causes any ongoing reception or transmission to be aborted. |
| bit5 | OTO:<br>One-to-one external clock selection bit | This bit sets an external clock directly to the LIN-USART's serial clock. This function is used for operating mode 2 (synchronous) slave mode operation. |
| bit4 | EXT:<br>External clock selection bit | This bit executes internal or external clock source for the reload counter |
| bit3 | REST:<br>Restart of transmission reload counter bit | If a 1 is written to this bit the reload counter is restarted. Writing 0 to it has no effect. Reading from this bit always returns 0. |
| bit2 | UPCL:<br>USART programmable clear bit (Software reset) | Writing a 1 to this bit resets LIN-USART immediately. The register settings are preserved. Possible reception or transmission will cut off.<br>All flags (TDRE, RDRF, LBD, PE, ORE, FRE) are cleared and the Reception Data Register (RDRn) contains $00_H$.<br>Writing 0 to this bit has no effect. Reading from it always returns 0.<br>LIN-USART reset should performed after disabling the reception and transmission interrupt enable bits. |
| bit1 | SCKE:<br>Serial clock output enable | • This bit controls the serial clock I/O ports.<br>• When this bit is 0, SCKn pin operates as general purpose I/O port or serial clock input pin. When this bit is 1 and no other peripheral resource uses this pin as output , the pin operates as serial clock output pin and outputs clock in operating mode 2 (synchronous).<br>**Note:**<br>When using SCKn pin as serial clock input (SCKE=0) pin, set the corresponding bit of DDR as input port and set the corresponding bit of PIER = 1. Also, select external clock (EXT = 1) using the external clock selection bit.<br>**Reference:**<br>When the SCKn pin is assigned to serial clock output (SCKE=1) and no other peripheral resource uses this pin as output, it functions as the serial clock output pin (SCKn) regardless of the status of general input-output ports. |
| bit0 | SOE:<br>Serial data output enable bit | • This bit enables or disables the output of serial data.<br>• When this bit is 0, SOTn pin operates as general purpose I/O pin. When this bit is 1and no other peripheral resource uses this pin as output, SOTn pin operates as serial data output pin (SOT).<br>**Reference:**<br>When the output of serial data is enabled (SOE=1) and no other peripheral resource uses this pin as output, SOTn pin functions as serial data output pin (SOT) regardless of the status of general input-output ports. |

## 20.4.3 Serial Status Register (SSRn)

This register checks the transmission and reception status and error status. It also enables and disables the transmission and receive interrupts.

### Serial status register (SSRn)

Figure 20-5. Configuration of the serial status register (SSRn)

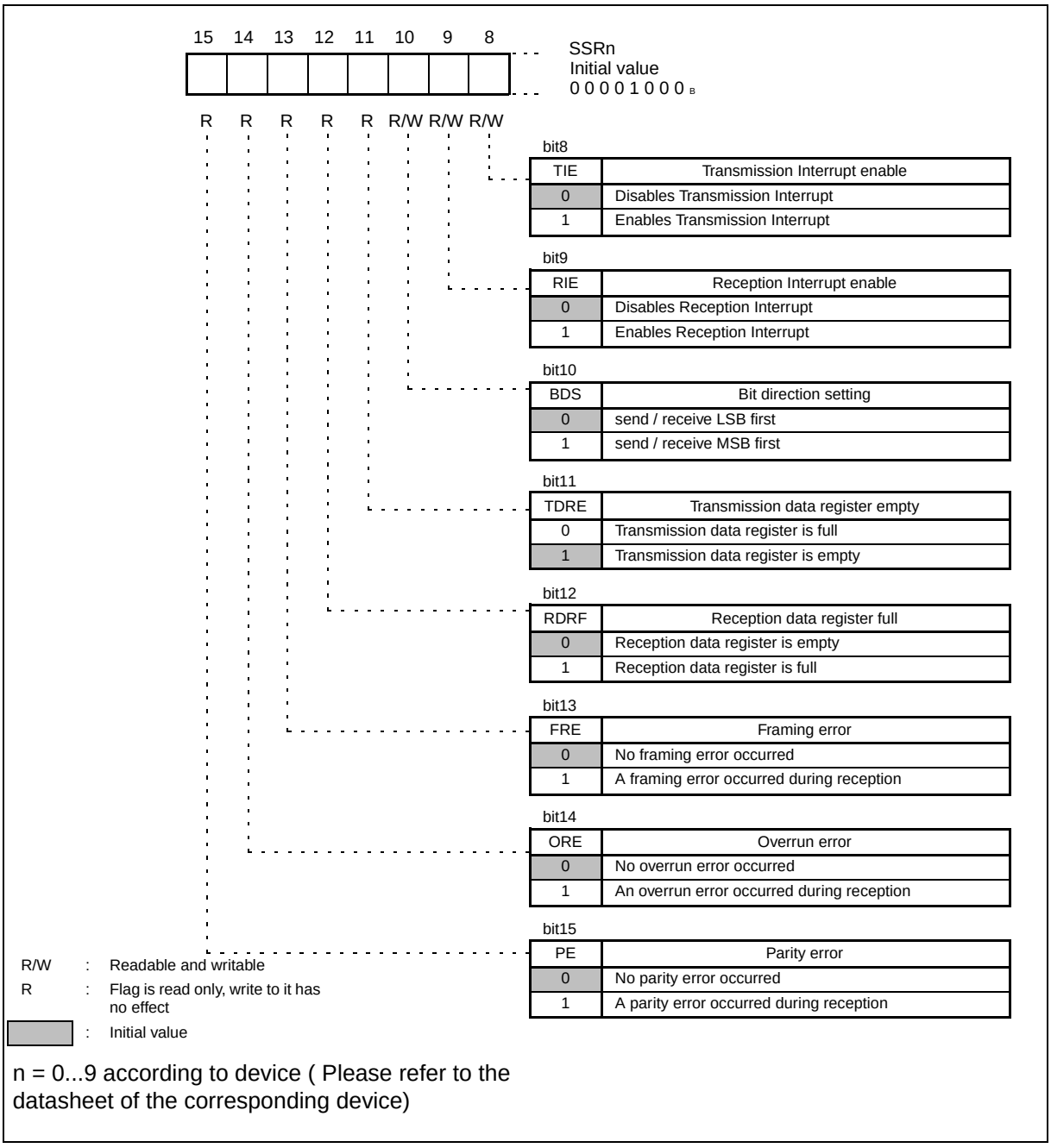


Table 20-7. Functions of each bit of status register (SSRn)

| Bit name | | Function |
|---|---|---|
| bit15 | PE:<br>Parity error flag bit | • This bit is set to 1 when a parity error occurs during reception at PE=1.<br>• It is cleared when 1 is written to the CRE bit of the serial control register (SCRn).<br>• In Mode 3, it is also cleared when ESCRn:LBD = 1.<br>• A receive interrupt request is output when this bit and the RIE bit are 1.<br>• Data in the reception data register (RDRn) is invalid when this flag is set. |
| bit14 | ORE:<br>Overrun error flag bit | • This bit is set to 1 when an overrun error occurs during reception.<br>• It is cleared when 1 is written to the CRE bit of the serial control register (SCRn).<br>• In Mode 3, it is also cleared when ESCRn:LBD = 1.<br>• A receive interrupt request is output when this bit and the RIE bit are 1.<br>• Data in the reception data register (RDRn) is invalid when this flag is set. |
| bit13 | FRE:<br>Framing error flag bit | • This bit is set to 1 when a framing error occurs during reception.<br>• It is cleared when 1 is written to the CRE bit of the serial control register 1 (SCRn).<br>• In Mode 3, it is also cleared when ESCRn:LBD = 1.<br>• A receive interrupt request is output when this bit and the RIE bit are 1.<br>• Data in the reception data register (RDRn) is invalid when this flag is set. |
| bit12 | RDRF:<br>Receive data full flag bit | • This flag indicates the status of the reception data register (RDRn).<br>• This bit is set to 1 when reception data is loaded into RDRn<br>• It is cleared when the reception data register (RDRn) is read.<br>• In Mode 3, it is also cleared when ESCRn:LBD = 1.<br>• A receive interrupt request is output when this bit and the RIE bit are 1.<br>**Note:**<br>ESIR:RDRF has the same behaviour as this bit, but it is not cleared when the reception data register (RDRn) is read. |
| bit11 | TDRE:<br>Transmission data empty flag bit | • This flag indicates the status of the transmission data register (TDRn).<br>• This bit is cleared to 0 when transmission data is written to TDRn and is set to 1 when data is loaded into the transmission shift register and transmission starts.<br>• A transmission interrupt request is generated if both this bit and the TIE bit are 1.<br>• If the LBR bit in the ECCRn register is set to "1" while the TDRE bit is "1", then this bit once changes to "0". After the completion of LIN synch break generator, the TDRE bit charges back to "1".<br>**Note:**<br>This bit is set to 1 (TDRn empty) as its initial value.<br>**Note:**<br>ESIR:TDRE has the same behaviour as this bit, but it is not cleared when transmission data is written to TDRn. |
| bit10 | BDS:<br>Transfer direction selection bit | • This bit selects whether to transfer serial data from the least significant bit (LSB first, BDS=0) or the most significant bit (MSB first, BDS=1).<br>**Note:**<br>The high-order and low-order sides of serial data are interchanged with each other during reading from or writing to the serial data register. If this bit is set to another value after the data is written to the RDRn register, the data becomes invalid.<br>• This bit is fixed to 0 in mode 3 (LIN) |
| bit9 | RIE:<br>receive interrupt request enable bit | • This bit enables/disables the receive interrupt. If any of the RDRF, PE, ORE and FRE bits is set and this bit is "1", then a receive interrupt is signaled to the interrupt controller. |

Table 20-7. Functions of each bit of status register (SSRn)

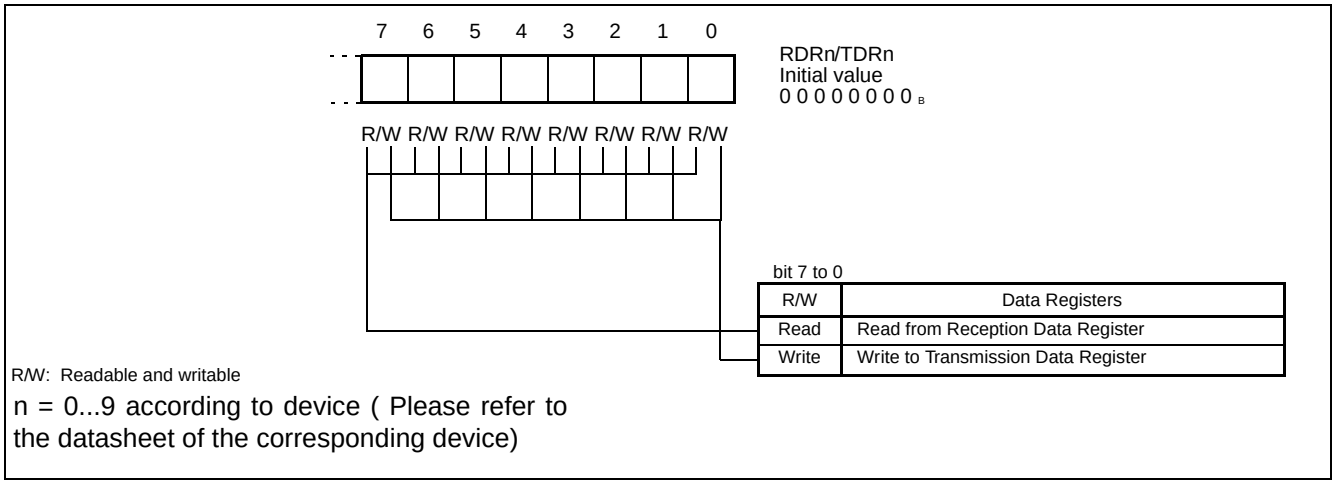
| Bit name | | Function |
|---|---|---|
| bit8 | TIE:<br>Transmission interrupt request enable bit | • This bit enables or disables the transmission interrupt.<br>• A transmission interrupt request is output when this bit and the TDRE bit are 1. |

## 20.4.4 Reception and Transmission Data Register (RDRn/TDRn)

The reception data register (RDRn) holds the received data. The transmission data register (TDRn) holds the transmission data. Both RDRn and TDRn registers are located at the same address.

### *20.4.4.1 Reception and transmission data registers (RDRn/TDRn)*

Figure 20-6. Transmission and reception data registers (RDRn/TDRn)




**Reception:**

RDRn is the register that contains reception data. The serial data signal transmitted to the SINn pin is converted in the shift register and stored there. When the data length is 7 bits, the uppermost bit (D7) contains 0. When reception is complete the data is stored in this register and the reception data full flag bit (SSRn:RDRF) is set to 1. If a receive interrupt request is enabled at this point, a receive interrupt occurs.

Read RDRn when the RDRF bit of the status register (SSRn) is 1. The RDRF bit is cleared automatically to 0 when RDRn is read. Also the receive interrupt is cleared if it is enabled and no error has occurred.

Data in RDRn is invalid when a reception error occurs (SSRn: PE, ORE, or FRE = 1).

**Transmission:**

When data to be transmitted is written to the transmission data register in transmission enable state, it is transferred to the transmission shift register, then converted to serial data, and transmitted from the serial data output terminal (SOTn pin). If the data length is 7 bits, the uppermost bit (D7) is not sent.

When transmission data is written to this register, the transmission data empty flag bit (SSRn:TDRE) is cleared to 0. When transfer to the transmission shift register is complete and starts, the bit is set to 1. When the TDRE bit is 1, the next part of transmission data can be written. If output transmission interrupt requests have been enabled, a transmission interrupt is generated. Write the next part of transmission data when a transmission interrupt is generated or the TDRE bit is 1.

**Note:**

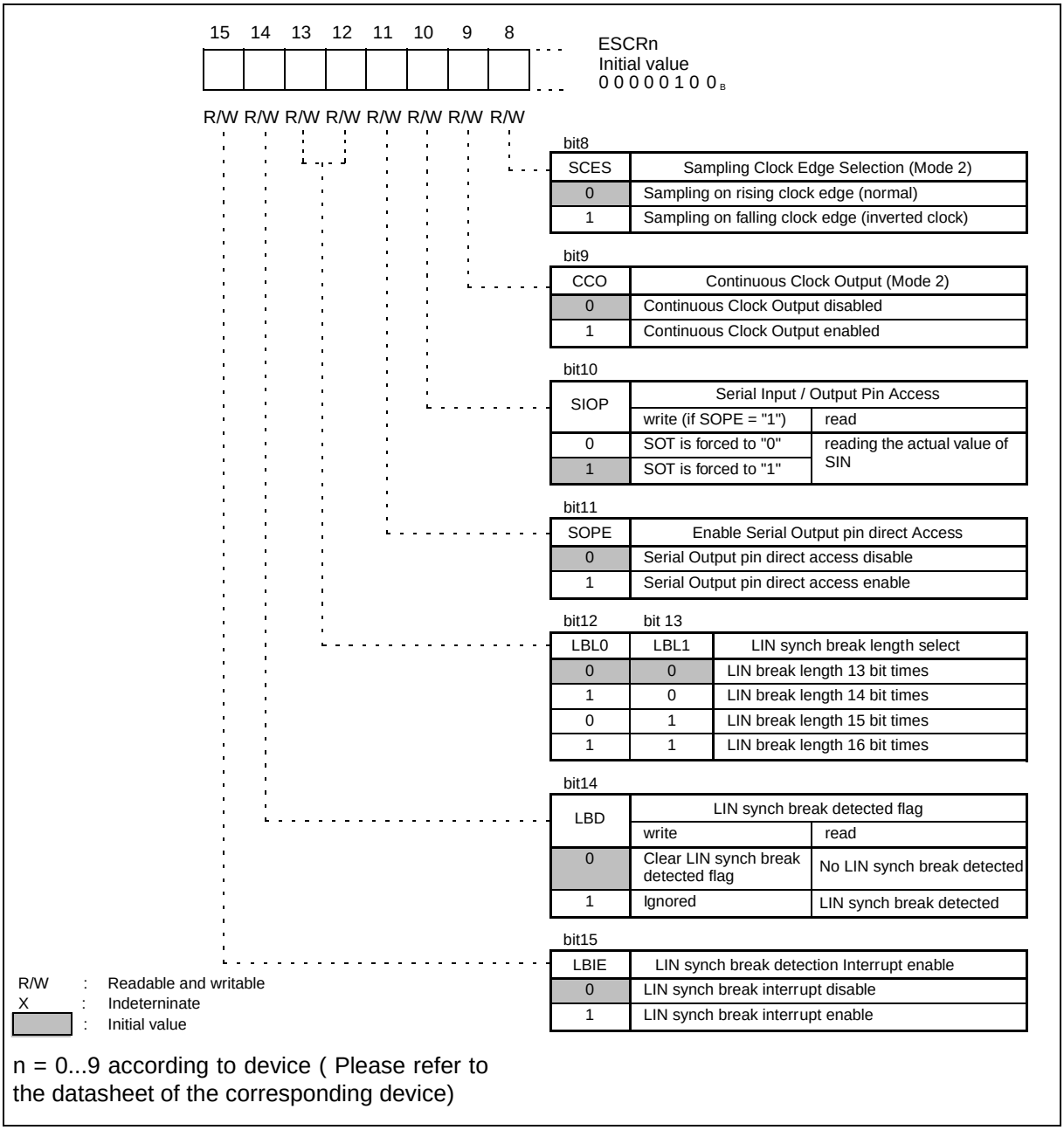
TDRn is a write-only register and RDRn is a read-only register. These registers are located in the same address, so the read value is different from the write value. Therefore, instructions that perform a read-modify-write (RMW) operation, such as the INC/DEC instruction, cannot be used.

## 20.4.5 Extended Status/Control Register (ESCRn)

This register provides several LIN functions, direct access to the SINn and SOTn pin and setting for USART synchronous clock mode.

### Extended status/control register (ESCRn)

Figure 20-7. Configuration of the extended status/control register (ESCRn)




n = 0...9 according to device ( Please refer to the datasheet of the corresponding device)

Table 20-8. Function of each bit of the extended status/control register (ESCRn)

| Bit name | | Function |
|---|---|---|
| bit15 | LBIE:<br>LIN synch break detection interrupt enable bit | • This bit enables/disables LIN synch break interrupt<br>• LIN synch break interrupt is connected to the receive interrupt. When in mode 3 the LBD bit is set and this bit is "1", a receive interrupt is signaled to the interrupt controller. This bit is fixed to "0" in operation mode 1 and 2. |
| bit14 | LBD:<br>LIN synch break detected flag | • This bit goes 1 if a LIN synch break was detected in operating mode 3. Writing a 0 to it clears this bit and the corresponding interrupt, if it is enabled.<br>• It is recommended to write "0" to the RXE bit in the SCRn register before using this bit.<br>• Read-modify-write instructions always return 1. Note that this does not indicate a LIN synch break. |
| bit13<br>bit12 | LBL1/0:<br>LIN synch break length selection | • These two bits determine how many serial bit times the LIN synch break is generated by USART. Receiving a LIN synch break is always fixed to 11 bit times. |
| bit11 | SOPE:<br>Serial Output pin direct access enable*1 | • Setting this bit to 1 enables the direct write to the SOTn pin, if SMRn:SOE = 1. *1 |
| bit10 | SIOP:<br>Serial Input/Output Pin direct access *1 | • Normal read instructions always return the actual value of the SINn pin. Writing to it sets the bit value to the SOTn pin, if SOPE = 1.<br>• During a Read-Modify-Write instruction the bit returns the SOTn value in the read cycle. *1 |
| bit9 | CCO:<br>Continuous Clock Output enable bit | • This bit enables a continuous serial clock at the SCKn pin if USART operates in master mode 2 (synchronous) and the SCKn pin is configured as a clock output.<br>**Note:**<br>When CCO bit is “1”, use ECCRn:SSM bit as setting to “1”. |
| bit8 | SCES:<br>Serial clock edge selection bit | • This bit inverts the serial clock signal in operation mode 2 (synchronous communication).<br>• Receiving data is sampled at the falling edge of the internal clock. If ECCRn:MS register is "0" (master mode) and SMRn:SCKE is "1" (clock output enabled), the output clock signal is also inverted.<br>• During operation mode 0,1,3 set this bit to 0. |

*1: See Table 20-9.

Table 20-9. Description of the interaction of SOPE and SIOP

| SOPE | SIOP | Writing to SIOP | Reading from SIOP |
|---|---|---|---|
| 0 | R/W | has no effect on SOTn, but holds the written value | returns current value of SINn |
| 1 | R/W | write “0” or “1” to SOTn | returns current value of SINn |
| 1 | RMW | reads current value of SOTn and writes it back | |

## 20.4.6 Extended Communication Control Register (ECCRn)

The extended communication control register provides bus idle recognition interrupt settings, synchronous clock settings, and the LIN break generation.

### Extended communication control register (ECCRn)

Figure 20-8. Configuration of the extended communication control register (ECCRn)

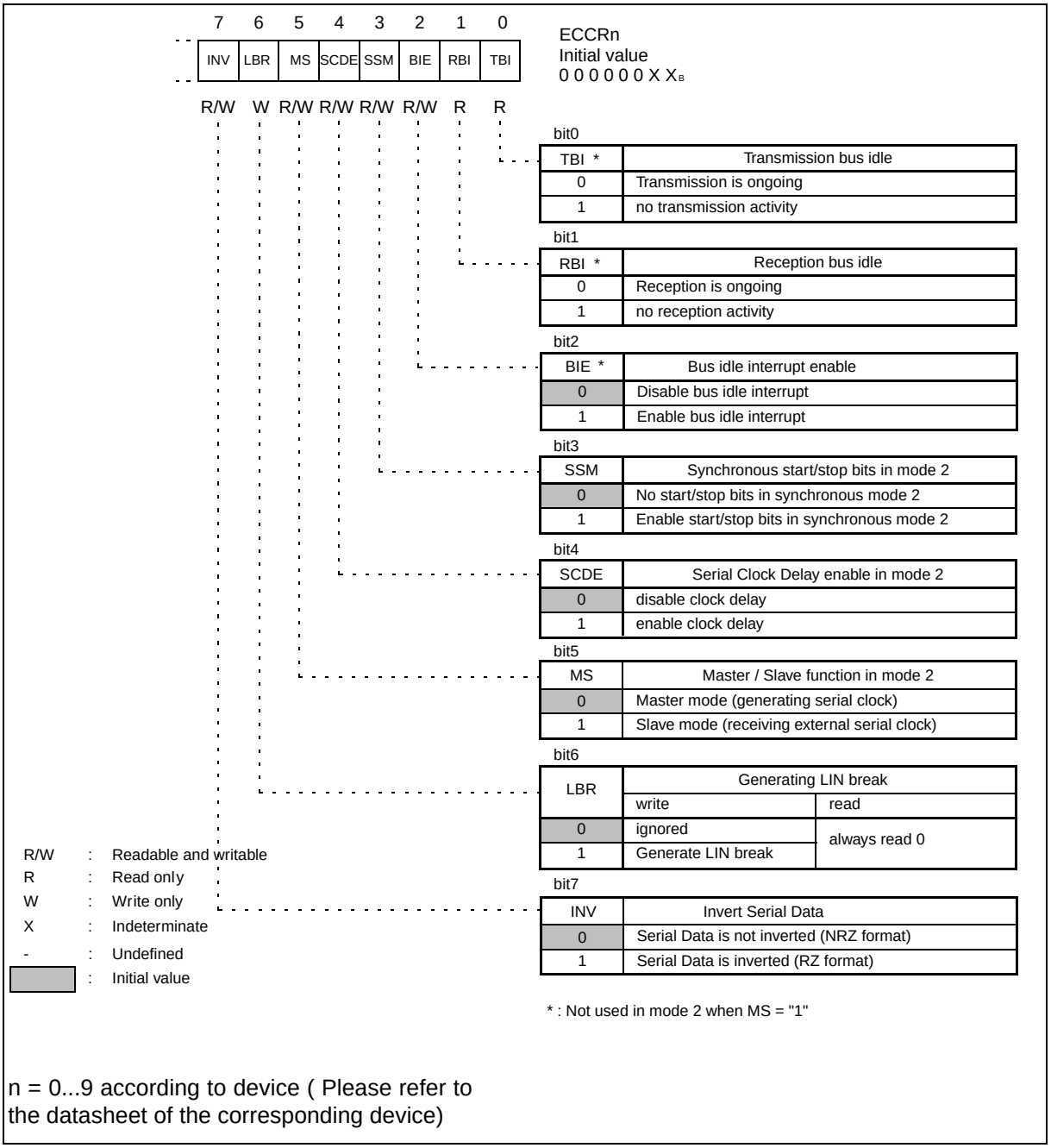


Table 20-10. Function of each bit of the Extended communication control register (ECCRn)

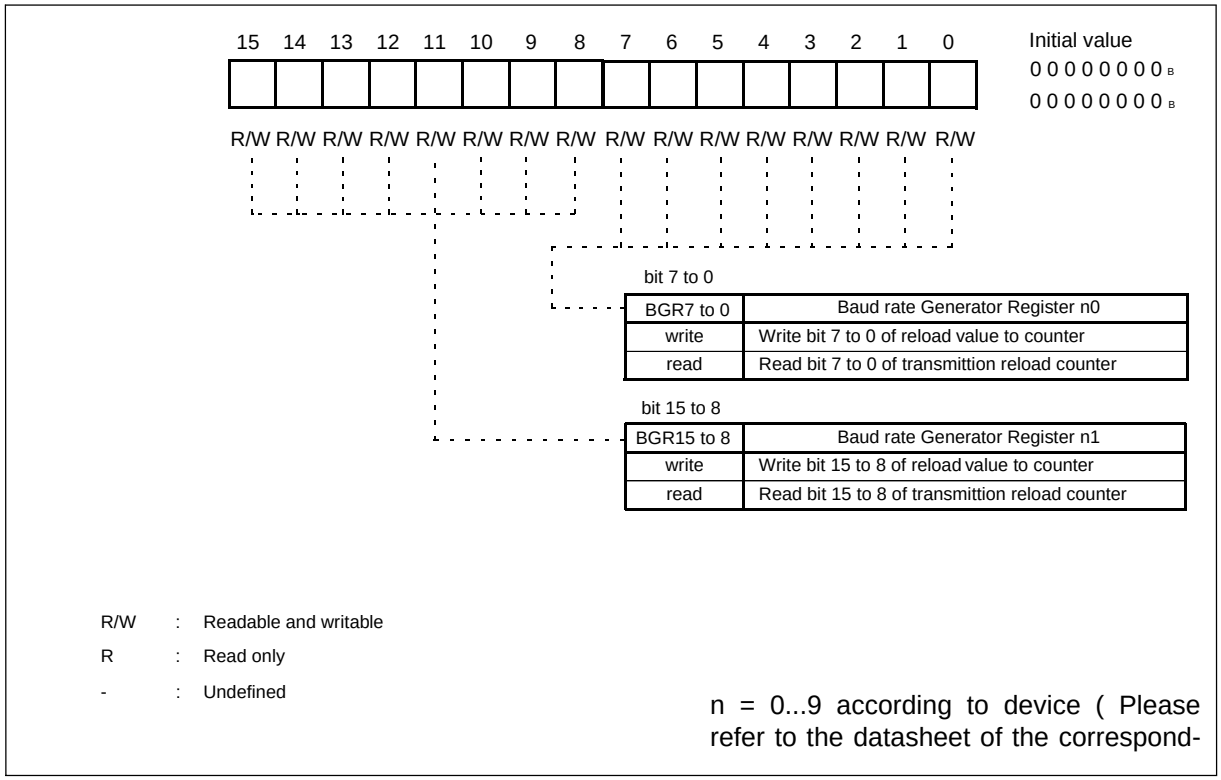
| Bit name | | Function |
|---|---|---|
| bit7 | INV: Invert serial data | • This bit inverts the serial data at SINn and SOTn pin. SCKn is not affected (see ESCRn: SCES).<br>• Writing "0": The serial data format is NRZ (default)<br>• Writing "1": The serial data is inverted (RZ format)<br>• RMW instructions do not affect this bit. |
| bit6 | LBR: Generating LIN synch break bit | • Writing a 1 to this bit generates a LIN synch break of the length selected by the LBL0/1 bits of the ESCRn, if operation mode 3 is selected. Setting to “0” in operation mode 0. |
| bit5 | MS:<br>Master/Slave mode selection bit | • This bit selects master or slave mode of USART in synchronous mode 2.<br>• If master mode is selected, USART generates the synchronous clock by itself.<br>• If slave mode is selected, USART receives external serial clock.<br>• This bit is fixed to “0” in operation mode 0, 1 and 3.<br>**Note:**<br>If slave mode is selected, the clock source must be external and set to “One-to-One” (SMRn:SCKE = 0, EXT = 1, OTO = 1). |
| bit4 | SCDE:<br>Serial clock delay enable bit | • If this bit is set the serial output clock is delayed as shown in Figure 20-20 if USART operates in master mode 2. |
| bit3 | SSM:<br>Start/Stop bit mode enable | • This bit adds start and stop bits to the synchronous data format in operation mode 2. It is ignored in mode 0, 1, and 3. |
| bit2 | BIE:<br>Bus idle interrupt enable | • This bit enables an interrupt on a bus idle condition.<br>• It enables a receive interrupt, if there is neither reception nor transmission ongoing (RBI = 1, TBI = 1) to indicate that the bus is idle.<br>• Do not use this bit in mode 2 when MS = 1.<br>**Note:**<br>When using the bus idle interrupt functionality, set ESIR:AICD = "1" to prevent that the interrupt is cleared automatically when the bus is no longer idle. |
| bit1 | RBI:<br>Reception bus idle flag bit | • This bit is “1” if there is no reception activity on the SINn pin.<br>• It is "0" when reception activity is ongoing on the SINn pin.<br>• Do not use this bit in mode 2. |
| bit0 | TBI:<br>Transmission bus idle flag bit | • This bit is “1” if there is no transmission activity on the SOTn pin.<br>• Do not use this bit in mode 2 when MS = "1". |

## 20.4.7 Baud Rate/Reload Counter Register (BGRn)

The baud rate/reload counter registers set the division ratio for the serial clock. Also the actual count of the transmission reload counter can be read.

### Baud rate generator register (BGRn)

Figure 20-9. Baud rate generator register (BGRn)




### Baud rate/reload counter register

The baud rate/reload counter registers determine the division ratio for the serial clock.

Both registers can be read or written via byte or word access.

## 20.4.8 Extended Serial Interrupt Register (ESIRn)

The extended serial interrupt register contains control bits to change the interrupt handling of the USART for improved interrupt handling and to enable DMA to handle USART data transfers.

### Extended serial interrupt register (ESIRn)

Figure 20-10. Configuration of the extended serial interrupt register (ESIRn)

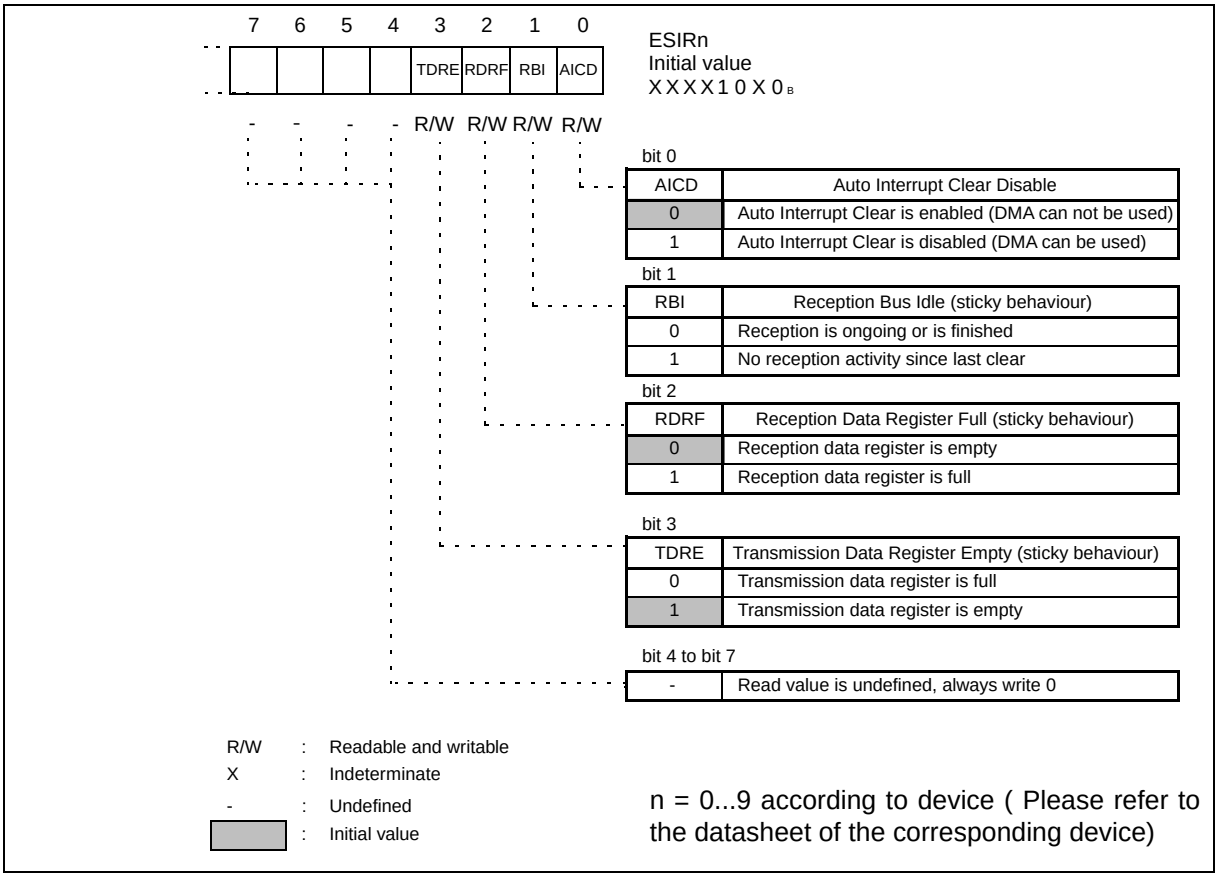


Table 20-11. Function of each bit of the extended serial interrupt register (ESIRn)

| Bit name | | Function |
|---|---|---|
| bit7 to bit4 | - | • These bits are undefined.<br>• Always write “0”. Read-modify-write is not affected. |
| bit3 | TDRE:<br>Transmission Data Register Empty (sticky behaviour) | • This flag has the same function as SSR:TDRE but it is not cleared when data is written to the transmission data register TDR.<br>• This flag indicates the status of the transmission data register (TDR).<br>• This bit is set to 1 when data is loaded into the transmission shift register and transmission starts.<br>• This bit must be cleared by writing "0".<br>• When AICD = "0" and SSR:TIE = "1", then SSR:TDRE is used to generate a transmit interrupt.<br>• When AICD = "1" and SSR:TIE = "1", then ESIR:TDRE is used to generate a transmit interrupt.<br>**Note:**<br>This bit is set to 1 (TDR empty) as its initial value. |
| bit2 | RDRF:<br>Reception Data Register Full (sticky behaviour) | • This flag has the same function as SSR:RDRF but it is not cleared when data is read from the reception data register RDR.<br>• This flag indicates the status of the reception data register (RDR).<br>• This bit is set to 1 when reception data is loaded into RDR.<br>• This bit must be cleared by writing "0".<br>• When AICD = "0" and SSR:RIE = "1", then SSR:RDRF is used to generate a receive interrupt.<br>• When AICD = "1" and SSR:RIE = "1", then ESIR:RDRF is used to generate a receive interrupt. |
| bit1 | RBI:<br>Reception Bus Idle (sticky behaviour) | • This flag has the same function as ECCR:RBI but it is not cleared when the reception bus is no longer idle.<br>• When there is no reception activity ongoing, this bit is set to "1".<br>• It must be cleared by writing "0".<br>• When AICD = "0" and ECCR:BIE = "1", then ECCR:RBI is used to generate a receive interrupt.<br>• When AICD = "1" and ECCR:BIE = "1", then ESIR:RBI is used to generate a receive interrupt.<br>**Note:**<br>When an interrupt shall be generated by a bus idle condition, set AICD = "1". When AICD = "0", ECCR:RBI would be used for interrupt generation. As soon as the bus is no longer idle, ECCR:RBI is "0" and the interrupt may be removed even before the interrupt handling routine could acknowledge it. |
| bit0 | AICD:<br>Auto Interrupt Clear Disable | • When this bit is "0", then SSR:TDRE, SSR:RDRF and ECCR:RBI are used to generate interrupts. In this mode, the USART can not be used together with DMA.<br>• When this bit is "1", then ESIR:TDRE, ESIR:RDRF and ESIR:RBI are used to generate interrupts. In this mode, the USART can be used together with DMA.<br>**Note:**<br>When AICD = "0", the USART can not be used with DMA. If DMA is used to handle data transfers of the USART, set AICD = "1". |

## 20.5 USART Interrupts

USART uses both reception and transmission interrupts. An interrupt request can be generated for either of the following causes:

- Receive data is set in the Reception Data Register (RDRn), or a reception error occurs.
- Transmission data is transferred from the Transmission Data Register (TDRn) to the transmission shift register and started.
- A LIN break is detected

DMA is available for these interrupts.

### 20.5.1 LIN-USART interrupts

Table 20-12. Interrupt control bits and interrupt causes of LIN-USART

| Reception/ transmission/ICU | Interrupt request flag bit | Flag Register | Operation mode | | | | Interrupt cause | Interrupt cause enable bit | How to clear the Interrupt Request |
|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | | | |
| Reception | RDRF | SSRn | ❍ | ❍ | ❍ | ❍ | receive data is written to RDRn | SSRn:RIE | Receive data is read |
| | ORE | SSRn | ❍ | ❍ | ❍ | ❍ | Overrun error | | "1" is written to clear rec. error bit (SCRn: CRE) |
| | FRE | SSRn | ❍ | ❍ | 1 | ❍ | Framing error | | |
| | PE | SSRn | ❍ | ✕ | 1 | ✕ | Parity error | | |
| | LBD | ESCRn | ✕ | ✕ | ✕ | ❍ | LIN synch break detected | ESCRn:LBIE | “0” is written to ESCRn: LBD |
| Transmission | TDRE | SSRn | ❍ | ❍ | ❍ | ❍ | TDRn empty | SSRn:TIE | Write data to TDRn |
| Bus Idle[3] | RBI, TBI | ECCRn,ESIRn | ❍ | ❍ | ❍ | ❍ | Neither transmission nor reception activity | ECCRn:BIE | Write "0" to ESIRn:RBI, write data to TDRn |
| Input Capture Unit | ICPy[2] | ICSy[2] | ✕ | ✕ | ✕ | ❍ | 1st falling edge of LIN synch field | ICSy:ICEy[2] | disable ICPy temporarily |
| | | | | | | | 5th falling edge of LIN synch field | | disable ICPy |

❍: Used

✕: Unused

[1]: Only available if ECCRn/SSM = 1

[2]: For the Input Capture Unit number y refer to chapter Input Capture Unit Source Select for LIN-USART on page 27

[3]: It is not recommended to use bus idle interrupt with ESIR:AICD = 0, because in this case, as soon as reception activity starts, this interrupt is cleared.

#### Receive interrupt

If one of the following events occurs in reception mode, the corresponding flag bit of the Serial Status Register (SSRn) and the Extended Serial Interrupt Register (ESIRn) is set to “1”:

- Data reception is complete, i. e. the received data was transferred from the serial input shift register to the Reception Data Register (RDRn) and data can be read: SSRn:RDRF, ESIRn:RDRF
- Overrun error, i. e. SSRn:RDRF = 1 and RDR was not read by the CPU and received next serial data: SSRn:ORE
- Framing error, i. e. a stop bit was expected, but a "0"-bit was received: SSRn:FRE
- Parity error, i. e. a wrong parity bit was detected: SSRn:PE

If ESIR:AICD = 0 and at least one of the flag bits SSRn:RDRF, SSRn:ORE, SSRn:FRE, SSRn: PE is set to “1” and the receive interrupt is enabled (SSRn:RIE = 1), a receive interrupt request is generated.

If ESIR:AICD = 1 and at least one of the flag bits ESIRn:RDRF, SSRn:ORE, SSRn:FRE, SSRn: PE is set to “1” and the receive interrupt is enabled (SSRn:RIE = 1), a receive interrupt request is generated.

If the Reception Data Register (RDRn) is read, the SSRn:RDRF flag is automatically cleared to “0”. Note that this is the only way to reset the SSRn:RDRF flag.

The error flags are cleared to “0”, if a “1” is written to the Clear Reception Error (CRE) flag bit of the Serial Control Register (SCRn). The RDRn contains only valid data if the RDRF flag is “1” and no error bits are set.

Note that the CRE flag is "write only" and by writing a "1" to it, it is internally held to "1" for one CLKP1 clock cycle.

### Transmission Interrupt

If transmission data is transferred from the Transmission Data Register (TDRn) to the transfer shift register and transfer is started, the Transmission Data Register Empty flag bit (TDRE) of the Serial Status Register (SSR) is set to “1”. In this case an interrupt request is generated, if the Transmission Interrupt Enable (TIE) bit of the SSR was set to “1” before.

Note, that the initial value of TDRE (after hardware or software reset) is “1”. So an interrupt is generated immediately then, if the TIE flag is set to “1”. Also note, that the only way to reset the TDRE flag is writing data to the Transmission Data Register (TDRn).

### Bus Idle Interrupt

If no reception is ongoing at the SINn pin, the bits ECCRn:RBI and ESIRn:RBI are set to "1". ECCRn:RBI is cleared as soon as reception is starting, while ESIRn:RBI must be cleared explicitely by writing "0" to this bit.

If no transmission is ongoing at the SOTn pin, the bit ECCRn:TBI is set to "1". When transmission is starting, ECCRn:TBI is cleared again.

If ESIRn:AICD = "0" and ECCRn:BIE = "1", the interrupt is generated as soon as both ECCRn:RBI and ECCRn:TBI are "1". It is cleared as soon as either ECCRn:RBI or ECCRn:TBI is cleared. Do not use the bus idle interrupt with AICD = "0", because ECCRn:RBI may be cleared before the interrupt handling software is able to clear the interrupt.

If ESIRn:AICD = "1" and ECCRn:BIE = "1", the interrupt is generated as soon as both ESIRn:RBI and ECCRn:TBI are "1". It is cleared as soon as either ESIRn:RBI or ECCRn:TBI is cleared. Use the bus idle interrupt with ESIRn:AICD = 1, because both ESIRn:RBI and ECCRn:TBI are acknowlegde by the interrupt service routine.

### LIN Synchronization Break Interrupt

This paragraph is only relevant, if USART operates in mode 3 as a LIN slave.

If the bus (serial input) goes “0” (dominant) for more than 11 bit times, the LIN Break Detected (LBD) flag bit of the Extended Status/Control Register (ESCRn) is set to “1”. Note, that in this case after 9 bit times the reception error flags are set to “1”, therefore the RXE flag has to set to “0”, if only a LIN synch break detect is desired.

The interrupt and the ESCRn:LBD flag are cleared after writing a "0" to the LBD flag. This has to be performed before input capture interrupt for LIN sync field.

### LIN Synchronization Field Edge Detection Interrupts

This paragraph is only relevant, if USART operates in mode 3 as a LIN slave. After a LIN break detection the next falling edge of the reception bus is indicated by USART. Simultaneously an internal signal connected to the ICU associated to the LIN-USART is set to “1” (please refer to chapter Input Capture Unit Source Select for LIN-USART on page 27 for an overview which ICU is associated to which LIN-USART).

This signal is reset to “0” after the fifth falling edge of the LIN Synchronization Field. In both cases the associated ICU generates an interrupt, if “both edge detection” and the ICU interrupt are enabled. The difference of the ICU counter values is the serial clock multiplied by 8. Dividing it by 8 results in the new detected and calculated baud rate for the dedicated reload counter. This value - 1 has then to be written to the Baud Rate Generator Registers (BGRn). There is no need to restart the reload counter, because it is automatically reset if a falling edge of a start bit is detected.

## 20.5.2 USART DMA functions

The USART can be operated with DMA, for reception or transmission. Please refer to DMA on page 95 for details about DMA.

### Note:

When USART is served by DMA, set ESIRn:AICD = "1".

In addition, in case of an error while reception, the ongoing DMA transfer would be stopped, if the SE bit of DMACS register of the corresponding DMA channel is set.

## 20.5.3 Receive interrupt Generation and Flag Set Timing

The following are the receive interrupt causes: completion of reception (SSRn:RDRF) and occurrence of a reception error (SSRn:PE, ORE, or FRE).

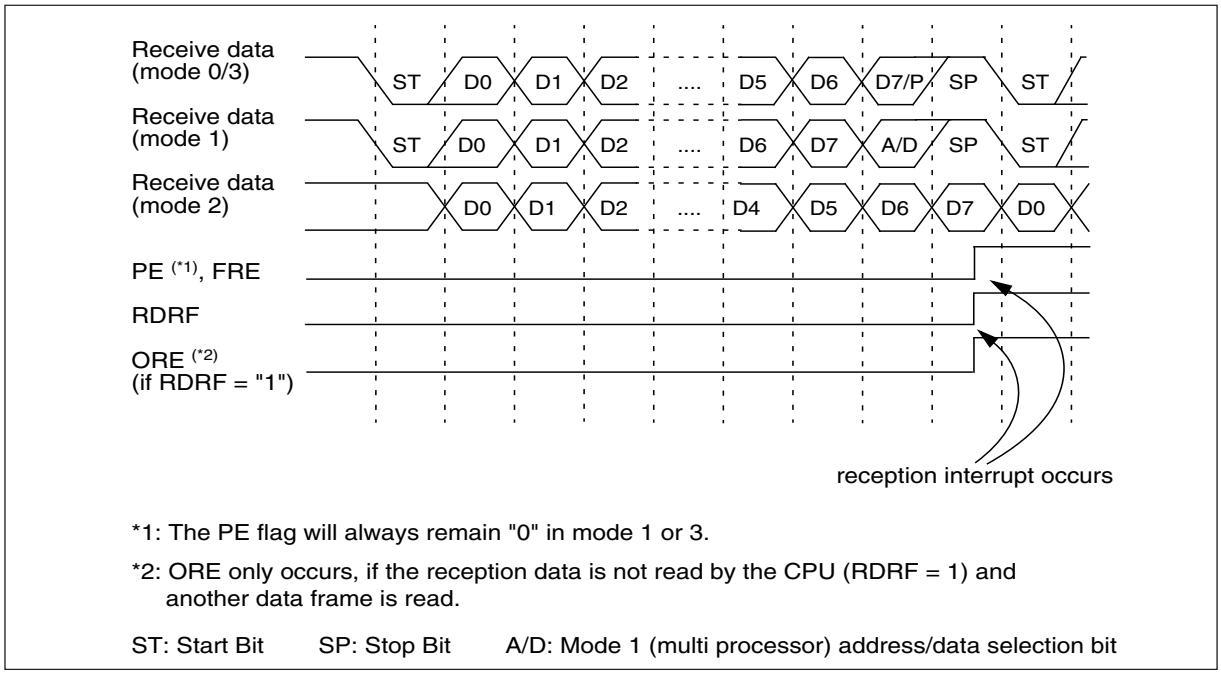
### Receive interrupt generation and flag set timing

Generally a receive interrupt is generated, if the received data is complete (when ESIRn:AICD = "0": SSRn:RDRF = "1", when ESIRn:AICD = "1": ESIRn:RDRF = "1") and the receive interrupt Enable (RIE) flag bit of the Serial Status Register (SSRn) was set to “1”. This interrupt is generated if the first stop bit is detected in mode 0, 1, 2 (if SSM = 1), 3, or the last data bit was read in mode 2 (if SSM = 0).

### Note:

If a reception error has occurred, the Reception Data Register (RDRn) contains invalid data in each mode.

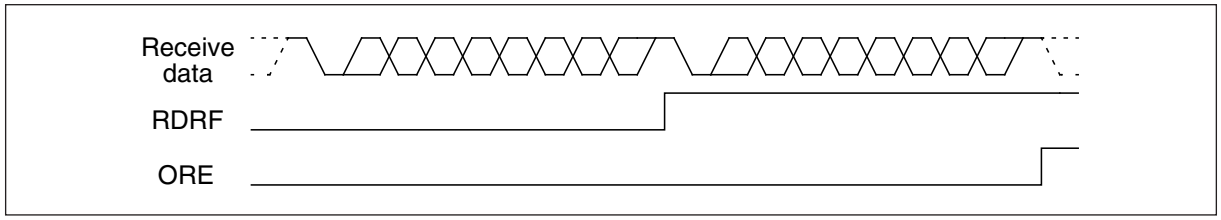
Figure 20-11. Reception operation and flag set timing




### Note:

The example in Figure 20-11 does not show all possible reception options for mode 0 and 3. Here it is: “7p1” and “8N1” (p = “E” [even] or “O” [odd]).

Figure 20-12. ORE set timing:



## 20.5.4 Transmission Interrupt Generation and Flag Set Timing

A transmission interrupt is generated when the transmission data is transferred from transmission data register (TDRn) to transmission shift register and started.

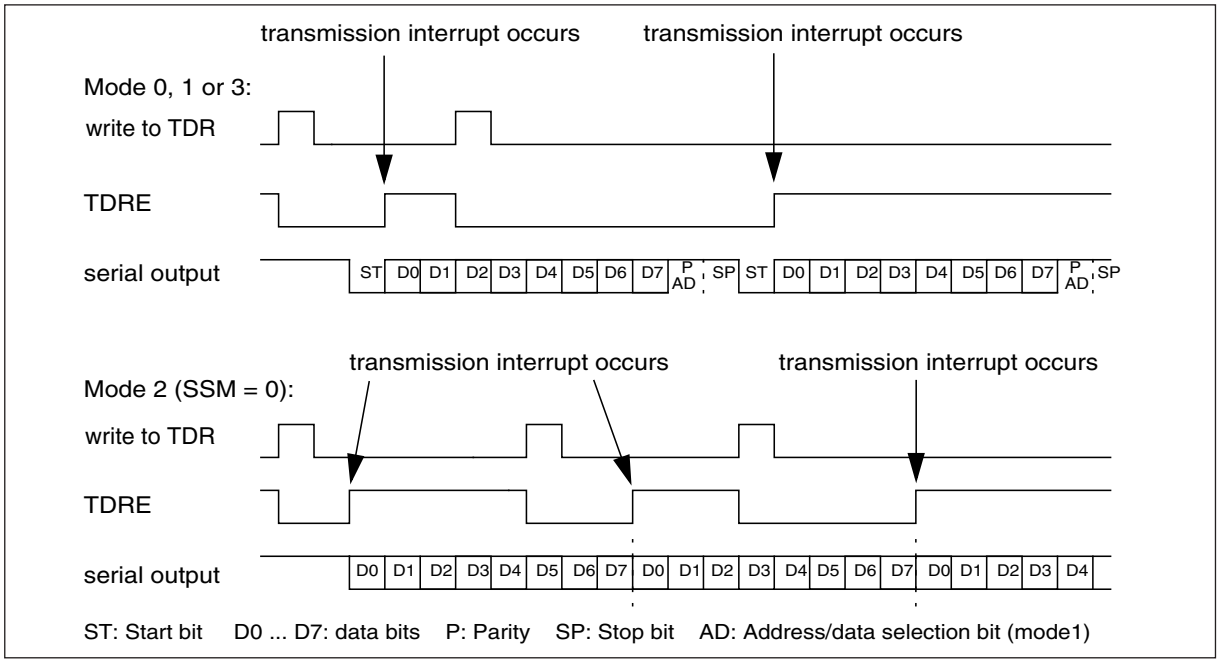
### Transmission interrupt generation and flag set timing

A transmission interrupt is generated, when the next data to be sent is ready to be written to the Transmission Data Register (TDRn), i. e. the TDRn is empty, and the transmission interrupt is enabled by setting the Transmission Interrupt Enable (TIE) bit of the Serial Status Register (SSRn) to “1”.

The Transmission Data Register Empty (TDRE) flag bit of the SSRn indicates an empty TDRn. Writing data to TDRn clears the TDRE bit.

The following figure demonstrates the transmission operation and flag set timing for the four modes of USART.

Figure 20-13. Transmission operation and flag set timing




**Note:**

The example in Figure 20-13 does not show all possible transmission options for mode 0. Here it is: “8p1” (p = “E” [even] or “O” [odd]). Parity is not provided in mode 3 or 2, if SSM = 0.

### Transmission interrupt request generation timing

If the TDRE flag is set to “1” when a transmission interrupt is enabled (SSRn: TIE=1), transmission interrupt request is generated.

**Note:**

A transmission completion interrupt is generated immediately after the transmission interrupt is enabled (TIE=1) because the TDRE bit is set to 1 as its initial value. Carefully specify the transmission interrupt enable timing.

## 20.6 USART Baud Rates

One of the following can be selected for the USART serial clock source:

- Dedicated baud rate generator (Reload Counter)
- External clock as it is (clock input to the SCKn pin)
- External clock connected to the baud rate generator (Reload Counter)

### 20.6.1 USART baud rate selection

The baud rate selection circuit is designed as shown below. One of the following three types of baud rates can be selected:

#### Baud rates determined using the dedicated baud rate generator (reload counter)

USART has two independent internal reload counters for transmission and reception serial clock. The baud rate can be selected via the 16-bit reload value determined by the Baud Rate Generator Registers (BGRn).

The reload counter divides the peripheral clock CLKP1 by the value set in the Baud Rate Generator Registers.

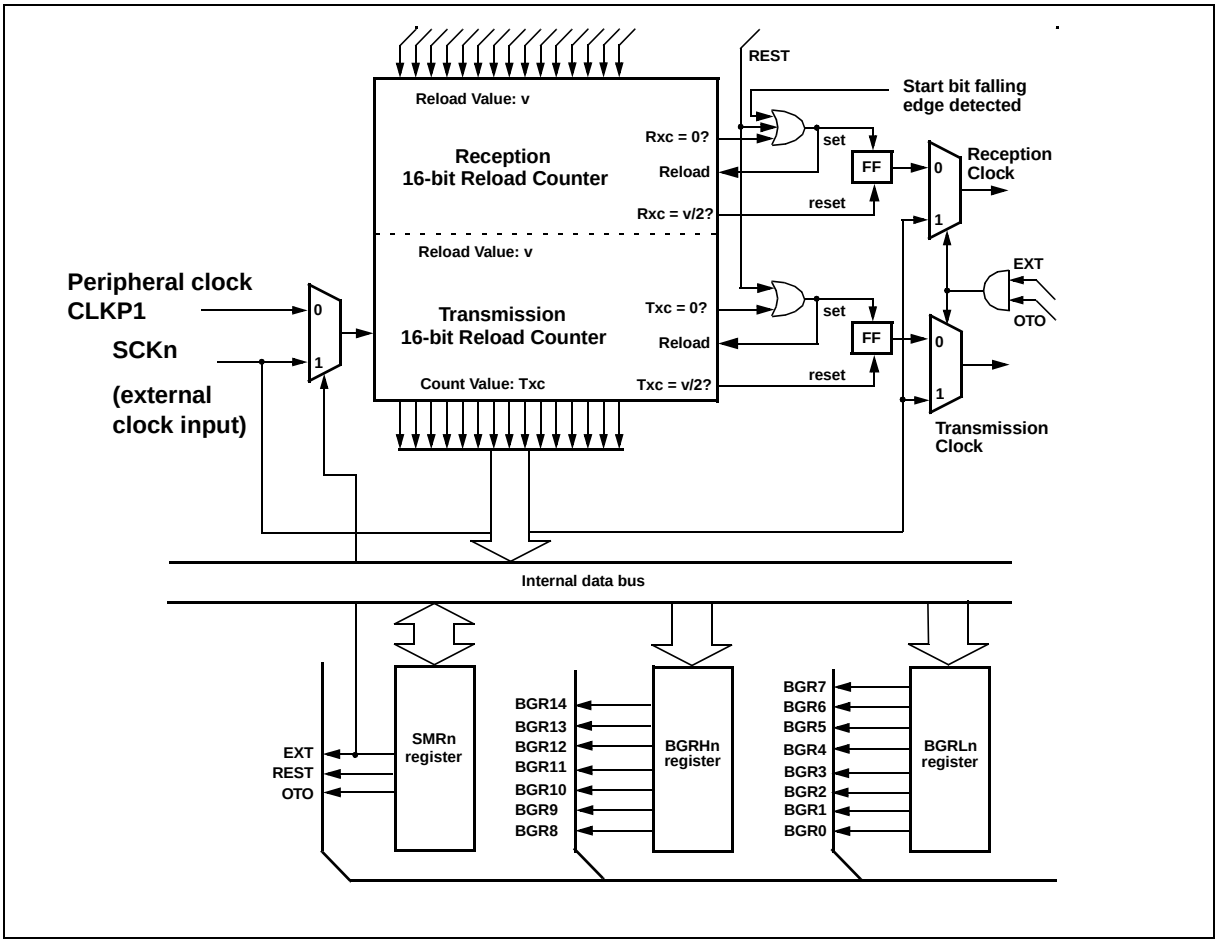
#### Baud rates determined using external clock (one-to-one mode)

The clock input from USART clock pulse input pins (SCK) is used as it is (synchronous). Any baud rate less than the peripheral clock CLKP1 divided by 4 and is divisible can be set externally.

#### Baud rates determined using the dedicated baud rate generator with external clock

An external clock source can also be connected internally to the reload counter. In this mode it is used instead of the internal peripheral clock CLKP1. This was designed to use quartz oscillators with special frequencies and having the possibility to divide them.

Figure 20-14. Baud rate selection circuit (reload counter)




## 20.6.2 Setting the Baud Rate

This section describes how the baud rates are set and the resulting serial clock frequency is calculated.

### *20.6.2.1 Calculating the baud rate*

The both 16-bit reload counters are programmed by the baud rate generator registers (BGRn). The following calculation formula should be used to set the desired baud rate:

Reload Value:

$$\mathbf{v = [\Phi / b] - 1,}$$

where $\Phi$ is the peripheral clock CLKP1, b the baud rate and [] gaussian brackets (mathematical rounding function).

#### Example of calculation

If the peripheral clock CLKP1 is 16 MHz and the desired baud rate is 19200 bps baud then the reload value v is:

$$\mathbf{v = [16*10^6 / 19200] - 1 = 832}$$

The exact baud rate can then be recalculated: $b_{exact} = \Phi / (v + 1)$, here it is: $16*10^6 / 833 = 19207.6831$

**Note:**

Setting the reload value to 0 stops the reload counter. For this reason the minimum division ratio is 2. For asynchronous communication, the reload value must be greater than equal to 4 because 5 times over-sampling is performed internally.

### *20.6.2.2 Suggested division ratios for different peripheral clock CLKP1 frequencies and baud rates*

The following settings are suggested for different peripheral clock CLKP1 frequencies and baud rates:

Table 20-13. Suggested baud rates and reload values at different peripheral clock CLKP1 frequencies.

| Baud rate | 8 MHz | | 10 MHz | | 16 MHz | | 20 MHz | | 24 MHz | |
|---|---|---|---|---|---|---|---|---|---|---|
| | value | dev. | value | dev. | value | dev. | value | dev. | value | dev. |
| 4M | - | - | - | - | - | - | 4 | 0 | 5 | 0 |
| 2M | - | - | 4 | 0 | 7 | 0 | 9 | 0 | 11 | 0 |
| 1M | 7 | 0 | 9 | 0 | 15 | 0 | 19 | 0 | 23 | 0 |
| 500000 | 15 | 0 | 19 | 0 | 31 | 0 | 39 | 0 | 47 | 0 |
| 460800 | - | - | - | - | - | - | - | - | 51 | -0.16 |
| 250000 | 31 | 0 | 39 | 0 | 63 | 0 | 79 | 0 | 95 | 0 |
| 230400 | - | - | - | - | - | - | - | - | 103 | -0.16 |
| 153600 | 51 | -0.16 | 64 | -0.16 | 103 | -0.16 | 129 | -0.16 | 155 | -0.16 |
| 125000 | 63 | 0 | 79 | 0 | 127 | 0 | 159 | 0 | 191 | 0 |
| 115200 | 68 | -0.64 | 86 | 0.22 | 138 | 0.08 | 173 | 0.22 | 207 | -0.16 |
| 76800 | 103 | -0.16 | 129 | -0.16 | 207 | -0.16 | 259 | -0.16 | 311 | -0.16 |
| 57600 | 138 | 0.08 | 173 | 0.22 | 277 | 0.08 | 346 | -0.06 | 416 | 0.08 |
| 38400 | 207 | -0.16 | 259 | -0.16 | 416 | 0.08 | 520 | 0.03 | 624 | 0 |
| 28800 | 277 | 0.08 | 346 | <0.01 | 554 | -0.01 | 693 | -0.06 | 832 | -0.03 |
| 19200 | 416 | 0.08 | 520 | 0.03 | 832 | -0.03 | 1041 | 0.03 | 1249 | 0 |
| 10417 | 767 | <0.01 | 959 | <0.01 | 1535 | <0.01 | 1919 | <0.01 | 2303 | <0.01 |
| 9600 | 832 | 0.04 | 1041 | 0.03 | 1666 | 0.02 | 2083 | 0.03 | 2499 | 0 |
| 7200 | 1110 | <0.01 | 1388 | <0.01 | 2221 | <0.01 | 2777 | <0.01 | 3332 | <0.01 |
| 4800 | 1666 | 0.02 | 2082 | -0.02 | 3332 | <0.01 | 4166 | <0.01 | 4999 | 0 |
| 2400 | 3332 | <0.01 | 4166 | <0.01 | 6666 | <0.01 | 8332 | <0.01 | 9999 | 0 |
| 1200 | 6666 | <0.01 | 8334 | 0.02 | 13332 | <0.01 | 16666 | <0.01 | 19999 | 0 |
| 600 | 13332 | <0.01 | 16666 | <0.01 | 26666 | <0.01 | - | - | - | - |
| 300 | 26666 | <0.01 | - | - | - | - | - | - | - | - |

**Note:**

1) Deviations are given in %.

2) Maximum Synchronous Baud Rate: Peripheral clock CLKP1 frequency divided by 5.

### *20.6.2.3 Using external clock*

If the EXT bit of the SMRn register is set, an external clock is selected, which has to be connected to the SCKn pin. The external clock is used in the same way as the peripheral clock CLKP1 to the baud rate reload counter.

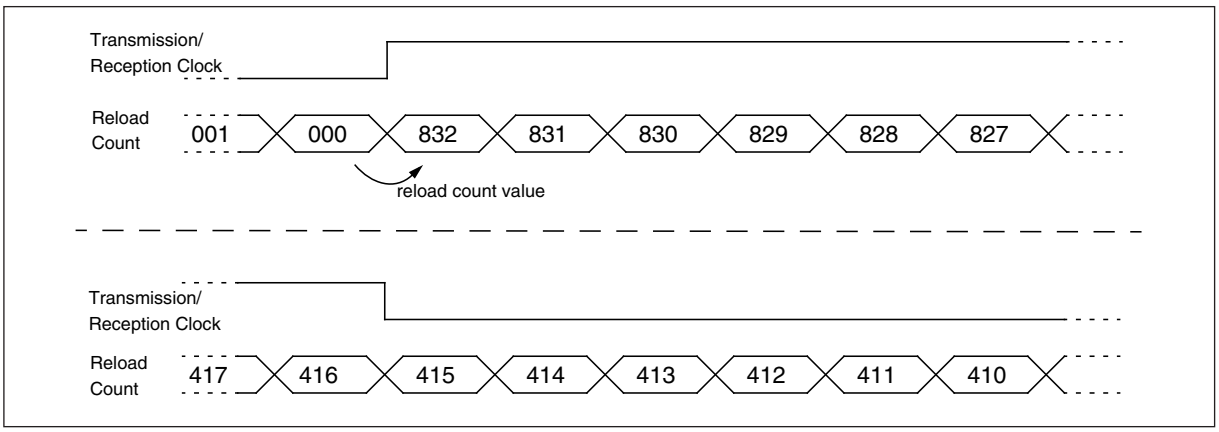
If One-to-one External Clock Input Mode (SMRn:OTO) is selected the SCKn signal is directly connected to the USART serial clock inputs. This is needed for the USART synchronous mode 2 operating as slave device.

Note, that in any case the resulting clock signal is synchronized to the peripheral clock CLKP1 in the USART module. This means that indivisible clock rates will result in phase unstable signals.

#### 20.6.2.4 Counting example

Assume the reload value is 832. The Figure 20-15 demonstrates the behavior of both Reload Counters:

Figure 20-15. Counting example of the reload counters




**Note:**

The falling edge of the Serial Clock Signal always occurs after | (v + 1) / 2 |.

### 20.6.3 Restarting the Reload Counter

The Reload Counters can be restarted of the following reasons:

Transmission and reception reload counter:

- Global MCU reset
- USART programmable clear (SMRn:UPCL bit)
- User programmable restart (SMRn:REST bit)

Reception reload counter:

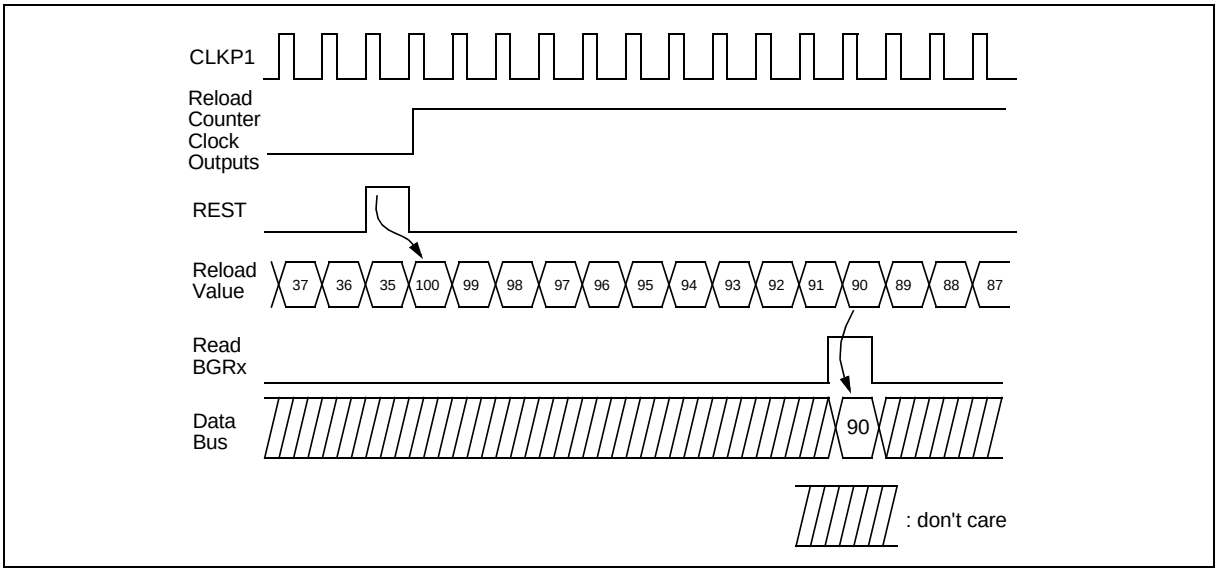
- Start bit falling edge detection in asynchronous mode

#### 20.6.3.1 Programmable restart

If the REST bit of the Serial Mode Register (SMRn) is set by the user, both Reload Counters are restarted at the next clock cycle. This feature is intended to use the Transmission Reload Counter as a small timer.

The following figure illustrates a possible usage of this feature (assume that the reload value is 100.)

Figure 20-16. Reload counter restart example




In this example the number of peripheral clock CLKP1 cycles (cyc) after REST is then:

$$cyc = v - c + 1 = 100 - 90 + 1 = 11,$$

where v is the reload value and c is the read counter value.

### Note:

If USART is reset by setting SMRn:UPCL, the Reload Counters will restart too.

- Automatic restart
  In asynchronous USART mode if a falling edge of a start bit is detected the Reception Reload Counter is restarted. This is intended to synchronize the serial input shifter to the incoming serial data stream.

### Clearing reload counters

The baud rate reload/counter register (BGRn) and the baud rate reload counters are cleared to "0" by the MCU global reset and the counters stops. The reload counters are cleared to "0" by writing "1" to the UPCL bit in the SMRn register. However the value stored in the reload register is kept unchanged and the counters start from reload value immediately. Writing "0" to the REST bit does not clear the counters and they restart from reload value immediately.

## 20.7 Operation of USART

USART operates in operation mode 0 for normal bidirectional serial communication, in mode 2 and 3 in bidirectional communication as master or slave, and in mode 1 as master or slave in multiprocessor communication.

### Operation of USART

- Operation modes

  There are four USART operation modes: modes 0 to 3. As listed in Table 20-14, an operation mode can be selected according to the communication method.

Table 20-14. USART operation mode

| Operation mode | | Data length | | Synchronization of mode | Length of stop bit | data bit direction *1 |
|---|---|---|---|---|---|---|
| | | parity disabled | parity enabled | | | |
| 0 | normal mode | 7 or 8 | | asynchronous | 1 or 2 | L/M |
| 1 | multiprocessor | 7 or 8 + 1*2 | - | asynchronous | 1 or 2 | L/M |
| 2 | normal mode | 8 | | synchronous | 0, 1 or 2 | L/M |
| 3 | LIN mode | 8 | - | asynchronous | 1 | L |

*1: means the data bit transfer format: LSB or MSB first

*2: "+1" means the indicator bit of the address/data selection in the multiprocessor mode, instead of parity.

### Note:

Mode 1 operation is supported both for master or slave operation of USART in a master-slave connection system. In Mode 3 the USART function is locked to 8N1-Format, LSB first.

If the mode is changed, USART cuts off all possible transmission or reception and awaits then new action.

### Inter-CPU connection method

External Clock One-to-one connection (normal mode) and master-slave connection (multiprocessor mode) can be selected. For either connection method, the data length, whether to enable parity, and the synchronization method must be common to all CPUs. Select an operation mode as follows:

- In the one-to-one connection method, operation mode 0 or 2 must be used in the two CPUs. Select operation mode 0 for asynchronous transfer mode and operation mode 2 for synchronous transfer mode.
  Note, that one CPU has to be configured as master and the other has to be configured as slave in synchronous mode 2.
- Select operation mode 1 for the master-slave connection method and use it either for the master or slave system.

### Synchronization methods

In asynchronous operation USART reception clock is automatically synchronized to the falling edge of a received start bit.

In synchronous mode the synchronization is performed either by the clock signal of the master device or by USART itself if operating as master.

### Signal mode

USART can treat data only in non-return to zero (NRZ) format.

### Operation enable bit

USART controls both transmission and reception using the operation enable bit for transmission (SCRn: TXE) and reception (SCRn: RXE).

- If reception operation is disabled during reception (data is input to the reception shift register), finish frame reception and read the received data of the reception data register (RDRn). Then stop the reception operation.

- If the transmission operation is disabled during transmission (data is output from the transmission shift register), wait until there is no data in the transmission data register (TDRn) before stopping the transmission operation.

## 20.7.1 Operation in Asynchronous Mode (Op. Modes 0 and 1)

When USART is used in operation mode 0 (normal mode) or operation mode 1 (multiprocessor mode), the asynchronous transfer mode is selected.

### *20.7.1.1 Operation in asynchronous mode*

#### Transfer data format

Generally each data transfer in the asynchronous mode operation begins with the start bit (low-level on bus) and ends with at least one stop bit (high-level). The direction of the bit stream (LSB first or MSB first) is determined by the BDS bit of the Serial Status Register (SSRn). The parity bit (if enabled) is always placed between the last data bit and the (first) stop bit.

In operation mode 0 the length of the data frame can be 7 or 8 bits, with or without parity, and 1 or 2 stop bits.

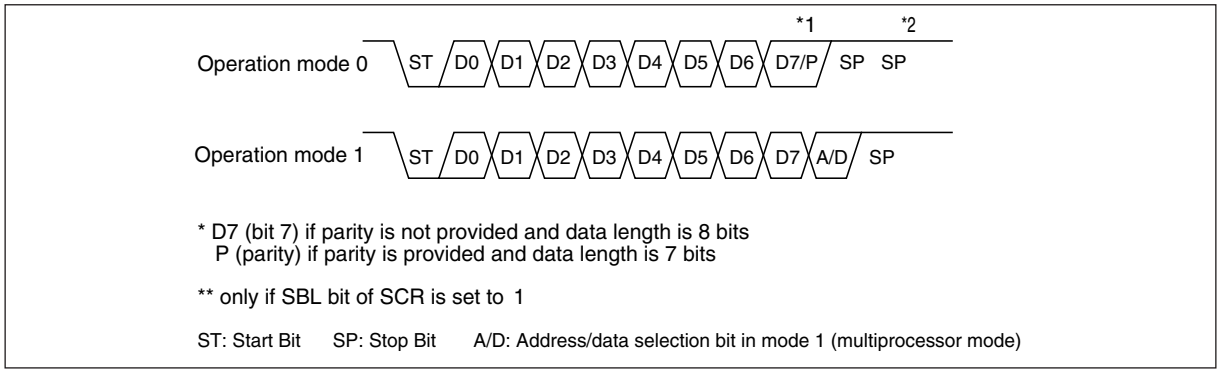
In operation mode 1 the length of the data frame can be 7 or 8 bits with a following address-/data-selection bit instead of a parity bit. 1 or 2 stop bits can be selected.

The calculation formula for the bit length of a transfer frame is:

Length = 1 + d + p + s

(d = number of data bits [7 or 8], p = parity [0 or 1], s = number of stop bits [1 or 2]

Figure 20-17. Transfer data format (operation modes 0 and 1))




#### Note:

If BDS bit of the Serial Status Register (SSRn) is set to “1” (MSB first), the bit stream processes as: D7, D6, ..., D1, D0, (P).

During Reception both stop bits are detected, if selected. But the Reception data register full (RDRF) flag will go “1” at the first stop bit. The bus idle flag (RBI of ECCRn) goes “1” after the second stop bit if no further start bit is detected. (The second stop bit belongs to “bus activity”, although it is just mark level.)

#### Transmission operation

If the Transmission Data Register Empty (TDRE) flag bit of the Serial Status Register (SSRn) is “1”, transmission data is allowed to be written to the Transmission Data Register (TDRn). When data is written, the TDRE flag goes “0”. If the transmission operation is enabled by the TXE-Bit (“1”) of the Serial Control Register (SCRn), the data is written next to the transmission shift register and the transmission starts at the next clock cycle of the serial clock, beginning with the start bit. Thereby the TDRE flag goes “1”, so that new data can be written to the TDRn.

If transmission interrupt is enabled (TIE = 1), the interrupt is generated by the TDRE flag. Note, that the initial value of the TDRE flag is “1”, so that in this case if TIE is set to “1” an interrupt will occur immediately.

When the character length is set to 7 bits (CL=0), the unused bit of the TDRn is always the MSB, independently from the bit direction setting in the BDS bit (LSB first or MSB first).

#### Reception operation

Reception operation is performed when it is enabled by the Reception Enable (RXE) flag bit of the SCRn. If a start bit is detected, a data frame is received according to the format specified by the SCRn. In case of errors, the corresponding error flags are set (PE, ORE, FRE). After the reception of the data frame the data is transferred from the serial shift register to the Reception Data Register (RDRn) and the Receive Data Register Full (RDRF) flag bits of SSRn and ESIRn registers are set.

If receive interrupt is enabled (RIE = 1) and ESIRn:AICD = "0", the interrupt is  generated by SSRn:RDRF.

If receive interrupt is enabled (RIE = 1) and ESIRn:AICD = "1", the interrupt is  generated by ESIRn:RDRF.

The data then has to be read by the CPU. By doing so, the SSRn:RDRF flag is cleared.

When ESIRn:AICD = "0", this also clears the interrupt.

When ESIRn:AICD = "1", the interrupt mst be cleared by writing "1" to ESIRn:RDRF.

When the character length is set to 7 bits (CL=0), the unused bit of the RDRn is always the MSB, independently from the bit direction setting in the BDS bit (LSB first or MSB first).

#### Note:

Only when the RDRF flag bit is set and no errors have occurred the Reception Data Register (RDRn) contains valid data.

#### Used clock

Use the internal clock or external clock. Select the baudrate generator (SMRn:EXT = 0 or 1, SMRn:OTO = 0) for desired baudrate.

#### Stop bit, error detection, and parity:

Number of stop bit, 1 or 2 can be specified by the SBL bit of the SCRn register. When receiving and 2-bit is specified as the stop bit, the second stop bit is checked in addition to the first stop bit. The RBI (bus idle) flag is set after the second stop bit. However the RDRF flag is set when the first stop bit is received. In mode 0, parity error, overrun error and framing error are checked. In mode 1, parity check is not supported and overrun error and framing error are checked. The PEN bit of the SCRn register enables/disables the parity bit and the P bit specifies even or odd parity in mode 0.

## 20.7.2 Operation in Synchronous Mode (Operation Mode 2)

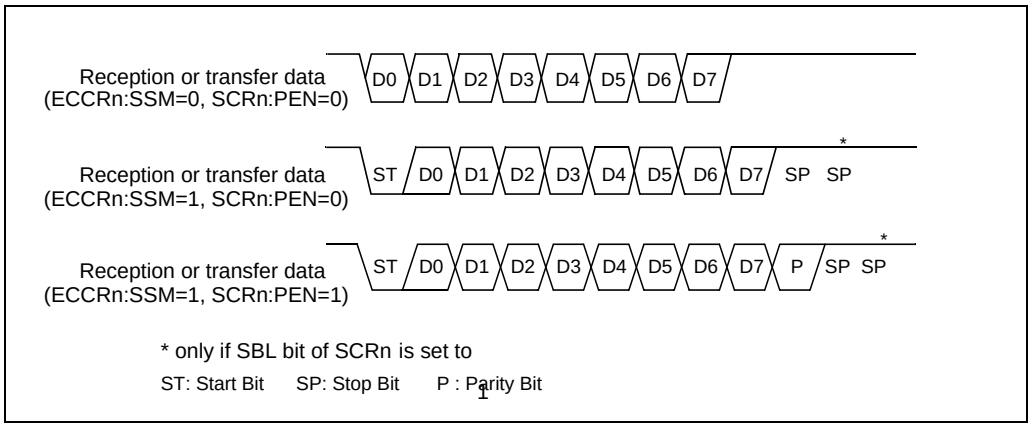
The clock synchronous transfer method is used for USART operation mode 2 (normal mode).

### *20.7.2.1 Operation in synchronous mode (operation mode 2)*

#### Transfer data format

In the synchronous mode, 8-bit data is transferred without start or stop bits if the SSM bit of the Extended Communication Control Register (ECCRn) is 0. The figure below illustrates the data format during a transmission in the synchronous operation mode.

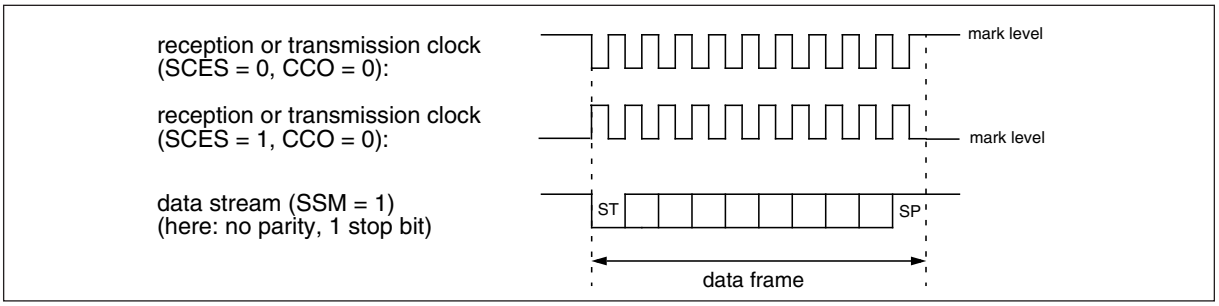
Figure 20-18. Transfer data format (operation mode 2)




### *20.7.2.2 Clock inversion and start/stop bits in mode 2*

If the SCES bit of the Extended Status/Control Register (ESCRn) is set the serial clock is inverted. Therefore in slave mode USART samples the data bits at the falling edge of the received serial clock. Note, that in master mode if SCES is set the clock signal's mark level is “0”. If the SSM bit of the Extended Communication Control Register (ECCRn) is set the data format gets additional start and stop bits like in asynchronous mode.

Figure 20-19. Transfer data format with clock inversion

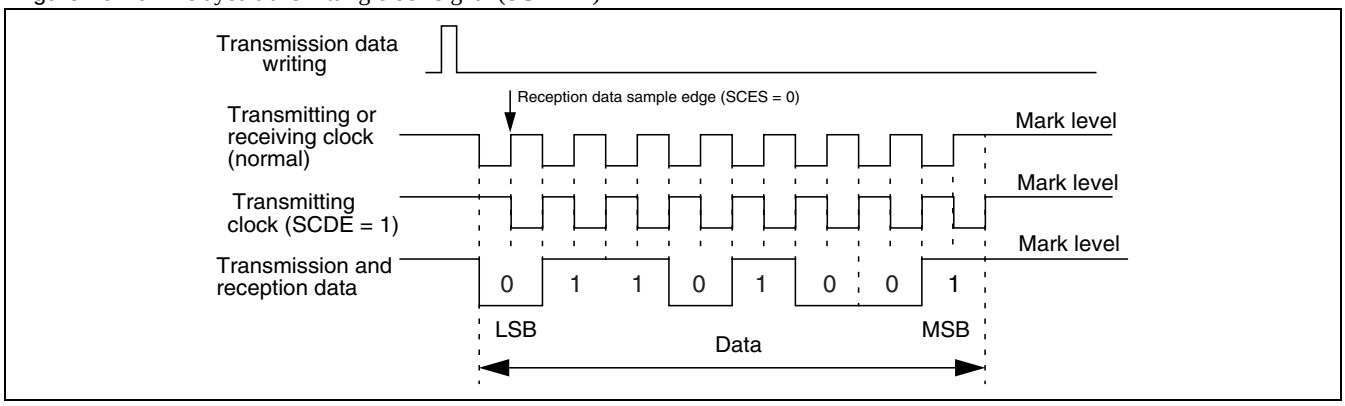



### *20.7.2.3 Clock supply:*

In operation mode 2, the number of clock cycles for the clock signal must be the same as the number of bits for the data including start and stop bits. If the MS bit of the ECCRn register is "0" (master mode) and the SCKE bit of the SMRn register is "1" (clock output enabled), the consistent clock cycles are generated automatically. If the MS bit of the ECCRn register is "1" (slave mode), make sure that correct clock cycles are generated by the other communication device. While there is no communication, the clock signal must be kept at "1" as the mark level.

If the SCDE bit of the ECCRn register is "1", the clock output signal is delayed by the half of the serial clock cycle as shown in figure 20.7-4. The operation is prepared for communication devices which use the falling edge of the serial clock signal for the data sampling.

Figure 20-20. Delayed transmitting clock signal (SCDE=1)



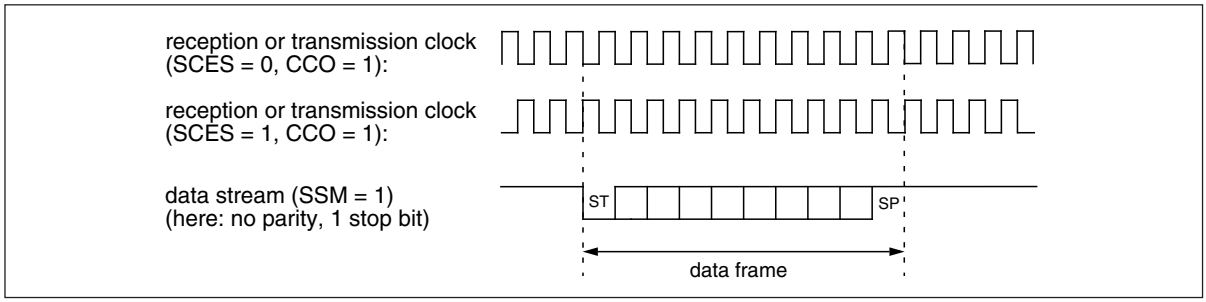

If the SCES bit of the ESCRn register is "1", the serial clock signal is inverted. Receiving data is sampled at the falling edge of the serial clock.

If the MS bit of the ECCRn register is "0" (master mode) and the SCKE bit of the SMRn register is "1" (clock output enabled), the output clock signal is also inverted.

While there is no communication, the clock signal must be kept at "0" as the mark level.

If the CCO bit of the ESCRn register is "1", the serial clock is signaled even while there is no data communication. Therefore it is recommended to specify the start/stop bits as shown in Figure 20-21.

Figure 20-21. Continuous clock output in mode 2




### *20.7.2.4 Error detection:*

If no start/stop bits is selected (ECCRn: SSM = 0) only overrun errors are detected.

### *20.7.2.5 Communication:*

For initialization of the synchronous mode, following settings have to be done:

**Baud rate generator registers (BGRn):**

Set the desired reload value for the dedicated baud rate reload counter.

**Serial mode control register (SMRn):**

MD1, MD0: “$10_B$” (Mode 2)

SCKE: “1” for the dedicated Baud Rate Reload Counter

“0” for external clock input

SOE:“1” for transmission and reception

“0” for reception only

**Serial control register (SCRn):**

RXE, TXE: set both of these flags to “0”

A/D: no Address/Data selection - don’t care

CL: automatically fixed to 8-bit data - don’t care

CRE: “1” to clear receive error flags.

**-- when SSM=0 (default):**

PEN, P, SBL: don’t care

**-- when SSM=1:**

PEN: “1” if parity bit is added/detected, “0” if not

P: “0” for even parity, “1” odd parity

SBL: “1” for 2 stop bits, “0” for 1 stop bit.

**Serial status register (SSRn):**

BDS: “0” for LSB first, “1” for MSB first

RIE: “1” if interrupts are used; “0” receive interrupts are disabled.

TIE: “1” if interrupts are used; “0” transmission interrupts are disabled.

**Extended communication control register (ECCRn):**

SSM: “0” if no start/stop bits are desired (normal); “1” for adding start/stop bits (special)

MS: “0” for master mode (USART generates the serial clock); “1” for slave mode (USART receives serial clock from the master device)

**Serial control register (SCRn):**

RXE, TXE: set one or both of these control bits to"1" to begin communication.

## 20.7.3 Operation with LIN Function (Operation Mode 3)

USART can be used either as LIN-Master or LIN-Slave. For this LIN function a special mode is provided. Setting the USART to mode 3 configures the data format to 8N1-LSB-first format.

### *20.7.3.1 Operation in asynchronous LIN mode (operation mode 3)*

**USART as LIN master**

In LIN master mode the master determines the baud rate of the whole sub bus, therefore slaves devices have to synchronize to the master. Therefore the desired baud rate remains fixed in master operation after initialization.

Writing a “1” into the LBR bit of the Extended Communication Control Register (ECCRn) generates a 13 - 16 bit time low-level on the SOT pin, which is the LIN synchronization break and the start of a LIN message. Thereby the TDRE flag of the Serial Status Register (SSRn) goes “0” and is reset to “1” after the break, and generates a transmission interrupt for the CPU (if TIE of SSRn is “1”).

The length of the Synchronization break to be sent can be determined by the LBL1/0 bits of the ESCRn as follows:

Table 20-15. LIN break length

| LBL1 | LBL0 | Length of Break |
|---|---|---|
| 0 | 0 | 13 Bit times |
| 1 | 0 | 14 Bit times |
| 0 | 1 | 15 Bit times |
| 1 | 1 | 16 Bit times |

The Synch Field is sent as byte data of 0x55 after the LIN break. To prevent a transmission interrupt, the 0x55 can be written to the TDRn just after writing the “1” to the LBR bit, although the TDRE flag is “0”. The internal transmission shifter waits until the LIN break has finished and shifts the TDRn value out afterwards. In this case no interrupt is generated after the LIN break and before the start bit.

### USART as LIN slave

In LIN slave mode USART has to synchronize to the master’s baud rate. If Reception is disabled (RXE = 0) but LIN break Interrupt is enabled (LBIE = 1) USART will generate a receive interrupt, if a synchronization break from the LIN master is detected, and indicates it with the LBD flag of the ESCRn. Writing “0” to this bit clears the receive interrupt request. The LIN slave may need to calculate the baud rate from the synch field. In this case, the time between the first falling edge to the fifth falling edge of the synch field is measured by the input capture module. For this purpose, the input capture module is connected to the LIN-USART with an internal signal. This internal signal changes from "0" to "1" at the first falling edge then "1" to "0" at the fifth falling edge. Therefore the input capture module should be set to detect both rising and falling edge. Also the input signal from the LIN-USART should be selected. The time measured by the input capture module represents 8 times of the baud rate clock cycle.

Therefore, baud rate setting value is summarized as follows:

without timer overflow : BGRn value = (b-a)/8

with timer overflow : BGRn value = (max + b-a)/8

where max is the timer maximum value at the overflow occurs.

where a is the value of the ICU counter register after the first Interrupt

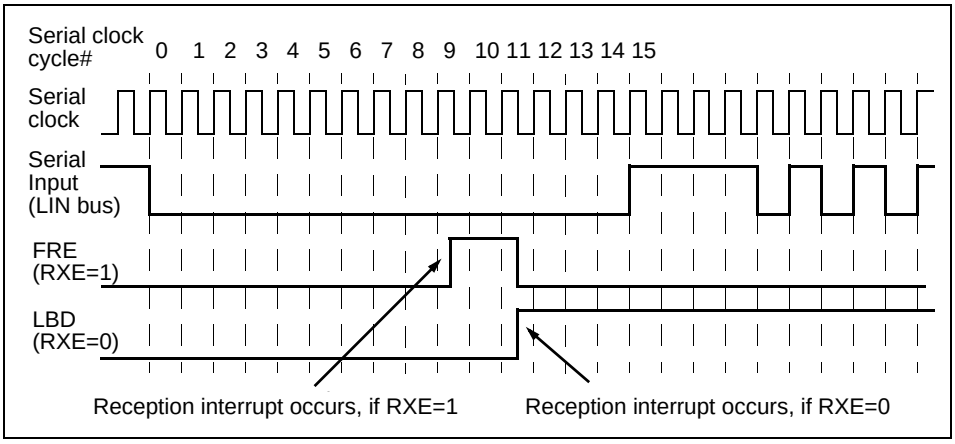
where b is the value of the ICU counter register after the second Interrupt

For the correspondence between other USARTs and ICUs, see 16-bit Free-Running Timer on page 302.

### LIN Synch Break Detection Interrupt and Flags

If a LIN Synch synchronization break is detected in the slave mode, the LIN Break Detected (LBD) flag of the ESCRn is set to “1”. This causes an interrupt, if the LIN Break Interrupt Enable (LBIE) bit is set.

Figure 20-22. LIN synch break detection and flag set timing.



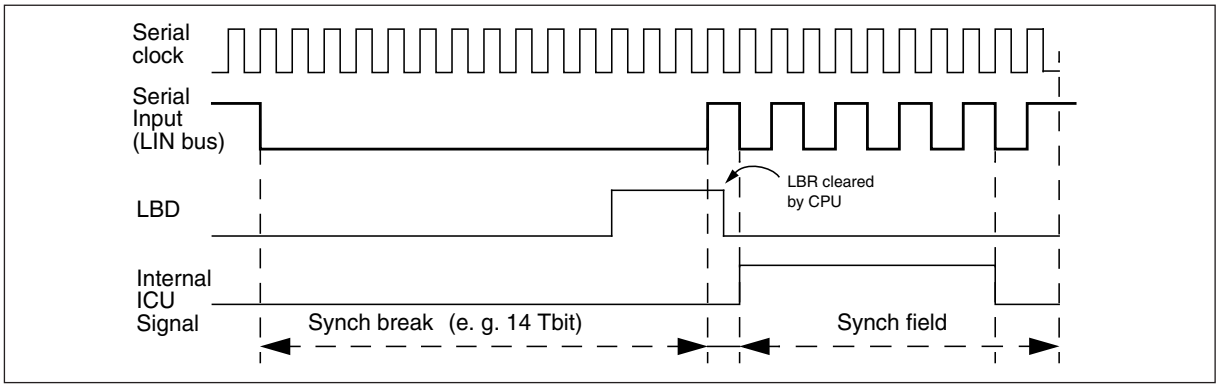

The figure above demonstrates the LIN synch break detection and flag set timing.

Note, that if reception is enabled (RXE = 1) and receive interrupt is enabled (RIE = 1) the Reception Data Framing Error (FRE) flag bit of the SSRn will cause a receive interrupt 2 bit times (“8N1”) earlier than the LIN break interrupt, so it is recommended to turn off RXE, if a LIN break is expected.

LBD is only supported in operation mode 3.

The Figure 20-23 shows a typical start of a LIN message frame and the behavior of the USART.

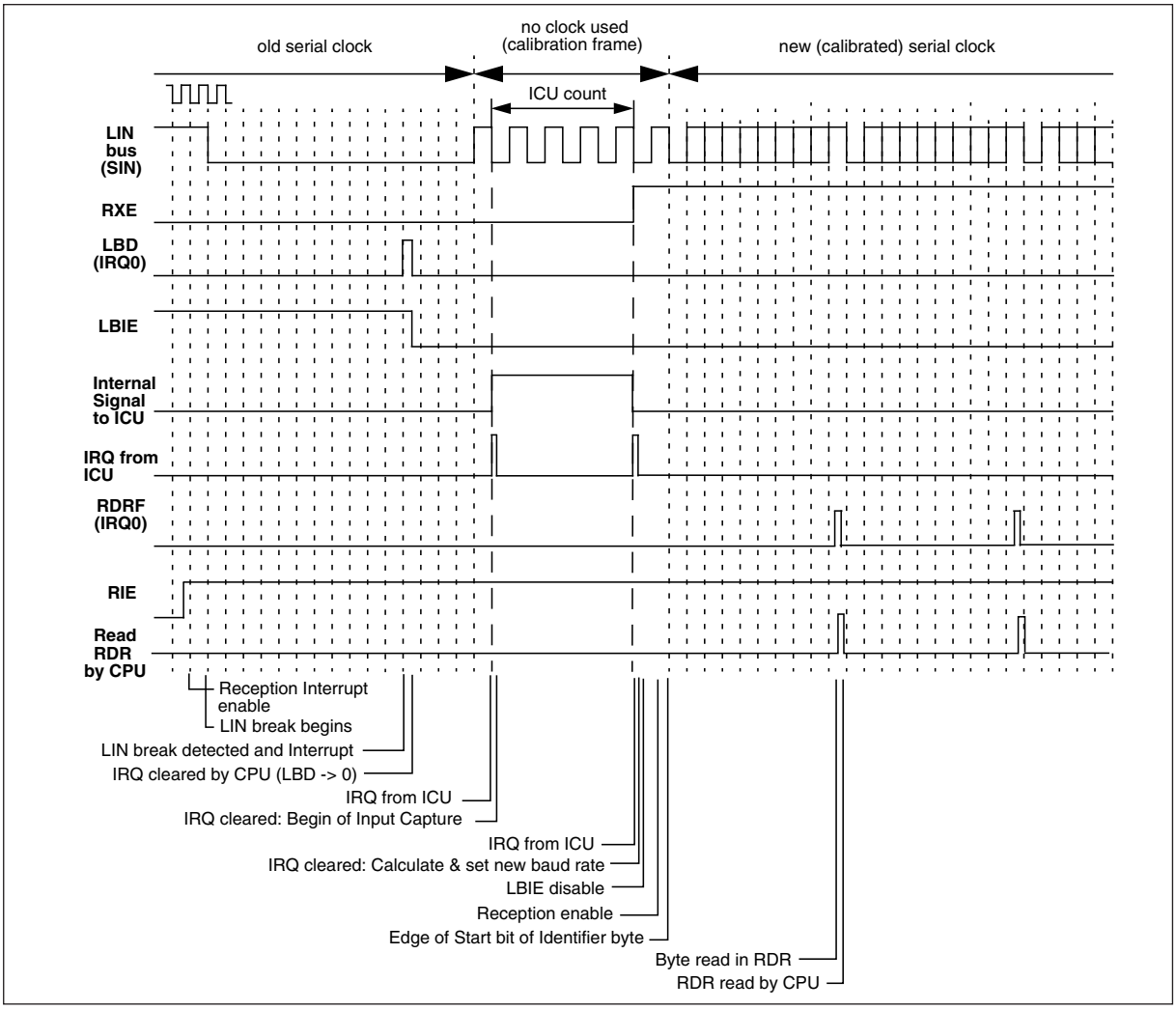
Figure 20-23. USART behavior as slave in LIN mode




LIN Synch Break Detection Interrupt and Flags

### LIN bus timing

Figure 20-24. LIN bus timing and USART signals




## 20.7.4 Direct Access to Serial Pins

USART allows the user to directly access to the transmission pin (SOTn) or the reception pin (SINn).

### USART direct pin access

The USART provides the ability for the software to access directly to serial input or output pin. The software can always monitor the incoming serial data by reading the SIOP bit of the ESCRn. If setting the Serial Output Pin direct access Enable (SOPE) bit of the ESCRn the software can force the SOTn pin to a desired value. Note that this access is only possible if the transmission shift register is empty (i. e. no transmission activity).

In LIN mode this function can be used for reading back the own transmission and is used for error handling if something is physically wrong with the single-wire LIN-bus.

### Notes:

- Write the desired value to ESCRn:SIOP before enabling the output pin access to prevent undesired output level because ESCRn:SIOP holds the last written value.

- During a Read-Modify-Write operation the ESCRn:SIOP bit returns the actual value of the SOTn pin in the read cycle instead of the value of SINn during a normal read instruction.

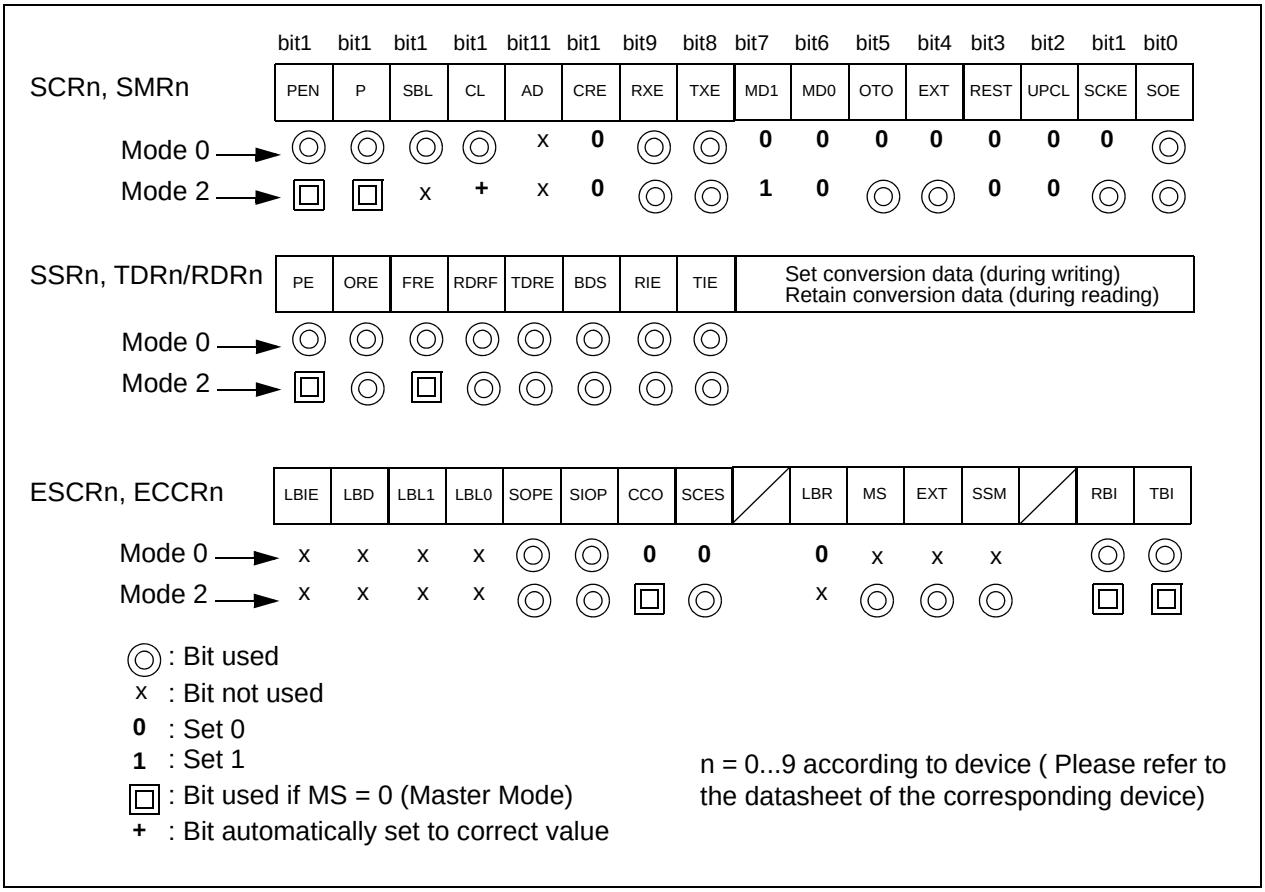
## 20.7.5 Bidirectional Communication Function (Normal Mode)

In operation mode 0 or 2, normal serial bidirectional communication is available. Select operation mode 0 for asynchronous communication and operation mode 2 for synchronous communication.

### *20.7.5.1 Bidirectional communication function*

The settings shown in Figure 20-25 are required to operate USART in normal mode (operation mode 0 or 2).

Figure 20-25. Settings for USART operation mode 0 and 2




**Inter-CPU connection**

As shown in Figure 20-26, interconnect two CPUs in USART mode 2.

Figure 20-26. Connection example of USART mode 2 bidirectional communication

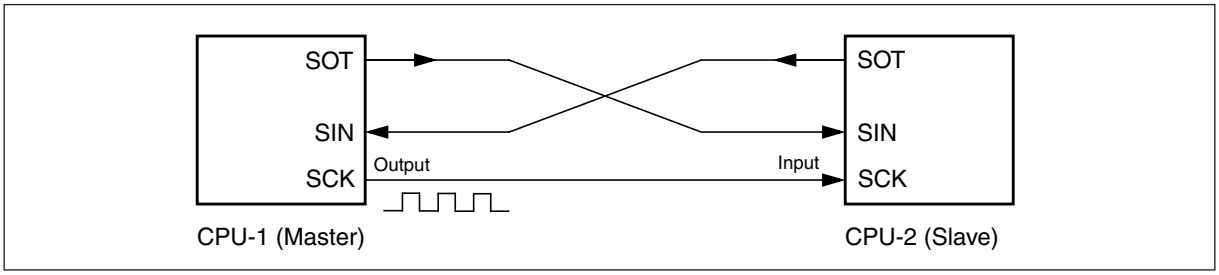
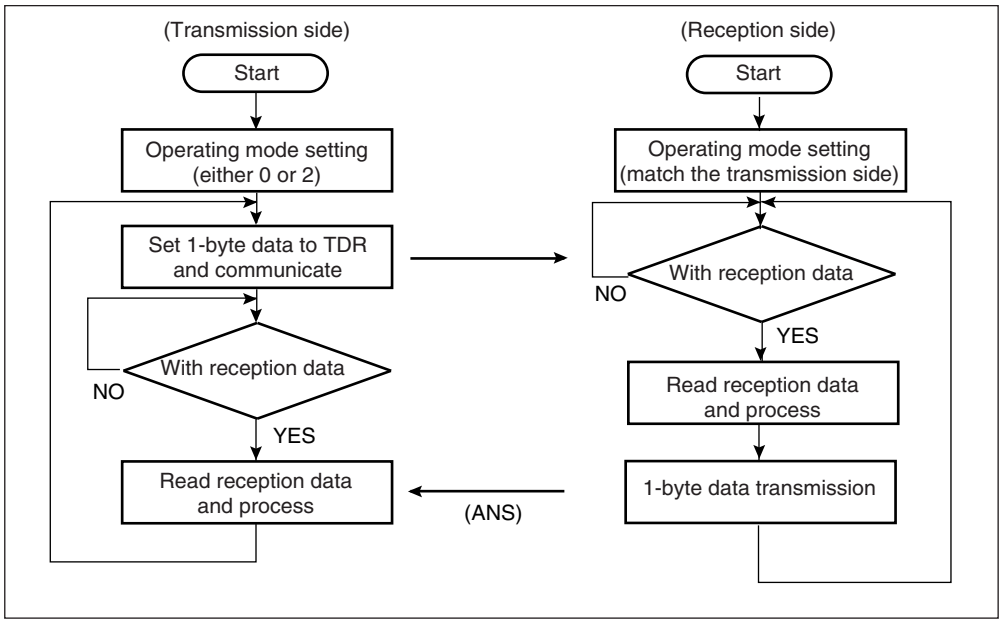


Figure 20-27. Example of master-slave communication flowchart



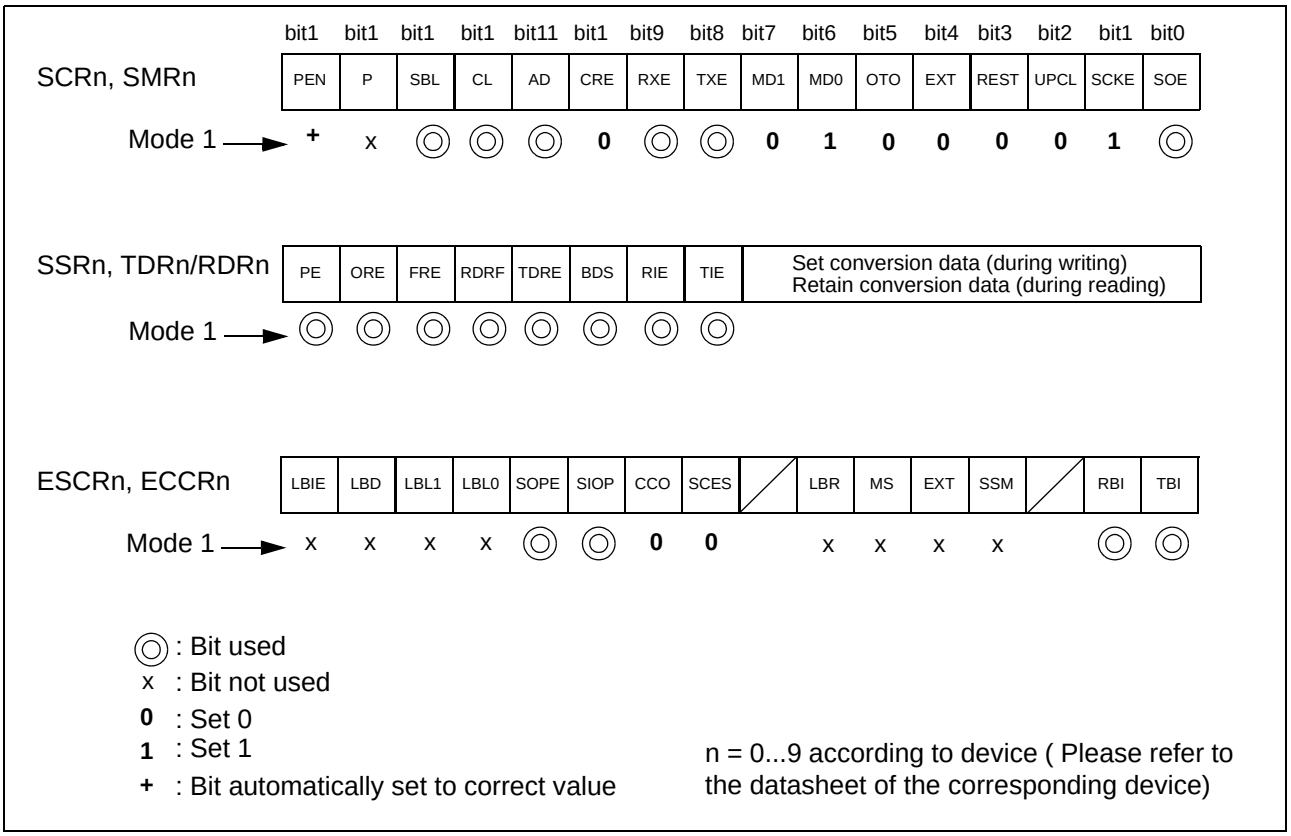
## 20.7.6 Master-Slave Communication Function (Multiprocessor Mode)

USART communication with multiple CPUs connected in master-slave mode is available for both master or slave systems.

### *20.7.6.1 Master-slave communication function*

The settings shown in Figure 20-28 are required to operate USART in multiprocessor mode (operation mode 1).

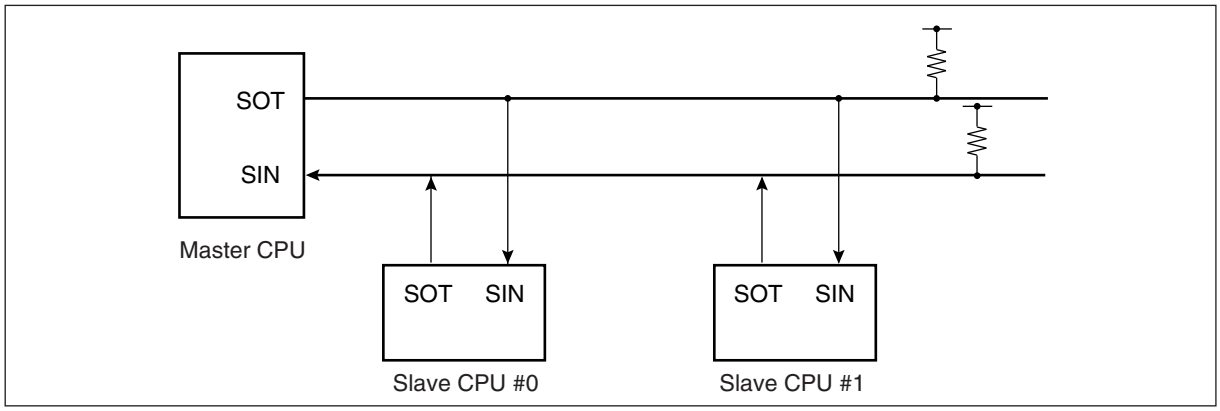
Figure 20-28. Settings for USART operation mode 1




**Inter-CPU connection**

As shown in Figure 20-29, a communication system consists of one master CPU and multiple slave CPUs connected to two communication lines. USART can be used for the master or slave CPU.

Figure 20-29. Connection example of USART master-slave communication



### Function selection

Select the operation mode and data transfer mode for master-slave communication as shown in Table 20-16.

Table 20-16. Selection of the master-slave communication function

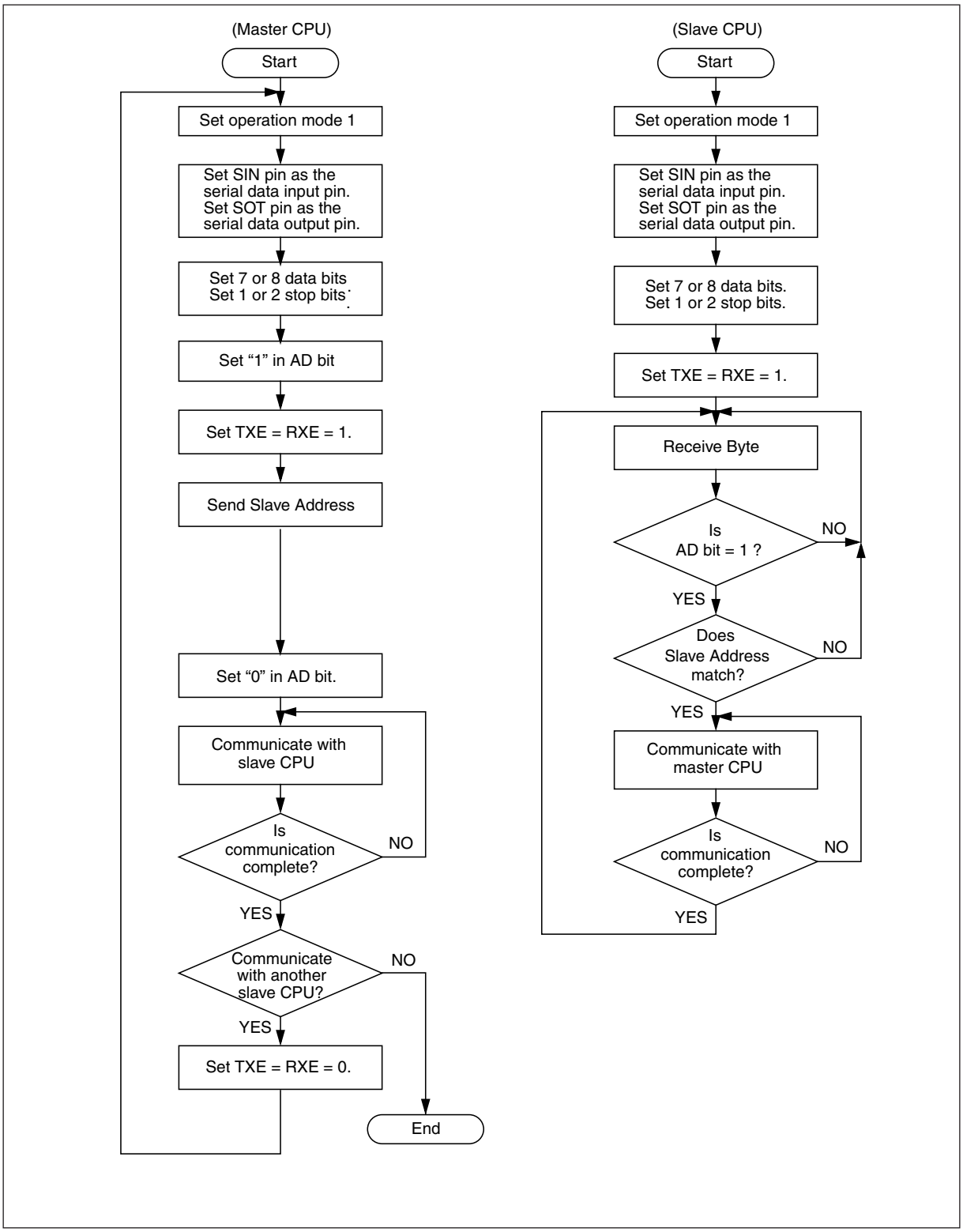
| | Operation mode | | Data | Parity | Synchronization method | Stop bit | Bit direction |
|---|---|---|---|---|---|---|---|
| | Master CPU | Slave CPU | | | | | |
| Address transmission and reception | Mode 1 (transmit/receive AD-bit) | Mode 1 (transmit/receive AD-bit) | AD="1" + 7- or 8-bit address | None | Asynchronous | 1 or 2 bits | LSB or MSB first |
| Data transmission and reception | | | AD="0" + 7- or 8-bit data | | | | |

### Communication procedure

When the master CPU transmits address data, communication starts. The A/D bit in the address data is set to 1, and the communication destination slave CPU is selected. Each slave CPU checks the address data using a program. When the address data indicates the address assigned to a slave CPU, the slave CPU communicates with the master CPU.

Figure 20-30 shows a flowchart of master-slave communication (multiprocessor mode).

Figure 20-30. Master-slave communication flowchart



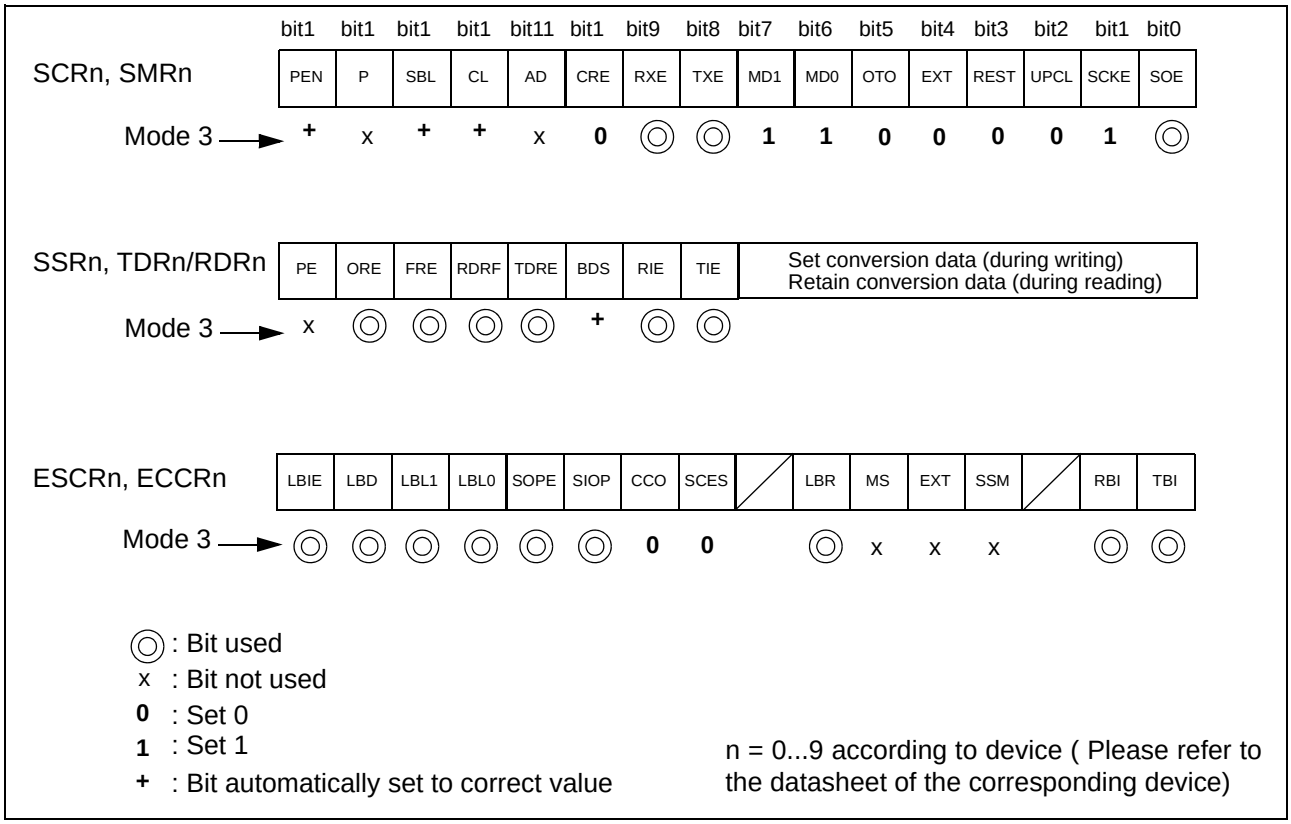
## 20.7.7 LIN Communication Function

USART communication with LIN devices is available for both LIN master or LIN slave systems.

### *20.7.7.1 LIN-master-slave communication function*

The settings shown in the figure below are required to operate USART in LIN communication mode (operation mode 3).

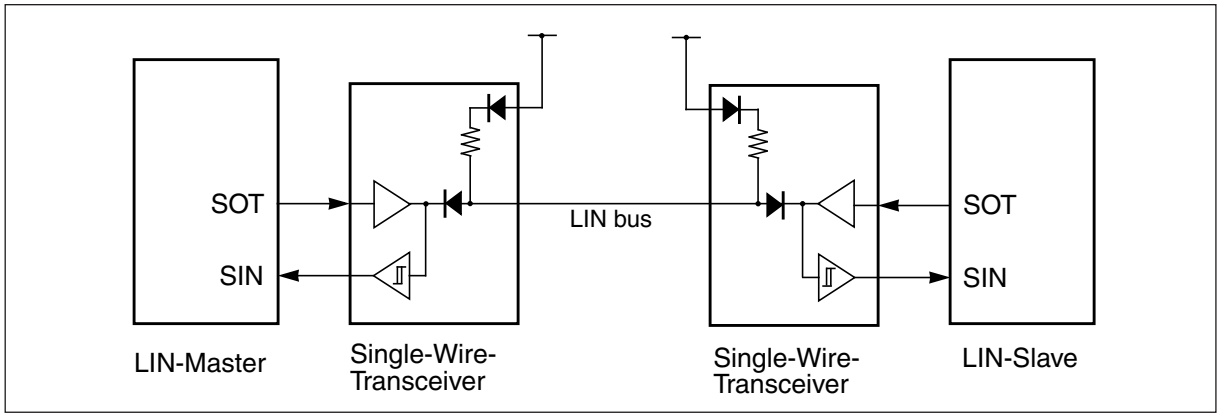
Figure 20-31. Settings for USART in operation mode 3 (LIN)




**LIN device connection**

As shown in the figure below, a communication system of one LIN-Master device and a LIN-Slave device. USART can operate both as LIN-Master or LIN-Slave.

Figure 20-32. Connection example of a small LIN-Bus system

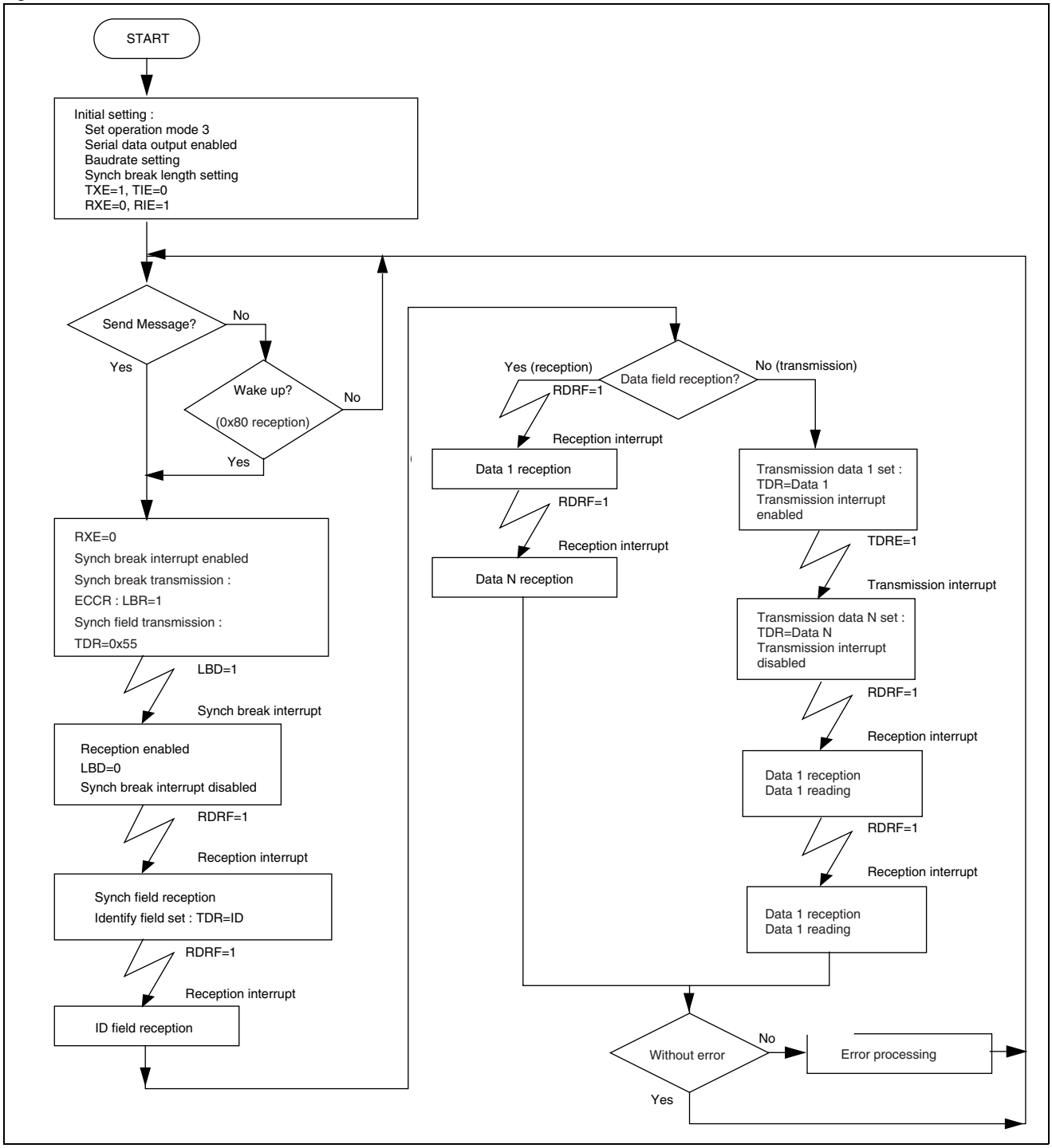


## 20.7.8 Sample Flowcharts for USART in LIN communication (Operation Mode 3)

This section contains sample flowcharts for USART in LIN communication.

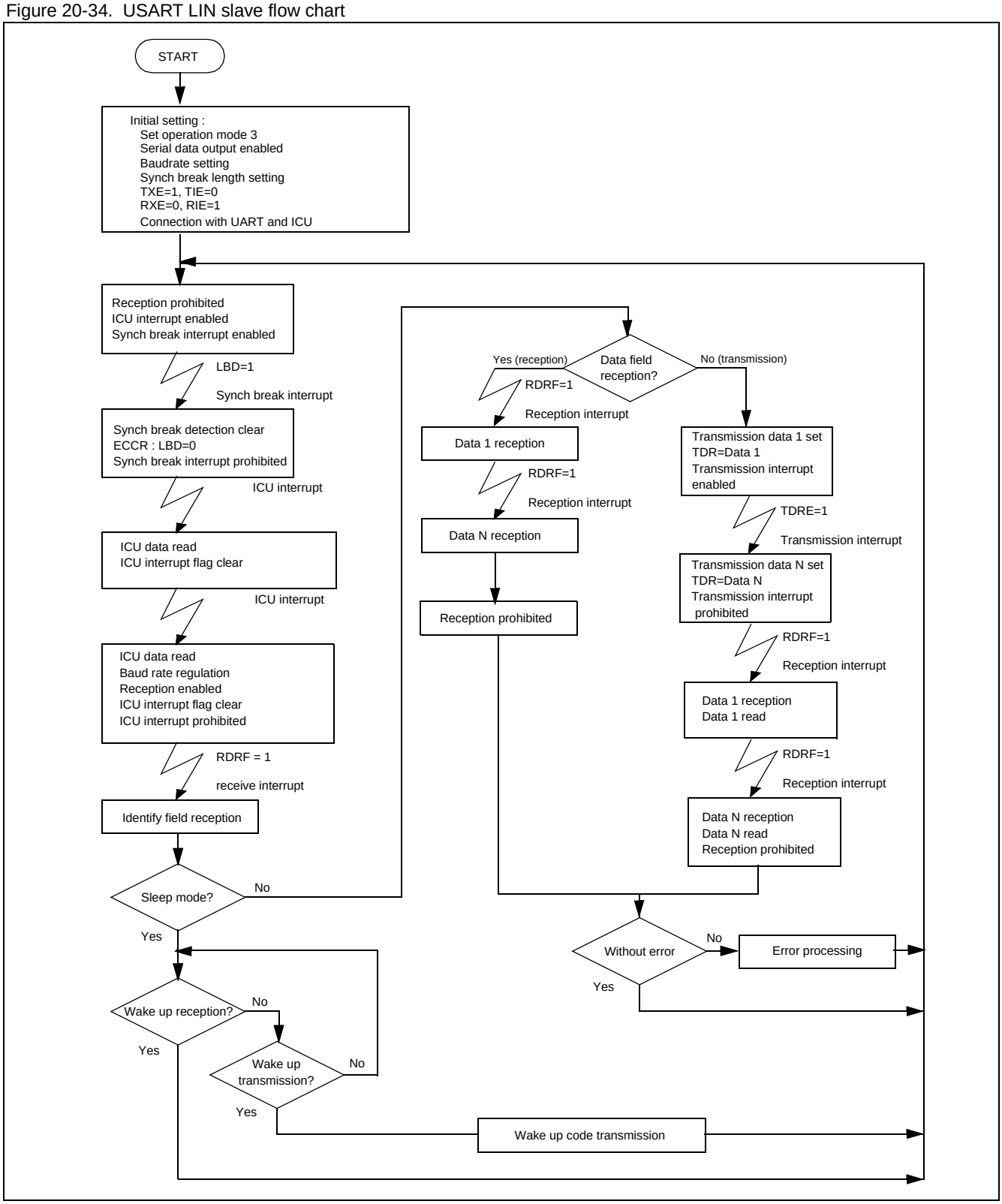
### *20.7.8.1 USART as master device*

Figure 20-33. USART LIN master flow chart



### *20.7.8.2 USART as slave device*

Figure 20-34. USART LIN slave flow chart



# 20.8 Notes on Using USART

Notes on using USART are given below.

## 20.8.1 Notes on using USART

### Enabling operation

In USART, the control register (SCRn) has TXE (transmission) and RXE (reception) operation enable bits. Both, transmission and reception operations, must be enabled before the communication starts because they have been disabled as the default value (initial value). The operation can also be canceled by disabling these bits.

### Communication mode setting

Set the communication mode while the system is not operating. If the mode is changed during transmission or reception, the transmission or reception is stopped and possible data will be lost.

### Transmission interrupt enabling timing

The default (initial value) of the transmission data empty flag bit (SSRn:TDRE) is “1” (no transmission data and transmission data write enable state). A transmission interrupt request is generated as soon as the transmission interrupt request is enabled (SSRn:TIE=1). Be sure to set the TIE flag to “1” after setting the transmission data to avoid an immediate interrupt.

### Using LIN operation mode 3

The LIN features are available in mode 3 (transmitting, receiving synch break), but using mode 3 sets the USART data format automatically to LIN format (8N1, LSB first). Note, that the length of the synch break for transmission is variable but for reception it is fixed 11-bit time.

### Changing operation settings

It is strongly recommended to reset the USART after changing operation settings. Particularly if (for example) start-/stop-bits added to or removed from the data format.

It is recommended to disable the communication (RXE = "0", TXE = "0"), if the USART setting or mode is changed or the USART is initialized.

### Note:

If settings in the Serial Mode Register (SMRn) are desired, it is not useful to set the UPCL bit at the same time to reset the USART. The correct operation settings are not guaranteed in this case. Thus it is recommended to set the bits of the SMRn and then to set them again plus the UPCL bit.

### LIN slave settings

Set the baud rate before receiving the first LIN synch break for the slave operation. Otherwise, duration of the synch break can not be correctly checked against the minimum requirement of the LIN specification (13 master bit time and 11 slave bit time).

### Bus idle function

The Bus Idle Function cannot be used in synchronous slave mode mode 2.

### AD bit (serial control register (SCRn): address/data type select bit)

Special care has to be taken when using the AD bit (Address-Data-Bit for multiprocessor mode 1) of the Serial Control Register. This bit is both a control and a flag bit, because writing to it sets the AD bit for transmission, whereas reading from it returns the last received AD bit. Internally, the received and the transmitted value are stored in different registers, but in Read-Modify-Write instructions, the received value is read, modified and then written back for transmission. This can lead to a wrong value in the AD bit, when one of the other bits in the same register is accessed by an instruction of this kind.

### Clearing reception errors

Clearing reception errors resets the reception state machine. Therefore check any reception errors before the next start-bit or start condition is met, to not disturb any ongoing reception.

### LIN Synch field wait state

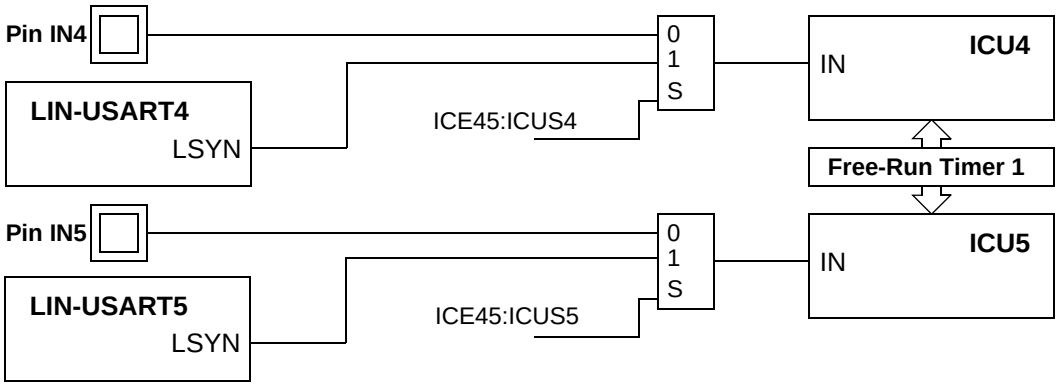
In mode 3 (LIN operation), the LBD bit in the ESCRn register is set to "1" if the input signal is kept at "0" for more than equal to 11-bit time. Then the USART waits for the following synch field to be received. If the USART is set into this state for other reasons than the synch break, it should be initialized by the software reset (SMRnSCRn:UPCL=1).

In mode 3, LIN-Break detection is always working in the background and is level sensitive. Be careful in case of a bus error (bus always dominant). The LIN-Break detection flag (LBD) will go "1" or stay "1" after each 11 bit times independent from enabled or disabled LIN break detection interrupt. If you use LIN-Break detection interrupt, be sure to check and clear always this flag in your receive interrupt handler.

### Baud Rate detection using the Input Capture Units (ICU)

The USARTs provide the signal LSYN that can be connected to the ICU so that LSYN's pulse length can be measured to derive the baud rate. The connection of the LSYN signals to the ICUs is controlled by the Input capture edge register ICE(2n)(2n+1). For assignment of ICUs and USARTs please check chapter Input Capture Unit Source Select for LIN-USART on page 27.

Figure 20-35. Baud Rate Detection Using the Input Capture Units




If the ICUSx bit in the ICE(2n)(2n+1) register is cleared and the peripheral resource input is enabled, the ICU is connected to its corresponding input pin IN. If the ICUSx bit is set to '1' the corresponding LIN-USART is connected to the ICU.

The user has to take into account that: Different ICUs share one free running timer (prescaler).

Effects of reception errors and CRE bit.

CRE resets reception state machine and next falling edge at SINn starts reception of new byte. Therefore either set CRE bit immediately (within half bit time) after receiving errors to prevent data stream desynchronization or wait an application dependent time after receiving errors and set CRE, when SINn is idle.

Figure 20-36. Timing of the CRE bit

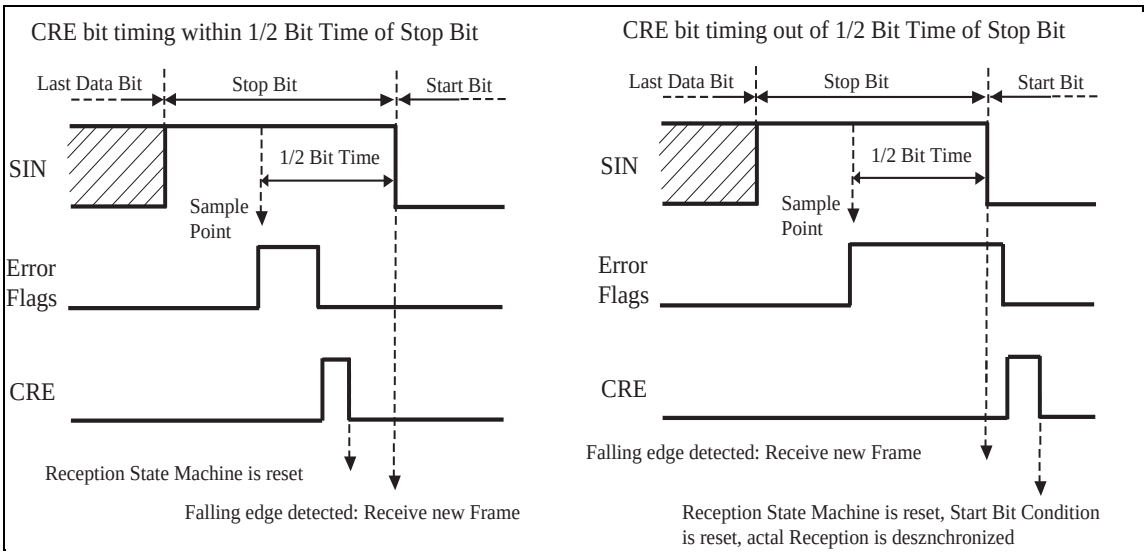
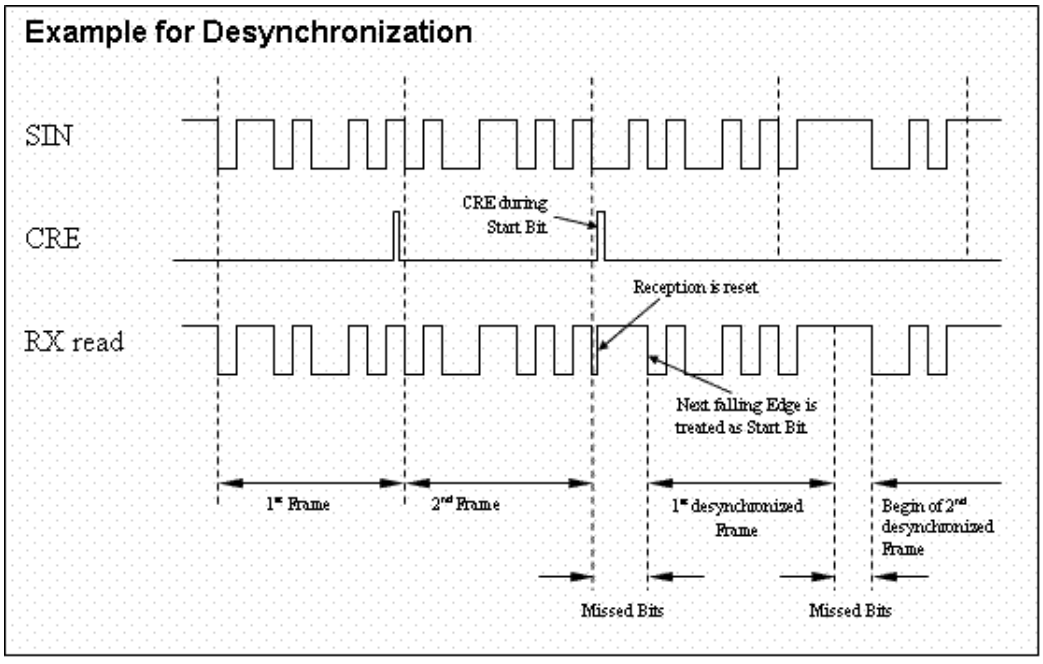


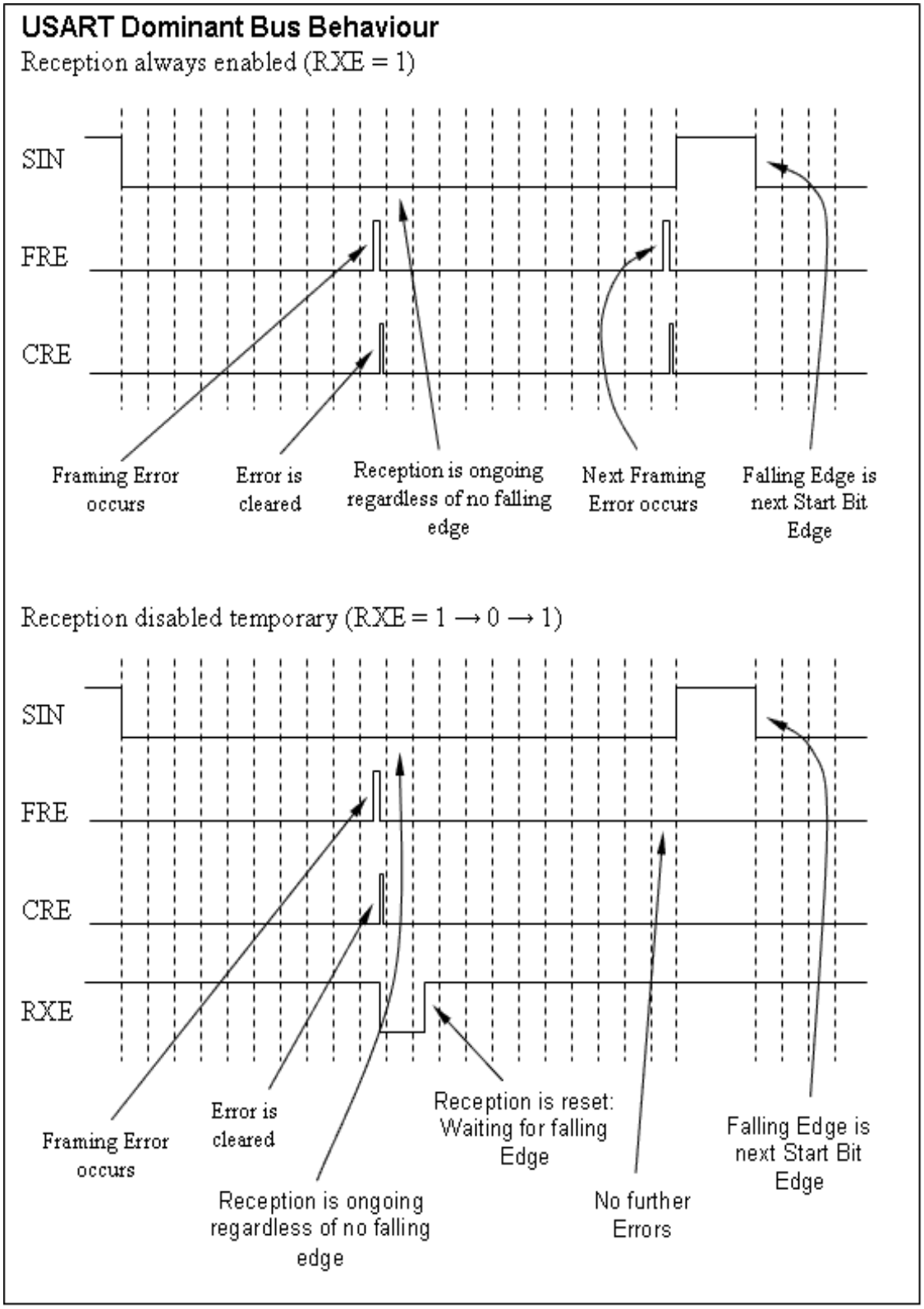

Figure 20-37. Data Stream Synchronization




**Note:**

In case a framing error occurred (stop bit: SINn = "0") and the next start bit (SINn = "0") follows immediately, this start bit is recognized regardless of no falling edge before. This is used to keep the USART synchronized to the data stream and to determine bus always dominant errors (See Figure 20-38 upper figure) by producing next framing errors, if a recessive stop bit is expected. If this behaviour is not wanted, please disable the reception temporarily (RXE = 1 -> 0 -> 1) after framing error. In this case, reception goes on at next falling edge on SINn. (See Figure 20-38 lower figure).

Figure 20-38. USART Dominant Bus Behaviour



# 21. 400 kHz I²C Interface

This section describes the functions and operation of the fast I²C interface.

21.1 I2C Interface Overview

21.2 I2C Interface Registers

21.3 I2C Interface Operation

21.4 Programming Flow Charts

## 21.1 I²C Interface Overview

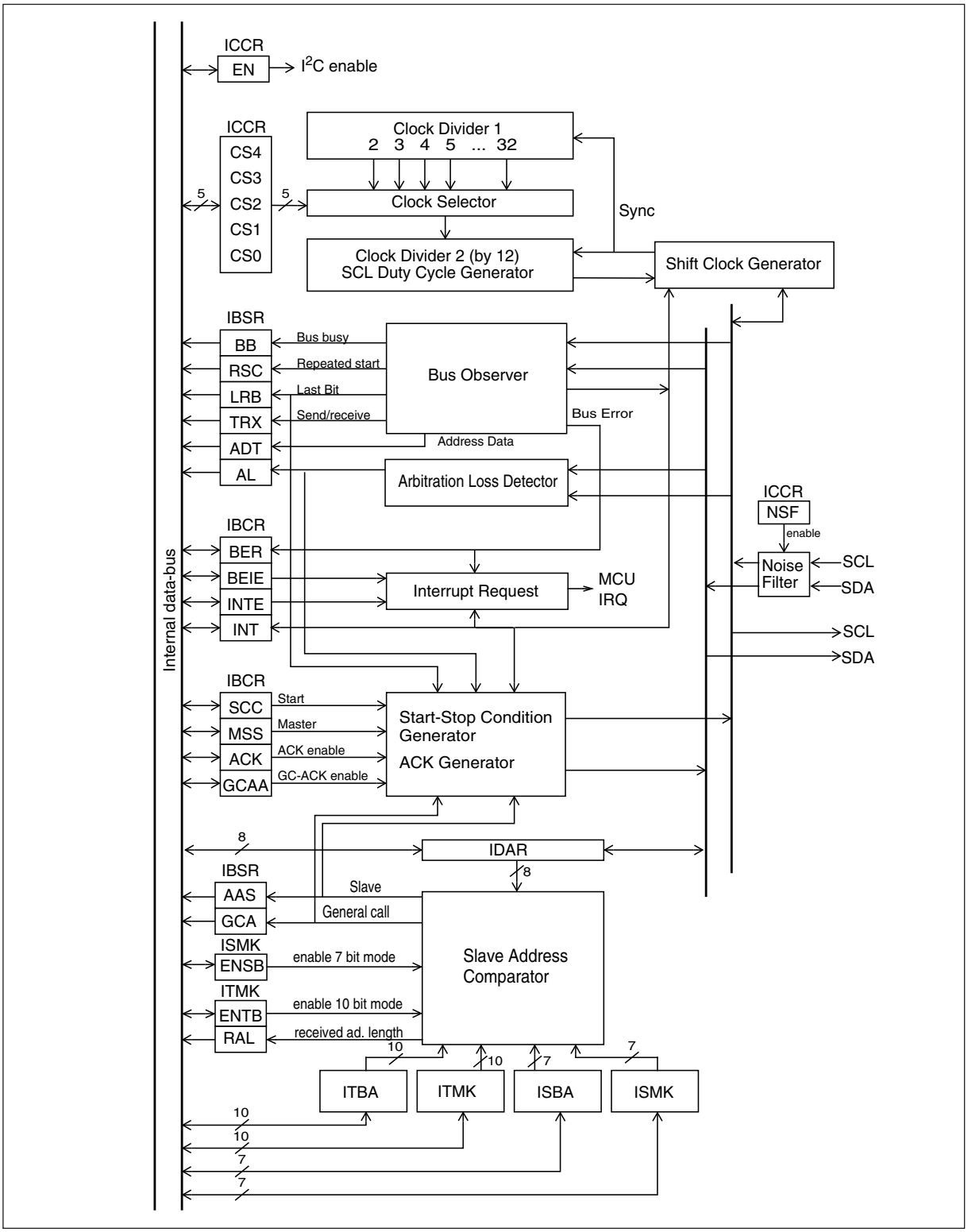
The I²C interface is a serial I/O port supporting the Inter IC bus, operating as a master/slave device on the I²C bus.

### Features

- Master/slave transmitting and receiving functions
- Arbitration function
- Clock synchronization function
- General call addressing support
- Transfer direction detection function
- Repeated start condition generation and detection function
- Bus error detection function
- 7 bit addressing as master and slave
- 10 bit addressing as master and slave
- Possibility to give the interface a seven and a ten bit slave address
- Acknowledging upon slave address reception can be disabled (Master-only operation)
- Address masking to give interface several slave addresses (in 7 and 10 bit mode)
- Up to 400 KBit transfer rate
- Possibility to use built-in noise filters for SDA and SCL
- Can receive data at 400 KBit if peripheral clock CLKP1 is higher than 6 MHz regardless of prescaler setting
- Can generate MCU interrupts on transmission and bus error events
- Supports being slowed down by a slave on bit and byte level

The I²C interface does not support SCL clock stretching on bit level since it can receive the full 400 kBit data rate if the peripheral clock CLKP1 is higher than 6 MHz regardless of the prescaler setting. However, clock stretching on byte level is performed since SCL is pulled low during an interrupt (INT="1" in IBCR register).

Figure 21-1. Block diagram

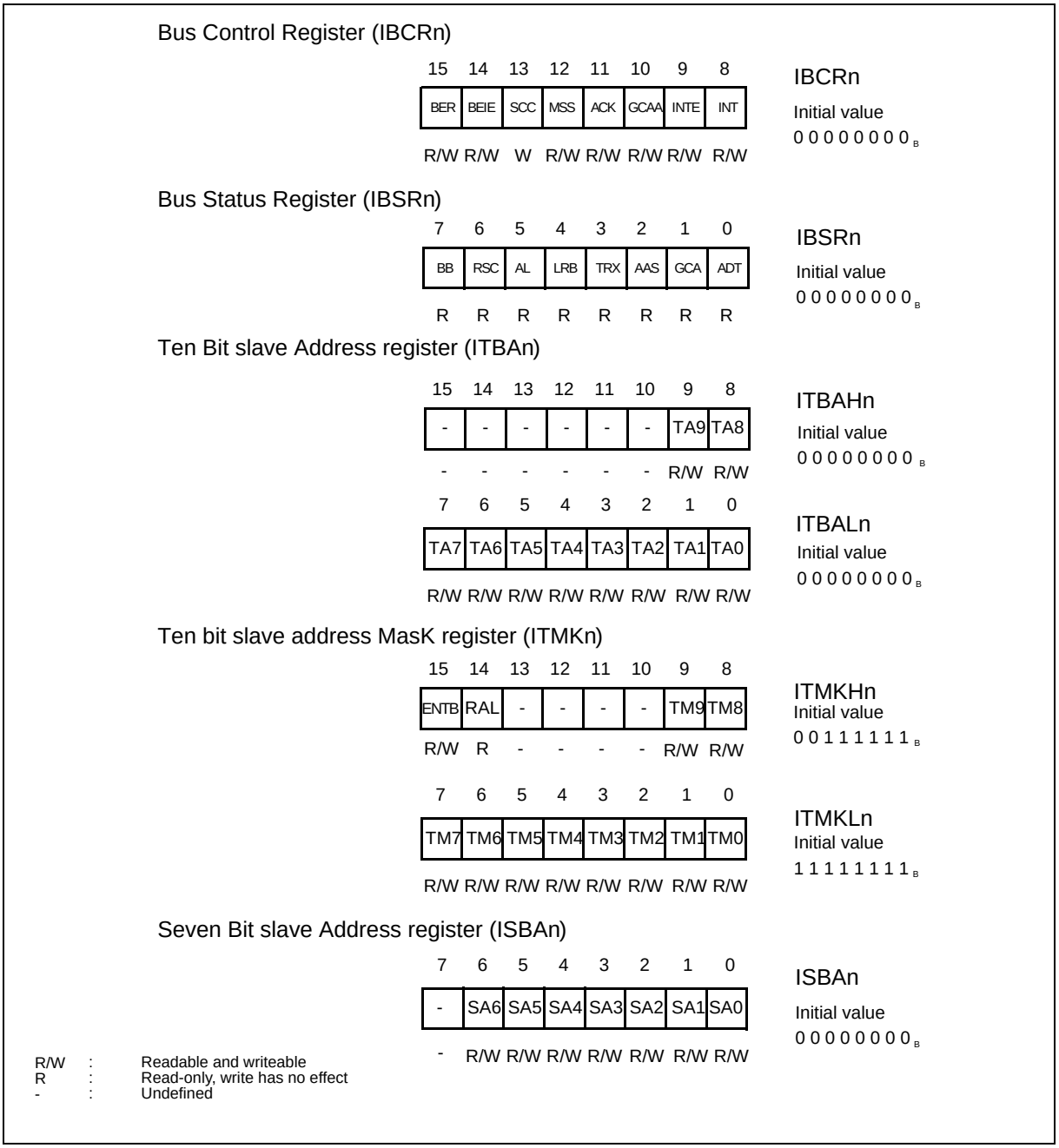


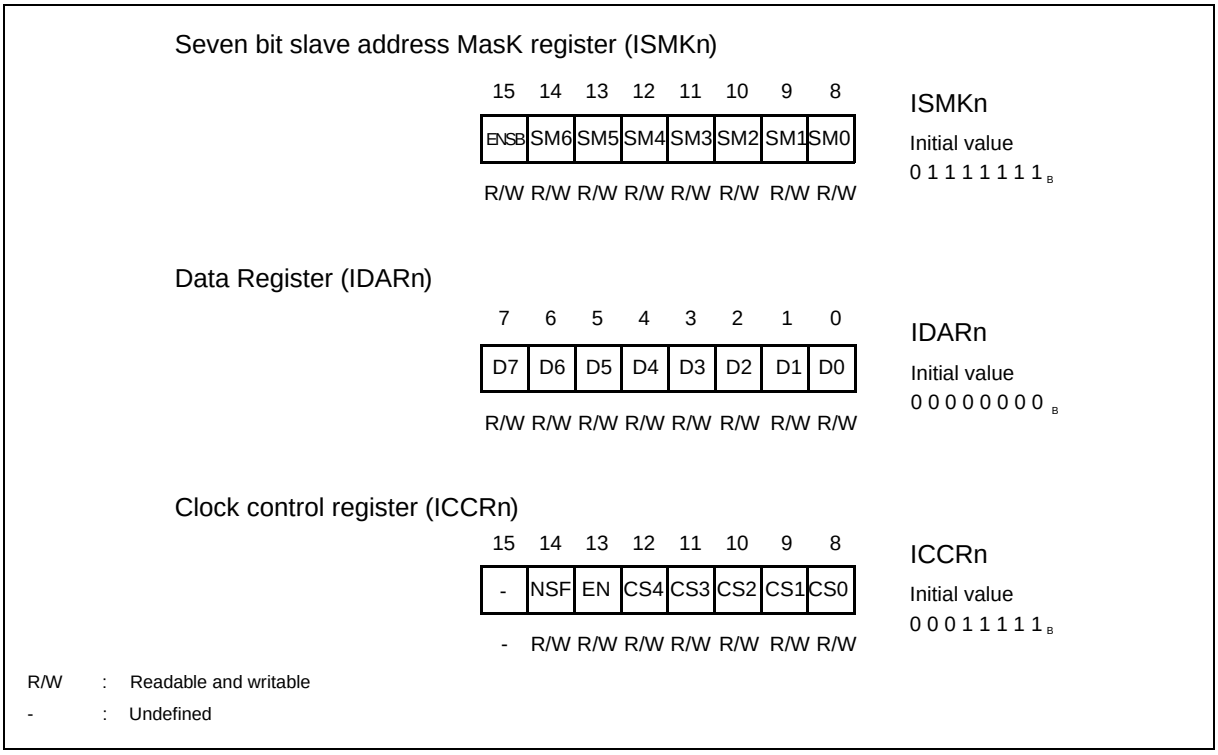
## 21.2 I²C Interface Registers

This section describes the function of the I²C interface registers in detail.

### I²C Interface registers

Figure 21-2. I²C Interface registers






## 21.2.1 Bus Status Register (IBSRn)

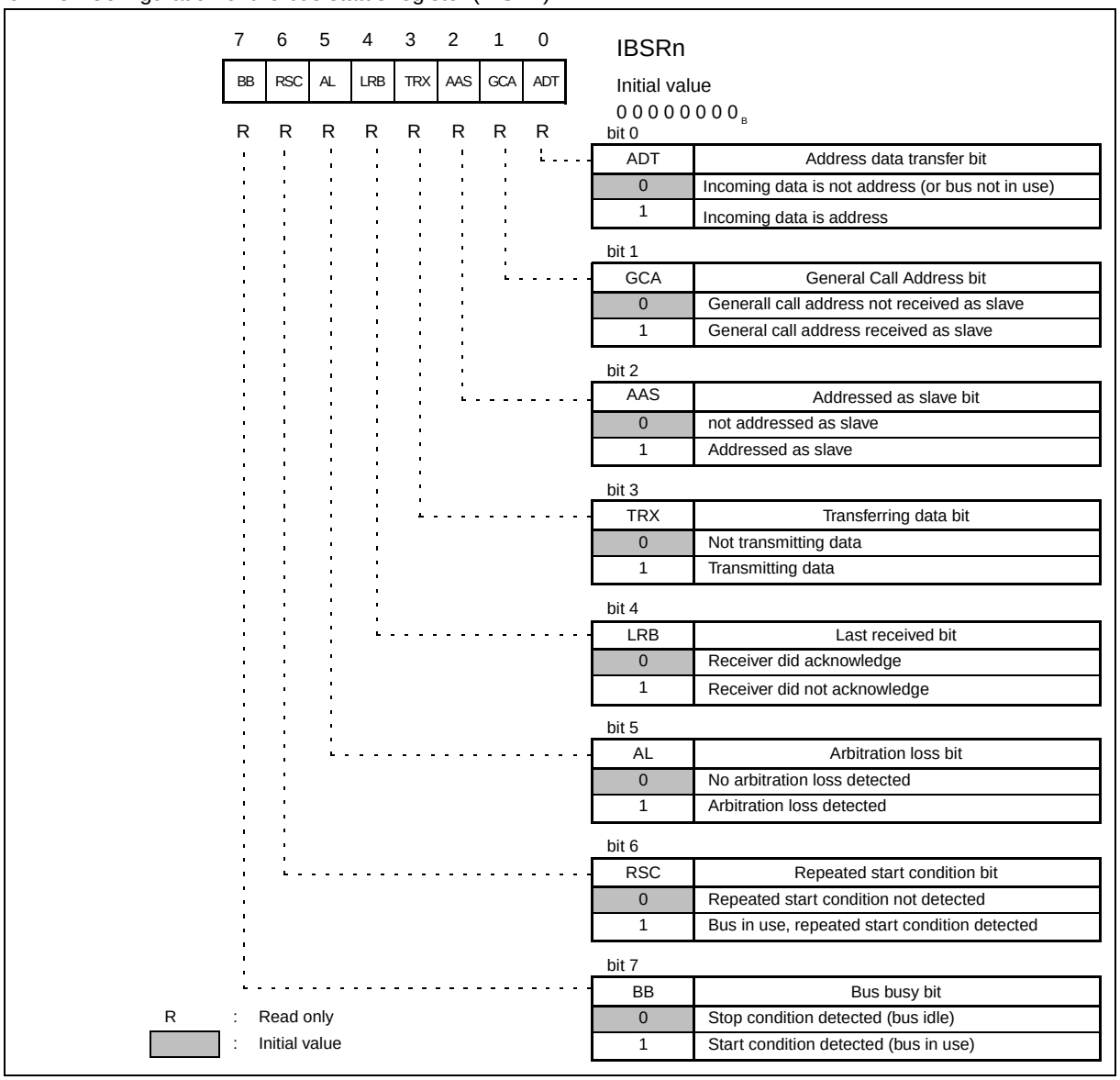
The bus status register (IBSRn) has the following functions:

- Bus busy detection
- Repeated start condition detection
- Arbitration loss detection
- Acknowledge detection
- Data transfer direction indication
- Addressing detection as slave
- General call address detection
- Address/data detection

### Bus status register (IBSRn)

This register is read-only, all bits are controlled by the hardware. All bits are cleared if the interface is not enabled (EN = "0" in ICCRn).

Figure 21-3. Configuration of the bus status register (IBSRn)




## Bus status register (IBSRn) contents

Table 21-1. Function of each bit of the bus status register (IBSRn) (Sheet 1 of 2)

| Bit name | | Function |
|---|---|---|
| bit 7 | BB:<br>Bus busy bit | This bit indicates the status of the $I^2C$ bus.<br>"0": Stop condition detected (bus idle)<br>"1": Start condition detected (bus in use)<br>This bit is set to "1" if a start condition is detected. It is reset upon a stop condition. |
| bit 6 | RSC:<br>Repeated start condition bit | This bit indicates detection of a repeated start condition.<br>"0": Repeated start condition not detected.<br>"1": Repeated start condition detected (bus in use).<br>This bit is cleared at the end of an address data transfer (ADT="0") or detection of a stop condition. |

Table 21-1. Function of each bit of the bus status register (IBSRn) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 5 | AL:<br>Arbitration loss bit | This bit indicates an arbitration loss.<br>"0": No arbitration loss detected.<br>"1": Arbitration loss occurred during master sending.<br>This bit is cleared by writing "0" to the INT bit or by writing "1" to the MSS bit in the IBCR register.<br>An arbitration loss occurs if:<br>- the data sent does not match the data read on the SDA line at the rising SCL edge<br>- a repeated start condition is generated by another master in the first bit of a data byte<br>- The interface could not generate a start or stop condition because signal transition caused from "1" to "0" by a certain external condition observed at the SCL line. |
| bit 4 | LRB:<br>Last received bit | This bit is used to indicate the acknowledge from the receiving device.<br>"0": Receiver acknowledged.<br>"1": Receiver did not acknowledge.<br>It is changed by the hardware upon reception of bit 9 (acknowledge bit) and is also cleared by a start or stop condition. |
| bit 3 | TRX:<br>Transferring data bit | This bit indicates data transmission operation.<br>"0": Not transmitting data.<br>"1": Transmitting data.<br>It is set to "1":<br>- if a start condition was generated in master mode<br>- if addressed as slave in read access.<br>It is set to "0" if:<br>- the bus is idle (BB="0")<br>- an arbitration loss occurred<br>- "1" is written to the SCC bit during master interrupt (MSS="1" and INT="1")<br>- the MSS bit being cleared during master interrupt (MSS="1" and INT="1")<br>- the interface is in slave mode and the last transferred byte was not acknowledged<br>- the interface is in slave mode and it is receiving data<br>- the interface is in master mode and is reading data from a slave |
| bit 2 | AAS:<br>Addressed as slave bit | This bit indicates detection of a slave addressing.<br>"0": Not addressed as slave.<br>"1": Addressed as slave.<br>This bit is cleared by a (repeated-) start or stop condition. It is set if the interface detects its seven and/or ten bit slave address. |
| bit 1 | GCA:<br>General call address bit | This bit indicates detection of a general call address (0x00).<br>"0": General call address not received as slave.<br>"1": General call address received as slave.<br>This bit is cleared by a (repeated-) start or stop condition. |
| bit 0 | ADT:<br>Address data transfer bit | This bit indicates the detection of an address data transfer.<br>"0": Incoming data is not address data (or bus is not in use).<br>"1": Incoming data is address data.<br>This bit is set to "1" by a start condition. It is cleared after the second byte if a ten bit slave address header with write access is detected, else it is cleared after the first byte.<br>This bit is also cleared when:<br>-"0" is written to the MSS bit during a master interrupt (MSS="1" and INT="1" in IBCR)<br>-"1" is written to the SCC bit during a master interrupt (MSS="1" and INT="1" in IBCR)<br>- the INT bit is being cleared<br>- the beginning of every byte transfer if the interface is not involved in the current transfer as master or slave |

## 21.2.2 Bus Control Register (IBCRn)

The Bus Control Register (IBCRn) has the following functions:

Interrupt enabling flags

Interrupt generation flag

Bus error detection flag

Repeated start condition generation

Master / slave mode selection

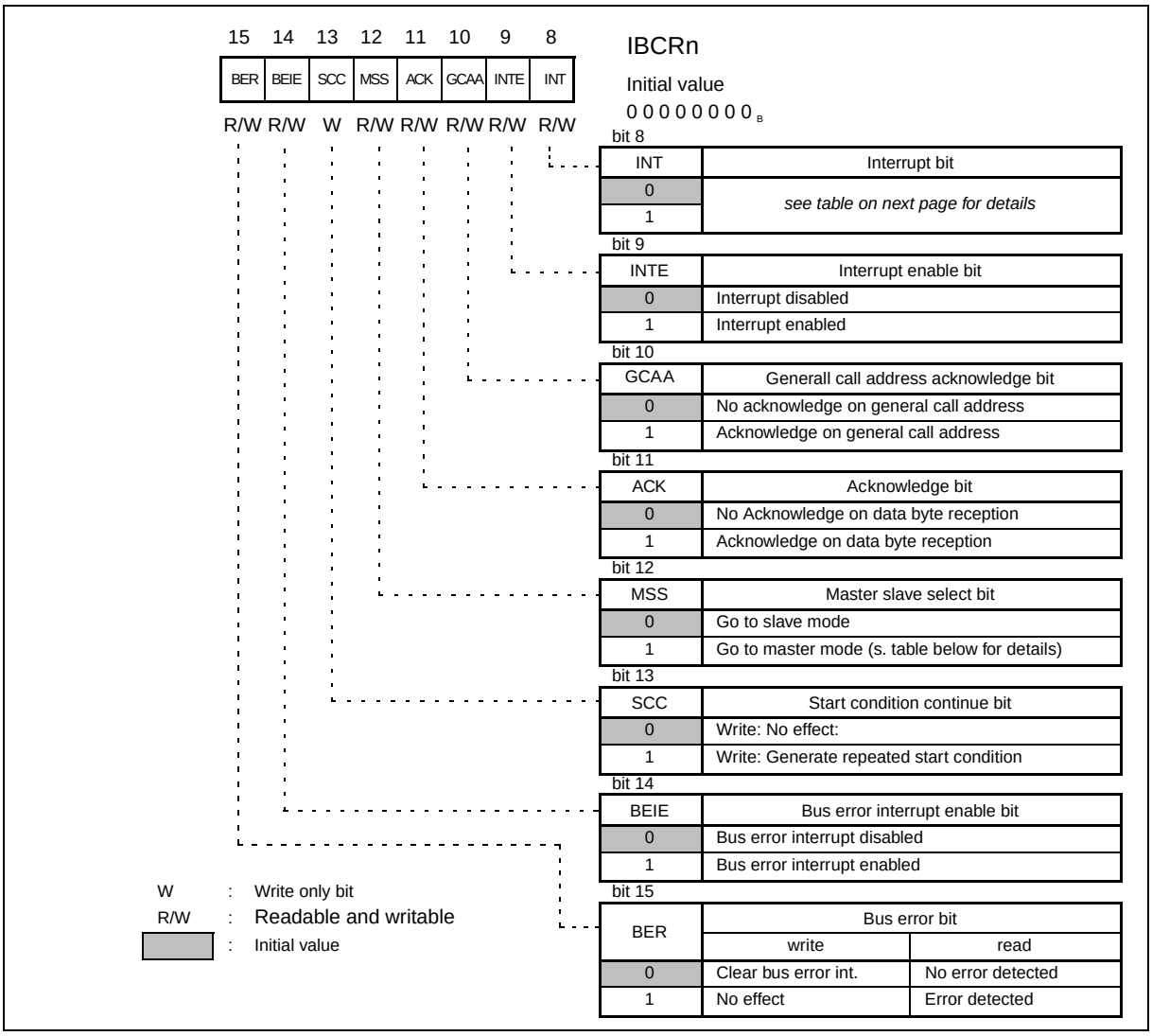
General call acknowledge generation enabling

Data byte acknowledge generation enabling

### *21.2.2.1 Bus control register (IBCRn)*

Write access to this register should only occur while the INT="1" or if a transfer is to be started. The user should not write to this register during an ongoing transfer since changes to the ACK or GCAA bits could result in bus errors. All bits in this register except the BER and the BEIE bit are cleared if the interface is not enabled (EN="0" in ICCR).

Figure 21-4. Configuration of the bus control register (IBCRn)




bit 8

| INT | Interrupt bit |
|---|---|
| 0 | *see table on next page for details* |
| 1 | |

bit 9

| INTE | Interrupt enable bit |
|---|---|
| 0 | Interrupt disabled |
| 1 | Interrupt enabled |

bit 10

| GCAA | Generall call address acknowledge bit |
|---|---|
| 0 | No acknowledge on general call address |
| 1 | Acknowledge on general call address |

bit 11

| ACK | Acknowledge bit |
|---|---|
| 0 | No Acknowledge on data byte reception |
| 1 | Acknowledge on data byte reception |

bit 12

| MSS | Master slave select bit |
|---|---|
| 0 | Go to slave mode |
| 1 | Go to master mode (s. table below for details) |

bit 13

| SCC | Start condition continue bit |
|---|---|
| 0 | Write: No effect: |
| 1 | Write: Generate repeated start condition |

bit 14

| BEIE | Bus error interrupt enable bit |
|---|---|
| 0 | Bus error interrupt disabled |
| 1 | Bus error interrupt enabled |

bit 15

| BER | Bus error bit | |
|---|---|---|
| | write | read |
| 0 | Clear bus error int. | No error detected |
| 1 | No effect | Error detected |

### *21.2.2.2 Bus control register (IBCRn) contents*

Table 21-2. Function of each bit of the bus control register (IBCRn) (Sheet 1 of 2)

| Bit name | | Function |
|---|---|---|
| bit 15 | BER:<br>Bus error bit | This bit is the bus error interrupt flag. It is set by the hardware and cleared by the user.<br>It always reads "1" in a Read-Modify-Write access.<br>Write access:<br>"0": Clear bus error interrupt flag<br>"1": No effect<br>Read access:<br>"0": No bus error detected<br>"1": One of the error conditions described below detected<br>When this bit is set, the EN bit in the ICCR register is cleared, the $I^2C$ interface goes to pause status, data transfer is interrupted and all bits in the IBSR and the IBCR registers except BER and BEIE are cleared. The BER bit must be cleared before the interface may be reenabled.<br>This bit is set to "1" if:<br>- start or stop conditions are detected at wrong places: during an address data transfer or during the transfer of the bits two to nine (acknowledge bit)<br>- a ten bit address header with read access is received before a ten bit write access |
| bit 14 | BEIE:<br>Bus error interrupt enable bit | This bit enables the bus error interrupt. It only can be changed by the user.<br>"0": Bus error interrupt disabled<br>"1": Bus error interrupt enabled<br>Setting this bit to "1" enables MCU interrupt generation when the BER bit is set to "1". |
| bit 13 | SCC<br>Start condition continue bit | This bit is used to generate a repeated start condition. It is write only - it always reads "0".<br>Write access:<br>"0": No effect<br>"1": Generate repeated start condition during master transfer<br>A repeated start condition is generated if a "1" is written to this bit while an interrupt in master mode (MSS="1" and INT="1") and the INT bit is cleared automatically. |
| bit 12 | MSS:<br>Master slave select bit | This is the master/slave mode selection bit. It can only be set by the user, but it can be cleared by the user and the hardware.<br>"0": Go to slave mode<br>"1": Go to master mode, generate start condition and send address data byte in IDAR<br>register. It is cleared if an arbitration loss event occurs during master sending.<br>If a "0" is written to it during a master interrupt (MSS="1" and INT="1"), the INT bit is cleared automatically, a stop condition will be generated and the data transfer ends. Note that the MSS bit is reset immediately, the generation of the stop condition can be checked by polling the BB bit in the IBSR register.<br>If a "1" is written to it while the bus is idle (MSS="0" and BB="0"), a start condition is generated and the contents of the IDAR register (which should be address data) is sent.<br>If a "1" is written to the MSS bit while the bus is in use (BB="1" and TRX="0" in IBSR; MSS="0" in IBCR), the interface waits until the bus is free and then starts sending.<br>If the interface is addressed as slave with write access (data reception) in the meantime, it will start sending after the transfer ended and the bus is free again. If the interface is sending data as slave in the meantime (AAS="1" and TRX="1" in IBSR), it will not start sending data if the bus is free again. It is important to check whether the interface was addressed as slave (AAS="1" in IBSR), sent the data byte successfully (MSS="1" in IBCR) or failed to send the data byte (AL="1" in IBSR) at the next interrupt! |

Table 21-2. Function of each bit of the bus control register (IBCRn) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 11 | ACK:<br>Acknowledge bit | This bit enables the acknowledge generation on data byte reception. It only can be changed by the user.<br>"0": The interface will not acknowledge on data byte reception<br>"1": The interface will acknowledge on data byte reception<br>This bit is not valid when receiving address bytes in slave mode - if the interface detects its 7 or 10 bit slave address, it will acknowledge if the corresponding enable bit (ENTB in ITMK or ENSB in ISMK) is set.<br>Write access to this bit should occur during an interrupt (INT="1") or if the bus is idle (BB="0" in the IBSR register) only. |
| bit 10 | GCAA:<br>General call address acknowledge bit | This bit enables acknowledge generation when a general call address is received. It only can be changed by the user.<br>"0": The interface will not acknowledge on general call address byte reception.<br>"1": The interface will acknowledge on general call address byte reception.<br>Write access to this bit should occur during an interrupt (INT="1") or if the bus is idle (BB="0" in IBSR register) or the interface is disabled (EN="0" in ICCR register) only. |
| bit 9 | INTE:<br>Interrupt enable bit | This bit enables the MCU interrupt generation. It only can be changed by the user.<br>"0": Interrupt disabled<br>"1": Interrupt enabled<br>Setting this bit to "1" enables MCU interrupt generation when the INT bit is set to "1" (by the hardware). |
| bit 8 | INT:<br>Interrupt flag bit | This bit is the transfer end interrupt request flag. It is changed by the hardware and can be cleared by the user. It always reads "1" in a Read-Modify-Write access.<br>Write access:<br>"0": Clear transfer end interrupt request flag<br>"1": No effect<br>Read access:<br>"0": Transfer not ended or not involved in current transfer or bus is idle<br>"1": Set at the end of a 1-byte data transfer or reception including the acknowledge bit under the following conditions:<br>Device is bus master.<br>Device is addressed as slave.<br>General call address received.<br>Arbitration loss occurred.<br>Set at the end of an address data reception (after first byte if seven bit address received, after second byte if ten bit address received) including the acknowledge bit if the device is addressed as slave.<br>While this bit is "1" the SCL line will hold an "L" level signal. Writing "0" to this bit clears the setting, releases the SCL line, and executes transfer of the next byte or a repeated start or stop condition is generated. Additionally, this bit is cleared if a "1" is written to the SCC bit or the MSS bit is being cleared. |

### *21.2.2.3 SCC, MSS and INT bit competition*

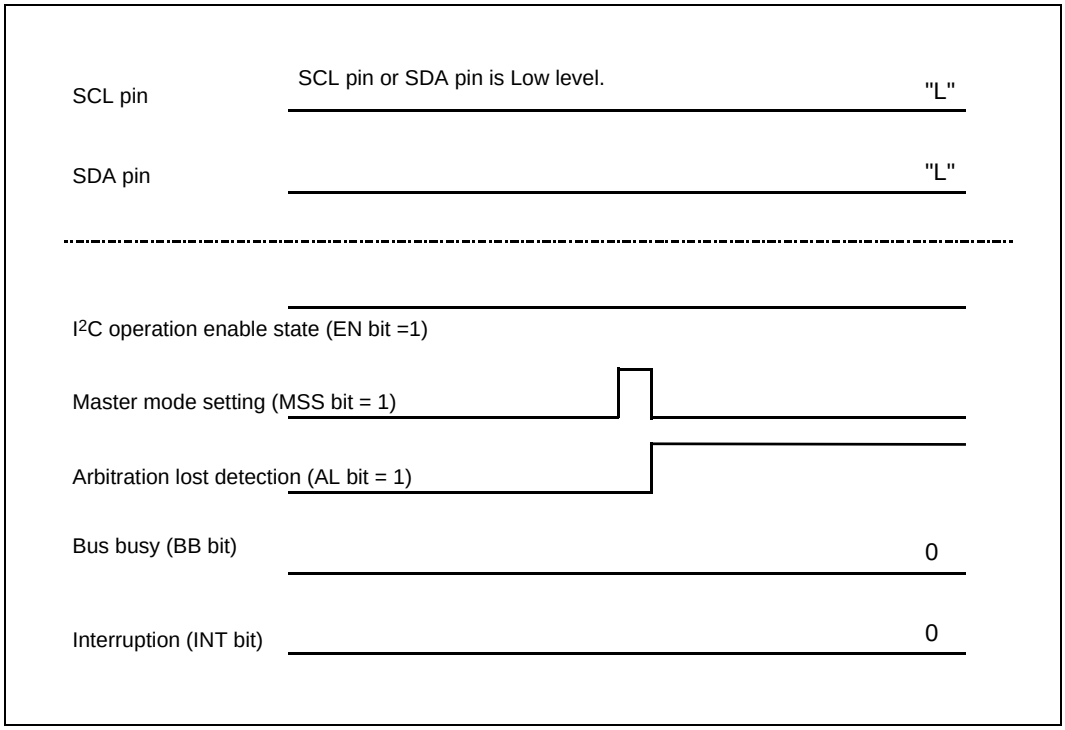
Simultaneously writing to the SCC, MSS and INT bits causes a competition to transfer the next byte, to generate a repeated start condition or to generate a stop condition. In these cases the order of priority is as follows:

- Next byte transfer and stop condition generation. When "0" is written to the INT bit and "0" is written to the MSS bit, the MSS bit takes priority and a stop condition is generated.
- Repeated start condition generation and stop condition generation. When a “1” is written to the SCC bit and “0” to the MSS bit, the MSS bit clearing takes priority. A stop condition is generated and the interface enters slave mode. If specific conditions are met, the AL (arbitration lost) bit does not set the INT (interrupt) bit. Those conditions are presented in Figure 21-5 and Figure 21-6.

**Case 1: When SCL and SDA signals are kept at "L";**

The AL bit is set immediately after the MSS bit set to "1" while the BB bit is indicating "0" (no start condition is detected). However the AL bit will not set the INT bit under this circumstances.

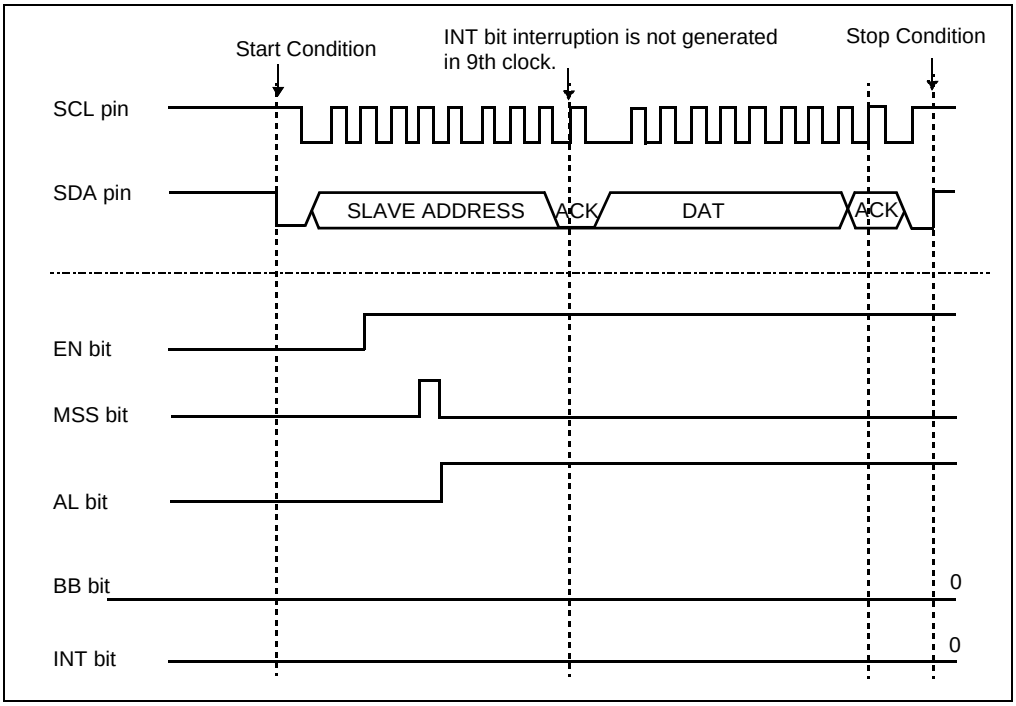
Figure 21-5. Diagram of timing at which an interrupt upon detection of "AL bit = 1" does not occur




### Case 2: When I²C interface is enabled while there is ongoing communication with another bus master;

The interface participates in the I²C bus while the bus is occupied with ongoing communication if the EN bit is set from "0" to "1". In this case the BB bit stays "0" (no start condition is detected) and setting the MSS bit to "1" results in the AL bit indicating "1". However the AL bit will not set the INT bit under this circumstances.

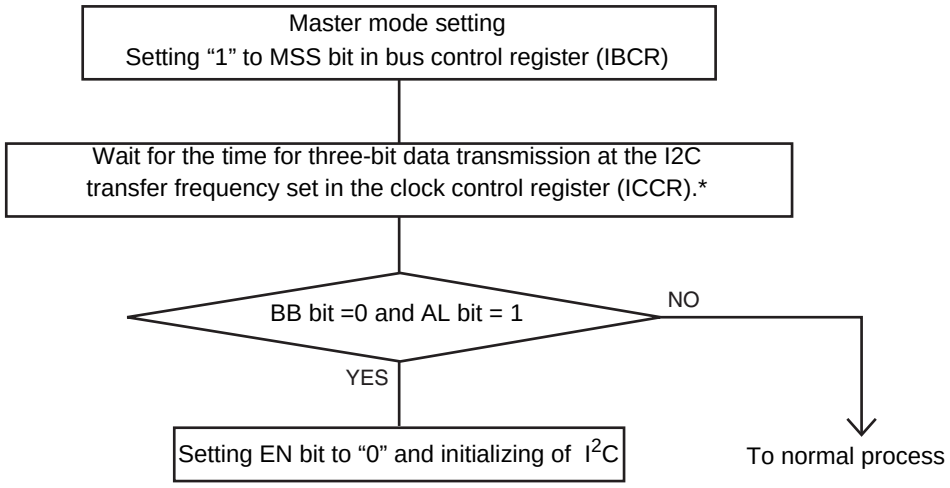
Figure 21-6. Diagram of timing at which an interrupt upon detection of "AL bit = 1" does not occur




If a symptom as described above can occur, follow the procedure below for software processing.

1. Execute the instruction that generates a start condition (set the MSS bit to 1).
2. Use, for example, the timer function to wait for the time for three - bit data transmission at the $I^2C$ transfer frequency set in the ICCR register.*
   Example: Time for three-bit data transmission at an $I^2C$ transfer frequency of 100 kHz = $(1 / (100 \times 10^3)) \times 3 = 30$
3. Check the AL and BB bits in the IBSR register and, if the AL and BB bits are 1 and 0, respectively, set the EN bit in the ICCR register to 0 to initialize $I^2C$. When the AL and BB bits are not so, perform normal processing.

A sample flow is given below.

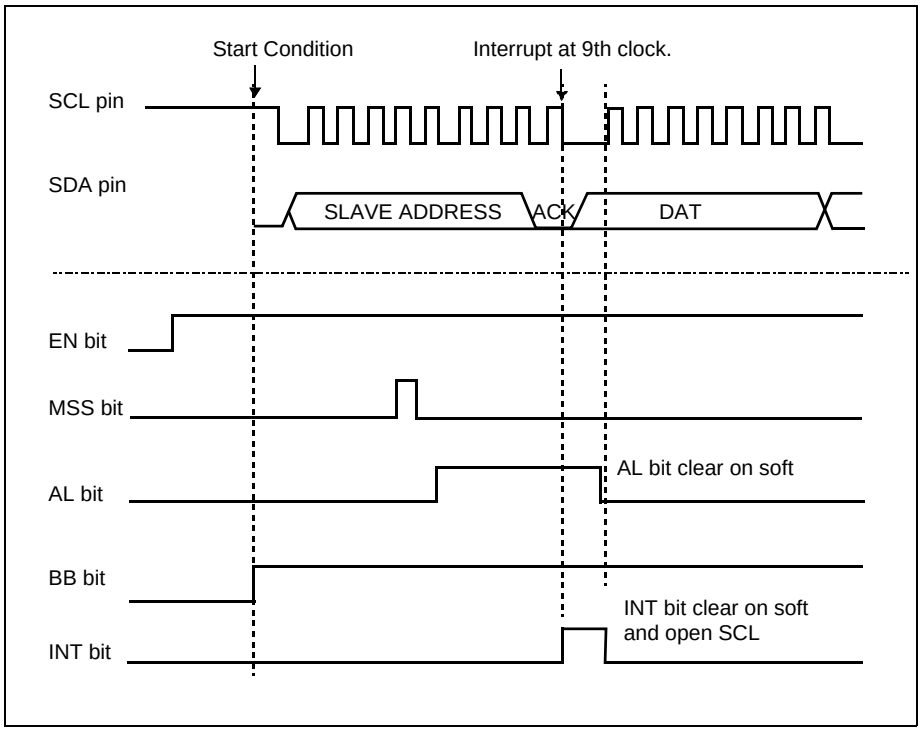



*:Arbitration lost is detected within 3-bit time after setting the MSS bit to "1".

### Example of occurrence of an interrupt (INT bit = 1) upon detection of "AL bit = 1"

When an instruction which generates a start condition is executed (setting the MSS bit to 1) with "bus busy" detected (BB bit = 1) and arbitration is lost, the INT bit interrupt occurs upon detection of "AL bit = 1".

Figure 21-7. Diagram of timing at which an interrupt upon detection of "AL bit = 1" occurs



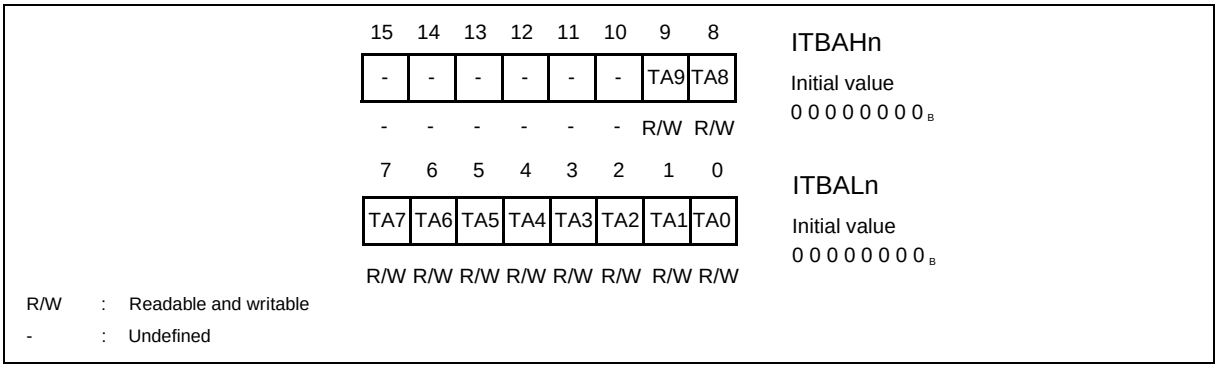

## 21.2.3 Ten Bit Slave Address Register (ITBAn)

The Ten Bit Slave Address Register (ITBAn) designates the ten bit slave address.

### Ten Bit Slave Address Register (ITBAn)

Write access to this register is only possible if the interface is disabled (EN="0" in ICCR).

Figure 21-8. Configuration of the Ten Bit Slave Address Register (IBTAn)



### Ten Bit Slave Address Register (ITBAn) contents

Table 21-3. Function of each bit of the ten bit slave address register (ITBAn)

| Bit name | | Function |
|---|---|---|
| bits 15 to 10 | Undefined | These bits always return "0". |
| bits 9 to 0 | TA9 to TA0:<br>Ten bit slave address | When address data is received in slave mode, it is compared to the ITBA register if the ten bit address is enabled (ENTB="1" in the ITMK register). An acknowledge is sent to the master after reception of a ten bit address header with write access[a]. Then, the second incoming byte is compared to the TBAL register. If a match is detected, an acknowledge signal is sent to the master device and the AAS bit is set.<br>Additionally, the interface acknowledges upon the reception of a ten bit header with read access[b] after a repeated start condition.<br>All bits of the slave address may be masked using the ITMK register. The received ten bit slave address is written back to the ITBA register, it is only valid while the AAS bit in the IBSR register is “1”. |

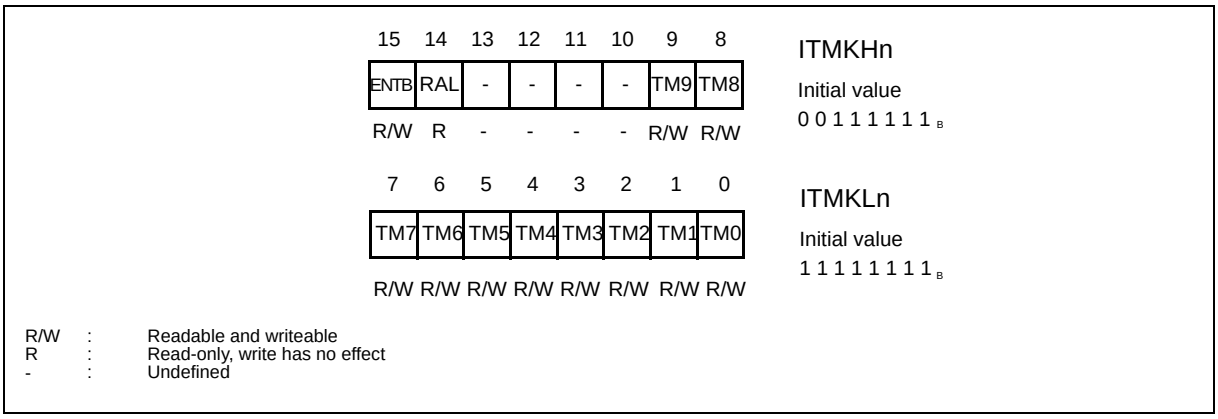
a. A ten bit header (write access) consists of the following bit sequence: 11110, TA9, TA8, 0.
b. A ten bit header (read access) consists of the following bit sequence: 11110, TA9, TA8, 1.

## 21.2.4 Ten Bit Slave Address Mask Register (ITMKn)

The Ten Bit Slave Address Mask Register (ITMKn) contains the ten bit slave address mask and the ten bit slave address enable bit.

### Ten bit address mask register (ITMKn)

Figure 21-9. Configuration of the ten bit address mask register (ITMKn)



### Ten bit address mask register (ITMKn) contents

Table 21-4. Function of each bit of the ten bit address mask register (ITMKn)

| Bit name | | Function |
|---|---|---|
| bit 15 | ENTB:<br>Enable ten bit slave address bit | This bit enables the ten bit slave address (and the acknowledging upon its reception). Write access to this bit is only possible if the interface is disabled (EN="0" in ICCR).<br>"0": Ten bit address disabled<br>"1": Ten bit address enabled |
| bit 14 | RAL:<br>Received slave address length bit | This bit indicates whether the interface was addressed as a seven or ten bit slave. It is read-only.<br>"0": Addressed as seven bit slave<br>"1": Addressed as ten bit slave<br>This bit can be used to determine whether the interface was addressed as a seven or ten bit slave if both slave addresses are enabled (ENTB="1" and ENSB="1"). Its contents is only valid if the AAS bit in the IBSR register is "1". This bit is also reset if the interface is disabled (EN="0" in ICCR). |
| bit 13 to 10 | Undefined | These bits always return "1" during reading. |
| bit 9 to 0 | TM9 to TM0:<br>Ten bit slave address mask bits | This register is used to mask the ten bit slave address of the interface. Write access to these bits is only possible if the interface is disabled (EN="0" in ICCR).<br>"0": Bit is not used in slave address comparison<br>"1": Bit is used in slave address comparison<br>This can be used to make the interface acknowledge on multiple ten bit slave addresses. Only the bits set to "1" in this register are used in the ten bit slave address comparison. The received slave address is written back to the ITBA register and thus may be determined by reading the ITBA register if the AAS bit in the IBSR register is “1”.<br>**Note:**<br>If the address mask is changed after the interface has been enabled, the slave address should also be set again since it could have been overwritten by a previously received slave address. |

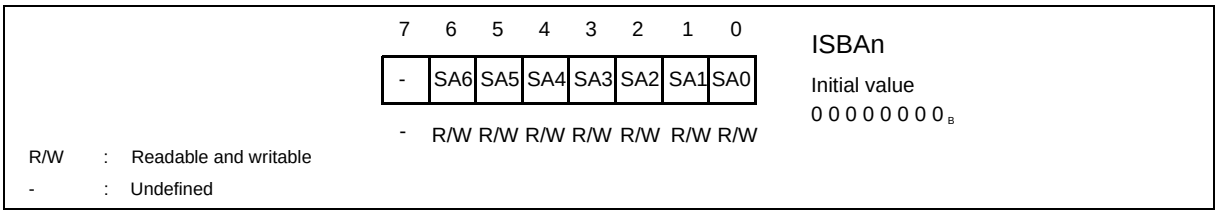
## 21.2.5 Seven Bit Slave Address Register (ISBAn)

The Seven Bit Slave Address Register (ISBAn) contains the seven bit slave address.

### Seven Bit Slave Address Register (ISBAn)

Write access to this register is only possible if the interface is disabled (EN="0" in ICCR).

Figure 21-10. Configuration of the Seven Bit Slave Address Register (ISBAn)




### Seven Bit Slave Address Register (ISBAn) contents

Table 21-5. Function of each bit of the Seven Bit Slave Address Register (ISBAn)

| Bit name | | Function |
|---|---|---|
| bit 7 | Undefined | This bit always returns "0" during reading. |
| bit 6 to 0 | SA6 to SA0:<br>Seven bit slave address bits | When address data is received in slave mode, it is compared to the ISBA register if the seven bit address is enabled (ENSB="1" in the ISMK register). If a match is detected, an acknowledge signal is sent to the master device and the AAS bit is set.<br>All bits of the slave address may be masked using the ISMK register. The received seven bit slave address is written back to the ISBA register, it is only valid while the AAS bit in the IBSR register is “1”.<br>The interface does not compare the contents of this register to the incoming data if a ten bit header or a general call is received. |

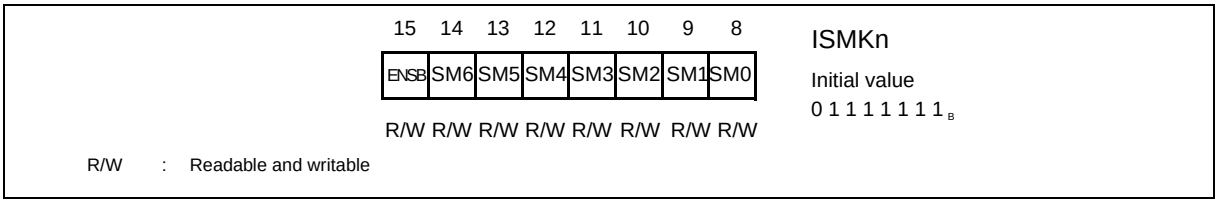
## 21.2.6 Seven Bit Slave Address Mask Register (ISMKn)

The Seven Bit Slave Address Mask Register (ISMKn) contains the seven bit slave address mask and the seven bit slave address enable bit.

### Seven Bit Slave Address Mask Register (ISMKn)

This register contains the seven bit slave address mask and the seven bit mode enable bit. Write access to this register is only possible if the interface is disabled (EN="0" in ICCR).

Figure 21-11. Configuration of the Seven Bit Slave Address Mask register (ISMKn)




### Seven Bit Slave Address Mask Register (ISMKn) contents

Table 21-6. Function of each bit of the Seven Bit Slave Address Mask Register (ISMKn)

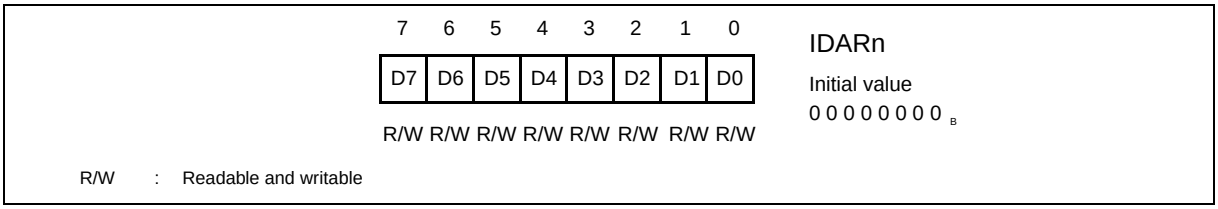
| Bit name | | Function |
|---|---|---|
| bit 15 | ENSB:<br>Enable seven bit slave address bit | This bit enables the seven bit slave address (and the acknowledging upon its reception).<br>"0": Seven bit slave address disabled<br>"1": Seven bit slave address enabled |
| bit 14 to 8 | SM6 to SM0:<br>Seven bit slave address mask bits | This register is used to mask the seven bit slave address of the interface.<br>"0": Bit is not used in slave address comparison.<br>"1": Bit is used in slave address comparison.<br>This can be used to make the interface acknowledge on multiple seven bit slave addresses. Only the bits set to “1” in this register are used in the seven bit slave address comparison. The received slave address is written back to the ISBA register and may thus may be determined by reading the ISBA register if the AAS bit in the IBSR register is “1”.<br>**Note:**<br>If the address mask is changed after the interface has been enabled, the slave address should also be set again since it could have been overwritten by a previously received slave address. |

## 21.2.7 Data Register (IDARn)

The Data Register for the 400 kHz $I^2C$ Interface (IDARn) is used to send and receive data by the CPU.

### Data Register (IDARn)

Figure 21-12. Configuration of the Data Register (IDARn)



### Data Register (IDARn) contents

Table 21-7. Function of each bit of the Data Register (IDARn)

| Bit name | | Function |
|---|---|---|
| bits 7 to 0 | D7 to D0:<br>Data bits | The data register is used in serial data transfer, and transfers data MSB-first. This register is double buffered on the write side, so that when the bus is in use (BB="1"), write data can be loaded to the register for serial transfer. The data byte is loaded into the internal transfer register if the INT bit in the IBCR register is being cleared or the bus is idle (BB="0" in IBSR). In a read access, the internal register is read directly, therefore received data values in this register are only valid if INT="1" in the IBCR register. |

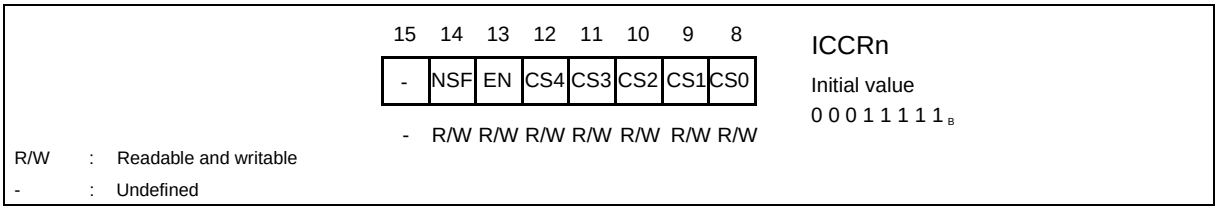
## 21.2.8 Clock Control Register (ICCRn)

The Clock Control Register (ICCRn) has the following functions:

- Enable test mode
- Enable IO pad noise filters
- Enable $I^2C$ interface operation
- Setting the serial clock frequency

### Clock control register (ICCRn)

Figure 21-13. Configuration of the Clock Control Register (ICCRn)



### Clock Control Register (ICCRn) contents

Table 21-8. Function of each bit of the Clock Control Register (ICCRn)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 15</td><td>Undefined</td><td>This bit always returns "0" during reading.</td></tr>
<tr><td>bit 14</td><td>NSF:<br>IO pad noise filter enable bit</td><td>This bit enables the noise filters built into the SDA and SCL IO pads.<br>The noise filter will suppress single spikes with a pulse width of 0 ns (minimum) and between 1 and 1.5 cycles of the peripheral clock CLKP1. The maximum depends on the phase relationship between I$^2$C signals (SDA, SCL) and peripheral clock CLKP1.<br>It should be set to “1” if the interface is transmitting or receiving at data rates above 100 KBit.</td></tr>
<tr><td>bit 13</td><td>EN:<br>Enable bit</td><td>This bit enables the I$^2$C interface operation. It can only be set by the user but may be cleared by the user and the hardware.<br>"0": Interface disabled<br>"1": Interface enabled<br>When this bit is set to “0” all bits in the IBSR register and IBCR register (except the BER and BEIE bits) are cleared and the module is disabled and the I$^2$C lines are left open. It is cleared by the hardware if a bus error occurs (BER="1" in IBCR).<br>Note:<br>The interface immediately stops transmitting or receiving if it is being disabled. This might leave the I$^2$C bus in an undesired state.</td></tr>
<tr><td>bits 12 to 8</td><td>CS4 to CS0:<br>Clock prescaler bits</td><td>These bits select the serial bit rate. They can only be changed if the interface is disabled (EN="0") or the EN bit is being cleared in the same write access.<br><br>
<table>
<tr><td>n</td><td>CS4</td><td>CS3</td><td>CS2</td><td>CS1</td><td>CS0</td><td></td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>Bitrate: Φ / 28(+1)</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>Bitrate: Φ / 40(+1)</td></tr>
<tr><td>3</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>Bitrate: Φ / 52(+1)</td></tr>
<tr><td>4</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>Bitrate: Φ / 64(+1)</td></tr>
<tr><td colspan="6">...</td><td></td></tr>
<tr><td>31</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>Bitrate: Φ / 400(+1)</td></tr>
</table>
(+1) means: Add 1 to divisor, if noise filter is enabled</td></tr>
</table>

### Clock prescaler settings

The calculation formula for CS0 to CS4 is determined as follows:

$$\text{Bitrate} = \frac{\phi}{n*12 + 16} \quad n>0 \quad \phi\text{: Peripheral clock CLKP1, Noise filter disabled}$$

$$\text{Bitrate} = \frac{\phi}{n*12 + 17} \quad n>0 \quad \phi\text{: Peripheral clock CLKP1, Noise filter enabled}$$

Table 21-9. Prescaler settings:

| n | CS4 | CS3 | CS2 | CS1 | CS0 |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 | 0 |

Table 21-9. Prescaler settings:

| n | CS4 | CS3 | CS2 | CS1 | CS0 |
|---|---|---|---|---|---|
| 3 | 0 | 0 | 0 | 1 | 1 |
| ... | | | | | |
| 31 | 1 | 1 | 1 | 1 | 1 |

**Note:**

Do not use n=0 prescaler setting, it violates SDA/SCL timings.

### Common peripheral clock CLKP1 frequencies

Table 21-10shows the most common peripheral clock CLKP1 frequencies with their prescaler settings and the resulting sending bit rate:

Table 21-10. Common peripheral clock CLKP1 frequencies

| CLKP1 frequency [MHz] | 100 Kbit (Noise filter disabled) | | 400 Kbit (Noise filter enabled) | |
|---|---|---|---|---|
| | n | Bit rate [Kbit] | n | Bit rate [Kbit] |
| 24 | 19 | 98 | 4 | 369 |
| 20 | 16 | 96 | 3 | 377 |
| 16 | 12 | 100 | 2 | 390 |
| 40/3 = 13.3 | 10 | 98 | 2 | 325 |
| 12 | 9 | 96 | 2 | 292 |
| 64/6 = 10.6 | 8 | 94 | 1 | 367 |
| 10 | 7 | 100 | 1 | 344 |
| 8 | 6 | 90 | 1 | 275 |

## 21.3 I²C Interface Operation

The I²C bus executes communication using two bi-directional bus lines, the serial data line (SDA) and serial clock line (SCL). The I²C interface has two open-drain I/O pins (SDA/SCL) corresponding to these lines, enabling wired logic applications.

### Start conditions

When the bus is free (BB="0" in IBSR, MSS="0" in IBCR), writing “1” to the MSS bit places the I²C interface in master mode and generates a start condition.

If a “1” is written to it while the bus is idle (MSS="0" and BB="0"), a start condition is generated and the contents of the IDAR register (which should be address data) is sent.
Repeated start conditions can be generated by writing “1” to the SCC bit when in bus master mode and interrupt status (MSS="1" and INT="1" in IBCR).

If a “1” is written to the MSS bit while the bus is in use (BB="1" and TRX="0" in IBSR; MSS="0"and INT="0"in IBCR), the interface waits until the bus is free and then starts sending.

If the interface is addressed as slave with write access (data reception) in the meantime, it will start sending after the transfer ended and the bus is free again. If the interface is sending data as slave in the meantime, it will not start sending data if the bus is free again. It is important to check whether the interface was addressed as slave (MSS="0" in IBCR and AAS="1" in IBSR), sent the data byte successfully (MSS="1" in IBCR) or failed to send the data byte (AL="1" in IBSR) at the next interrupt.

Writing "1" to the MSS bit or SCC bit in any other situation has no significance.

### Stop conditions

Writing “0” to the MSS bit in master mode (MSS="1" and INT="1" in IBCR) generates a stop condition and places the device in slave mode. Writing “0” to the MSS bit in any other situation has no significance.

After clearing the MSS bit, the interface tries to generate a stop condition which might fail if a certain external condition causes signal transition caused from "1" to "0" at the SCL line before the generation of this stop condition. In this case, the AL bit is set to "1" and interrupt is signaled at the end of the next byte.

### Slave address detection

In slave mode, after a start condition is generated the BB is set to “1” and data sent from the master device is received into the IDAR register.

After the reception of eight bits, the contents of the IDAR register is compared to the ISBA register using the bit mask stored in ISMK if the ENSB bit in the ISMK register is “1”. If a match results, the AAS bit is set to “1” and an acknowledge signal is sent to the master. Then bit 0 of the received data (bit 0 of the IDAR register) is inverted and stored in the TRX bit.

If the ENTB bit in the ITMK register is "1" and a ten bit address header (11110, TA1, TA0, write access) is detected, the interface sends an acknowledge signal to the master and stores the inverted last data bit in the TRX register. No interrupt is generated. Then, the next transferred byte is compared (using the bit mask stored in ITMK) to the lower byte of the ITBA register. If a match is found, an acknowledge signal is sent to the master, the AAS bit is set and an interrupt is generated.

If the interface was addressed as slave and detects a repeated start condition, the AAS bit is set after reception of the ten bit address header (11110, TA1, TA0, read access) and an interrupt is generated.

Since there are separate registers for the ten and seven bit address and their bit masks, it is possible to make the interface acknowledge on both addresses by setting the ENSB (in ISMK) and ENTB (in ITMK) bits. The received slave address length (seven or ten bit) may be determined by reading the RAL bit in the ITMK register (this bit is valid if the AAS bit is set only).

It is also possible to give the interface no slave address by setting both bits to "0" if it is only used as a master.

All slave address bits may be masked with their corresponding mask register (ITMK or ISMK).

### Slave address masking

Only the bits set to “1” in the mask registers (ITMK / ISMK) are used for address comparison, all other bits are ignored. The received slave address can be read from the ITBA (if ten bit address received, RAL="1") or ISBA (if seven bit address received, RAL="0") register if the AAS bit in the IBSR register is “1”.

If the bit masks are cleared, the interface can be used as a bus monitor since it will always be addressed as slave. Note that this is not a real bus monitor because it acknowledges upon any slave address reception, even if there is no other slave listening.

### Addressing slaves

In master mode, after a start condition is generated the BB and TRX bits are set to “1” and the contents of the IDAR register is sent in MSB first order. After address data is sent and an acknowledge signal was received from the slave device, bit 0 of the sent data (bit 0 of the IDAR register after sending) is inverted and stored in the TRX bit. Acknowledgement by the slave may be checked using the LRB bit in the IBSR register. This procedure also applies to a repeated start condition.

In order to address a ten bit slave for write access, two bytes have to be sent. The first one is the ten bit address header which consists of the bit sequence "1 1 1 1 0 A9 A8 0", it is followed by the second byte containing the lower eight bits of the ten bit slave address (A7 to A0).

A ten bit slave is accessed for reading by sending the above byte sequence and generating a repeated start condition (SCC bit in IBCR) followed by a ten bit address header with read access (1 1 1 1 0 A9 A8 1).

Summary of the address data bytes:

7 bit slave, write access: Start condition - A6 A5 A4 A3 A2 A1 A0 0.

7 bit slave, read access: Start condition - A6 A5 A4 A3 A2 A1 A0 1.

10 bit slave, write access: Start condition - 1 1 1 1 0 A9 A8 0 - A7 A6 A5 A4 A3 A2 A1 A0.

10 bit slave, read access: Start condition - 1 1 1 1 0 A9 A8 1 - A7 A6 A5 A4 A3 A2 A1 A0 - repeated start - 1 1 1 1 0 A9 A8 1.

### Arbitration

During sending in master mode, if another master device is sending data at the same time, arbitration is performed. If a device is sending the data value "1" and the data on the SDA line has an "L" level value, the device is considered to have lost arbitration, and the AL bit is set to "1." Also, the AL bit is set to "1" if a start condition is detected at the first bit of a data byte but the interface did not want to generate one or the generation of a start or stop condition failed by some reason.

Arbitration loss detection clears both the MSS and TRX bit and immediately places the device in slave mode so it is able to acknowledge if its own slave address is being sent.

### Acknowledgement

Acknowledge bits are sent from the receiver to the transmitter. The ACK bit in the IBCR register can be used to select whether to send an acknowledgment when data bytes are received.

When data is send in slave mode (read access from another master), if no acknowledgement is received from the master, the TRX bit is set to "0" and the device goes to receiving mode. This enables the master to generate a stop condition as soon as the slave has released the SCL line.

In master mode, acknowledgement by the slave can be checked by reading the LRB bit in the IBSR register.

## 21.4 Programming Flow Charts

Each programming flow charts for the 400 kHz $I^2C$ interface is shown below.

### Programming flow charts

Figure 21-14. Example of slave addressing and sending data

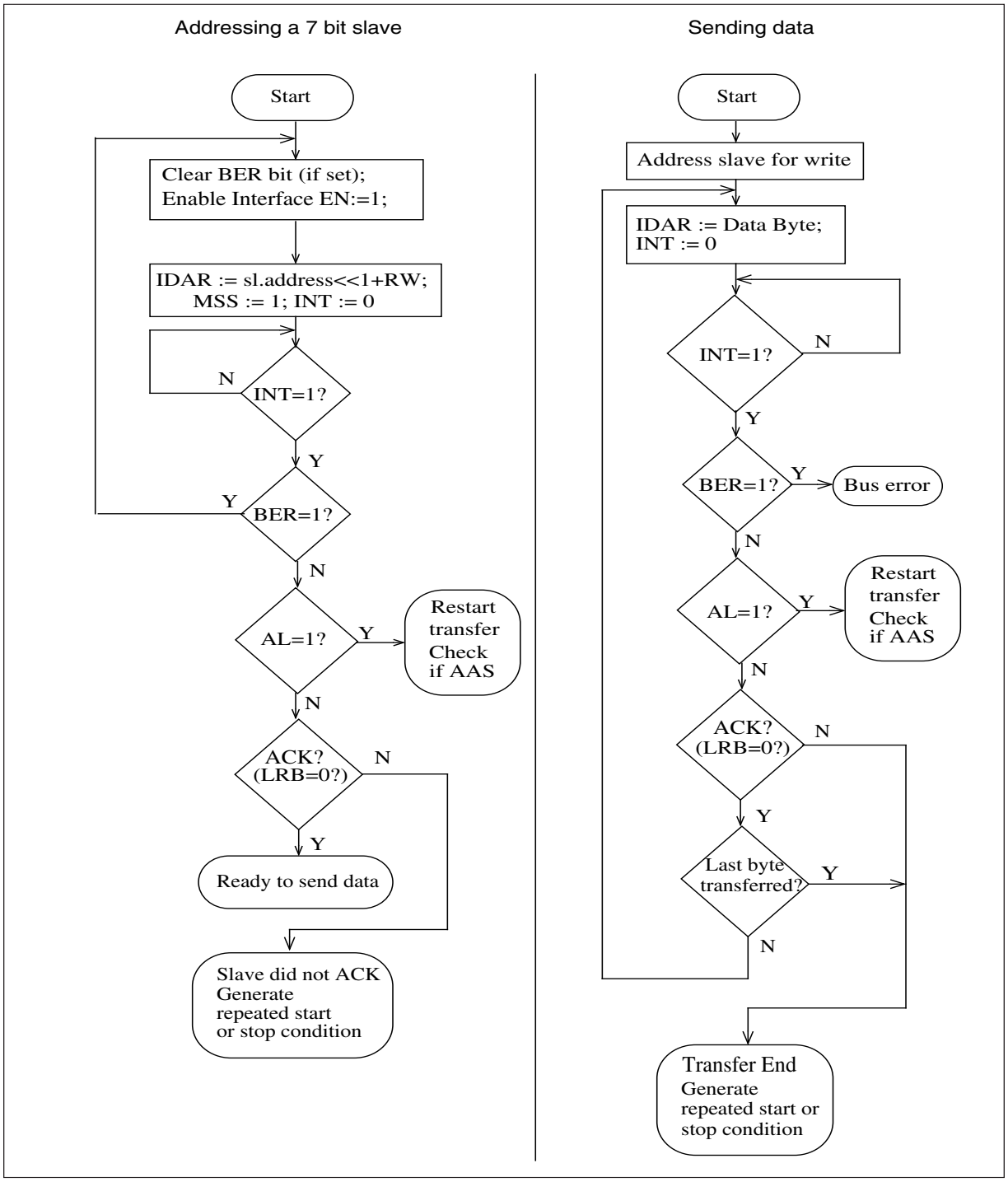
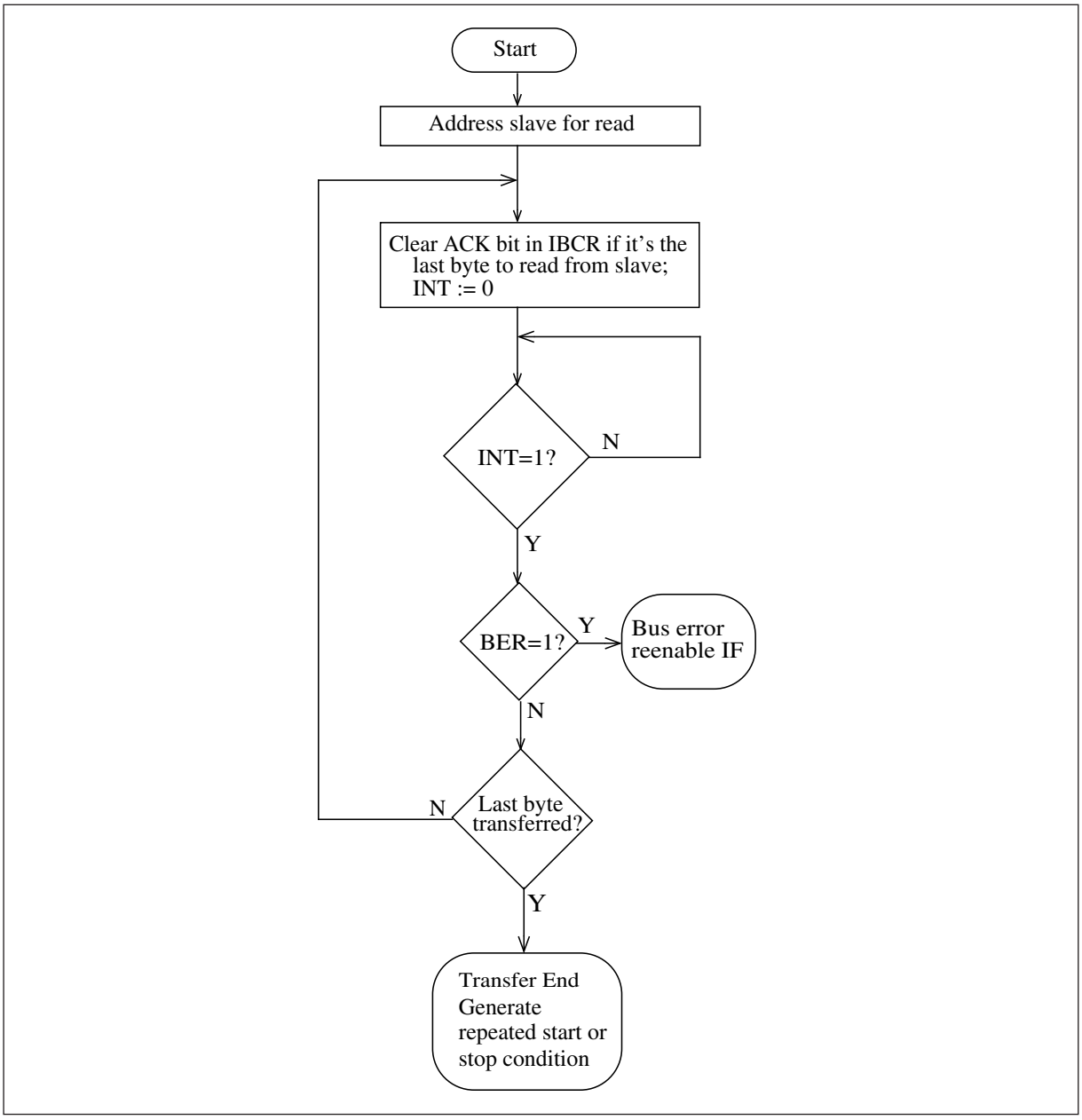


Figure 21-15. Example of receiving data



# 22. CAN Controller

This chapter explains the functions and operations of the CAN controller.

22.1 Overview

22.2 Functional Description

22.3 Register Description

22.4 CAN Device Related Registers

22.5 CAN Protocol Related Registers

22.6 Message Interface Register Sets

22.7 Message Object in the Message Memory

22.8 Message Handler Registers

22.9 CAN Application

## 22.1 Overview

The CAN performs communication according to the CAN protocol version 2.0 part A and B. The bit rate can be programmed to values up to 1MBit/s. For the connection to the physical layer additional transceiver hardware is required.

### Features

The CAN implements the following features:

- Supports CAN protocol version 2.0 part A and B
- Bit rates up to 1 MBit/s
- 32 (up to 128) Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode (concatenation of Message Objects)
- Maskable interrupt
- Disabled Automatic Retransmission mode for Time Triggered CAN applications
- Programmable loop-back mode for self-test operation

### Note:

In this chapter, the suffix "n" in the register names denotes the CAN controller number.

### Abbreviations

This chapter uses the following terms and abbreviations:

| Term | Meaning |
|---|---|
| CAN | Controller Area Network |
| BSP | Bit Stream Processor |
| BTL | Bit Timing Logic |
| CRC | Cyclic Redundancy Check |
| DLC | Data Length Code |
| EML | Error Management Logic |
| FSM | Finite State Machine |
| TTCAN | Time Triggered CAN |

## 22.2 Functional Description

This chapter provides an overview of the CAN module's operating modes and how to use them.

### Software Initialisation

The software initialization is started by setting the bit INIT in the CAN Control Register CTRLRLn, either by software or by a hardware reset, or by going Bus_Off.

While INIT is set, all message transfer from and to the CAN bus is stopped, the status of the CAN bus output CAN_TX is recessive (HIGH). The counters of the EML are unchanged. Setting INIT does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register BTRn and each Message Object. If a Message Object is not needed, it is sufficient to set it's MSGVAL bit to not valid. Otherwise, the whole Message Object has to be initialized.

Access to the Bit Timing Register BTRn and to the BRP Extension Register BRPERn for the configuration of the bit timing is enabled when both bits INIT and CCE in the CAN Control Register CTRLRLn are set.

Resetting INIT (by CPU only) finishes the software initialisation. Afterwards the Bit Stream Processor BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (= Bus Idle) before it can take part in bus activities and starts the message transfer.

The initialization of the Message Objects is independent of INIT and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer.

To change the configuration of a Message Object during normal operation, the CPU has to start by setting MSGVAL bit to not valid. When the configuration is completed, MSGVAL bit is set to valid again.

### CAN Message Transfer

Once the CAN is initialized and INIT is reset to zero, the CAN's CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits, which are masked to "don't care", may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers, the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits set up during configuration) exists for the message, only the data bytes are updated and then TXRQST bit with NEWDAT bit are set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, they are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started. Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

### Disabled Automatic Retransmission

According to the CAN Specification (see ISO11898, 6.3.3 Recovery Management), the CAN provides means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means that automatic retransmission is enabled. It can be disabled to enable the CAN to work within a Time Triggered CAN (TTCAN, see ISO11898-1) environment. The Disabled Automatic Retransmission mode is enabled by programming bit DAR in the CAN Control Register CTRLRn to "1". In this operation mode the programmer has to consider the different behaviour of bits TXRQST and NEWDAT in the Control Registers of the Message Buffers:

-When a transmission starts bit TXRQST of the respective Message Buffer is reset, while bit NEWDAT remains set.

When the transmission completed successfully bit NEWDAT is reset.

When a transmission failed (lost arbitration or error) bit NEWDAT remains set. To restart the transmission the CPU has to set TXRQST back to one.

When the host requests the transmission of several messages at the same time, only two of these messages will be transmitted. For all other requested transmit messages, the TXRQST bits will be reset, but no transmission will be started, NEWDAT and INTPND will be left unchanged. For the two messages that are transmitted, the TXRQST and NEWDAT bits will be reset and, if enabled by TXIE, INTPND will be set.

### Test Mode

The Test Mode is entered by setting bit TEST in the CAN Control Register to "1". In Test Mode the bits TX1, TX0, LBACK, SILENT and BASIC in the Test Register TESTRn are writable. Bit RX monitors the state of pin CAN_RX and therefore is only readable. All Test Register functions are disabled when bit TEST is reset to zero.

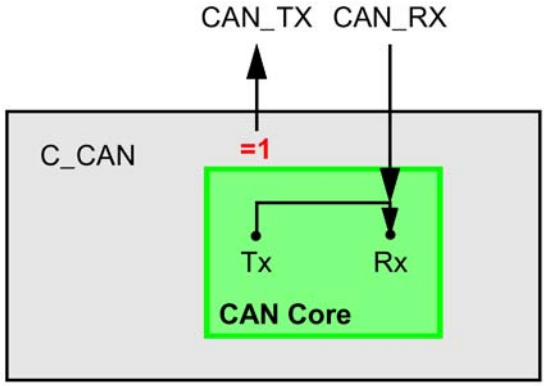
### Silent Mode

The CAN Core can be set in Silent Mode by programming the Test Register TESTRn bit SILENT to "1".

In Silent Mode, the CAN is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames). Figure 22-1 shows the connection of signals CAN_TX and CAN_RX to the CAN Core in Silent Mode.

In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode.

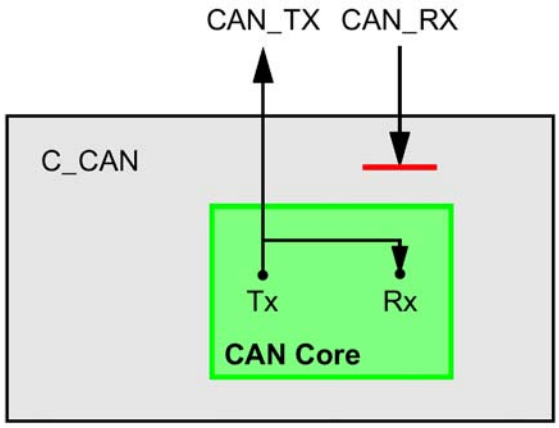
Figure 22-1. CAN Core in Silent Mode



### Loop Back Mode

The CAN Core can be set in Loop Back Mode by programming the Test Register TESTRn bit LBACK to one. In Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer. Figure 22-2 shows the connection of signals CAN_TX and CAN_RX to the CAN Core in Loop Back Mode.

Figure 22-2. CAN Core in Loop Back Mode

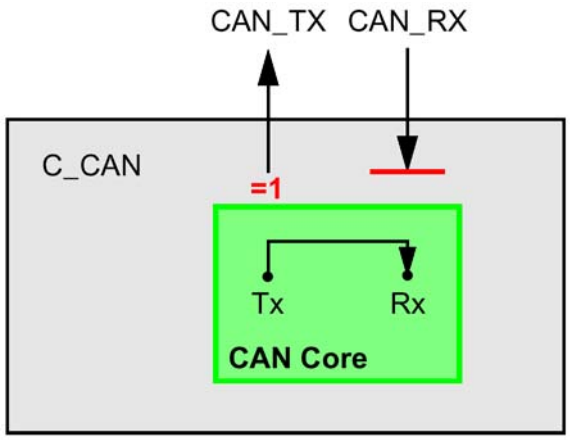



This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in Loop Back Mode. In this mode the CAN Core performs an internal feedback from its Tx output to its Rx input. The actual value of the CAN_RX input pin is disregarded by the CAN Core. The transmitted messages can be monitored at the CAN_TX pin.

### Loop Back combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming bits LBACK and SILENT to "1" at the same time. This mode can be used for a “Hot Selftest”, meaning the CAN can be tested without affecting a running CAN system connected to the pins CAN_TX and CAN_RX. In this mode the CAN_RX pin is disconnected from the CAN Core and the CAN_TX pin is held recessive. Figure 22-3 shows the connection of signals CAN_TX and CAN_RX to the CAN Core in case of the combination of Loop Back Mode with Silent Mode.

Figure 22-3. CAN Core in Loop Back combined with Silent Mode




### Basic Mode

The CAN Core can be set in Basic Mode by programming the Test Register TESTRn bit BASIC to "1". In this mode the CAN module runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the BUSY bit of the IF1 Command Request Register IF1CREQn to ‘1’. The IF1 Registers are locked while the BUSY bit is set. The BUSY bit indicates that the transmission is pending.

As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the BUSY bit is reset and the locked IF1 Registers are released.

A pending transmission can be aborted at any time by resetting the BUSY bit in the IF1 Command Request Register IF1CREQn while the IF1 Registers are locked. If the CPU has reset the BUSY bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering.

Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the BUSY bit of the IF2 Command Request Register IF1CREQn to ‘1’, the contents of the shift register are stored into the IF2 Registers.

In Basic Mode the evaluation of all Message Object related control and status bits and of the control bits of the IFx Command Mask Registers IF1CMSKn is turned off. The message number of the Command Request Registers is not evaluated. The NEWDAT and MSGLST bits of the IF2 Message Control Register  IF2MCTRn retain their function, DLC3-0 will show the received DLC, the other control bits will be read as ‘0’.

### Software control of Pin CAN_TX

Four output functions are available for the CAN transmit pin CAN_TX. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor CAN_Core’s bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin CAN_RX, can be used to check the CAN bus’ physical layer.

The output mode of pin CAN_TX is selected by programming the Test Register TESTRn bits TX1 and TX0 as described in section 22.5 CAN Protocol Related Registers.

The three test functions for pin CAN_TX interfere with all CAN protocol functions. CAN_TX must be left in its default function when CAN message transfer or any of the test modes Loop Back Mode, Silent Mode, or Basic Mode are selected.

## 22.3 Register Description

This section lists the CAN registers and describes the function of each register in detail.

### Programmer’s Model

The CAN module allocates an address space of 256 bytes (64 words). The CAN registers can be accessed from the CPU in byte and word.

The two sets of interface registers (IF1 and IF2) control the CPU access to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and message reception/transmission.

If several CAN modules are present on a device then they are located linearly in the address space with a constant offset of 256 bytes. Please refer to the data sheet for the base address of each CAN module on the device.

Table 22-1.  CAN Register Summary

| Offset | Byte Register Name | Word Register Name | Description | Initial value |
|---|---|---|---|---|
| CANn base address + 0x0 | CTRLRLn | CTRLRn | CAN n - Control register | 000X0001 |
| CANn base address + 0x1 | CTRLRHn | | CAN n - Control register (reserved) | XXXXXXXX |
| CANn base address + 0x2 | STATRLn | STATRn | CAN n - Status register | 00000000 |
| CANn base address + 0x3 | STATRHn | | CAN n - Status register (reserved) | XXXXXXXX |
| CANn base address + 0x4 | ERRCNTLn | ERRCNTn | CAN n - Error Counter (Transmit) | 00000000 |
| CANn base address + 0x5 | ERRCNTHn | | CAN n - Error Counter (Receive) | 00000000 |

Table 22-1. CAN Register Summary

| Offset | Byte Register Name | Word Register Name | Description | Initial value |
|---|---|---|---|---|
| CANn base address + 0x6 | BTRLn | BTRn | CAN n - Bit Timing Register | 00000001 |
| CANn base address + 0x7 | BTRHn | | CAN n - Bit Timing Register | X0100011 |
| CANn base address + 0x8 | INTRLn | INTRn | CAN n - Interrupt Register | 00000000 |
| CANn base address + 0x9 | INTRHn | | CAN n - Interrupt Register | 00000000 |
| CANn base address + 0xA | TESTRLn | TESTRn | CAN n - Test Register | 000000XX |
| CANn base address + 0xB | TESTRHn | | CAN n - Test Register (reserved) | XXXXXXXX |
| CANn base address + 0xC | BRPERLn | BRPERn | CAN n - BRP Extension register | XXXX0000 |
| CANn base address + 0xD | BRPERHn | | CAN n - BRP Extension register (reserved) | XXXXXXXX |
| IF1 registers | | | | |
| CANn base address + 0x10 | IF1CREQLn | IF1CREQn | CAN n - IF1 Command request register | 00000001 |
| CANn base address + 0x11 | IF1CREQHn | | CAN n - IF1 Command request register | 0XXXXXXX |
| CANn base address + 0x12 | IF1CMSKLn | IF1CMSKn | CAN n - IF1 Command Mask register | 00000000 |
| CANn base address + 0x13 | IF1CMSKHn | | CAN n - IF1 Command Mask register (reserved) | XXXXXXXX |
| CANn base address + 0x14 | IF1MSK1Ln | IF1MSK1n | CAN n - IF1 Mask Register | 11111111 |
| CANn base address + 0x15 | IF1MSK1Hn | | CAN n - IF1 Mask Register | 11111111 |
| CANn base address + 0x16 | IF1MSK2Ln | IF1MSK2n | CAN n - IF1 Mask Register | 11111111 |
| CANn base address + 0x17 | IF1MSK2Hn | | CAN n - IF1 Mask Register | 11X11111 |
| CANn base address + 0x18 | IF1ARB1Ln | IF1ARB1n | CAN n - IF1 Arbitration register | 00000000 |
| CANn base address + 0x19 | IF1ARB1Hn | | CAN n - IF1 Arbitration register | 00000000 |
| CANn base address + 0x1A | IF1ARB2Ln | IF1ARB2n | CAN n - IF1 Arbitration register | 00000000 |
| CANn base address + 0x1B | IF1ARB2Hn | | CAN n - IF1 Arbitration register | 00000000 |
| CANn base address + 0x1C | IF1MCTRLn | IF1MCTRn | CAN n - IF1 Message Control Register | 0XXX0000 |
| CANn base address + 0x1D | IF1MCTRHn | | CAN n - IF1 Message Control Register | 00000000 |
| CANn base address + 0x1E | IF1DTA1Ln | IF1DTA1n | CAN n - IF1 Data A1 | 00000000 |
| CANn base address + 0x1F | IF1DTA1Hn | | CAN n - IF1 Data A1 | 00000000 |
| CANn base address + 0x20 | IF1DTA2Ln | IF1DTA2n | CAN n - IF1 Data A2 | 00000000 |
| CANn base address + 0x21 | IF1DTA2Hn | | CAN n - IF1 Data A2 | 00000000 |
| CANn base address + 0x22 | IF1DTB1Ln | IF1DTB1n | CAN n - IF1 Data B1 | 00000000 |
| CANn base address + 0x23 | IF1DTB1Hn | | CAN n - IF1 Data B1 | 00000000 |
| CANn base address + 0x24 | IF1DTB2Ln | IF1DTB2n | CAN n - IF1 Data B2 | 00000000 |
| CANn base address + 0x25 | IF1DTB2Hn | | CAN n - IF1 Data B2 | 00000000 |
| IF2 registers | | | | |
| CANn base address + 0x40 | IF2CREQLn | IF2CREQn | CAN n - IF2 Command request register | 00000001 |
| CANn base address + 0x41 | IF2CREQHn | | CAN n - IF2 Command request register | 0XXXXXXX |
| CANn base address + 0x42 | IF2CMSKLn | IF2CMSKn | CAN n - IF2 Command Mask register | 00000000 |
| CANn base address + 0x43 | IF2CMSKHn | | CAN n - IF2 Command Mask register (reserved) | XXXXXXXX |
| CANn base address + 0x44 | IF2MSK1Ln | IF2MSK1n | CAN n - IF2 Mask Register | 11111111 |
| CANn base address + 0x45 | IF2MSK1Hn | | CAN n - IF2 Mask Register | 11111111 |
| CANn base address + 0x46 | IF2MSK2Ln | IF2MSK2n | CAN n - IF2 Mask Register | 11111111 |
| CANn base address + 0x47 | IF2MSK2Hn | | CAN n - IF2 Mask Register | 11X11111 |

Table 22-1. CAN Register Summary

| Offset | Byte Register Name | Word Register Name | Description | Initial value |
|---|---|---|---|---|
| CANn base address + 0x48 | IF2ARB1Ln | IF2ARB1n | CAN n - IF2 Arbitration register | 00000000 |
| CANn base address + 0x49 | IF2ARB1Hn | | CAN n - IF2 Arbitration register | 00000000 |
| CANn base address + 0x4A | IF2ARB2Ln | IF2ARB2n | CAN n - IF2 Arbitration register | 00000000 |
| CANn base address + 0x4B | IF2ARB2Hn | | CAN n - IF2 Arbitration register | 00000000 |
| CANn base address + 0x4C | IF2MCTRLn | IF2MCTRn | CAN n - IF2 Message Control Register | 0XXX0000 |
| CANn base address + 0x4D | IF2MCTRHn | | CAN n - IF2 Message Control Register | 00000000 |
| CANn base address + 0x4E | IF2DTA1Ln | IF2DTA1n | CAN n - IF2 Data A1 | 00000000 |
| CANn base address + 0x4F | IF2DTA1Hn | | CAN n - IF2 Data A1 | 00000000 |
| CANn base address + 0x50 | IF2DTA2Ln | IF2DTA2n | CAN n - IF2 Data A2 | 00000000 |
| CANn base address + 0x51 | IF2DTA2Hn | | CAN n - IF2 Data A2 | 00000000 |
| CANn base address + 0x52 | IF2DTB1Ln | IF2DTB1n | CAN n - IF2 Data B1 | 00000000 |
| CANn base address + 0x53 | IF2DTB1Hn | | CAN n - IF2 Data B1 | 00000000 |
| CANn base address + 0x54 | IF2DTB2Ln | IF2DTB2n | CAN n - IF2 Data B2 | 00000000 |
| CANn base address + 0x55 | IF2DTB2Hn | | CAN n - IF2 Data B2 | 00000000 |
| | | | | |
| CANn base address + 0x80 | TREQR1Ln | TREQR1n | CAN n - Transmission Request Register | 00000000 |
| CANn base address + 0x81 | TREQR1Hn | | CAN n - Transmission Request Register | 00000000 |
| CANn base address + 0x82 | TREQR2Ln | TREQR2n | CAN n - Transmission Request Register | 00000000 |
| CANn base address + 0x83 | TREQR2Hn | | CAN n - Transmission Request Register | 00000000 |
| CANn base address + 0x90 | NEWDT1Ln | NEWDT1n | CAN n - New Data Register | 00000000 |
| CANn base address + 0x91 | NEWDT1Hn | | CAN n - New Data Register | 00000000 |
| CANn base address + 0x92 | NEWDT2Ln | NEWDT2n | CAN n - New Data Register | 00000000 |
| CANn base address + 0x93 | NEWDT2Hn | | CAN n - New Data Register | 00000000 |
| CANn base address + 0xA0 | INTPND1Ln | INTPND1n | CAN n - Interrupt Pending Register | 00000000 |
| CANn base address + 0xA1 | INTPND1Hn | | CAN n - Interrupt Pending Register | 00000000 |
| CANn base address + 0xA2 | INTPND2Ln | INTPND2n | CAN n - Interrupt Pending Register | 00000000 |
| CANn base address + 0xA3 | INTPND2Hn | | CAN n - Interrupt Pending Register | 00000000 |
| CANn base address + 0xB0 | MSGVAL1Ln | MSGVAL1n | CAN n - Message Valid Register | 00000000 |
| CANn base address + 0xB1 | MSGVAL1Hn | | CAN n - Message Valid Register | 00000000 |
| CANn base address + 0xB2 | MSGVAL2Ln | MSGVAL2n | CAN n - Message Valid Register | 00000000 |
| CANn base address + 0xB3 | MSGVAL2Hn | | CAN n - Message Valid Register | 00000000 |
| CANn base address + 0xCE | COERn | | CAN n - Output enable register | XXXXXXX0 |

## Hardware Reset Description

After hardware reset, the registers of the CAN hold the values described as initial value in Table 22-1.

Additionally, the busoff state is reset and the output CAN_TX is set to recessive (HIGH). The value 0x0001 (INIT = ‘1’) in the CAN Control Register CTRLRLn enables the software initialisation. The CAN does not influence the CAN bus until the CPU resets INIT to ‘0’.

The data stored in the Message RAM is not affected by a hardware reset. After power-on, the contents of the Message RAM is undefined.

## 22.4 CAN Device Related Registers

These registers are related to the device which hosts the CANbus controller.

### 22.4.1 CAN Output Enable Register (COERn)

The CAN Output Enable Register (COERn) controls whether the device pins is used as CANbus controller's TX pin or not.

#### CAN Output Enable Register (COERn)

Figure 22-4. Configuration of the CAN Output Enable Register (COERn)

| CAN Output Enable Register | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0xCE | res | res | res | res | res | res | res | OE | COERn |
| Read/write ⇒ | (R/W0) | (R/W0) | (R/W0) | (R/W0) | (R/W0) | (R/W0) | (R/W0) | (R/W) | |
| Default value⇒ | (X) | (X) | (X) | (X) | (X) | (X) | (X) | (0) | |

#### Function of the CAN Output Enable Register (COERn)

[bit7-bit1] Reserved Bits

Always write "0". Read value is not defined. Read-Modify-Write is not affected.

| [bit0] | OE | Output Enable |
|---|---|---|
| | 0 | CAN TX is disabled. Pin can be used as General Purpose Port or for other peripheral function |
| | 1 | CAN TX is enabled. |

## 22.5 CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

### 22.5.1 CAN Control Register (CTRLRn)

The CAN Control Register (CTRLRn) controls the basic operation modes of the CAN controller.

#### CAN Control Register (CTRLRn)

Figure 22-5. Configuration of the CAN Control Register (CTRLRn)

| CAN Control Register high byte | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x01 | res | res | res | res | res | res | res | res | CTRLRHn |
| Read/write ⇒ | (R) | (R) | (R) | (R) | (R) | (R) | (R) | (R) | |
| Default value⇒ | (X) | (X) | (X) | (X) | (X) | (X) | (X) | (X) | |

| CAN Control Register low byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x00 | TEST | CCE | DAR | res | EIE | SIE | IE | INIT | CTRLRLn |
| Read/write ⇒ | (R/W) | (R/W) | (R/W) | (R) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Default value⇒ | (0) | (0) | (0) | (X) | (0) | (0) | (0) | (1) | |

### Function of the CAN Control Register (CTRLRn)

| | | |
|---|---|---|
| [bit15 - bit8] | | Reserved Bits<br>Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit7] | TEST | Test Mode Enable |
| | 0 | Normal Operation. |
| | 1 | Test Mode. |
| [bit6] | CCE | Configuration Change Enable |
| | 0 | The CPU has no write access to the Bit Timing Register. |
| | 1 | The CPU has write access to the Bit Timing Register (while Init = "1") |
| [bit5] | DAR | Disable Automatic Retransmission |
| | 0 | Automatic Retransmission of disturbed messages enabled. |
| | 1 | Automatic Retransmission disabled. |

**Note:**

When the C_CAN is configured to work in DAR-mode (Disable Automatic Retransmission) and the host requests the transmission of several messages at the same time, only two of these messages will be transmitted. For all other requested transmit messages, the TXRQST bits will be reset, but no transmission will be started, NEWDAT and INTPND will be left unchanged. For the two messages that are transmitted, the TXRQST and NEWDAT bits will be reset and, if enabled by TXIE, INTPND will be set.

| | | |
|---|---|---|
| [bit4] | res | reserved bit<br>Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit3] | EIE | Error Interrupt Enable |
| | 0 | Disabled - No Error Status Interrupt will be generated. |
| | 1 | Enabled - A change in the bits BOff or EWarn in the Status Register will generate an interrupt. |
| [bit2] | SIE | Status Change Interrupt Enable |
| | 0 | Disabled - No Status Change Interrupt will be generated. |
| | 1 | Enabled - An interrupt will be generated when a message transfer is successfully completed or a CAN bus error is detected. |
| [bit1] | IE | Module Interrupt Enable |
| | 0 | Disabled - Module Interrupt is always inactive. |
| | 1 | Enabled - Interrupts will be generated. The request remains active until all pending interrupts are processed. |
| [bit0] | INIT | Initialization |
| | 0 | Normal Operation |
| | 1 | Initialization is started. |

**Note:**

The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting INIT. If the device goes busoff, it will set INIT of its own accord, stopping all bus activities. Once INIT has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 * 11 consecutive recessive bits) before resuming normal operation. At the end of the busoff recovery sequence, the Error Management Counters will be reset.

**Note:**

During the waiting time after the resetting of INIT, each time a sequence of 11 recessive bits has been monitored, a Bit0Error code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at dominant level or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

## 22.5.2 Status Register (STATRn)

The Status Register (STATRn) shows the status of the CAN controller.

### Status Register (STATRn)

Figure 22-6. Configuration of the Status Register (STATRn)

| Status Register high byte | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x03 | res | res | res | res | res | res | res | res | STATRHn |
| Read/write ⇒ | (R) | (R) | (R) | (R) | (R) | (R) | (R) | (R) | |
| Default value⇒ | (X) | (X) | (X) | (X) | (X) | (X) | (X) | (X) | |

| Status Register low byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x02 | BOFF | EWARN | EPASS | RXOK | TXOK | LEC | | | STATRLn |
| Read/write ⇒ | (R) | (R) | (R) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Default value⇒ | (0) | (0) | (0) | (0) | (0) | (0) | (0) | (0) | |

### Function of the Status Register (STATRn)

| | | |
|---|---|---|
| [bit15 - bit8] | | Reserved Bits<br>Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit7] | BOFF | Busoff Status |
| | 0 | The CAN module is not busoff. |
| | 1 | The CAN module is in busoff state. |
| [bit6] | EWARN | Warning Status |
| | 0 | Both error counters are below the error warning limit of 96. |
| | 1 | At least one of the error counters in the EML has reached the error warning limit of 96. |
| [bit5] | EPASS | Error Passive |
| | 0 | The CAN Core is error active. |
| | 1 | The CAN Core is in the error passive state as defined in the CAN Specification. |
| [bit4] | RXOK | Received a Message Successfully |
| | 0 | Since this bit was last reset by the CPU, no message has been successfully received. This bit is never reset by the CAN Core. |
| | 1 | Since this bit was last reset (to zero) by the CPU, a message has been successfully received (independent of the result of acceptance. |
| [bit3] | TXOK | Transmitted a Message Successfully |
| | 0 | Since this bit was last reset by the CPU, no message has been successfully transmitted. This bit is never reset by the CAN Core. |
| | 1 | Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted. |

| [bit2 - bit0] | LEC | Last Error Code (Type of the last error to occur on the CAN bus) | |
|---|---|---|---|
| | 0 | No Error | |
| | 1 | Stuff Error | More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. |
| | 2 | Form Error | A fixed format part of a received frame has the wrong format. |
| | 3 | AckError | The message this CAN Core transmitted was not acknowledged by another node. |
| | 4 | Bit1Error | During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant. |
| | 5 | Bit0Error | During the transmission of a message (or acknowledge bit or active error flag, or over-load flag), the device wanted to send a dominant level (data or identifier bit logical value '0'), but the monitored Bus value was recessive. During busoff recovery this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at dominant level or continuously disturbed). |
| | 6 | CRCError | The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data. |
| | 7 | unused | When the LEC shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC. |

The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0' when a message has been transferred (reception or transmission) without error. The unused code '7' may be written by the CPU to check for updates.

### Status Interrupts

A Status Interrupt is generated by bits BOFF and EWARN (Error Interrupt) or by RXOK, TXOK, and LEC (Status Change Interrupt) assuming that the corresponding enable bits in the CAN Control Register are set. A change of bit EPASS or a write to RXOK, TXOK,or LEC will never generate a Status Interrupt.

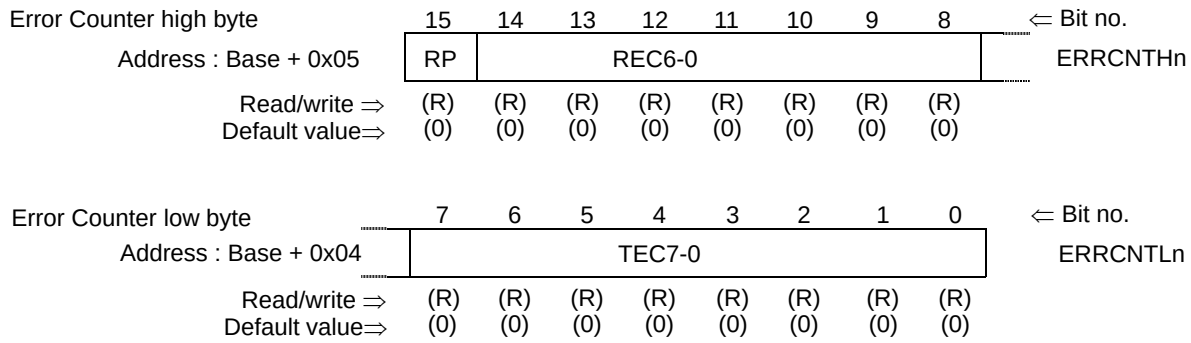
Reading the Status Register will clear the Status Interrupt value (8000h) in the Interrupt Register, if it is pending.

## 22.5.3 Error Counter (ERRCNTn)

The Error Counter Register (ERRCNTn) contains the Receive and Transmit Error Counters REC and TEC.

### Error Counter (ERRCNTn)

Figure 22-7. Configuration of the Error Counter (ERRCNTn)




### Function of the Error Counter (ERRCNTn)

| [bit15] | RP | Receive Error Passive |
|---|---|---|
| | 0 | The Receive Error Counter is below the error passive level. |
| | 1 | The Receive Error Counter has reached the error passive level as defined in the CAN Specification. |

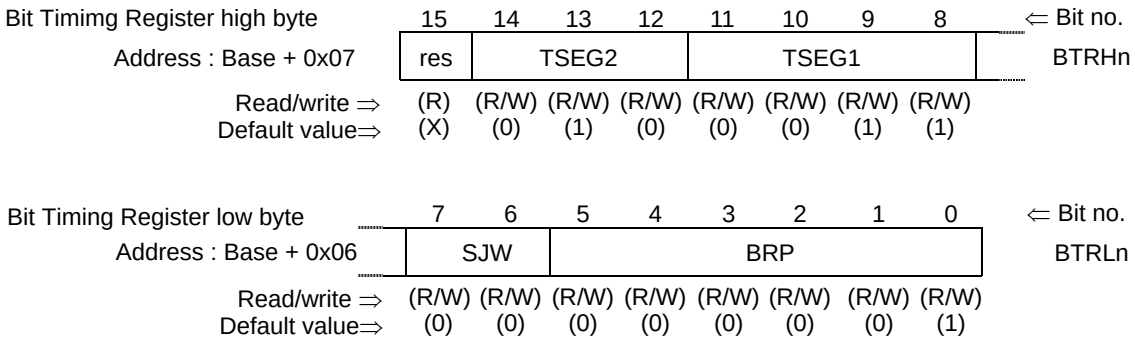
| | | |
|---|---|---|
| [bit14 - bit8] | REC6-0 | Receive Error Counter<br>Actual state of the Receive Error Counter. Values between 0 and 127. |
| [bit7 - bit0] | TEC7-0 | Transmit Error Counter<br>Actual state of the Transmit Error Counter. Values between 0 and 255. |

## 22.5.4 Bit Timing Register (BTRn)

The Bit Timing Register (BTRn) enables controlling of the CAN bus controller bit timing.

### Bit Timing Register (BTRn)

Figure 22-8. Configuration of the Bit Timing Register (BTRn)




### Function of the Bit Timing Register (BTRn)

| | | |
|---|---|---|
| [bit15] | res | Reserved bit<br>Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit114 - bit12] | TSEG2<br>0x0-0x7 | The time segment after the sample point<br>Valid values for TSEG2 are [ 0 … 7 ]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used. |
| [bit11 - bit8] | TSEG1<br>0x01-0x0F | The time segment before the sample point<br>Valid values for TSEG1 are [ 1 … 15 ]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used |
| [bit7 - 6] | SJW<br>0x0-0x3 | (Re)Synchronisation Jump Width<br>Valid programmed values are 0-3. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used. |
| [bit5 - bit0] | BRP<br>0x00-0x3F | Baud Rate Prescaler<br>The value by which the peripheral clock CLKP2 frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are[0 … 63]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used. |

### Note:

With a peripheral clock CLKP2 frequency of 8 MHz, the reset value of 0x2301 configures the CAN for a bit rate of 500 kBit/s. The registers are only writable if bits CCE and INIT in the CAN Control Register are set.

## 22.5.5 Test Register (TESTRn)

The Test Register (TESTRn) offers functionality to test the CAN bus system.

### Test Register (TESTRn)

Figure 22-9. Configuration of the Test Register (TESTRn)

| Test Register high byte | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x0B | res | res | res | res | res | res | res | res | TESTRHn |
| Read/write ⇒ | (R) | (R) | (R) | (R) | (R) | (R) | (R) | (R) | |
| Default value⇒ | (X) | (X) | (X) | (X) | (X) | (X) | (X) | (X) | |

| Test Register low byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x0A | RX | TX1 | TX0 | LBACK | SILENT | BASIC | res | res | TESTRLn |
| Read/write ⇒ | (R) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R) | (R) | |
| Default value⇒ | (0) | (0) | (0) | (0) | (0) | (0) | (X) | (X) | |

### Function of the Test Register (TESTRn)

| | | |
|---|---|---|
| [bit15-bit8] | | Reserved bits |
| | | Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit7] | RX | Monitors the actual value of the CAN_RX Pin |
| | 0 | The CAN bus is dominant (CAN_RX = ‘0’). |
| | 1 | The CAN bus is recessive (CAN_RX = ‘1’). |
| [bit6-bit5] | TX1-0 | Control of CAN_TX pin |
| | 00 | Reset value, CAN_TX is controlled by the CAN Core. |
| | 01 | Sample Point can be monitored at CAN_TX pin. |
| | 10 | CAN_TX pin drives a dominant (‘0’) value. |
| | 11 | CAN_TX pin drives a recessive (‘1’) value. |
| [bit4] | LBACK | Loop Back Mode |
| | 0 | Loop Back Mode is disabled. |
| | 1 | Loop Back Mode is enabled. |
| [bit3] | SILENT | Silent Mode |
| | 0 | Normal operation. |
| | 1 | The module is in Silent Mode. |
| [bit2] | BASIC | Basic Mode |
| | 0 | Basic Mode disabled. |
| | 1 | IF1 Registers used as Tx Buffer, IF2 Registers used as Rx Buffer. |
| [bit1-bit0] | res | Reserved Bits |
| | | Always write "0". Read value is not defined. Read-Modify-Write is not affected. |

### Note:

Write access to the Test Register is enabled by setting bit TEST in the CAN Control Register CTRLRn. The different test functions may be combined, but TX1-0 /= “00” disturbs message transfer.

### 22.5.6 BRP Extension Register (BRPERn)

The BRP Extension Register (BRPERn) contains an extension for the baud rate generator prescaler.

#### BRP Extension Register (BRPERn)

Figure 22-10. Configuration of the BRP Extension Register (BRPERn)

| BRP Extension Register high byte | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x0D | res | res | res | res | res | res | res | res | BRPERHn |
| Read/write ⇒ | (R) | (R) | (R) | (R) | (R) | (R) | (R) | (R) | |
| Default value⇒ | (X) | (X) | (X) | (X) | (X) | (X) | (X) | (X) | |

| BRP Extension Register low byte | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| Address : Base + 0x0C | res | res | res | res | BRPE | | | | BRPERLn |
| Read/write ⇒ | (R) | (R) | (R) | (R) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Default value⇒ | (X) | (X) | (X) | (X) | (0) | (0) | (0) | (0) | |

#### Function of the BRP Extension Register (BRPERn)

| | | |
|---|---|---|
| [bit15-bit4] | res | Reserved Bits<br>Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit3-bit0] | BRPE | Baud Rate Prescaler Extension |
| | 0x00-0x0F | By programming BRPE the Baud Rate Prescaler can be extended to values up to 1023. The actual interpretation by the hardware is that one more than the value programmed by BRPE (MSBs) and BRP (LSBs) is used. |

## 22.6 Message Interface Register Sets

To avoid access conflicts between the CPU and the CAN bus controller in accessing the Message RAM, there are two sets of Interface Registers which are used to control the CPU access to the Message RAM.

#### Overview of Message Interface Register Sets

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object (see section 22.7 Message Object in the Message Memory) or parts of the Message Object may be transferred between the Message RAM and the IFx Message Buffer registers in one single transfer.

The function of the two interface register sets is identical (except for test mode BASIC). They can be used the way that one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other. Table 22-2 gives an overview of the two Interface Register sets.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

Table 22-2. IF1 and IF2 Message Interface Register Sets

| Address | IF1 register set | Address | IF2 register set |
|---|---|---|---|
| CAN Base + 0x10 | IF1 Command Request (IF1CREQn) | CAN Base + 0x40 | IF2 Command Request (IF2CREQn) |
| CAN Base + 0x12 | IF1 Command Mask (IF1CMSKn) | CAN Base + 0x42 | IF2 Command Mask (IF2CMSKn) |
| CAN Base + 0x14 | IF1 Mask 1 (IF1MSK1n) | CAN Base + 0x44 | IF2 Mask 1 (IF2MSK1n) |
| CAN Base + 0x16 | IF1 Mask 2 (IF1MSK2n) | CAN Base + 0x46 | IF2 Mask 2 (IF2MSK2n) |
| CAN Base + 0x18 | IF1 Arbitration 1 (IF1ARB1n) | CAN Base + 0x48 | IF2 Arbitration 1 (IF2ARB1n) |
| CAN Base + 0x1A | IF1 Arbitration 2 (IF1ARB2n) | CAN Base + 0x4A | IF2 Arbitration 2 (IF2ARB2n) |
| CAN Base + 0x1C | IF1 Message Control (IF1MCTRn) | CAN Base + 0x4C | IF2 Message Control (IF2MCTRn) |
| CAN Base + 0x1E | IF1 Data A1 (IF1DTA1n) | CAN Base + 0x4E | IF2 Data A1 (IF2DTA1n) |
| CAN Base + 0x20 | IF1 Data A2 (IF1DTA2n) | CAN Base + 0x50 | IF2 Data A2 (IF2DTA2n) |
| CAN Base + 0x22 | IF1 Data B1 (IF1DTB1n) | CAN Base + 0x52 | IF2 Data B1 (IF2DTB1n) |
| CAN Base + 0x24 | IF1 Data B2 (IF1DTB2n) | CAN Base + 0x54 | IF2 Data B2 (IF2DTB2n) |

## 22.6.1 IFx Command Request Registers (IFxCREQn)

A message transfer is started as soon as the CPU has written the message number to the Command Request Register (IFx-CREQn). With this write operation the CPU is notified that a transfer is in progress. If a CPU access to the CAN happens while the transfer is in progress then this access is delayed until the transfer has finished. After 3 to 6 CAN_CLK periods, the transfer between the Interface Register and the Message RAM has completed and the upcoming CPU access is executed.

### IFx Command Request Registers (IFxCREQn)

Figure 22-11. Configuration of the IFx Command Request Registers (IFxCREQn)

IFx Command Request Register high byte — Address (x = 1): CANn base + 0x11, (x = 2): CANn base + 0x41 — IFxCREQHn

| ⇐ Bit no. | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | BUSY | res | res | res | res | res | res | res |
| Read/write ⇒ | (R/(W)) | (R) | (R) | (R) | (R) | (R) | (R) | (R) |
| Default value⇒ | (0) | (X) | (X) | (X) | (X) | (X) | (X) | (X) |

IFx Command Request Register low byte — Address (x = 1): CANn base + 0x10, (x = 2): CANn base + 0x40 — IFxCREQLn

| ⇐ Bit no. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MSGN | | | | | | | |
| Read/write ⇒ | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) |
| Default value⇒ | (0) | (0) | (0) | (0) | (0) | (0) | (0) | (1) |

### Function of the IFx Command Request Registers (IFxCREQn)

| | | |
|---|---|---|
| [bit15] | BUSY | Busy Flag |
| | 0 | Reset to zero when read/write action has finished |
| | 1 | Set to one when writing to the IFx Command Request Register |
| [bit14-bit8] | res | Reserved Bits |
| | | Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit5-bit0] | MSGN | Message Number (for 32 message buffer CANs, please refer to the data sheet for the number of message buffers of the device.) |
| | 0x00 | Not a valid Message Number, interpreted as 0x20. |
| | 0x01-0x20 | Valid Message Number, the Message Object in the Message RAM is selected for data transfer. |
| | 0x21-0xFF | Not a valid Message Number, interpreted as 0x01-0x1F. |

| | | |
|---|---|---|
| [bit5-bit0] | MSGN | Message Number (for 64 message buffer CANs, , please refer to data sheet for the number of message buffers of the device.) |
| | 0x00 | Not a valid Message Number, interpreted as 0x20. |
| | 0x01-0x40 | Valid Message Number, the Message Object in the Message RAM is selected for data transfer. |
| | 0x41-0xFF | Not a valid Message Number, interpreted as 0x01-0x3F. |
| [bit7-bit0] | MSGN | Message Number (for 128 message buffer CANs, please refer to the data sheet for the number of message buffers of the device.) |
| | 0x00 | Not a valid Message Number, interpreted as 0x80. |
| | 0x01-0x80 | Valid Message Number, the Message Object in the Message RAM is selected for data transfer. |
| | 0x81-0xFF | Not a valid Message Number, interpreted as 0x01-0x7F. |

**Note:**

The Busy Flag can only be used in BASIC mode (see Basic Mode on page 464). When using the Message RAM (BASIC=0) the hardware interface controls the access for read and write and this flag is always read as ‘0’.

**Note:**

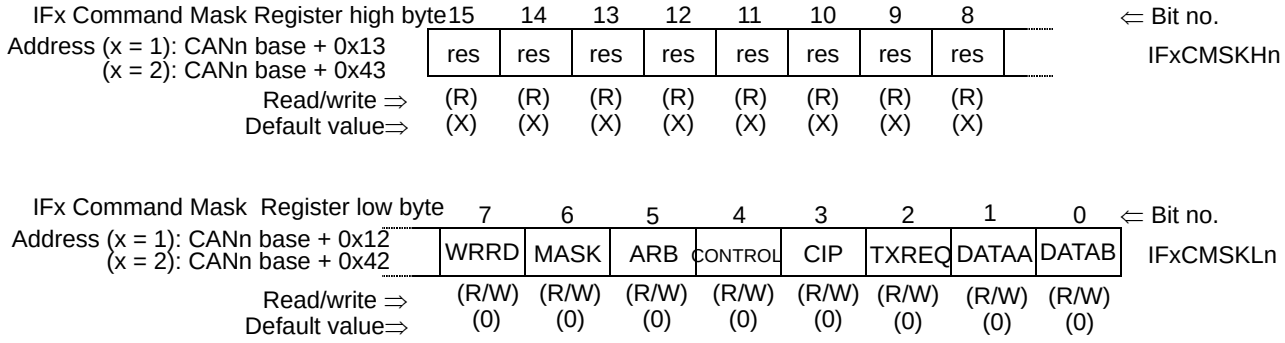
When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

## 22.6.2 IFx Command Mask Register (IFxCMSKn)

The control bits of the IFx Command Mask Register (IFxCMSKn) specify the transfer direction and select which of the IFx Message Buffer Registers are source or target of the data transfer.

### IFx Command Mask Register (IFxCMSKn)

Figure 22-12. Configuration of the IFx Command Mask Register (IFxCMSKn)




### Function of the IFx Command Mask Register (IFxCMSKn)

| | | |
|---|---|---|
| [bit15-bit8] | res | Reserved Bits<br>Always write "0". Read value is not defined. Read-Modify-Write is not affected. |
| [bit7] | WRRD | Write / Read |
| | 0 | Read: Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers. |
| | 1 | Write: Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register. |

The other bits of IFx Command Mask Register have different functions depending on the transfer direction:

### Direction = Write

| | | |
|---|---|---|
| [bit6] | MASK | Access Mask Bits |
| | 0 | Mask bits unchanged. |
| | 1 | Transfer Identifier Mask + MDir + MXtd to Message Object. |
| [bit5] | ARB | Access Arbitration Bits |
| | 0 | Arbitration bits unchanged. |
| | 1 | Transfer Identifier + Dir + Xtd + MsgVal to Message Object. |
| [bit4] | CONTROL | Access Control Bits |
| | 0 | Control Bits unchanged. |
| | 1 | Transfer Control Bits to Message Object. |
| [bit3] | CIP | Clear Interrupt Pending Bit |
| | | When writing to a Message Object, this bit is ignored. |
| [bit2] | TXREQ | Access Transmission Request Bit |
| | 0 | TXRQST bit unchanged |
| | 1 | set TXRQST bit |
| [bit1] | DATAA | Access Data Bytes 0-3 |
| | 0 | Data Bytes 0-3 unchanged. |
| | 1 | Transfer Data Bytes 0-3 to Message Object. |
| [bit0] | DATAB | Access Data Bytes 4-7 |
| | 0 | Data Bytes 4-7 unchanged. |
| | 1 | Transfer Data Bytes 4-7 to Message Object. |

### Direction = Read

| | | |
|---|---|---|
| [bit6] | MASK | Access Mask Bits |
| | 0 | Mask bits unchanged. |
| | 1 | Transfer Identifier Mask + MDir + MXtd to IFx Message Buffer Register. |
| [bit5] | ARB | Access Arbitration Bits |
| | 0 | Arbitration bits unchanged. |
| | 1 | Transfer Identifier + Dir + Xtd + MsgVal to IFx Message Buffer Register. |
| [bit4] | CONTROL | Access Control Bits |
| | 0 | Control Bits unchanged. |
| | 1 | Transfer Control Bits to IFx Message Buffer Register. |
| [bit3] | CIP | Clear Interrupt Pending Bit |
| | 0 | INTPND bit remains unchanged. |
| | 1 | Clear INTPND bit in the Message Object. |
| [bit2] | TXREQ | Access New Data Bit |
| | 0 | NEWDAT bit remains unchanged. |
| | 1 | Clear NEWDAT bit in the Message Object. |
| [bit1] | DATAA | Access Data Bytes 0-3 |
| | 0 | Data Bytes 0-3 unchanged. |
| | 1 | Transfer Data Bytes 0-3 to IFx Message Buffer Register. |

| | | |
|---|---|---|
| [bit0] | DATAB | Access Data Bytes 4-7 |
| | 0 | Data Bytes 4-7 unchanged. |
| | 1 | Transfer Data Bytes 4-7 to IFx Message Buffer Register. |

**Note:**

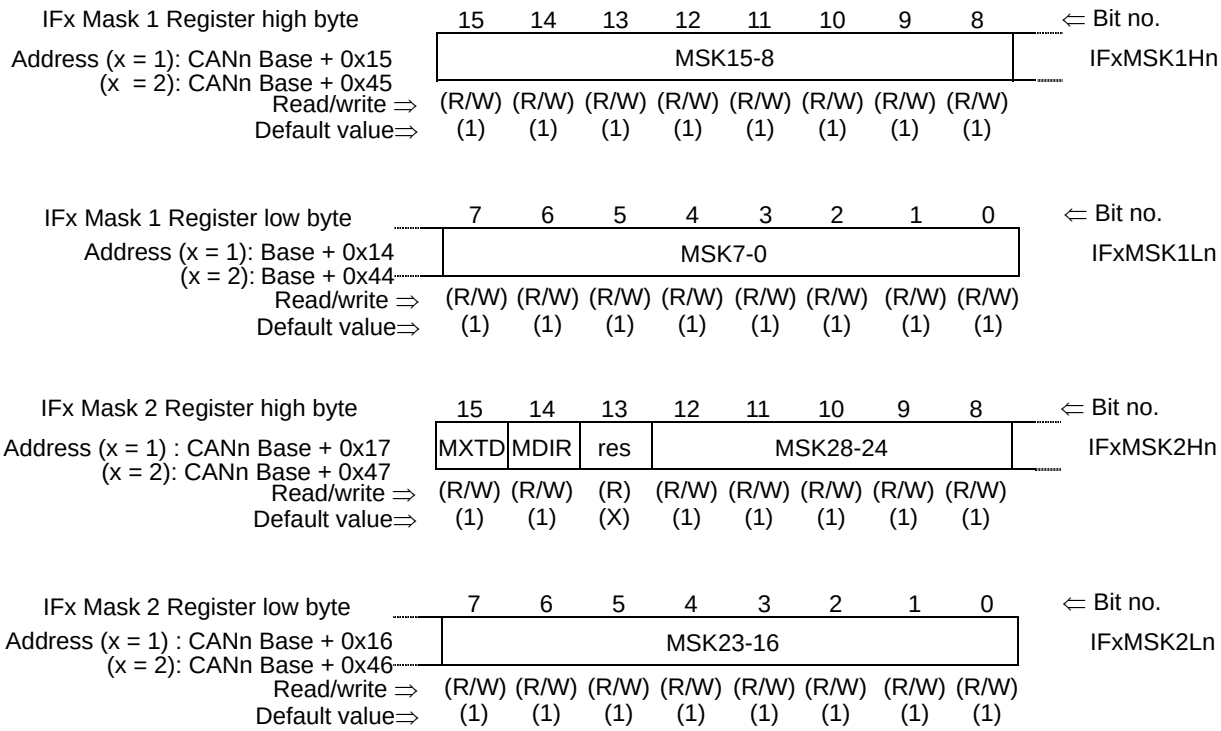
A read access to a Message Object can be combined with the reset of the control bits INTPND and NEWDAT. The values of these bits transferred to the IFx Message Control Register always reflect the status before resetting these bits.

## 22.6.3 IFx Mask Registers (IFxMSK1n, IFxMSK2n)

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The bits of the IFx Mask Registers (IFxMSK1n, IFxMSK2n) specify which parts of the message arbitration registers (IFxARB1n, IFxARB2n) are evaluated.

### IFx Mask Registers (IFxMSK1n, IFxMSK2n)

Figure 22-13. Configuration of the IFx Mask Registers (IFxMSK1n, IFxMSK2n)

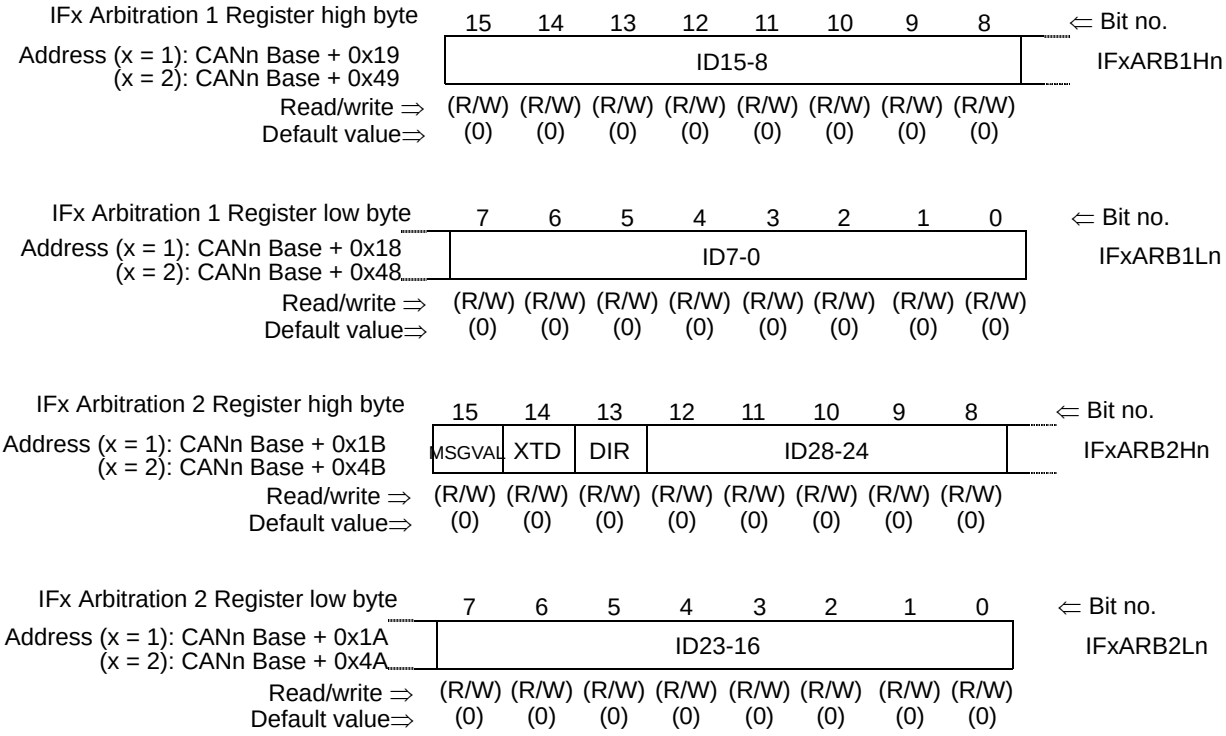


## 22.6.4 IFx Arbitration Registers (IFxARB1n, IFxARB2n)

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. In case of transmission, the bits of the message arbitration registers (IFxARB1n, IFxARB2n) specify the ID of the message. In case of reception, the bits of the message arbitration registers (IFxARB1n, IFxARB2n) specify the acceptance filter.

### IFx Arbitration Registers (IFxARB1n, IFxARB2n)

Figure 22-14. Configuration of the IFx Arbitration Registers (IFxARB1n, IFxARB2n)

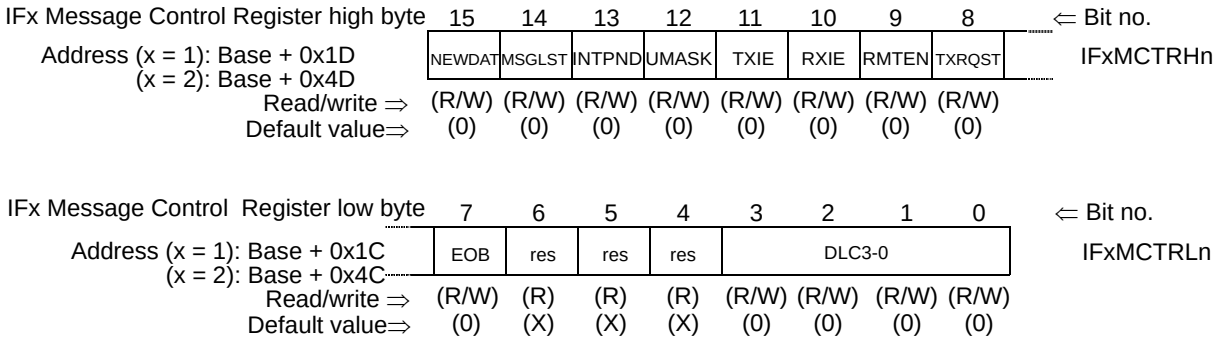



## 22.6.5 IFx Message Control Register (IFxMCTRn)

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The bits of the Message Control Register (IFxMCTRn) contain status information and control bits.

### IFx Message Control Register (IFxMCTRn)

Figure 22-15. Configuration of the IFx Message Control Register (IFxMCTRn)



### 22.6.6 IFx Data A and Data B Registers (IFxDTA1n, IFxDTA2n, IFxDTB1n, IFxDTB2n)

The data bytes of CAN messages are stored in the IFx Message Buffer Registers.

#### IFx Data A and Data B Registers (IFxDTAn, IFxDTBn)

In a CAN Data Frame, Data(0) is the first, Data(7) is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

The data bytes of CAN messages are stored in the IFx Message Buffer Registers in the following order:

Table 22-3.

| Register | Address | Address + 0 content | Address + 1 content |
|---|---|---|---|
| IFxDTA1n | x = 1: CANn base + 0x1E<br>x = 2: CANn base + 0x4E | Data(0) | Data(1) |
| IFxDTA2n | x = 1: CANn base + 0x20<br>x = 2: CANn base + 0x50 | Data(2) | Data(3) |
| IFxDTB1n | x = 1: CANn base + 0x22<br>x = 2: CANn base + 0x52 | Data(4) | Data(5) |
| IFxDTB2n | x = 1: CANn base + 0x24<br>x = 2: CANn base + 0x54 | Data(6) | Data(7) |

## 22.7 Message Object in the Message Memory

There are 32 Message Objects (up to 128 depending on the implementation) in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects, these accesses are handled via the IFx Interface Registers.

#### Structure of a Message Object in the Message Memory

Figure 22-16 gives an overview of the two structure of a Message Object.

Figure 22-16. Structure of a Message Object in the Message Memory

| Message Object | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UMASK | MSK28-0 | MXTD | MDIR | EOB | NEWDAT | | MSGLST | RXIE | TXIE | INTPND | RMTEN | TXRQST |
| MSGVAL | ID28-0 | XTD | DIR | DLC3-0 | Data0 | Data1 | Data2 | Data3 | Data4 | Data5 | Data6 | Data7 |

| MSGVAL | Message Valid |
|---|---|
| 0 | The Message Object is ignored by the Message Handler. |
| 1 | The Message Object is configured and should be considered by the Message Handler. |

**Note:**

The CPU must reset the MSGVAL bit of all unused Messages Objects during the initialization before it resets bit INIT in the CAN Control Register. This bit must also be reset before the identifier ID28-0, the control bits XTD, DIR, or the Data Length Code DLC3-0 are modified, or if the Message Object is no longer required.

| UMASK | Use Acceptance Mask |
|---|---|
| 0 | Mask ignored. |
| 1 | Use Mask (MSK28-0, MXTD, and MDIR) for acceptance filtering. |

**Note:**

If the UMASK bit is set to one, the Message Object's mask bits have to be programmed during initialization of the Message Object before MSGVAL is set to "1".

| | |
|---|---|
| ID28-0 | Message Identifier |
| ID28 - ID0 | 29-bit Identifier (“Extended Frame”). |
| ID28 - ID18 | 11-bit Identifier (“Standard Frame”). |

| | |
|---|---|
| MSK28-0 | Identifier Mask |
| 0 | The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering. |
| 1 | The corresponding identifier bit is used for acceptance filtering. |

| | |
|---|---|
| XTD | Extended Identifier |
| 0 | The 11-bit (“standard”) Identifier will be used for this Message Object. |
| 1 | The 29-bit (“extended”) Identifier will be used for this Message Object. |

| | |
|---|---|
| MXTD | Mask Extended Identifier |
| 0 | The extended identifier bit (XTD) has no effect on the acceptance filtering |
| 1 | The extended identifier bit (XTD) is used for acceptance filtering. |

**Note:**

When 11-bit (“standard”) Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits ID28 to ID18. For acceptance filtering, only these bits together with mask bits MSK28 to MSK18 are considered.

| | |
|---|---|
| DIR | Message Direction |
| 0 | Direction = receive: On TXRQST, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object. |
| 1 | Direction = transmit: On TXRQST, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TXRQST bit of this Message Object is set (if RMTEN = one). |

| | |
|---|---|
| MDIR | Mask Message Direction |
| 0 | The message direction bit (DIR) has no effect on the acceptance filtering. |
| 1 | The message direction bit (DIR) is used for acceptance filtering. |

The Arbitration Registers ID28-0, XTD, and DIR are used to define the identifier and type of outgoing messages and are used (together with the mask registers MSK28-0, MXTD and MDIR) for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction = receive (Data Frame) or Direction = transmit(Remote Frame). Extended frames can be stored only in Message Objects with XTD = one, standard frames in Message Objects with XTD = zero. If a received message (Data Frame or Remote Frame) matches with more than one valid Message Object, it is stored into that with the lowest message number. For details see section Acceptance Filtering of Received Messages on page 490.

| | |
|---|---|
| EOB | End of Buffer |
| 0 | Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer. |
| 1 | Single Message Object or last Message Object of a FIFO Buffer. |

**Note:**

This bit is used to concatenate two ore more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects (not belonging to a FIFO Buffer) this bit must always be set to 1. For details on the concatenation of Message Objects see section Configuration of a FIFO Buffer on page 493.

| | |
|---|---|
| NEWDAT | New Data |
| 0 | No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. |
| 1 | The Message Handler or the CPU has written new data into the data portion of this Message Object. |

| | |
|---|---|
| MSGLST | Message Lost (only valid for Message Objects with direction = receive) |
| 0 | No message lost since last time this bit was reset by the CPU. |
| 1 | The Message Handler stored a new message into this object when NewDat was still set, the CPU has lost a message. |
| RXIE | Receive Interrupt Enable |
| 0 | INTPND will be left unchanged after a successful reception of a frame. |
| 1 | INTPND will be set after a successful reception of a frame. |
| TXIE | Transmit Interrupt Enable |
| 0 | INTPND will be left unchanged after the successful transmission of a frame. |
| 1 | INTPND will be set after a successful transmission of a frame. |
| INTPND | Interrupt Pending |
| 0 | This message object is not the source of an interrupt. |
| 1 | This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority. |
| RMTEN | Remote Enable |
| 0 | At the reception of a Remote Frame, TXRQST is left unchanged. |
| 1 | At the reception of a Remote Frame, TXRQST is set. |
| TXRQST | Transmit Request |
| 0 | This Message Object is not waiting for transmission. |
| 1 | The transmission of this Message Object is requested and is not yet done. |

### Note:

When the lowest priority message buffer is used for transmission, setting TXRQST = "0" may cause a delay of transmission, when TXRQST is set to "1" again.

Depending on the exact time when TXRQST was set to 0, the message may not be transmitted immediately after setting TXRQST = "1", but after any of the following events:

1. there is activity ongoing on the CANbus

2. a transmission request is issued on another message object

3. the CANbus is initialized by the INIT bit

In general, there is no need to cancel an ongoing transmission by setting TXRQST = 0. If the content of a message object needs to be changed while TXRQST = 1, it is sufficient to update the message object via the CPU interface registers (Identifier, DLC, Data, with TXRQST and NEWDAT, optionally TXIE). The updated content will be transmitted at the next opportunity.

| | |
|---|---|
| DLC3-0 | Data Length Code |
| 0-8 | Data Frame has 0-8 data bytes. |
| 9-15 | Data Frame has 8 data bytes. |

### Note:

The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.

| | |
|---|---|
| Data0 | 1st data byte of a CAN Data Frame |
| Data1 | 2nd data byte of a CAN Data Frame |
| Data2 | 3rd data byte of a CAN Data Frame |
| Data3 | 4th data byte of a CAN Data Frame |

| | |
|---|---|
| Data4 | 5th data byte of a CAN Data Frame |
| Data5 | 6th data byte of a CAN Data Frame |
| Data6 | 7th data byte of a CAN Data Frame |
| Data7 | 8th data byte of a CAN Data Frame |

**Note:**

Byte Data0 is the first data byte shifted into the shift register of the CAN Core during a reception, byte Data7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

# 22.8 Message Handler Registers

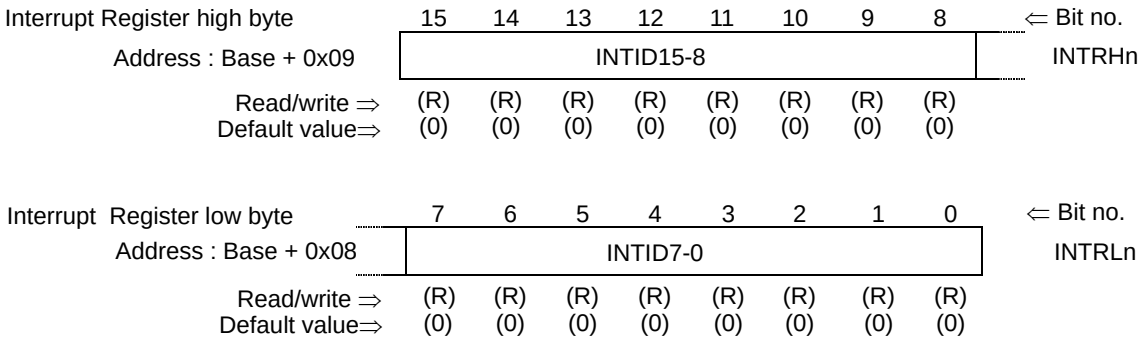
All Message Handler registers are read-only. Their contents (TXRQST, NEWDAT, INTPND, and MSGVAL bits of each Message Object and the Interrupt Identifier) is status information provided by the Message Handler FSM.

## 22.8.1 Interrupt Register (INTRn)

The Interrupt Register (INTRn) points to the pending interrupt of highest priority.

### Interrupt Register (INTRn)

Figure 22-17. Configuration of the Interrupt Register (INTRn)




### Function of the Interrupt Register (INTRn)

(For 32 message buffer CANs)

| | |
|---|---|
| INTID15-0 | Interrupt Identifier (the number here indicates the source of the interrupt) |
| 0x0000 | No interrupt is pending. |
| 0x0001-0x0020 | Number of Message Object which caused the interrupt. |
| 0x0021-0x7FFF | unused. |
| 0x8000 | Status Interrupt. |
| 0x8001-0xFFFF | unused. |

(For 128 message buffer CANs)

| | |
|---|---|
| INTID15-0 | Interrupt Identifier (the number here indicates the source of the interrupt) |
| 0x0000 | No interrupt is pending. |
| 0x0001-0x0080 | Number of Message Object which caused the interrupt. |
| 0x0081-0x7FFF | unused. |
| 0x8000 | Status Interrupt. |
| 0x8001-0xFFFF | unused. |

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If IntId is different from 0x0000 and IE is set, the interrupt line to the CPU is active. The interrupt line remains active until IntId is back to value 0x0000 (the cause of the interrupt is reset) or until IE is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

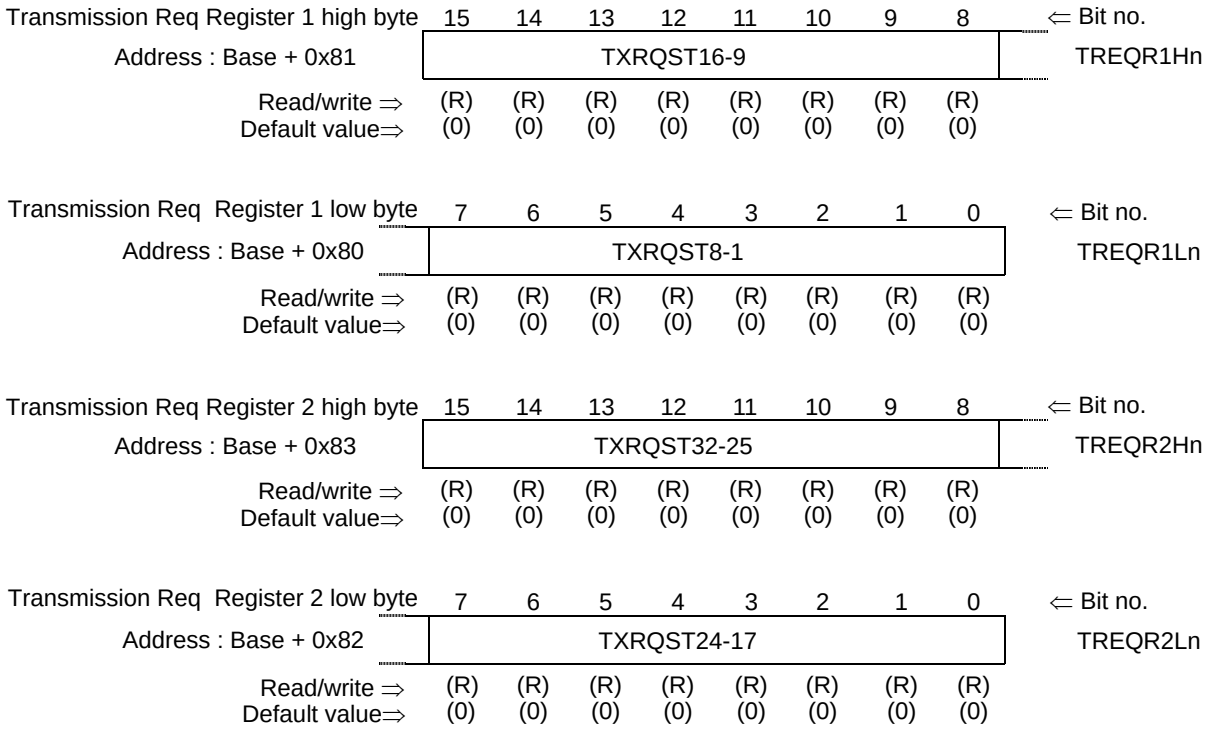
A message interrupt is cleared by clearing the Message Object's INTPND bit. The Status Interrupt is cleared by reading the Status Register.

## 22.8.2 Transmission Request Registers (TREQR1n, TREQR2n)

The Transmission Request Registers (TREQR1n, TREQR2n) control the transmission of message objects.

### Transmission Request Registers (TREQR1n, TREQR2n)

Figure 22-18. Transmission Request Registers (TREQR1n, TREQR2n)




### Function of the Transmission Request Registers (TREQR1n, TREQR2n)

| | |
|---|---|
| TXRQST32-1 | Transmission Request Bits (of all Message Objects) |
| 0 | This Message Object is not waiting for transmission. |
| 1 | The transmission of this Message Object is requested and is not yet done. |

These registers hold the TXRQST bits of the 32 Message Objects. By reading out the TXRQST bits, the CPU can check for which Message Object a Transmission Request is pending. The TXRQST bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

**Note:**

When the lowest priority message buffer is used for transmission, setting TXRQST = "0" may cause a delay of transmission, when TXRQST is set to "1" again.

Depending on the exact time when TXRQST was set to 0, the message may not be transmitted immediately after setting TXRQST = "1", but after any of the following events:

1. there is activity ongoing on the CANbus

2. a transmission request is issued on another message object

3. the CANbus is initialized by the INIT bit

In general, there is no need to cancel an ongoing transmission by setting TXRQST = 0. If the content of a message object needs to be changed while TXRQST = 1, it is sufficient to update the message object via the CPU interface registers (Identifier, DLC, Data, with TXRQST and NEWDAT, optionally TXIE). The updated content will be transmitted at the next opportunity.

If more than 32 message buffers are implemented, the following table gives an overview about the additional flags:

Table 22-4. Additional flags when more than 32 message buffers exist

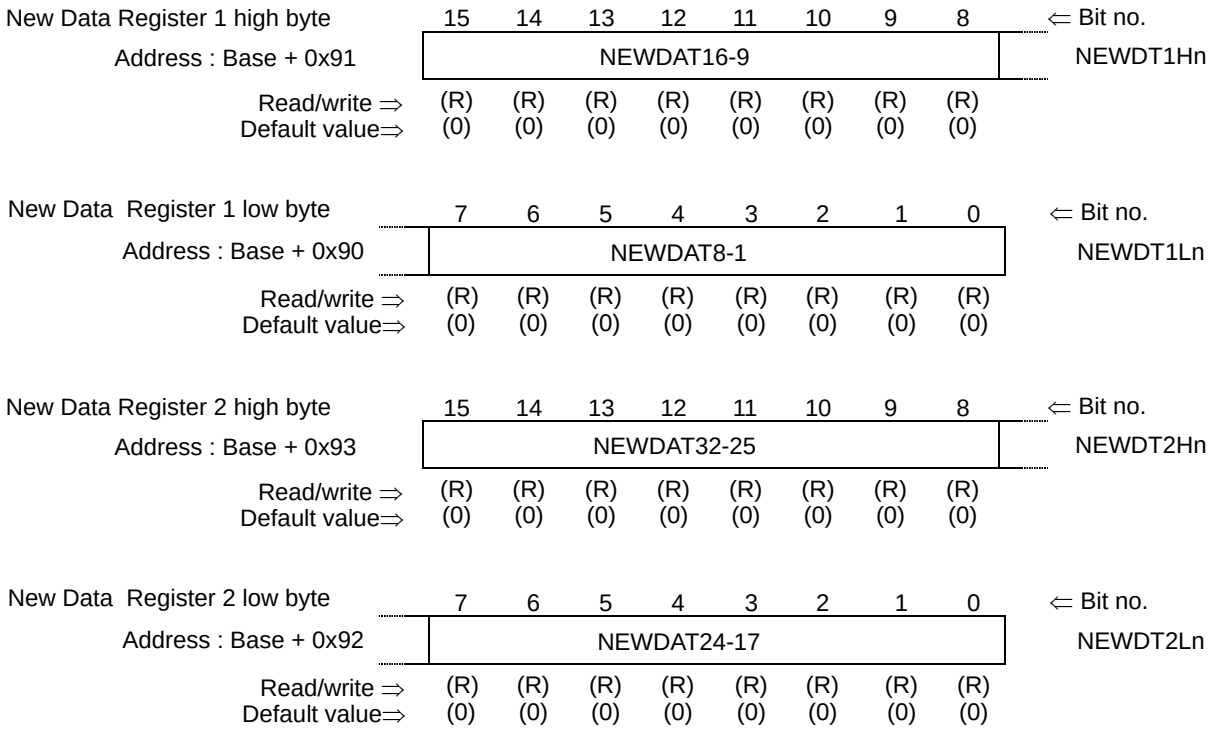
| | | addr+0 | addr+1 | addr+2 | addr+3 |
|---|---|---|---|---|---|
| TREQR 4 & 3 | TXRQST 64-33 (address 0x84) | TXRQST64-57 | TXRQST56-49 | TXRQST48-41 | TXRQST40-33 |
| TREQR 6 & 5 | TXRQST 96-65 (address 0x88) | TXRQST96-89 | TXRQST88-81 | TXRQST80-73 | TXRQST72-65 |
| TREQR 8 & 7 | TXRQST 128-97 (address 0x8C) | TXRQST128-121 | TXRQST120-113 | TXRQST112-105 | TXRQST104-97 |

## 22.8.3 New Data Registers (NEWDT1n, NEWDT2n)

The New Data Registers (NEWDT1n, NEWDT2n) indicate per message buffer that new data has been received.

### New Data Registers (NEWDT1n, NEWDT2n)

Figure 22-19. Configuration of the New Data Registers (NEWDT1n, NEWDT2n)




### Function of the New Data Registers (NEWDT1n, NEWDT2n)

| | |
|---|---|
| NEWDAT32-1 | New Data Bits (of all Message Objects) |
| 0 | No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the CPU. |
| 1 | The Message Handler or the CPU has written new data into the data portion of this Message Object. |

These registers hold the NEWDAT bits of the 32 Message Objects. By reading out the NEWDAT bits, the CPU can check for which Message Object the data portion was updated. The NEWDAT bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

If more than 32 message buffers are implemented, the following table gives an overview about the additional flags:

Table 22-5. Additional flags when more than 32 message buffers exist

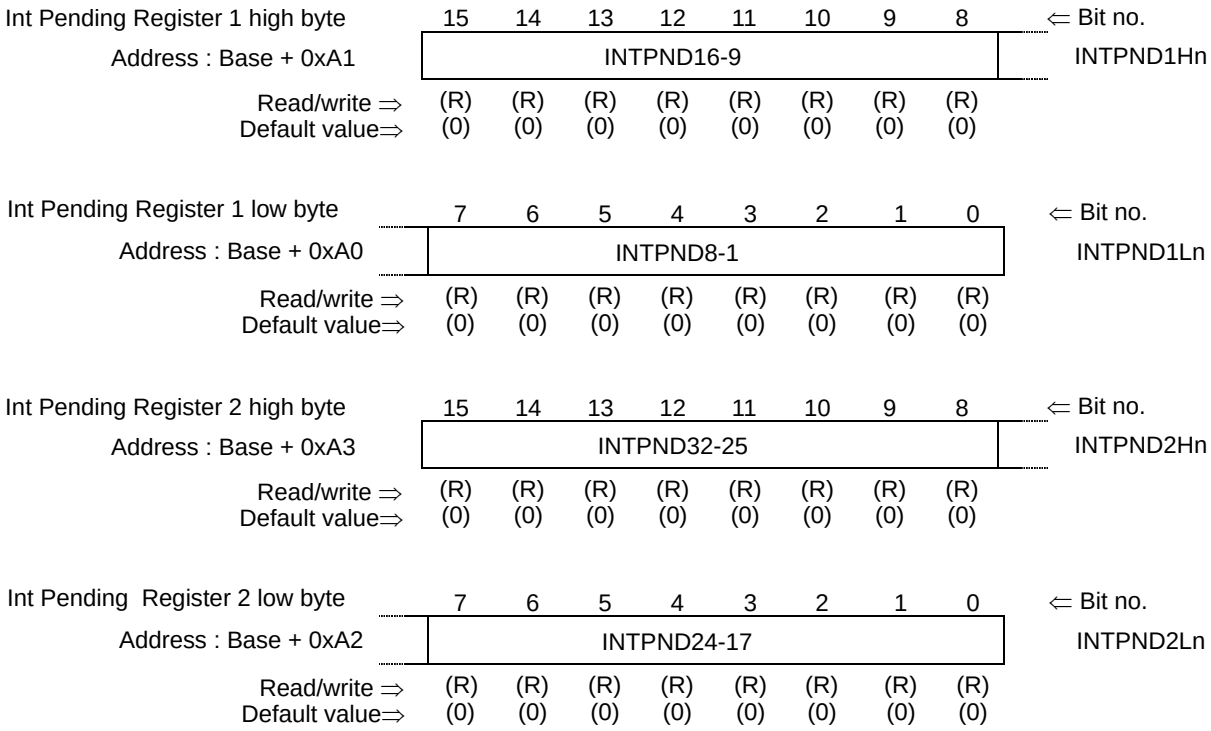
| | | addr+0 | addr+1 | addr+2 | addr+3 |
|---|---|---|---|---|---|
| NEWDT 4 & 3 | NEWDAT 64-33 (address 0x94) | NEWDAT64-57 | NEWDAT56-49 | NEWDAT48-41 | NEWDAT40-33 |
| NEWDT 6 & 5 | NEWDAT 96-65 (address 0x98) | NEWDAT96-89 | NEWDAT88-81 | NEWDAT80-73 | NEWDAT72-65 |
| NEWDT 8 & 7 | NEWDAT 128-97 (address 0x9C) | NEWDAT128-121 | NEWDAT120-113 | NEWDAT112-105 | NEWDAT104-97 |

## 22.8.4 Interrupt Pending Registers (INTPND1n, INTPND2n)

The Interrupt Pending Registers (INTPND1n, INTPND2n) indicate whether a message object caused an interrupt or not.

### Interrupt Pending Registers (INTPND1n, INTPND2n)

Figure 22-20. Configuration of Interrupt Pending Registers (INTPND1n, INTPND2n)




### Function of Interrupt Pending Registers (INTPND1n, INTPND2n)

| | |
|---|---|
| INTPND32-1 | Interrupt Pending Bits (of all Message Objects) |
| 0 | This message object is not the source of an interrupt. |
| 1 | This message object is the source of an interrupt. |

These registers hold the INTPND bits of the 32 Message Objects. By reading out the INTPND bits, the CPU can check for which Message Object an interrupt is pending. The INTPND bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of INTID in the Interrupt Register INTRn.

If more than 32 message buffers are implemented, the following table gives an overview about the additional flags:

Table 22-6. Additional flags when more than 32 message buffers exist

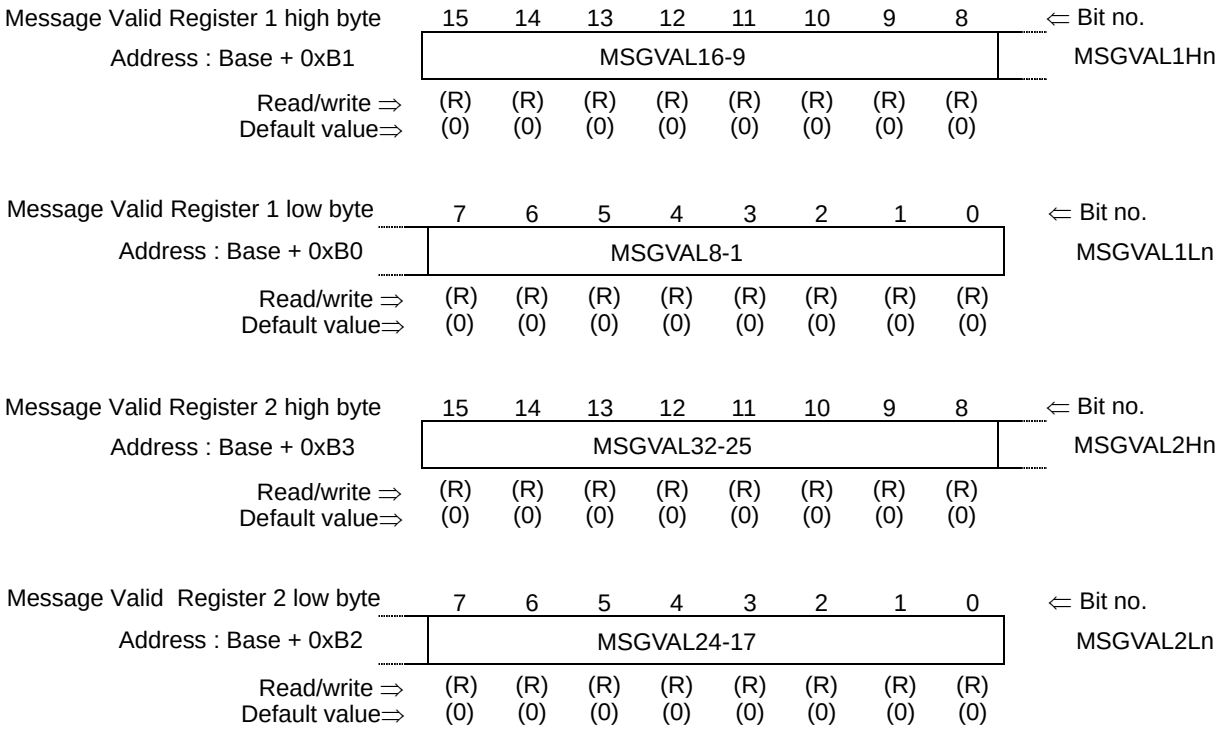
| | | addr+0 | addr+1 | addr+2 | addr+3 |
|---|---|---|---|---|---|
| INTPND 4 & 3 | INTPND 64-33 (address 0xA4) | INTPND64-57 | INTPND56-49 | INTPND48-41 | INTPND40-33 |
| INTPND 6 & 5 | INTPND 96-65 (address 0xA8) | INTPND96-89 | INTPND88-81 | INTPND80-73 | INTPND72-65 |
| INTPND 8 & 7 | INTPND 128-97 (address 0xAC) | INTPND128-121 | INTPND120-113 | INTPND112-105 | INTPND104-97 |

## 22.8.5 Message Valid Registers (MSGVAL1n, MSGVAL2n)

The Message Valid Registers (MSGVAL1n, MSGVAL2n) show the validity status for each message object.

### Message Valid Registers (MSGVAL1n, MSGVAL2n)

Figure 22-21. Configuration of Message Valid Registers (MSGVAL1n, MSGVAL2n)




### Function of the Message Valid Registers (MSGVAL1n, MSGVAL2n)

| | |
|---|---|
| MSGVAL32-1 | Message Valid Bits (of all Message Objects) |
| 1 | This Message Object is configured and should be considered by the Message Handler. |

These registers hold the MSGVAL bits of the 32 Message Objects. By reading out the MSGVAL bits, the CPU can check which Message Object is valid. The MSGVAL bit of a specific Message Object can be set/reset by the CPU via the IFx Message Interface Registers.

If more than 32 message buffers are implemented, the following table gives an overview about the additional flags:

Table 22-7. Additional flags when more than 32 message buffers exist

| | | addr+0 | addr+1 | addr+2 | addr+3 |
|---|---|---|---|---|---|
| MSGVAL 4 & 3 | MSGVAL 64-33 (address 0xB4) | MSGVAL64-57 | MSGVAL56-49 | MSGVAL48-41 | MSGVAL40-33 |
| MSGVAL 6 & 5 | MSGVAL 96-65 (address 0xB8) | MSGVAL96-89 | MSGVAL88-81 | MSGVAL80-73 | MSGVAL72-65 |
| MSGVAL 8 & 7 | MSGVAL 128-97 (address 0xBC) | MSGVAL128-121 | MSGVAL120-113 | MSGVAL112-105 | MSGVAL104-97 |

## 22.9 CAN Application

This section describes how to use the CAN module in the application

### Management of Message Objects

The configuration of the Message Objects in the Message RAM will (with the exception of the bits MSGVAL, NEWDAT, INTPND, and TXRQST) not be affected by resetting the chip. All the Message Objects must be initialized by the CPU or they must be not valid (MSGVAL = ‘0’) and the bit timing must be configured before the CPU clears the INIT bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFx Command Request Register, the IFx Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the INIT bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN_Core and the Message Handler State Machine control the CAN’s internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN_Core’s Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFx Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

### Message Handler State Machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IFx Registers.

The Message Handler FSM controls the following functions:

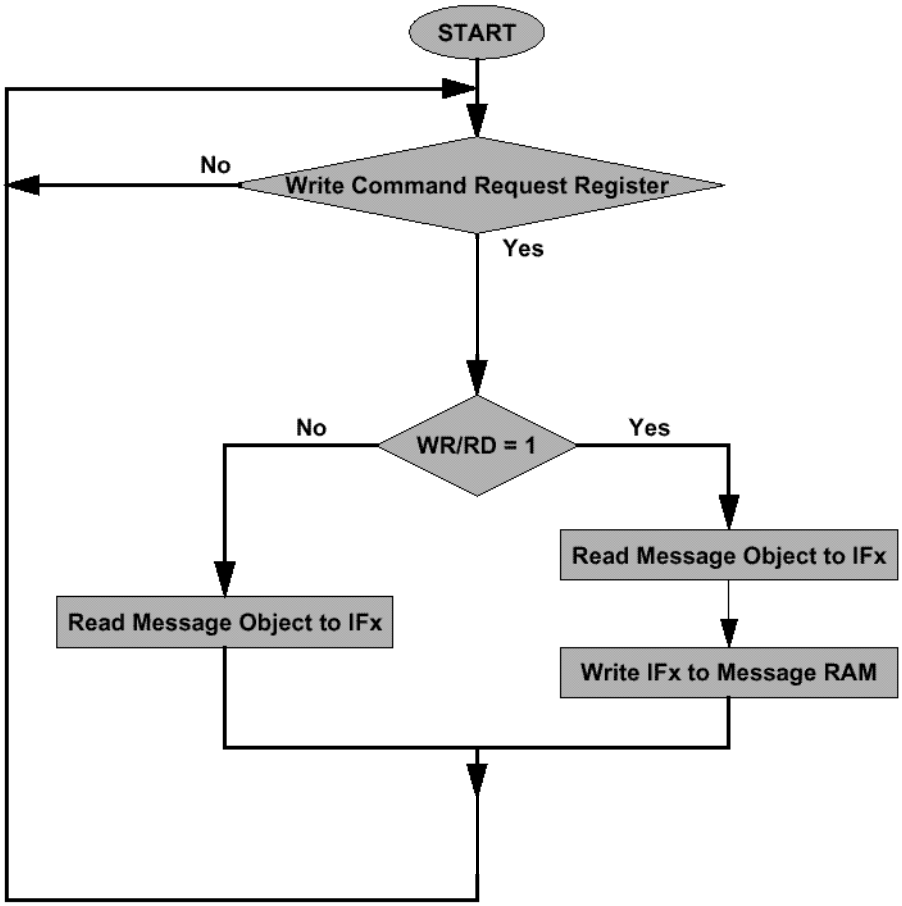
- Data Transfer from IFx Registers to the Message RAM
- Data Transfer from Message RAM to the IFx Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of TXRQST flags.
- Handling of interrupts.

### Data Transfer from/to Message RAM

When the CPU initiates a data transfer between the IFx Registers and Message RAM, the Message Handler sets an internal busy signal which delays a consecutive access. After the transfer has completed, the busy signal is set back and the consecutive access is executed.

The respective Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM it is not possible to write single bits/bytes of one Message Object, it is always necessary to write a complete Message Object into the Message RAM. Therefore the data transfer from the IFx Registers to the Message RAM requires a read-modify-write cycle. First the parts of the Message Object that are not to be changed are read from the Message RAM and then the complete contents of the Message Buffer Registers are transferred to the Message Object.

Figure 22-22. Data Transfer between IFx Registers and Message RAM




After the partial write of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be set to the actual contents of the selected Message Object.

After the partial read of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be left unchanged.

### Transmission of Messages

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFx Registers and Message RAM, the MSGVAL bits in the Message Valid Register and the TXRQST bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object's NEWDAT bit is reset.

After a successful transmission and if no new data was written to the Message Object (NEWDAT = '0') since the start of the transmission, the TXRQST bit will be reset. If TXIE is set, INTPND will be set after a successful transmission. If the CAN has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

### Acceptance Filtering of Received Messages

When the arbitration and control field (Identifier + IDE + RTR + DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including MSGVAL, UMASK, NEWDAT, and EOB) of Message Object 1 are loaded into the Acceptance Filtering unit and compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler FSM proceeds depending on the type of frame (Data Frame or Remote Frame) received.

### Reception of Data Frame

The Message Handler FSM stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but all arbitration bits and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NEWDAT bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU should reset NEWDAT when it reads the Message Object. If at the time of the reception the NEWDAT bit was already set, MSGLST is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RXIE bit is set, the INTPND bit is set, causing the Interrupt Register to point to this Message Object.

The TXRQST bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

### Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1) DIR = ‘1’ (direction = transmit), RMTEN = ‘1’, UMASK = ‘1’ or ’0’

At the reception of a matching Remote Frame, the TXRQST bit of this Message Object is set. The rest of the Message Object remains unchanged.

2) DIR = ‘1’ (direction = transmit), RMTEN = ‘0’, UMASK = ’0’

At the reception of a matching Remote Frame, the TXRQST bit of this Message Object remains unchanged; the Remote Frame is ignored.

3) DIR = ‘1’ (direction = transmit), RMTEN = ‘0’, UMASK = ’1’

At the reception of a matching Remote Frame, the TXRQST bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM and the NEWDAT bit of this Message Object is set. The data field of the Message Object remains unchanged; the Remote Frame is treated similar to a received Data Frame.

### Receive / Transmit Priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 (the highest implemented message object number) has the lowest priority. If more than one transmission request is pending, they are serviced due to the priority of the corresponding Message Object.

### Configuration of a Transmit Object

Figure 22-23 shows how a Transmit Object should be initialised.

Figure 22-23. Initialisation of a Transmit Object

| MSGVAL | ARB | DATA | MASK | EOB | DIR | NEWDAT | MSGLST | RXIE | TXIE | INTPND | RMTEN | TXRQST |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | appl. | appl. | appl. | 1 | 1 | 0 | 0 | 0 | appl. | 0 | appl. | 0 |

The Arbitration Registers (ID28-0 and XTD bit) are given by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18, ID17 - ID0 can then be disregarded.

If the TXIE bit is set, the INTPND bit will be set after a successful transmission of the Message Object.

If the RMTEN bit is set, a matching received Remote Frame will cause the TXRQST bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (DLC3-0, Data0-7) are given by the application, TXRQST and RMTEN may not be set before the data is valid.

The Mask Registers (MSK28-0, UMASK, MXTD, and MDIR bits) may be used (UMASK = "1") to allow groups of Remote Frames with similar identifiers to set the TXRQST bit. For details see section Transmission of Messages on page 490. Handle with care. The DIR bit should not be masked.

### Updating a Transmit Object

The CPU may update the data bytes of a Transmit Object any time via the IFx Interface registers, neither MSGVAL nor TXRQST have to be reset before the update.

### Note:

When the lowest priority message buffer is used for transmission, setting TXRQST = "0" may cause a delay of transmission, when TXRQST is set to "1" again. Please refer to the note at the description of TXRQST in section 22.8.2 Transmission Request Registers (TREQR1n, TREQR2n).

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFx Data A Register or IFx Data B Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFx Data Register or the Message Object is transferred to the IFx Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087 is written to the Command Mask Register and then the number of the Message Object is written to the Command Request Register, concurrently updating the data bytes and setting TXRQST.

To prevent the reset of TXRQST at the end of a transmission that may already be in progress while the data is updated, NEWDAT has to be set together with TXRQST. For details see section Transmission of Messages on page 490.

When NEWDAT is set together with TXRQST, NEWDAT will be reset as soon as the new transmission has started.

### Configuration of a Receive Object

Figure 22-24 shows how a Receive Object should be initialised.

Figure 22-24. Initialisation of a Receive Object

| MSGVAL | ARB | DATA | MASK | EOB | DIR | NEWDAT | MSGLST | RXIE | TXIE | INTPND | RMTEN | TXRQST |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | appl. | appl. | appl. | 1 | 0 | 0 | 0 | appl. | 0 | 0 | 0 | 0 |

The Arbitration Registers (ID28-0 and XTD bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18, ID17 - ID0 can then be disregarded. When a Data Frame with an 11-bit Identifier is received, ID17 - ID0 will be set to ‘0’.

If the RXIE bit is set, the INTPND bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLC3-0) is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The Mask Registers (MSK28-0, UMASK, MXTD, and MDIR bits) may be used (UMASK = "1") to allow groups of Data Frames with similar identifiers to be accepted. For details see section Reception of Data Frame on page 491. The DIR bit should not be masked in typical applications.

### Handling of Received Messages

The CPU may read a received message any time via the IFx Interface registers, the data consistency is guaranteed by the Message Handler state machine.

Typically the CPU will write first 0x007F to the Command Mask Register and then the number of the Message Object to the Command Request Register. That combination will transfer the whole received message from the Message RAM into the Message Buffer Register. Additionally, the bits NEWDAT and INTPND are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of NEWDAT shows whether a new message has been received since last time this Message Object was read. The actual value of MSGLST shows whether more than one message has been received since last time this Message Object was read. MSGLST will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TXRQST bit of a receive object will cause the transmission of a Remote Frame with the receive object's identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TXRQST bit is automatically reset.

### Configuration of a FIFO Buffer

With the exception of the EOB bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object, see section Configuration of a Receive Object on page 492.

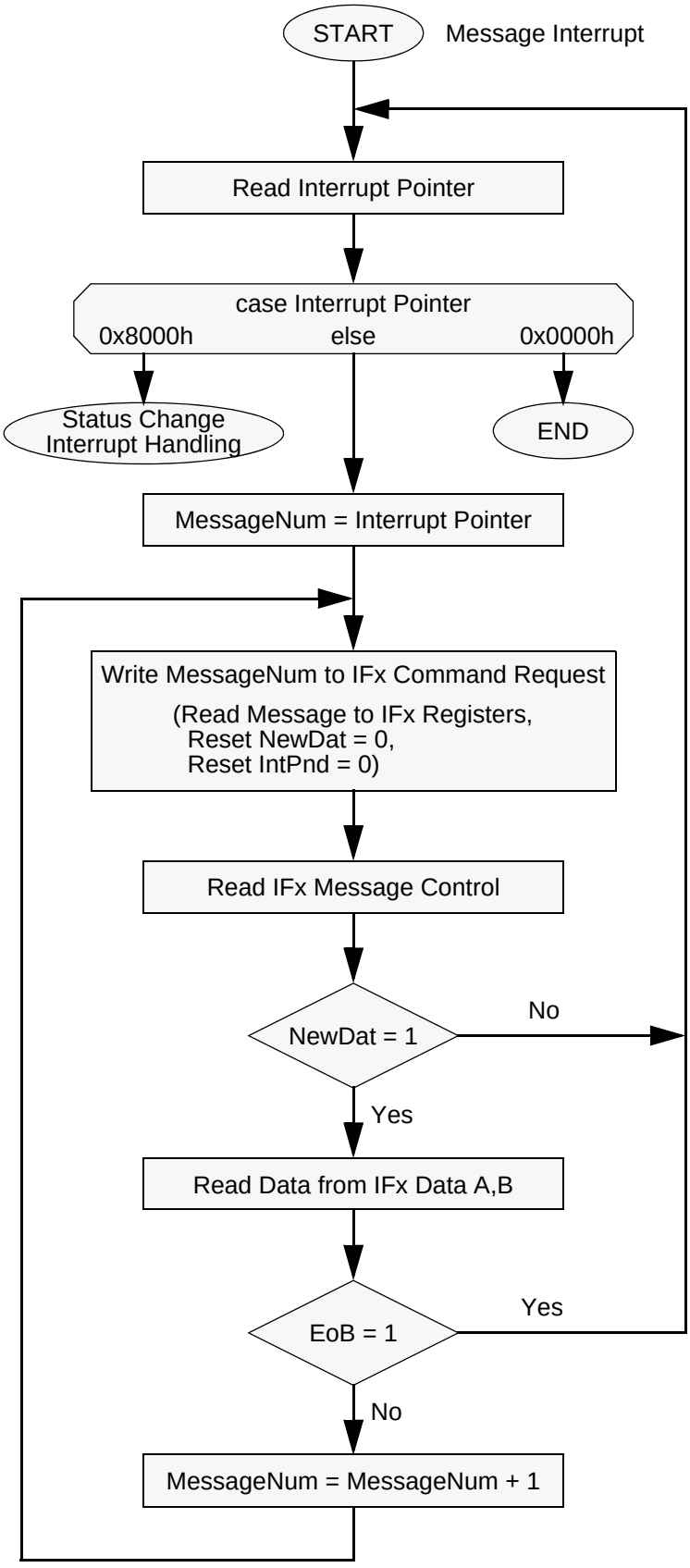
To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The EOB bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EOB bit of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

### Reading from a FIFO Buffer

When the CPU transfers the contents of Message Object to the IFx Message Buffer registers by writing its number to the IFx Command Request Register, the corresponding Command Mask Register should be programmed the way that bits NEWDAT and INTPND are reset to zero (TXRQST = '1' and CIP = '1'). The values of these bits in the Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number. Figure 22-25 shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.

Figure 22-25. CPU Handling of a FIFO Buffer


START
Message Interrupt
Read Interrupt Pointer
case Interrupt Pointer
0x8000h
else
0x0000h
Status Change Interrupt Handling
END
MessageNum = Interrupt Pointer
Write MessageNum to IFx Command Request
(Read Message to IFx Registers,
Reset NewDat = 0,
Reset IntPnd = 0)
Read IFx Message Control
NewDat = 1
No
Yes
Read Data from IFx Data A,B
EoB = 1
Yes
No
MessageNum = MessageNum + 1

### Handling of Interrupts

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the CPU has cleared it.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's INTPND bit. The Status Interrupt is cleared by reading the Status Register.

The interrupt identifier INTID in the Interrupt Register INTRn indicates the cause of the interrupt. When no interrupt is pending, the register will hold the value zero. If the value of the Interrupt Register is different from zero, then there is an interrupt pending and, if IE is set, the interrupt line to the CPU is active. The interrupt line remains active until the Interrupt Register is back to value zero (the cause of the interrupt is reset) or until IE is reset.

The value 0x8000 indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the status bits RXOK, TXOK and LEC, but a write access of the CPU to the Status Register can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects, INTID points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Register may cause an interrupt (bits EIE and SIE in the CAN Control Register) and whether the interrupt line becomes active when the Interrupt Register is different from zero (bit IE in the CAN Control Register). The Interrupt Register will be updated even when IE is reset.

The CPU has two possibilities to follow the source of a message interrupt. First it can follow the INTID in the Interrupt Register and second it can poll the Interrupt Pending Register (see section 22.8 Message Handler Registers).

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object's INTPND at the same time (bit CIP in the Command Mask Register IFxCMSKn). When INTPND is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

### Bit Time and Bit Rate

CAN supports bit rates in the range of lower than 1kBit/s up to 1000kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (fosc) may be different.

The frequencies of these oscillators are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specific oscillator tolerance range (df), the CAN nodes are able to compensate for the different bit rates by resynchronising to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see Figure 22-26). The Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1, and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see Table 1). The length of the time quantum (tq) , which is the basic time unit of the bit time, is defined by the CAN controller's system clock fsys and the Baud Rate Prescaler (BRP) : tq = BRP / fsys. The CAN's system clock fsys is the frequency of its CAN_CLK input.

The Synchronisation Segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-)Synchronisation Jump Width (SJW) defines how far a resynchronisation may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Figure 22-26. Bit Timing

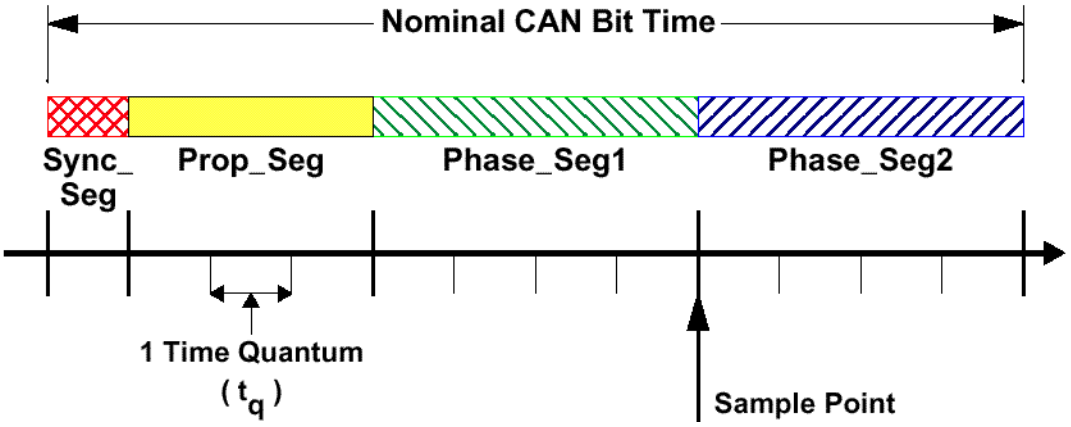



Table 22-8. Parameters of the CAN Bit Time

| Parameter | Range | Remark |
|---|---|---|
| BRP | [1..32] | defines the legth of the time quantum tq |
| Sync_Seg | 1 tq | fixed length, synchronisation of us into system clock |
| Prop_Seg | [1..8] tq | compensates for the physical delay times |
| Phase_Seg1 | [1..8] tq | may be lengthened temporarily by synchronisation |
| Phase_Seg2 | [1..8] tq | may be shortened temporarily by synchronisation |
| SJW | [1..4] tq | may not be longer than either Phase Buffer Segment |

### Note

This table describes the minimum programmable ranges required by the CAN protocol.

As described earlier in this chapter for BTRn register fields, programming is as follow:

Prop_Seg + Phase_Seg1 = TSEG1 value +1

Phase_Seg2 = TSEG2 value + 1

# 23. Clock Output Function

This chapter describes the functions and operations of the clock output function.

23.1 Overview of the Clock Output Function

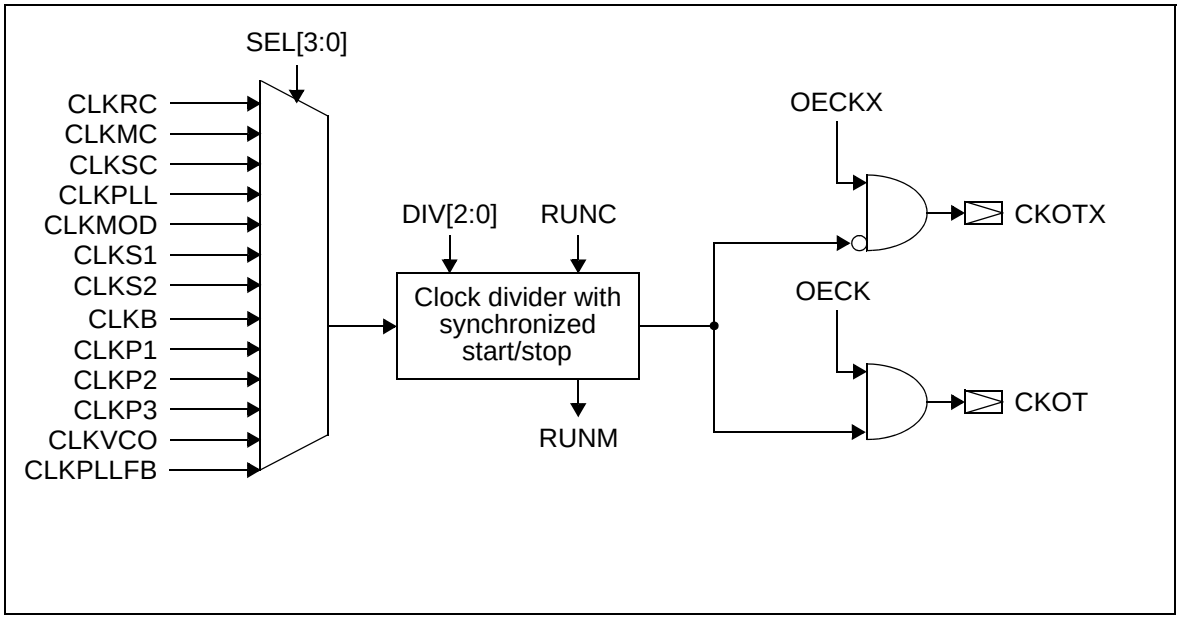
23.2 Clock Output Configuration and Activation Registers

23.3 Operation of the Clock Output Function

## 23.1 Overview of the Clock Output Function

The clock output function can be used to output up to two internal clocks to the four clock output pins CKOT0/CKOTX0 and CKOT1/CKOTX1 (one non-inverted and one inverted output pin for each clock). The clocks can be output directly or divided with the internal clock divider.

**Block Diagram of the Clock Output Function (one channel)**




For the definition of the clocks that can be selected, see Clocks on page 119.

CLKP3 is not available for all devices. Please refer to the datasheet of your device.

## 23.2 Clock Output Configuration and Activation Registers

The Clock Output Configuration Register (COCR) selects the output clock and controls the clock divider. The COCR0 register controls the CKOT0/CKOTX0 output clock and the COCR1 register controls the CKOT1/CKOTX1 output clock.

The Clock Output Activation Register (COAR) enables the CKOT0, CKOTX0, CKOT1 and CKOTX1 output pins and synchronously starts and stops the selected clocks.

## Clock Output Configuration Register (COCR)

Figure 23-1. Clock Output Configuration Register (COCR)

Address:
COCR1: $000417_H$
COCR0: $000416_H$

| 15/7 | 14/6 | 13/5 | 12/4 | 11/3 | 10/2 | 9/1 | 8/0 |
|---|---|---|---|---|---|---|---|
| - | DIV2 | DIV1 | DIV0 | SEL3 | SEL2 | SEL1 | SEL0 |
| - | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value
$X0000000_B$

bit11/3 bit10/2 bit9/1 bit8/0

| SEL3 | SEL2 | SEL1 | SEL0 | Clock Select bits |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | No clock |
| 0 | 0 | 0 | 1 | CLKRC (RC clock) |
| 0 | 0 | 1 | 0 | CLKMC (Main clock) |
| 0 | 0 | 1 | 1 | CLKSC (Sub clock) |
| 0 | 1 | 0 | 0 | CLKPLL (PLL output clock) |
| 0 | 1 | 0 | 1 | CLKMOD (Modulated PLL clock) |
| 0 | 1 | 1 | 0 | CLKS1 (System clock 1) |
| 0 | 1 | 1 | 1 | CLKS2 (System clock 2) |
| 1 | 0 | 0 | 0 | CLKB (Bus clock) |
| 1 | 0 | 0 | 1 | CLKP1 (Peripheral clock 1) |
| 1 | 0 | 1 | 0 | CLKP2 (Peripheral clock 2) |
| 1 | 0 | 1 | 1 | CLKVCO (VCO clock of PLL) |
| 1 | 1 | 0 | 0 | CLKPLLFB (PLL feedback clock) |
| 1 | 1 | 0 | 1 | CLKP3 (Peripheral clock 3) * |
| 1 | 1 | 1 | 0 | reserved |
| 1 | 1 | 1 | 1 | reserved |

bit14/6 bit13/5 bit12/4

| DIV2 | DIV1 | DIV0 | Clock Division bits |
|---|---|---|---|
| 0 | 0 | 0 | Output clock is not divided |
| 0 | 0 | 1 | Output clock is divided by 2 |
| 0 | 1 | 0 | Output clock is divided by 4 |
| 0 | 1 | 1 | Output clock is divided by 8 |
| 1 | 0 | 0 | Output clock is divided by 16 |
| 1 | 0 | 1 | Output clock is divided by 32 |
| 1 | 1 | 0 | Output clock is divided by 64 |
| 1 | 1 | 1 | Output clock is divided by 128 |

bit15/7

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

R/W : Readable and writable
: Initial value

*CLKP3 is not available for all devices. Please refer to the datasheet of your device.

Table 23-1. Function Description of Each Bit of the Clock Output Configuration Register (COCR0/1)

<table>
<tr><th colspan="2">Bit name</th><th>Function</th></tr>
<tr><td>bit 0 - bit 3 / bit 8 - bit 11</td><td>SEL0 to SEL3:<br>Clock Select bits</td><td>
• These bits select which clock is output to the CKOT0/1 pins according to the following table:
<table>
<tr><th>bit11/3</th><th>bit10/2</th><th>bit9/1</th><th>bit8/0</th><th></th></tr>
<tr><th>SEL3</th><th>SEL2</th><th>SEL1</th><th>SEL0</th><th>Clock Select bits</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>No clock</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>CLKRC (RC clock)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>CLKMC (Main clock)</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>CLKSC (Sub clock)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>CLKPLL (PLL output clock)</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>CLKMOD (Modulated PLL clock)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>CLKS1 (System clock 1)</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>CLKS2 (System clock 2)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>CLKB (Bus clock)</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>CLKP1 (Peripheral clock 1)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>CLKP2 (Peripheral clock 2)</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>CLKVCO (VCO clock of PLL)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>CLKPLLFB (PLL feedback clock)</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>CLKP3 (Peripheral clock 3) *</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td><td>reserved</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>reserved</td></tr>
</table>
• Any reset initializes these bits to "0000" (no clock is selected).<br>
• If the setting of these bits is changed while the clock output function is active (corresponding RUNC and OE bits set to "1"), a spike can be output.<br>
* CLKP3 is not available for all devices. Please refer to the datasheet of your device.
</td></tr>
<tr><td>bit 4 - bit 6 / bit 12 - bit 14</td><td>DIV0 to DIV2:<br>Clock Division bits</td><td>
• These bits configure the clock output divider according to the following table:
<table>
<tr><th>bit14/6</th><th>bit13/5</th><th>bit12/4</th><th></th></tr>
<tr><th>DIV2</th><th>DIV1</th><th>DIV0</th><th>Clock Division bits</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Output clock is not divided</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Output clock is divided by 2</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Output clock is divided by 4</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Output clock is divided by 8</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Output clock is divided by 16</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Output clock is divided by 32</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Output clock is divided by 64</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Output clock is divided by 128</td></tr>
</table>
• Any reset initializes these bits to "000" (Output clock not divided).<br>
• If the setting of these bits is changed while the clock output function is active (corresponding RUNC and OE bits set to "1"), a spike can be output.
</td></tr>
<tr><td>bit 7 / bit 15</td><td>Reserved</td><td>
• Always write "0" to this bit.<br>
• The read value of this bit is undefined.<br>
• Read modify write operations to this register are not affected.
</td></tr>
</table>

## Clock Output Activation Register (COAR)

Figure 23-2. Clock Output Activation Register (COAR)

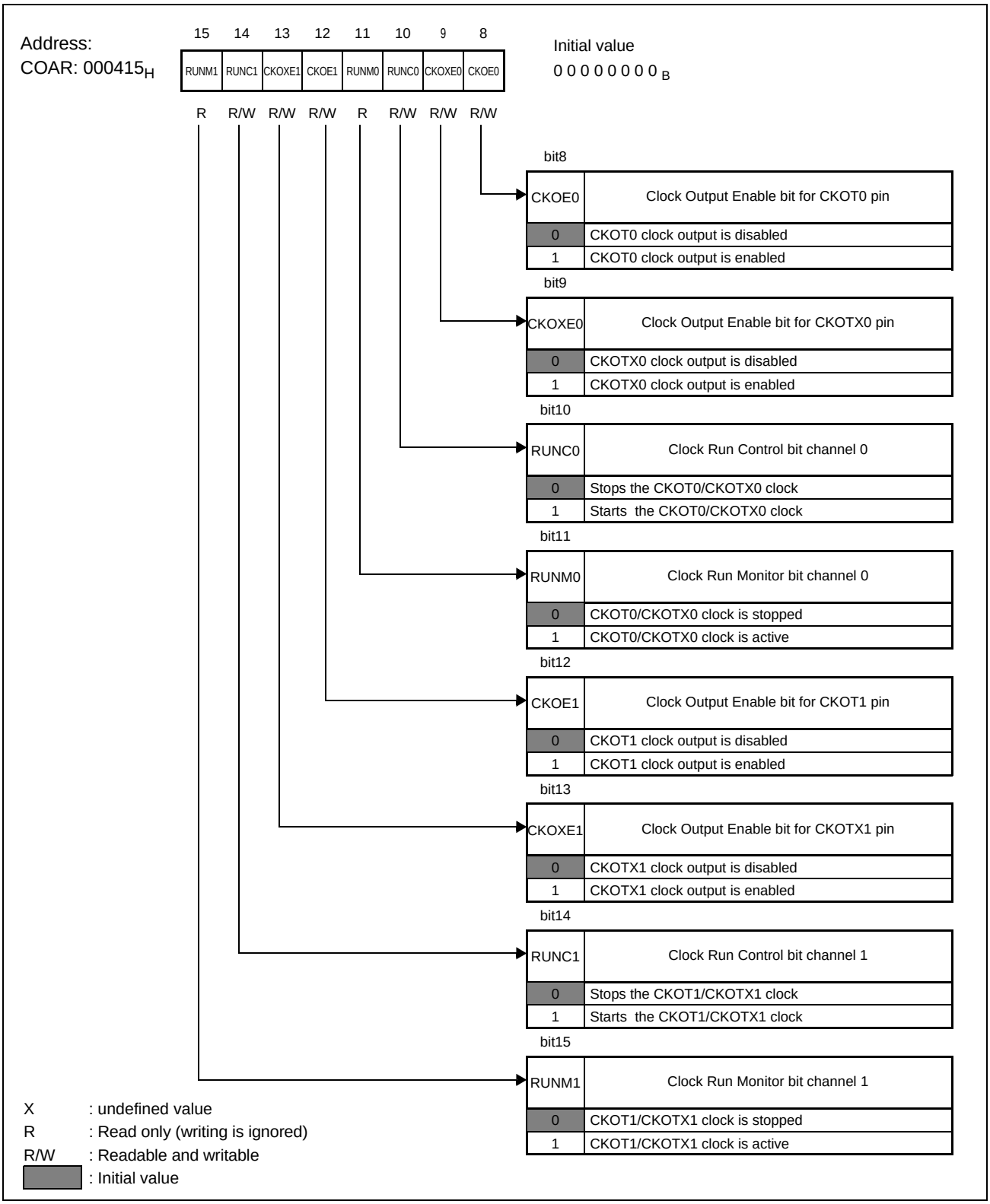


Table 23-2. Function Description of Each Bit of the Clock Output Activation Register (COAR) (Sheet 1 of 2)

| Bit name | | Function |
|---|---|---|
| bit 8 | CKOE0<br>Clock Output Enable bit for CKOT0 pin | • This bit controls the resource output enable of the direct clock output function.<br>• Writing "1" to this bit enables the CKOT0 output pin. Do not enable any other resource output shared with the CKOT0 pin when setting this bit to "1".<br>• Setting this bit to "0" disables the clock output (initial value).<br>• A read access returns the bit setting.<br>• If the setting of this bit is changed while the RUNM0 bit is "1", a spike can be output. |
| bit 9 | CKOXE0<br>Clock Output Enable bit for CKOTX0 pin | • This bit controls the resource output enable of the inverted clock output function.<br>• Writing "1" to this bit enables the CKOTX0 output pin. Do not enable any other resource output shared with the CKOTX0 pin when setting this bit to "1".<br>• Setting this bit to "0" disables the clock output (initial value).<br>• A read access returns the bit setting.<br>• If the setting of this bit is changed while the RUNM0 bit is "1", a spike can be output. |
| bit 10 | RUNC0<br>Clock Run Control bit channel 0 | • This bit controls the synchronous start/stop function of the CKOT0/CKOTX0 clock outputs.<br>• Setting this bit to "1" starts the clock outputs and writing "0" stops the clock outputs.<br>• This bit is initialized to "0" by any reset (clock outputs stopped).<br>• The RUNC0 and the CKOE0 and/or CKOXE0 bits must be set to "1" in order to output the selected clock at the CKOT0 and/or CKOTX0 pins.<br>• The RUNC0 bit can be written at any time. However the starting and stopping takes some clock cycles and is performed only when the selected clock is active.<br>• A read access returns the setting of this control bit. Use the RUNM0 monitor bit to check if the synchronized starting/stopping has finished. |
| bit 11 | RUNM0<br>Clock Run Monitor bit channel 0 | • Reading this bit returns the status of the start/stop synchronization circuit and indicates if starting/stopping the clock output has finished.<br>• Reading "1" means that the CKOT0/CKOTX0 clock output is started.<br>• Reading "0" means that the CKOT0/CKOTX0 clock output is stopped (the CKOT0 pin outputs a low level and the CKOTX0 pin outputs a high level).<br>• Writing to this bit has no effect. |
| bit 12 | CKOE1<br>Clock Output Enable bit for CKOT1 pin | • This bit controls the resource output enable of the direct clock output function.<br>• Writing "1" to this bit enables the CKOT1 output pin. Do not enable any other resource output shared with the CKOT1 pin when setting this bit to "1".<br>• Setting this bit to "0" disables the clock output (initial value).<br>• A read access returns the bit setting.<br>• If the setting of this bit is changed while the RUNM1 bit is "1", a spike can be output. |
| bit 13 | CKOXE1<br>Clock Output Enable bit for CKOTX1 pin | • This bit controls the resource output enable of the inverted clock output function.<br>• Writing "1" to this bit enables the CKOTX1 output pin. Do not enable any other resource output shared with the CKOTX1 pin when setting this bit to "1".<br>• Setting this bit to "0" disables the clock output (initial value).<br>• A read access returns the bit setting.<br>• If the setting of this bit is changed while the RUNM1 bit is "1", a spike can be output. |

Table 23-2. Function Description of Each Bit of the Clock Output Activation Register (COAR) (Sheet 2 of 2)

| Bit name | | Function |
|---|---|---|
| bit 14 | RUNC1<br>Clock Run Control bit channel 1 | • This bit controls the synchronous start/stop function of the CKOT1/CKOTX1 clock outputs.<br>• Setting this bit to "1" starts the clock outputs and writing "0" stops the clock outputs.<br>• This bit is initialized to "0" by any reset (clock outputs stopped).<br>• The RUNC1 and the CKOE1 and/or CKOXE1 bits must be set to "1" in order to output the selected clock at the CKOT1 and/or CKOTX1 pins.<br>• The RUNC1 bit can be written at any time. However the starting and stopping takes some clock cycles and is performed only when the selected clock is active.<br>• A read access returns the setting of this control bit. Use the RUNM1 monitor bit to check if the synchronized starting/stopping has finished. |
| bit 15 | RUNM1<br>Clock Run Monitor bit channel 1 | • Reading this bit returns the status of the start/stop synchronization circuit and indicates if starting/stopping the clock has finished.<br>• Reading "1" means that the CKOT1/CKOTX1 clock output is started.<br>• Reading "0" means that the CKOT1/CKOTX1 clock output is stopped (the CKOT1 pin outputs a low level and the CKOTX1 pin outputs a high level).<br>• Writing to this bit has no effect. |

## 23.3 Operation of the Clock Output Function

The clock output function can be used to output two independent clocks at the CKOT0/CKOTX0 and CKOT1/CKOTX1 output pins. The clocks can be activated and deactivated synchronously.

### Activating the clock output function

Activate the clock output function for the CKOT0/CKOTX0 and CKOT1/CKOTX1 pins as follows:

- Configure the corresponding COCR register by selecting the requested clock and division value.
- Enable the CKOT and/or CKOTX output pin by setting the corresponding output enable bit to "1". The start bit RUNC should be set to "1" at the same time or after setting the output enable.

The CKOT pins output "0" and the CKOTX pins output "1" directly after activating the output enable.

After setting the RUNC start bit to "1", the synchronization circuit is activated and starts outputting the selected clock at the corresponding CKOT/CKOTX pins. Reading the RUNM bit now as "0" indicates that the output clock is not yet started (synchronization mechanism is still in effect) and reading "1" means that the output clock is started.

### Disabling the clock output function

Disable the clock output function as follows:

- Stop the clock output by setting the RUNC bit to "0". This stopping is done by using the synchronization circuit. Confirm that the clock is actually stopped by reading the RUNM bit (set to "0").
- When the clock is stopped, it is allowed to change the configuration of the COCR register or to disable the resource output by setting the output enable bit to "0".

The output value at the CKOT/CKOTX pins after stopping the clock is again "0" for the CKOT pins and "1" for the CKOTX pins.

The synchronization circuit for starting/stopping the output clock needs an active source clock. If the source clock selected by the SEL[3:0] bits is disabled, then the status of the output clock cannot be changed. Setting the RUNC bit to another value is possible, however the RUNM monitor bit will not change its value.

### Changing the Clock Output Configuration Register contents

The COCR register should only be written if the corresponding clock output is stopped (RUNM is "0"). Otherwise a spike can be output.

# 24. Real Time Clock

This chapter explains the functions and operations of the Real Time Clock.

24.1 Outline of Real Time Clock

24.2 Real Time Clock Registers

24.3 Operation

24.4 Cautions

## 24.1 Outline of Real Time Clock

The Real Time Clock consists of the Timer Control register, Sub-second register, Second/Minute/Hour registers, 1/2 clock divider, 21-bit prescaler and Second/Minute/Hour counters.

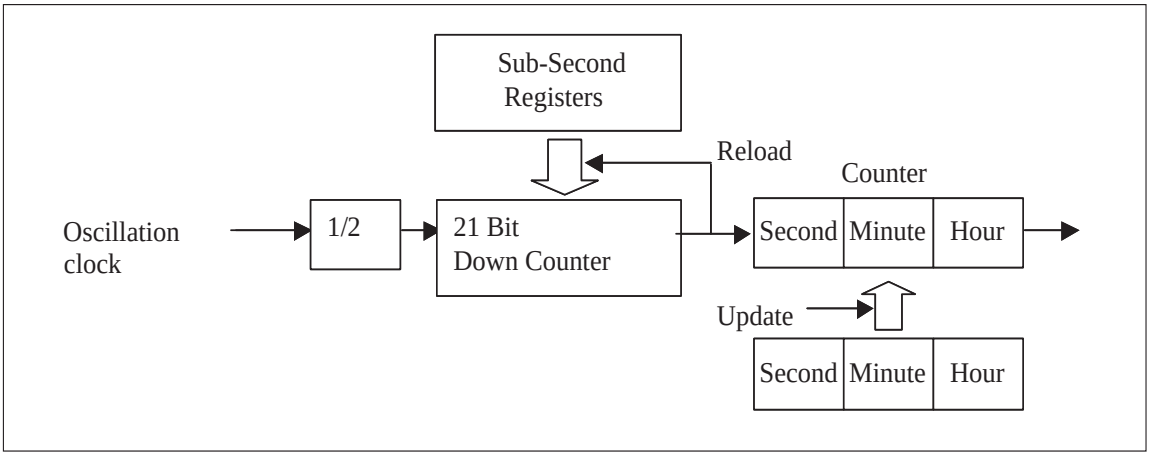
Real Time Clock (RTC) continues to count elapsed time even in the timer mode to provide the current real time (HH/MM/SS) based on main oscillation (4MHz), sub oscillation (32kHz) or RC oscillation (~100kHz/~2MHz). This allows precise time counting without a return from an interrupt during stand by periods.

### Features of Real Time Clock

- Information: Time count (HH/MM/SS). (This clock continues to operate even in Timer Mode.)
- Operational on main oscillation (4MHz), sub oscillation (32kHz) or RC oscillation (~100kHz or ~2MHz)
- Time unit: Selected clock divided by 2
- Operation clock:
  - For register access: CLKP1
  - For time count: Main-clock (CLKMC), Sub-clock (CLKSC) or RC clock (CLKRC)
- Time: Initial setting and adjustment are possible.
- Interrupt: Interrupts can be generated at any of the five intervals: 1 half-second, 1 second, 1 minute, 1 hour and 1 day.
- Others: By changing the value of the sub-second register, interrupts can be generated at any interval (from short to long).

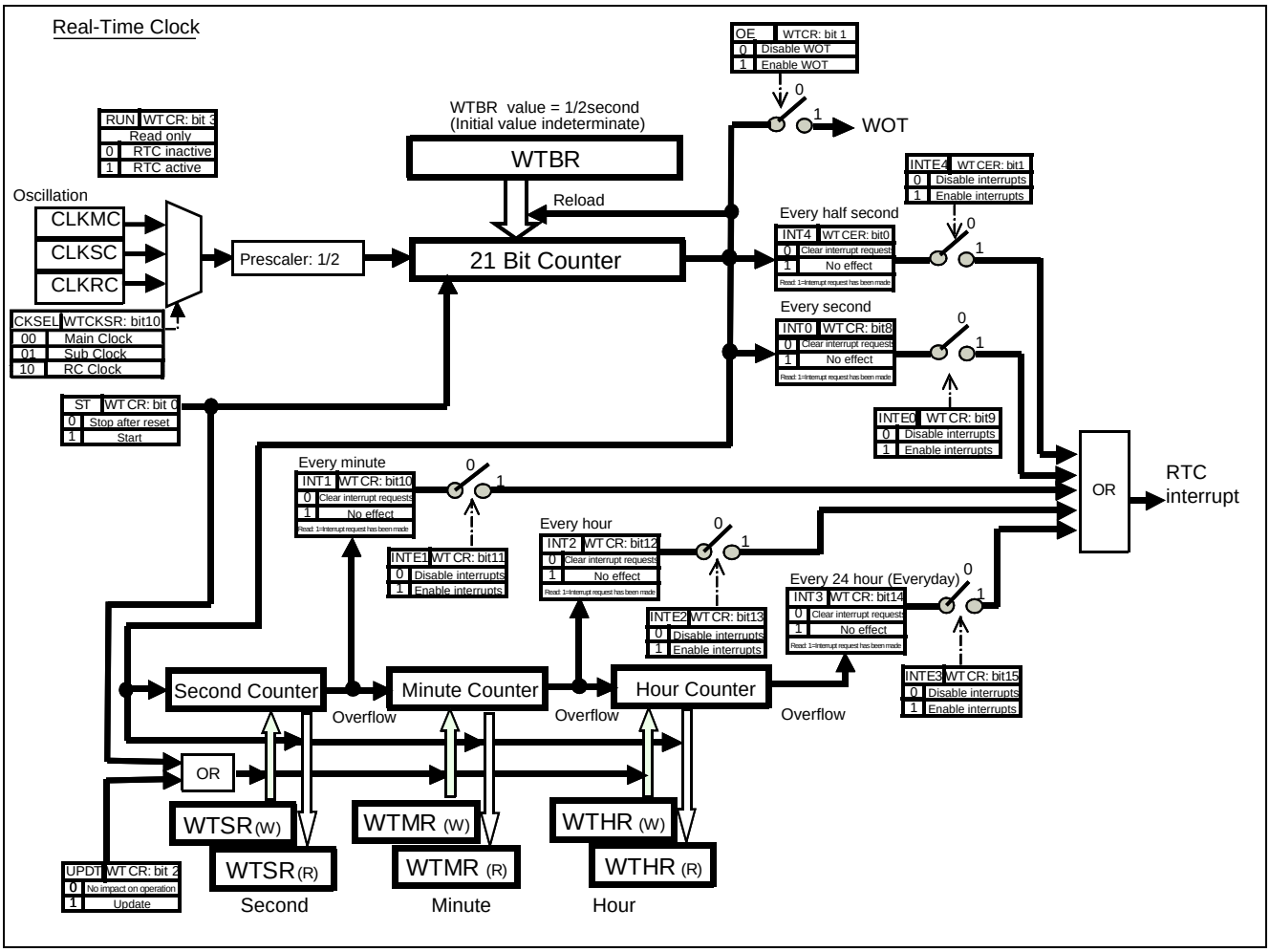
## Block diagram of Real Time Clock

Figure 24-1. Block diagram of Real Time Clock




## Configuration diagram

Figure 24-2. Configuration diagram



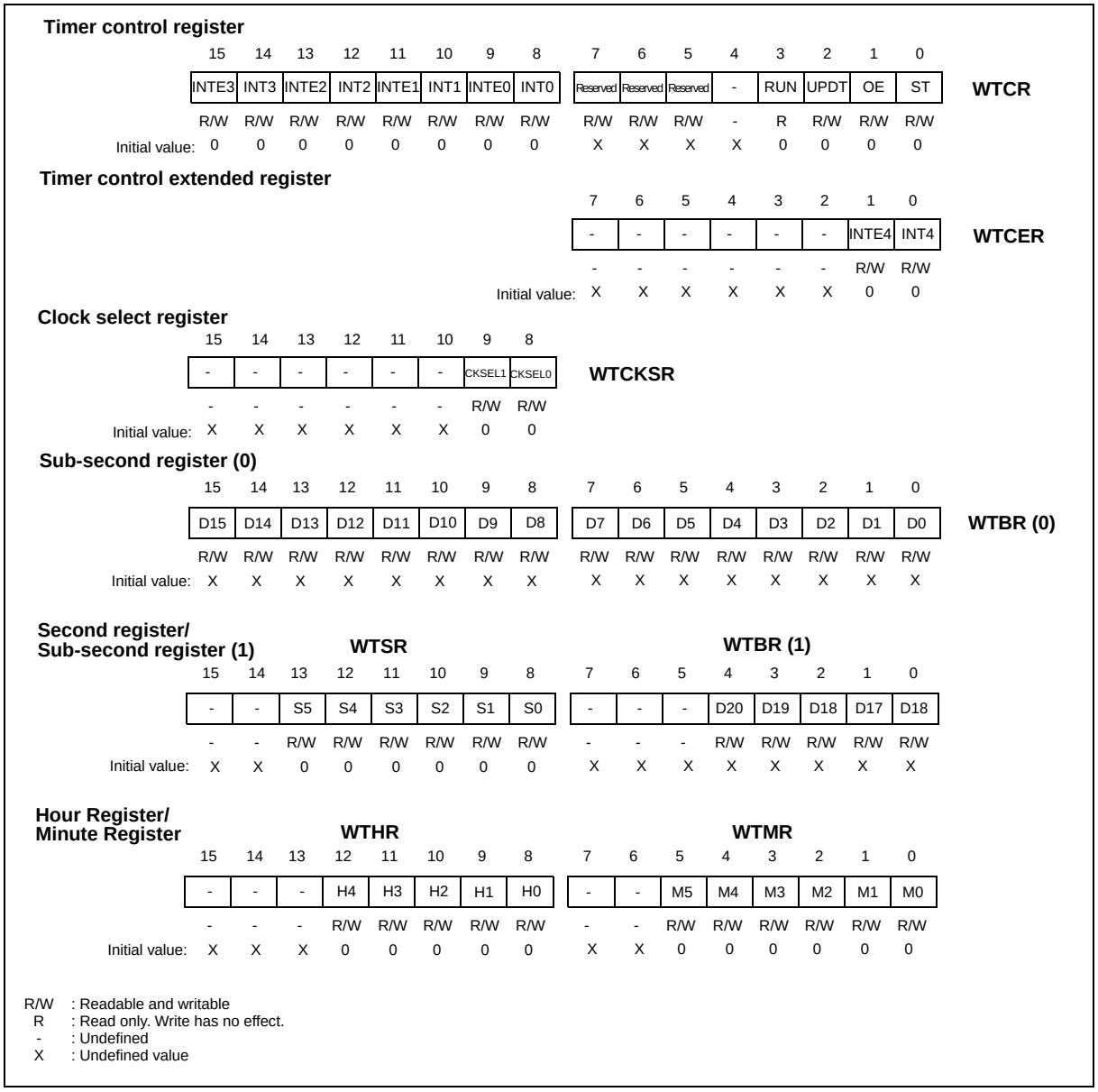
## 24.2 Real Time Clock Registers

The Real Time Clock has the following seven registers:

- Timer Control Register (WTCR)
- Timer Control Extended Register (WTCER)
- Clock Select Register (WTCKSR)
- Subsecond Register (WTBR)
- Second Register (WTSR)
- Minute Register (WTMR)
- Hour Register (WTHR)

### Real Time Clock registers

Figure 24-3. Real Time Clock registers



**Note:**

WTCR, WTCER, WTCKSR are initialized by all reset causes except unused and reserved bits.

WTBR is not reset.

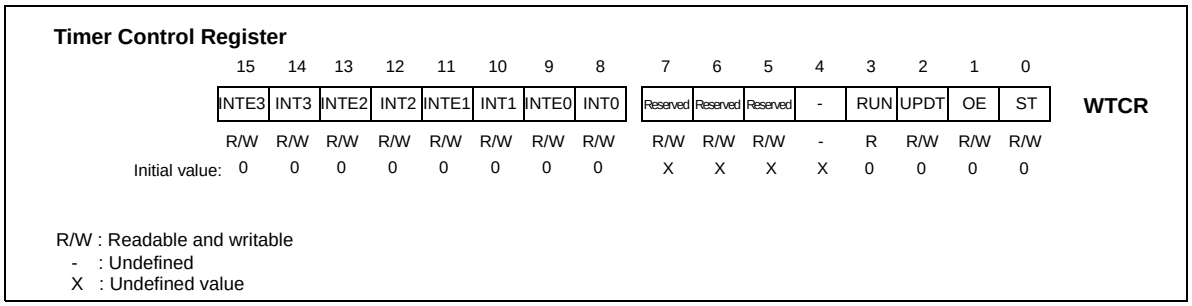
WTSR, WTMR, WTHR are initialized only by the Power-on reset, Low-voltage reset and external resets and not by any other reset, except for unused bits.

## 24.2.1 Timer Control Registers

The timer control registers start and stop the Real Time Clock, control interrupts, and set the external output pins.

### *24.2.1.1 Timer Control Register (WTCR)*

Figure 24-4. Timer Control Register (WTCR)




**Note:**

WTCR is initialized by all reset causes except for reserved or unused bits.

#### bit15: Enable interrupt requests at 1-day intervals

When the hour counter overflows, this flag is set to “1”.

| INTE3 | Operation |
|---|---|
| 0 | No interrupt requests |
| 1 | Generate interrupt requests at 1-day (24 hour) intervals. |

#### bit14: 1-day interrupt request flag

| INT3 | Status | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt requests | Clear the flag. |
| 1 | Generate interrupt requests at 1-day (24 hour) intervals. | Writing does not affect the operation. |

#### bit13: Enable interrupt requests at 1-hour intervals

When the minute counter overflows, this flag is set to “1”.

| INTE2 | Operation |
|---|---|
| 0 | No interrupt requests |
| 1 | Generate interrupt requests at 1-hour intervals. |

### bit12: 1-hour interrupt request flag

| INT2 | Status | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt requests | Clear the flag. |
| 1 | Generate interrupt requests at 1-hour intervals. | Writing does not affect the operation. |

### bit11: Enable interrupt requests at 1-minute intervals

When the minute counter overflows, this flag is set to “1”.

| INTE1 | Operation |
|---|---|
| 0 | No interrupt requests |
| 1 | Generate interrupt requests at 1-minute intervals. |

### bit10: 1-minute interrupt request flag

| INT1 | Operation | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt requests | Clear the flag. |
| 1 | Generate interrupt requests at 1-minute intervals. | Writing does not affect the operation. |

### bit9: Enable interrupt requests at 1-second intervals

When the 21 bit down counter is set to “0”, this flag is set to “1”.

| INTE0 | Operation |
|---|---|
| 0 | No interrupt requests |
| 1 | Generate interrupt requests at 1-second intervals. |

### bit8: 1-second interrupt request flag

| INT0 | Status | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt requests | Clear the flag. |
| 1 | Generate interrupt requests at 1-second intervals. | Writing does not affect the operation. |

### bit7-5: Reserved

Always write “0”. The read value is undefined. Read-modify-write is not affected.

### bit4: Undefined

Always write “0”. The read value is undefined. Read-modify-write is not affected.

### bit3: Operation status

| RUN | Operation status |
|---|---|
| 0 | The Real-time Clock module is inactive. |
| 1 | The Real-time Clock module is active. |

### bit2: Update

Before writing “1” to the update bit (UPDT), the hour/minute/second registers must be set to the values with which to update the hour/minute/second counters. The hour/minute/second registers are updated on reloading to the 21 bit down counter.

| UPDT | Status/Operation |
|---|---|
| 0 | The update has been completed. (Writing “0” does not affect the operation.) |
| 1 | Update the hour/minute/second counters with the values of the hour/minute/second registers, respectively. |

### bit1: Output Enable

| OE | Status/Operation |
|---|---|
| 0 | The WOT external pin can be used as a general purpose I/O or for another peripheral block. |
| 1 | The WOT external pin serves as the output for the 21-bit down counter. |

### bit0: Start

| ST | Operation |
|---|---|
| 0 | The Real-time Clock module stops to operate, and the 21 bit down counter and the hour/ minute/second counters are cleared. |
| 1 | The settings of the hour/minute/second registers are loaded to the hour/minute/second counters, and the Real-time Clock module starts to operate. |

### Application Notes:

The Sub-second register of the RTC module stores the reload value for the 21bit counter. This value is reloaded after the reload counter reaches "0". When modifying all three bytes, make sure the reload operation will not be performed in between the write instructions. Otherwise the 21-bit prescaler loads the incorrect value of the combination of new data and old data bytes. It is generally recommended that the Sub-Second register is updated while the ST bit is "0".

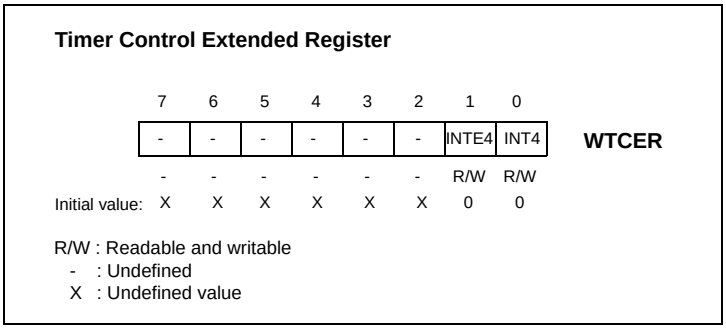
However, if this update is done immediately after an RTC second interrupt, there should be enough time to securely modify the registers until the next reload operation (next second interrupt) even if ST is not set to "0" and the module is in operation.

When updating the registers by using the ST bit the following must be taken into account:

The new value is written into the registers with the rising edge of the RUN bit. This RUN bit is clocked by the RTC clock (main clock, sub-oscillator clock or RC clock depending on device and mode). To make sure that the update is done properly, write the new values into the registers, set ST to 0, wait for the RUN bit to go low and then start the circuit again by setting ST to 1. RUN will go low at the second rising edge of the RTC clock after ST has been set to 0. It will rise again at the half second rising edge of RTC clock after ST has been set to 1. If this operation is to be done several times directly after each other, wait for RUN to go to high before setting ST to low again.

## 24.2.2 Timer Control Extended Register (WTCER)

Figure 24-5. Timer Control Extended Register (WTCER)




**Note:**

WTCER is initialized by all reset causes except for unused bits.

### bit7-2: Undefined

Writing does not affect the operation. The read value is undefined. Read-modify-write is not affected.

### bit1: Enable interrupt requests at half-second (500ms) intervals

When the 21-bit counter overflows, this flag is set to “1”.

| INTE4 | Operation |
|---|---|
| 0 | No interrupt requests |
| 1 | Generate interrupt requests at half-second (500 ms) intervals. |

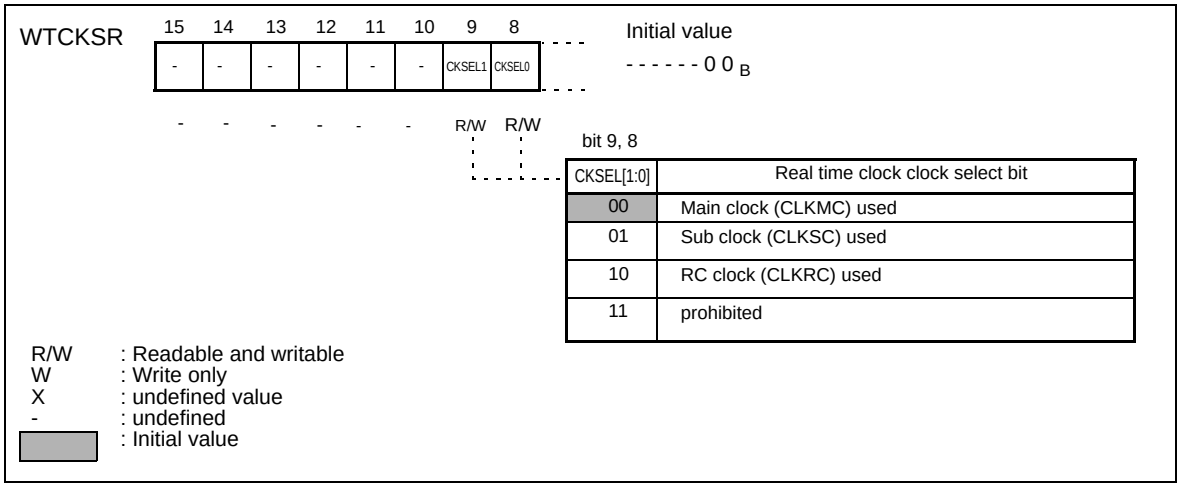
### bit0: half-second (500ms) interrupt request flag

| INT4 | Status | |
|---|---|---|
| | Read | Write |
| 0 | No interrupt requests | Clear the flag. |
| 1 | Generate interrupt requests at half-second (500 ms) intervals. | Writing does not affect the operation. |

## 24.2.3 Clock Select Register (WTCKSR)

The Clock Select Register is used to select the clock source for Real Time Clock.

Figure 24-6. Clock Select Register (WTCKSR)




**Note:**

WTCKSR is initialized by all reset causes except for unused bits.

**Application Notes:**

When the setting of WTCKR:CKSEL[1:0] shall be changed, the clock on which the Real time clock is

currently running must stay enabled for at least 3 further cycles after change. Otherwise

the Real time clock does not change to the new setting. The former clock can be switched

off afterwards.

Example: Real time clock shall be changed from Main clock to RC-Clock:

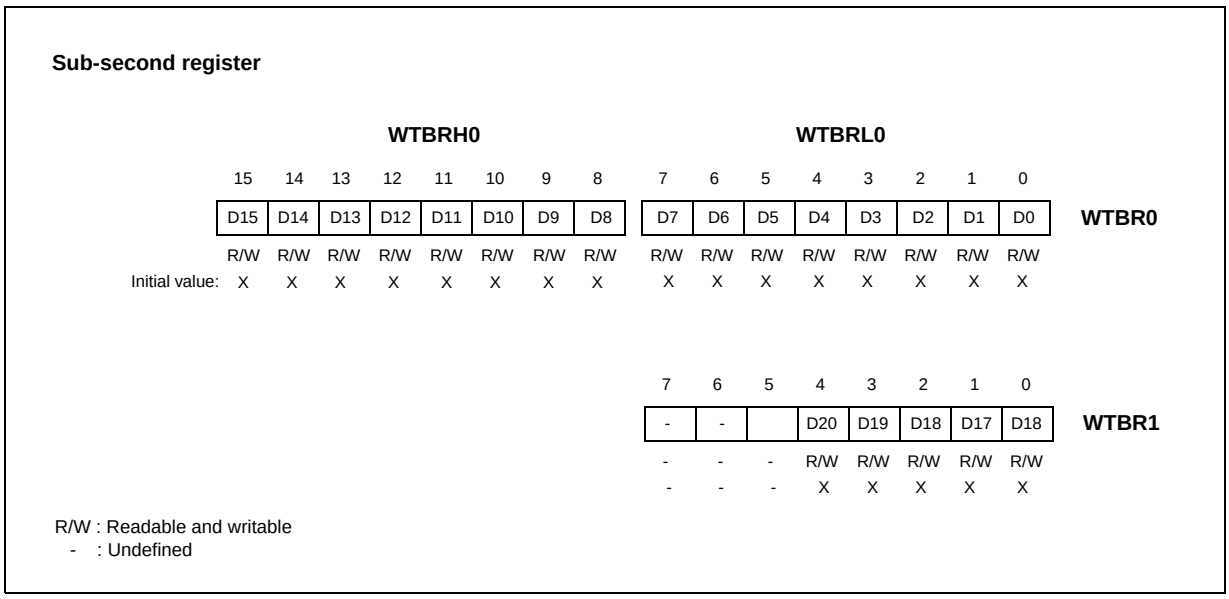
Main clock oscillator is enabled after power on. Therefore is must be enabled when WTCKSR:CKSEL[1:0] is changed to 'RC clock' by writing '10' to it. Otherwise RTC cannot switch to RC clock. The Main oscillator must stay enabled for at least three Main clock cycles after the write access to WTCKST:CKSEL[1:0].

## 24.2.4 Sub-Second Register

The Sub-Second Register stores a reload value for the 21-bit counter that divides the oscillation clock. The reload value is usually set so that the 21-bit counter will output exactly within a half second cycle. This register is not initialized by reset, but 21-bit counter is initialized by reset.

### *24.2.4.1 Sub-Second Register (WTBR)*

Figure 24-7. Sub-Second Register (WTBR)




**Application Notes:**

The Sub-Second Registers, WTBR, holds values to be reloaded to the 21 bit down counter. When the 21 bit down counter value becomes “0”, the settings of WTBR are reloaded to the 21 bit down counter.

Remark: The reload value to be set in the sub-second registers corresponds to the time for half a second. One second is reached after counting twice the reload value set in WTBR.

Table 24-1. Example configuration of WTBR registers for different clock configurations

| | WTBR1 | WTBRH0 | WTBRL0 |
|---|---|---|---|
| Main oscillator, 4MHz | $0F_H$ | $42_H$ | $3F_H$ |
| RC oscillator, 2MHz | $07_H$ | $A1_H$ | $1F_H$ |
| RC oscillator, 100kHz | $00_H$ | $61_H$ | $A7_H$ |
| Sub oscillator, 32.768kHz | $00_H$ | $1F_H$ | $FF_H$ |

## 24.2.5 Second/Minute/Hour Registers (WTSR, WTMR, WTHR)

The Second/Minute/Hour registers store the time information. It is a binary representation of the second, minute and hour.

Reading these registers simply returns the counter values. These registers are write accessible however, the written data is loaded in the counters after the UPDT bit is set to "1". These registers and counter are initialized by reset.

### *24.2.5.1 Second/minute/hour registers*

Figure 24-8. Configuration of the second/minute/hour registers (WTSR, WTMR, WTHR)

**Second register**

**WTSR**

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | - | - | S5 | S4 | S3 | S2 | S1 | S0 |
| | - | - | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial value: | X | X | 0 | 0 | 0 | 0 | 0 | 0 |

**Hour Register/ Minute Register**

**WTHR**

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| | - | - | - | H4 | H3 | H2 | H1 | H0 |
| | - | - | - | R/W | R/W | R/W | R/W | R/W |
| Initial value: | X | X | X | 0 | 0 | 0 | 0 | 0 |

**WTMR**

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | - | - | M5 | M4 | M3 | M2 | M1 | M0 |
| | - | - | R/W | R/W | R/W | R/W | R/W | R/W |
| Initial value: | X | X | 0 | 0 | 0 | 0 | 0 | 0 |

R/W : Readable and writable
R : Read only. Write has no effect.
- : Undefined

**Note:**

WTSR, WTMR, WTHR are initialized only by the Power-on reset, Low-voltage reset and external resets and not by any other reset except for unused bits.

**Application Notes:**

The values written to the hour/minute/second registers are the initial values to be loaded to the hour/minute/second counters.

By setting “1” to the update bit (WTCR:UPDT) or the start bit (WTCR:ST), the hour/minute/second register values are written to the hour/minute/second counters.

The hour/minute/second counter values are saved to the hour/minute/second registers every time when the second counter overflows, that is, at intervals of one minute. When the hour/minute/second counters are read, the saved count values, not written ones, are read.

The hour/minute/second registers consist of two separate sets of registers: one for reading and the other for writing. The write registers get updated when CPU writes into the hour/minute/second registers. The read registers get updated values from the internal logic and these are the values available when CPU tries to read from the hour/minute/second registers anytime. Hence, if it is intented to stop and continue the realtime clock some time after it has been set, make sure to update the write registers with the most recent values of the read registers.

Since there are three byte-registers, make sure the obtained values from the registers are consistent.

For example, obtained value of "1 hour, 59 minute, 59 second" could be "1 hour, 59 minute, 59 second" or "1 hour, 0 minute, 0 second" or "2 hour, 0 minute, 0 second".

Also when the CPU operation clock CLKB is a fraction of the Realtime clock’s operating clock, the read values from these registers may be inconsistent, because they change their value during the read operation. Therefore it is recommended to use the 1-second interrupt to trigger the read instructions, since after this interrupt these registers are not updated for 1 second.

Do not write impossible data (e.g. 60 seconds) into the WTSR, WTMR or WTHR register.

## 24.3 Operation

- This section describes the Real Time Clock operation.

### Real Time Clock operation

Figure 24-9. Real Time Clock operation

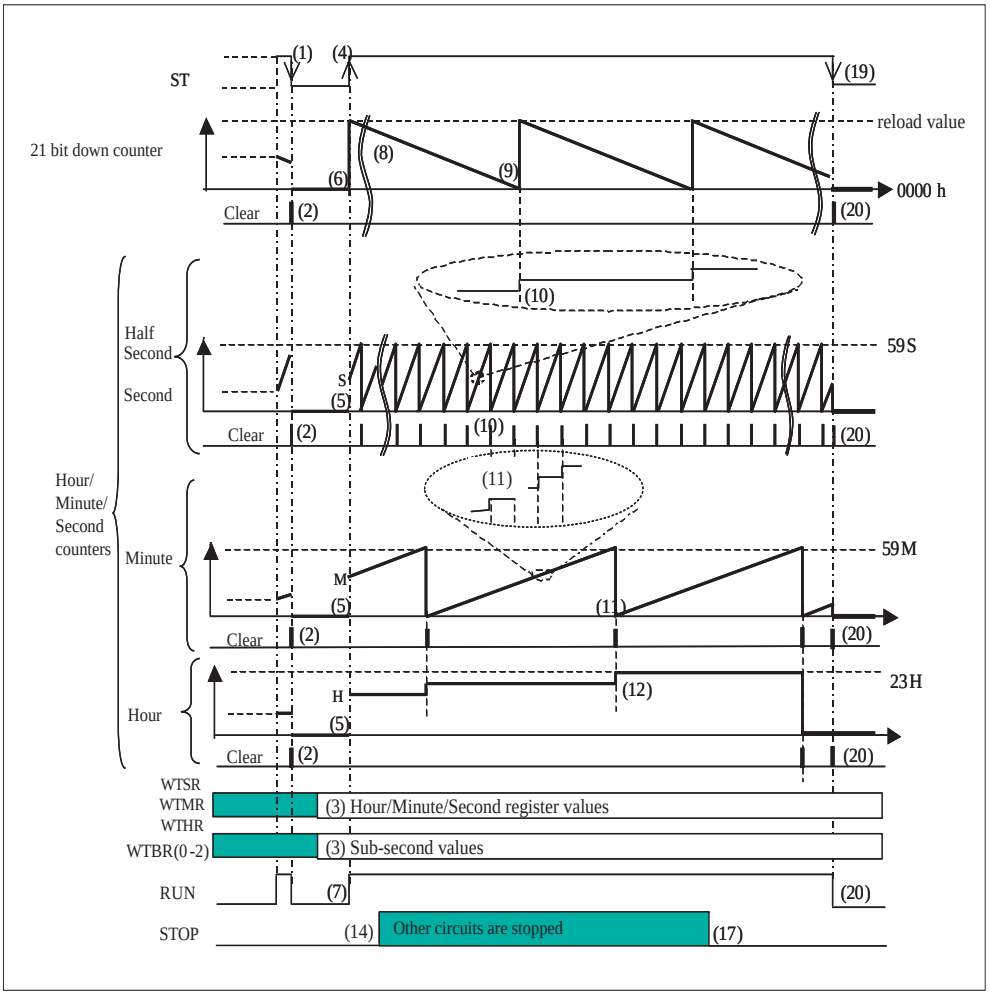
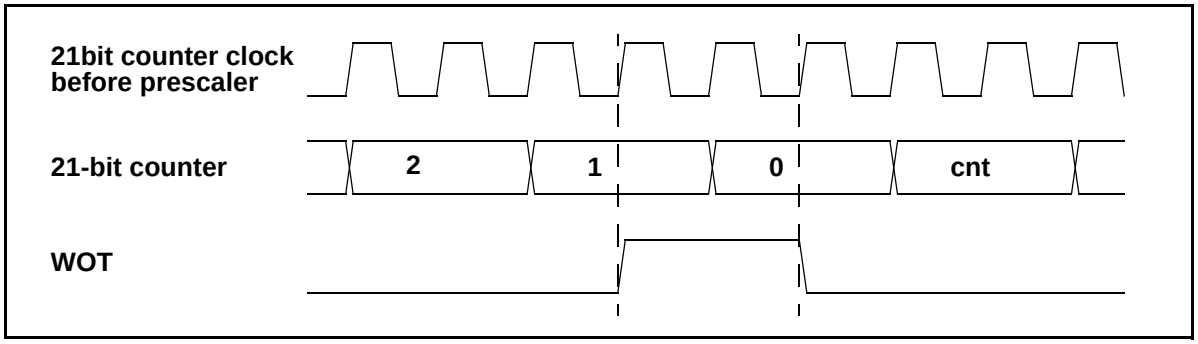



Figure 24-10. WOT pin operation



(1) The start bit (ST) is set to “1” and then “0”. (Register initialization operation)

(2) This (ST=“0”) resets to 0 and stops the 21 bit down counter and the hour/minute/second timers.

(3) The software writes hour, minute, and second values to the hour/minute/second registers, WTHR/WTMR/WTSR.

(4) The software writes the appropriate values to the sub-second registers, WTBRL0/WTBRH0/WTBR1.

(5) The interrupt request bits (INT0, INT1, INT2, INT3 and INT4) are initialized, and the interrupt request enable bits (INTE0, INTE1, INTE2, INT3 and INTE4) are set to “interrupts enabled”.

(6) The start bit (ST) is set to “1”.

(7) This (ST=“1”) causes the values of the hour/minute/second registers, WTHR/WTMR/WTSR, to be loaded to the hour/minute/second timers.

(8) The values of the sub-second registers, WTBRL0/WTBRH0/WTBR1, are loaded to the 21 bit down counter.

(9) The run flag (RUN) is set to “1”.

(10) The 21 bit down counter begins counting at the mainclock (CLKMC) divided by 2 (4/2 MHz), subclock (CLKSC) divided by 2 (32.768/2 KHz) or RC clock (CLKRC) divided by 2 (100/2 kHz or 2/2MHz).

(11) When the 21 bit down counter reaches “000000H”, the value of the sub-second registers is loaded to the 21 bit down counter, generating a half-second interrupt request. Each second half-second interrupt a second interrupt will be generated.

(12) When the second counter counts up to “59”, the counter is cleared next time when the counter counts up, at which the minute counter counts up, generating a 1-minute interrupt request.

(13) When the minute counter counts up to “59”, the counter is cleared next time when the counter counts up, at which the hour counter counts up, generating a 1-hour interrupt request.

(14) When the hour counter counts up to “23”, the counter is cleared next time when the counter counts up, at which a 1-day interrupt request is generated.

(15) The software changes the state of the MCU to Timer Mode by setting the bits SMCR:SMS[1:0] to “10” (see section Standby Mode Control Register (SMCR) on page 192 for details). The Real-time Clock continues to operate in the Timer Mode.

(16) The device recovers from Timer Mode mode (by interrupt request).

(17) This (ST=“0”) resets and stops the 21 bit down counter and the hour/minute/second counters.

## 24.4 Cautions

This section describes the cautions to be considered while using the Real Time Clock

### 24.4.1 Cautions

- Setting the interrupt request flags (WTCR:INT0, WTCR:INT1, WTCR:INT2, WTCR:INT4 and WTCER:INT4) to “1” due to overflow, and writing “0” to that bit have occurred at the same time, the flag is set to “1” (Flag setting takes precedence).
- Writing “1” to the update bit (WTCR:UPDT) and update completion have occurred at the same time, the update bit (UPDT) is set to “0”.
- When the second counter holds the value of 59, even if “1” is written to the update bit (WTCR:UPDT), the hour/minute/second counters are not updated, leaving the update bit to remain “0”.
  In order to update the hour/minute/second counters, it is recommended that “0” should be written to the start bit (WTCR:ST), the hour/minute/second counters be cleared to “0”, and then “1” be written to the start bit (ST).
- If you stop the peripheral clock (CLKP1) after updating the hour/minute/second counters using the update bit (WTCR:UPDT), read the hour/minute/second registers to confirm that they have been updated before stopping the peripheral clock CLKP1.
- When you start to use the Real Time Clock module, change the start bit (ST) from “1” to “0”, and clear the hour/minute/second counters and the 21 bit down counter to “0”.

- The Sub-second register of the RTC module stores the reload value for the 21bit prescaler. This value is reloaded after the reload counter reaches "0". When modifying all three bytes, make sure the reload operation will not be performed in between the write instructions. Otherwise the 21-bit prescaler loads the incorrect value of the combination of new data and old data bytes. It is generally recommended that the Sub-Second register is updated while the ST bit is "0".
- However, if this update is done immediately after an RTC second interrupt there should be enough time to securely modify the registers until the next reload operation (next second interrupt) even if ST is not set to "0" and the module is in operation.
- When updating the registers by using the ST bit the following must be taken into account: The new value is written into the registers with the rising edge of the RUN bit. This RUN bit is clocked by the RTC clock (32 kHz, 100kHz, 2MHz or 4 MHz depending on device and mode). To make sure that the update is done properly, write the new values into the registers, set ST to 0, wait for the RUN bit to go low and then start the circuit again by setting ST to 1. RUN will go low at the second rising edge of the RTC clock after ST has been set to 0. It will rise again at the half second rising edge of RTC clock after ST has been set to 1. If this operation is to be done several times directly after each other, wait for RUN to go to high before setting ST to low again.
- If a reload has occurred during updating the sub-second registers, WTBRL0, WTBRH0, WTBR1, an unexpected value may be reloaded to the 21 bit down counter. Therefore, it is recommended that the sub-second register, WTBR, should be updated with the start bit (WTCR:ST) set to “0”.
- If all the sub-second registers, WTBRL0, WTBRH0, WTBR1, are set to “0”, the 21 bit down counter does not operate, resulting in the Real-time Clock module to be inoperational.
- If a carry has occurred during reading from the hour/minute/second registers, WTHR/WTMR/WTSR, inappropriate values may be read. To avoid this, it is recommended that interrupts (INT0-4) should be used to read time (HH/MM/SS).
- In order for the Real Time Clock module to function properly, the frequency of the subclock (CLKSC) or the RC-clock (CLKRC) must be much lower than that of the peripheral clock (CLKP1). If not, correct values cannot be read from WTHR/WTMR/WTSR.
- Note that only byte-access is allowed to these register. So, when these registers are read at the very timing of changing over the hour or minute boundary as shown below, there is a possibility of misjudging the time. So, read several times to get a logically consistent value.

  Example: Read begins at the second register: 02:59:59=> 03:59:59 => 03:00:00
  Read begins at the hour register: 02:59:59 => 02:00:00 => 03:00:00

  In this case, the current time should be interpreted as 3 o'clock.

# 25. Clock Calibration Unit

This chapter explains the functions and operation of the Clock Calibration Unit

25.1 Outline

25.2 Register Description

25.3 Application Notes

## 25.1 Outline

The Clock Calibration Module provides possibilities to calibrate the sub  oscillator clock or the RC oscillator clock with respect to the main oscillator clock. This chapter gives an overview of the Clock Calibration Unit, describes the registers and provides some application notes.

### Description

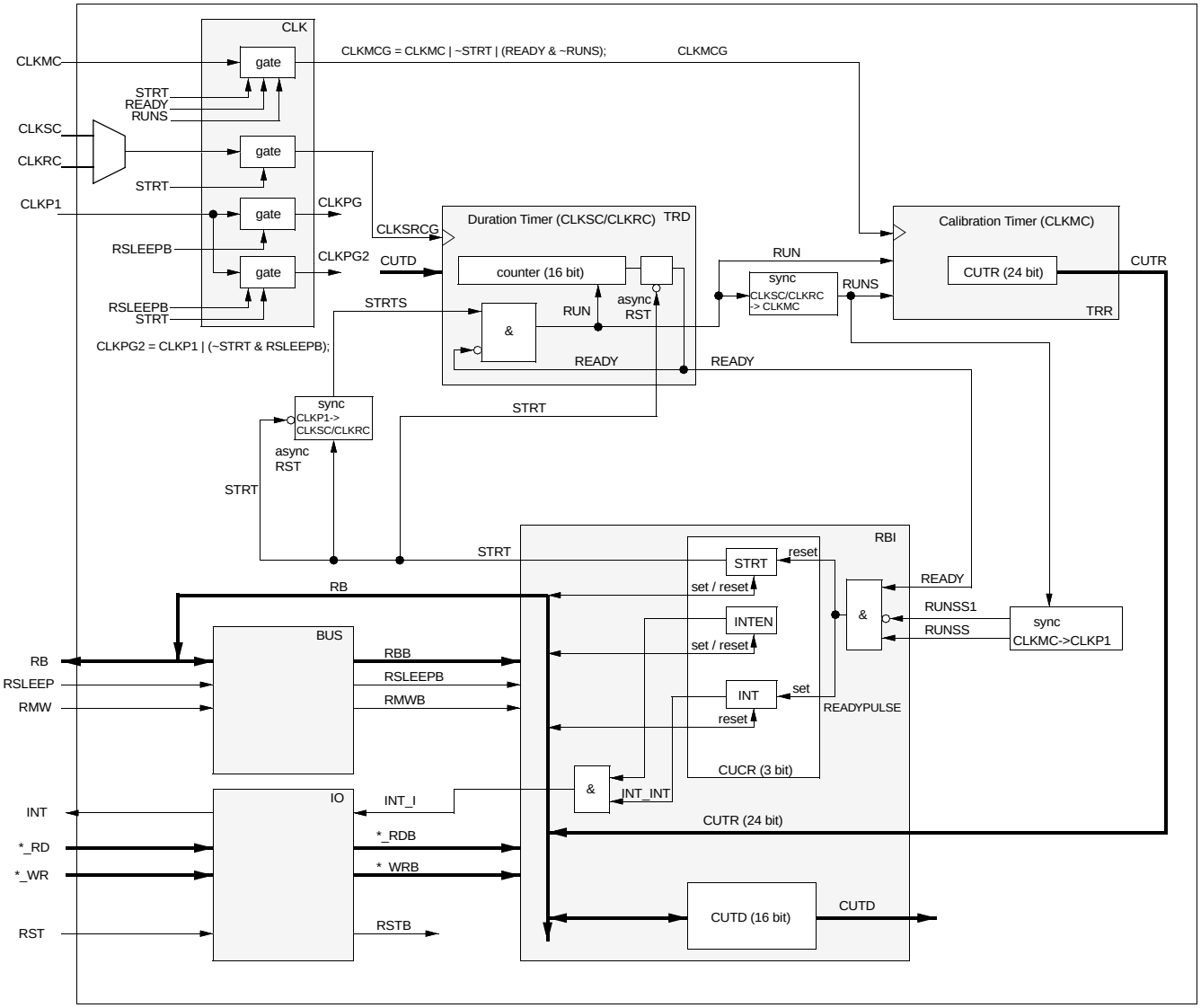
This hardware allows the software to measure time generated by the sub oscillator clock CLKSC or the RC oscillator clock CLKSC with the main oscillator clock CLKMC.

By utilizing this hardware in conjunction with software processing, the accuracy of the sub oscillator clock or RC oscillator clock can come closer to that of the main oscillator clock. The measurement result from the Clock Calibration Unit can be processed by the software and the setting required for the Real Time Clock Module can be obtained.

This module consists of two timers. The Duration Timer operates with the sub oscillator clock CLKSC or RC-clock CLKRC. The Calibration Timer operates with the main oscillator clock CLKMC. Typical values are 32kHz for CLKSC, 4MHz for CLKMC and 100kHz or 2MHz for CLKRC. The Duration Timer triggers the Calibration Timer and the resulting Calibration Timer value is stored into a register. The value stored in this register can be used for the subsequent software processing to calculate the desired Real Time Clock module's setting.

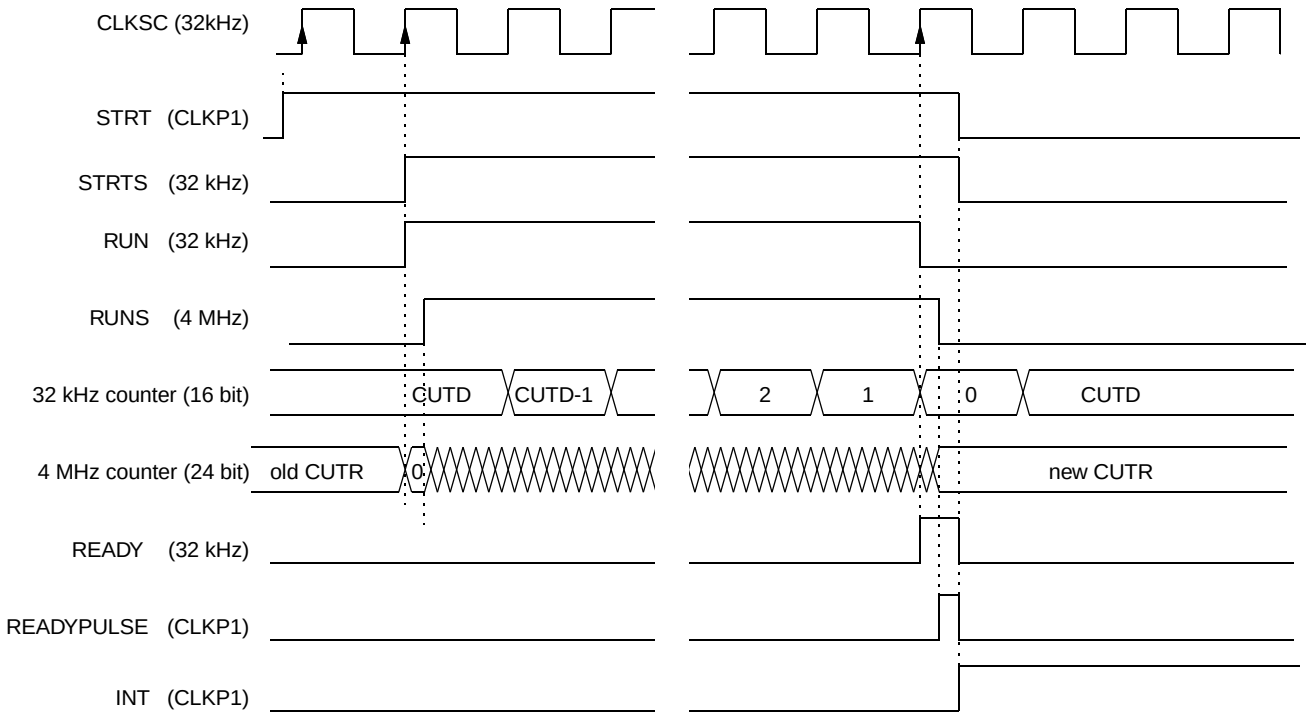
## Block Diagram

Figure 25-1. Block Diagram



## Timing

Figure 25-2. Timing of the measurement process




## Clocks

The module operates with 3 different clocks: The main oscillator clock CLKMC, the sub-clock CLKSC (or the RC clock CLKRC) and the peripheral clock CLKP1. For an overview about available clocks, please refer to Figure 6-1. Synchronization circuits adapt the different domains.

All 3 clocks are gated. CLKSC (or CLKRC) and CLKMC clock are switched off if STRT is 0. CLKPG is gated by RSLEEP and CLKPG2 by RSLEEP and STRT for the 2 bits, which are set/reset by hardware.

The clock frequencies have to fulfill the following requirements:

- $T_{CLKSC/CLKRC} > 2 \times T_{CLKMC} + 3 \times T_{CLKP1}$
- $T_{CLKMC} < 1/2 \times T_{CLKSC/CLKRC} - 3/2 \times T_{CLKP1}$
- $T_{CLKP1} < 1/3 \times T_{CLKSC/CLKRC} - 2/3 \times T_{CLKP1}$

Table 25-1. Example of valid clock ratios which fulfill the requirements

| | CLKSC | | CLKRC | | CLKMC | | CLKP1 | |
|---|---|---|---|---|---|---|---|---|
| maximum operation speed | 100 kHz | 10 us | 2 MHz | 500 ns | 10 MHz | 100 ns | 50 MHz | 20 ns |
| normal operation | 32 kHz | 31.25 us | 100 kHz | 10 us | 4 MHz | 250 ns | >2 MHz | 500 ns |

## 25.2 Register Description

This section lists the registers of the calibration unit and describes the function of each register in detail.

### 25.2.1 Clock Calibration Unit registers

| Register | | | |
|---|---|---|---|
| +0 | +1 | +2 | +3 |
| CUCR [R/W]<br>---0--00 -------- | | CUTD [R/W]<br>00000000 10000000 | |
| CUTR1 [R]<br>00000000 -------- | | CUTR2 [R]<br>00000000 00000000 | |

### 25.2.2 Clock Calibration Unit Control Register (CUCR)

Control Register low byte

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| | RES | - | - | STRT | - | CKSEL | INT | INTEN | CUCR |
| Read/write ⇒ | (W0/R) | (R) | (R) | (R/W) | (R) | (R/W) | (R/W) | (R/W) | |
| Default value⇒ | (X) | (0) | (0) | (0) | (0) | (0) | (0) | (0) | |

The Clock Calibration Unit Control Register (CUCR) has the following functions:

start / stop calibration measurement

enable / disable interrupt

indicates the end of the calibration measurement

#### BIT[0]: INTEN - Interrupt enable

| | |
|---|---|
| 0 | interrupt disabled (default) |
| 1 | interrupt enabled |

This is the interrupt enable bit corresponding to the INT bit. When this bit is set to 1 and the INT bit is set by the hardware, the calibration module signals an interrupt to the CPU. The INT-bit itself is not affected by the INTEN bit and is set by hardware even if interrupts are disabled (INTEN=0).

#### BIT[1]: INT - Interrupt

| | |
|---|---|
| 0 | calibration ongoing / module inactive (default) |
| 1 | calibration completed |

This bit indicates the end of the calibration. When the Duration Timer reaches zero after the start of calibration, the Calibration Timer Data Register stores the last Calibration Timer value and the INT bit is set to 1.

The read-modify-write operation to this bit results in reading 1. Writing 0 to this bit clears this flag (INT=0). Writing 1 to this bit has no effect.

The interrupt flag INT is not reset by hardware. Therefore it must be reset by software before starting a new calibration. Otherwise the end of the calibration process is only signalized by the STRT bit (the INT flag stays 1 also during calibration).

#### BIT[2]: CKSEL - Input clock select

| | |
|---|---|
| 0 | use CLKSC (default) |
| 1 | use CLKRC |

The RC-clock frequency depends on the setting of CKSCR:RCFS bit (RC-Oscillator frequency select).

Devices with sub clock: If the Clock setting shall be changed, the clock which is currently selected must be active for at lease 3 cycles if it was enabled at any time before. Otherwise the selected clock does not change to the new clock.

### BIT[4]: STRT - Calibration Start

| | |
|---|---|
| 0 | calibration stopped, module switched off (default) |
| 1 | start calibration |

When the STRT bit is set to 1 by the software, the calibration starts. The Duration Timer starts counting down from the value stored in the Duration Timer Data Register CUTD and the Calibration starts counting up from zero.

When the Duration Timer reaches zero, this bit is reset to 0 by the hardware.

If 0 is written into this bit by the software during the calibration process, the calibration is immediately stopped. If writing 0 by the software and reset to 0 by the hardware happens at the same time, the hardware operation supersede the software operation. This means the calibration is properly completed and the INT bit is set to “1”. Writing 1 to this bit during the calibration has no effect.

### BIT[7]: RES - Reserved bit

Always write “0”. The read value is undefined. Read-modify-write is not affected.

## 25.2.3 Clock Calibration Unit Duration Timer Data Register (CUTD)

Duration Timer Register high byte

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| | TDD15 | TDD14 | TDD13 | TDD12 | TDD11 | TDD10 | TDD9 | TDD8 | CUTDH |
| Read/write ⇒ | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Default value⇒ | (1) | (0) | (0) | (0) | (0) | (0) | (0) | (0) | |

Duration Timer Register low byte

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ⇐ Bit no. |
|---|---|---|---|---|---|---|---|---|---|
| | TDD7 | TDD6 | TDD5 | TDD4 | TDD3 | TDD2 | TDD1 | TDD0 | CUTDL |
| Read/write ⇒ | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | (R/W) | |
| Default value⇒ | (0) | (0) | (0) | (0) | (0) | (0) | (0) | (0) | |

The Clock Calibration Unit Duration Timer Data Register (CUTD) holds the value which determines the duration of calibration, i. e. it stores Duration Timer reload value.When the calibration is started, the stored value is loaded into the Low Speed Timer and the timer starts counting down until it reaches zero.

The Clock Calibration Unit Duration Timer operates with CLKSC or with CLKRC clock, depending on the setting of CUCR:CKSEL.

Default value is 0x8000 which corresponds to a measurement duration of 1 second, if CLKSC is used with a 32.768 kHz crystal.

This register should be written only when the calibration is inactive (STRT=0).

If CUTD is initialized with 0x0000 an underflow will occur and the measurement will take (0xFFFF + 1) * $T_{CLKSC/CLKRC}$

In order to achieve a measurement duration of 1 second at a 32kHz oscillation, the CUTD register has to be loaded with 0x8000 = 32768 dec. This number results from the exact oscillation frequency of the crystal, which is $F_{osc}$=32768 Hz. The ideal values of the measurement results (if a 4.00 MHz crystal is used) are shown in the following table.

**32kHz: Ideal measurement results depending on measurement duration**

| duration of calibration | CUTD value | CUTR value |
|---|---|---|
| 2 sec | 0x0000 | 0x7A1200 |
| 1.75 sec | 0xE000 | 0x6ACFC0 |
| 1.5 sec | 0xC000 | 0x5B8D80 |
| 1.25 sec | 0xA000 | 0x4C4B40 |
| 1 sec | 0x8000 | 0x3D0900 |
| 0.75 sec | 0x6000 | 0x2DC6C0 |
| 0.5 sec | 0x4000 | 0x1E8480 |
| 0.25 sec | 0x2000 | 0x0F4240 |

In order to achieve a measurement duration of half a second at a 100kHz oscillation, the CUTD register has to be loaded with 0xC350 = 50000 dec. This number results from the exact oscillation frequency of $F_{osc}$=100000 Hz. The ideal values of the measurement results (if a 4.00 MHz crystal is used) are shown in the following table.

**100kHz: Ideal measurement results depending on measurement duration**

| duration of calibration | CUTD value | CUTR value |
|---|---|---|
| 0.5 sec | 0xC350 | 0x1E8480 |
| 0.25 sec | 0x61A8 | 0x0F4240 |
| 0.125 sec | 0x30D4 | 0x07A120 |
| 0.1 sec | 0x2710 | 0x061A80 |

The duration of the whole process from writing a 1 into the STRT bit until STRT is reset by hardware is longer than the actual calibration measurement time, due to synchronization between the different clock domains. Process Duration < (CUTD + 3) * $T_{CLKSC/CLKRC}$ .

The calibration measurement time is CUTD * $T_{CLKRC}$.

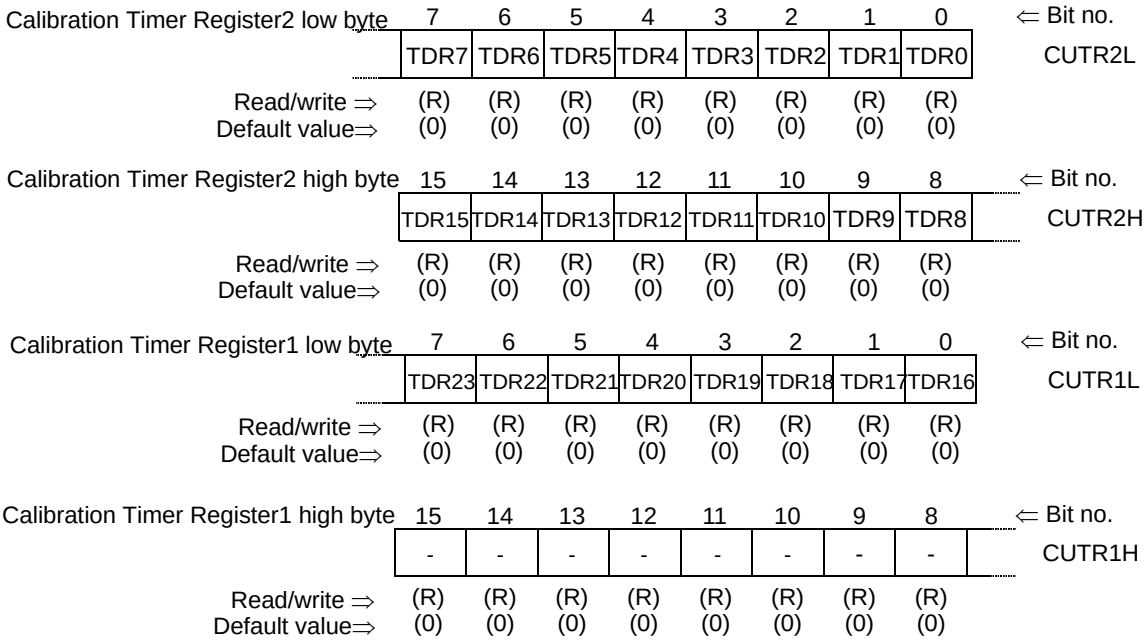
## 25.2.4 Clock Calibration Unit Calibration Timer Data Register (24 bits) (CUTR)

The Clock Calibration Unit Calibration Timer Data Register stores the result of the calibration. When the calibration is started, the Calibration Timer starts counting up from zero. When the Duration Timer reaches zero, the Calibration Timer stops counting and the register holds the calibration result until the next calibration is triggered by software.

**Precaution:**

Reading this register during calibration results in random values. The end of calibration is indicated by the INT-bit and the STRT-bit in the CUCR-register.

After INT has changed from 0 to 1 / STRT has changed from 1 to 0, the value of CUTR is valid.




Writing into this register by software has no effect.

The Calibration Timer operates with the main oscillator clock CLKMC.

## 25.3 Application Notes

This section lists application notes concerning accuracy of the calibration, power dissipation and measurement duration.

### 25.3.1 32 kHz

The setting of the Duration Timer Data Register can be calculated in the following way.

If the duration of 1 second is desired for the calibration, 8000Hex = 32768Dec should be set in the Duration Timer Data Register and it represents 32,768 pulses of the 32.768kHz oscillation clock.

This setting should result in the stored value of approximately 3D0900Hex in the Calibration Timer Data Register. This value represents 4,000,000 pulses of the 4MHz oscillator.

**Ideal measurement results (CUTR) with 32.768kHz and 4.0MHz oscillators**

| duration of calibration | CUTD value | CUTR value |
|---|---|---|
| 2 sec | 0x0000 | 0x7A1200 |
| 1.75 sec | 0xE000 | 0x6ACFC0 |
| 1.5 sec | 0xC000 | 0x5B8D80 |
| 1.25 sec | 0xA000 | 0x4C4B40 |

| duration of calibration | CUTD value | CUTR value |
|---|---|---|
| 1 sec | 0x8000 | 0x3D0900 |
| 0.75 sec | 0x6000 | 0x2DC6C0 |
| 0.5 sec | 0x4000 | 0x1E8480 |
| 0.25 sec | 0x2000 | 0x0F4240 |

### 25.3.2 100 kHz

The setting of the Duration Timer Data Register can be calculated in the following way.

If the duration of 0.5 second is desired for the calibration, C350Hex = 50000Dec should be set in the Duration Timer Data Register and it represents 50,000 pulses of the 100kHz oscillation clock.

This setting should result in the stored value of approximately 1E8480Hex in the Calibration Timer Data Register. This value represents 2,000,000 pulses of the 4MHz oscillator.

**Ideal measurement results (CUTR) with 100kHz and 4.0MHz oscillators**

| duration of calibration | CUTD value | CUTR value |
|---|---|---|
| 0.5 sec | 0xC350 | 0x1E8480 |
| 0.25 sec | 0x61A8 | 0x0F4240 |
| 0.125 sec | 0x30D4 | 0x07A120 |
| 0.1 sec | 0x2710 | 0x061A80 |

The purpose to use of the Clock Calibration Unit is to reduce power dissipation of the MCU while maintaining an accurate clock, for example for a daytime display.

### 25.3.3 Accuracy

The accuracy of the calibration is dependent on the clock frequency used by the Calibration Timer and the duration of the calibration. The maximum error of the Calibration Timer is +/-1 digit. If the clock frequency is 4MHz and the duration of the calibration is 1 second, the achieved accuracy is calculated in the following way:

0.25us (Clock cycle time) / 1 second (duration)=0.25 ppm.

In general:

Accuracy = (Clock cycle time of Calibration Timer) / (Duration of calibration)

### 25.3.4 Power Dissipation

Suppose the current consumption $I_{RUN}$ in run mode is 20 times the consumption $I_{RTC}$ in RTC mode ($I_{RUN}$ = 20 x $I_{RTC}$ ).

If the MCU is woken up from RTC mode by software to trigger the calibration measurement every minute and the duration of the calibration is set to 1 second, the increase in the power dissipation can be 20 x $I_{RTC}$/60 = 1/3 x $I_{RTC}$.

Therefore the software has to make sure that the increase should not affect the hardware limitations coming from the system requirements. For example, the software has to be designed to trigger the calibration least frequently.

It is generally recommended that the theoretical increase in power dissipation is not more than 5% particularly in the RTC mode of the MCU.

### 25.3.5 Measurement Limits

- The limit to the duration of the calibration is roughly 2 seconds if the Duration Timer is operating with a 32kHz clock (0.5 seconds if operating with 100kHz). On the other hand, the Calibration Timer can measure time up to 4 seconds if it is working with a 4MHz clock.

# 26. LCD Controller/Driver

This chapter describes the functions and operations of the LCD Controller/Driver.

26.1 Configuration of LCD Controller/Driver

26.2 LCD Controller Registers

26.3 LCD Enable Registers (LCDER, LCDVER)

26.4 LCD Controller/Driver Display RAM

26.5 Operation of LCD Controller/Driver

26.6 Cautions

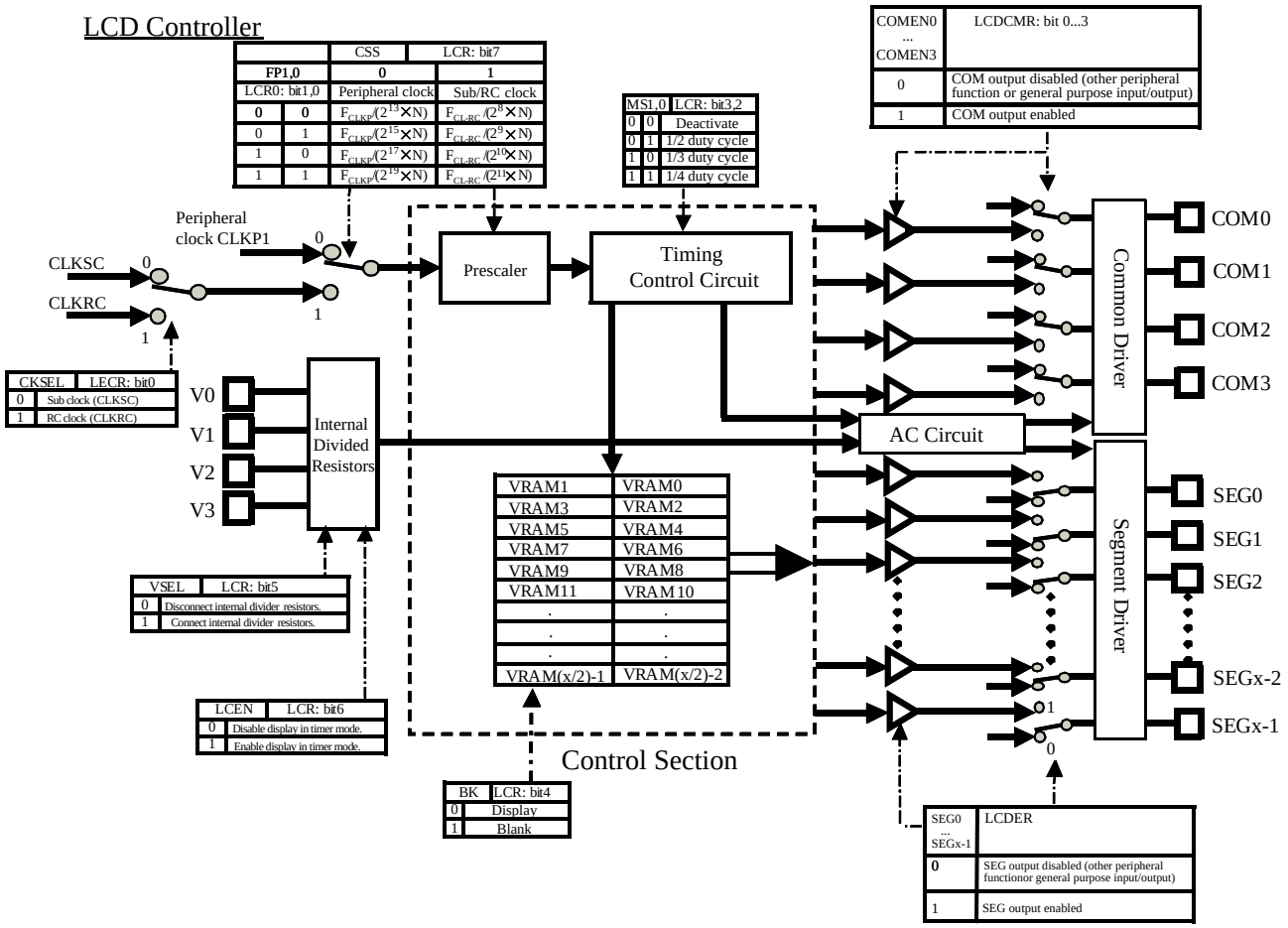
## 26.1 Configuration of LCD Controller/Driver

The LCD Controller/Driver consists of the blocks listed below. Functionally, it consists of the controller section, which generates a segment signal and common signal based on display RAM data, and the driver section, which drives the LCD.

- LCD control register (LCR)
- Display RAM
- Prescaler
- Timing controller
- AC circuit
- Common driver
- Segment driver
- Divide resistor

## Configuration Diagram

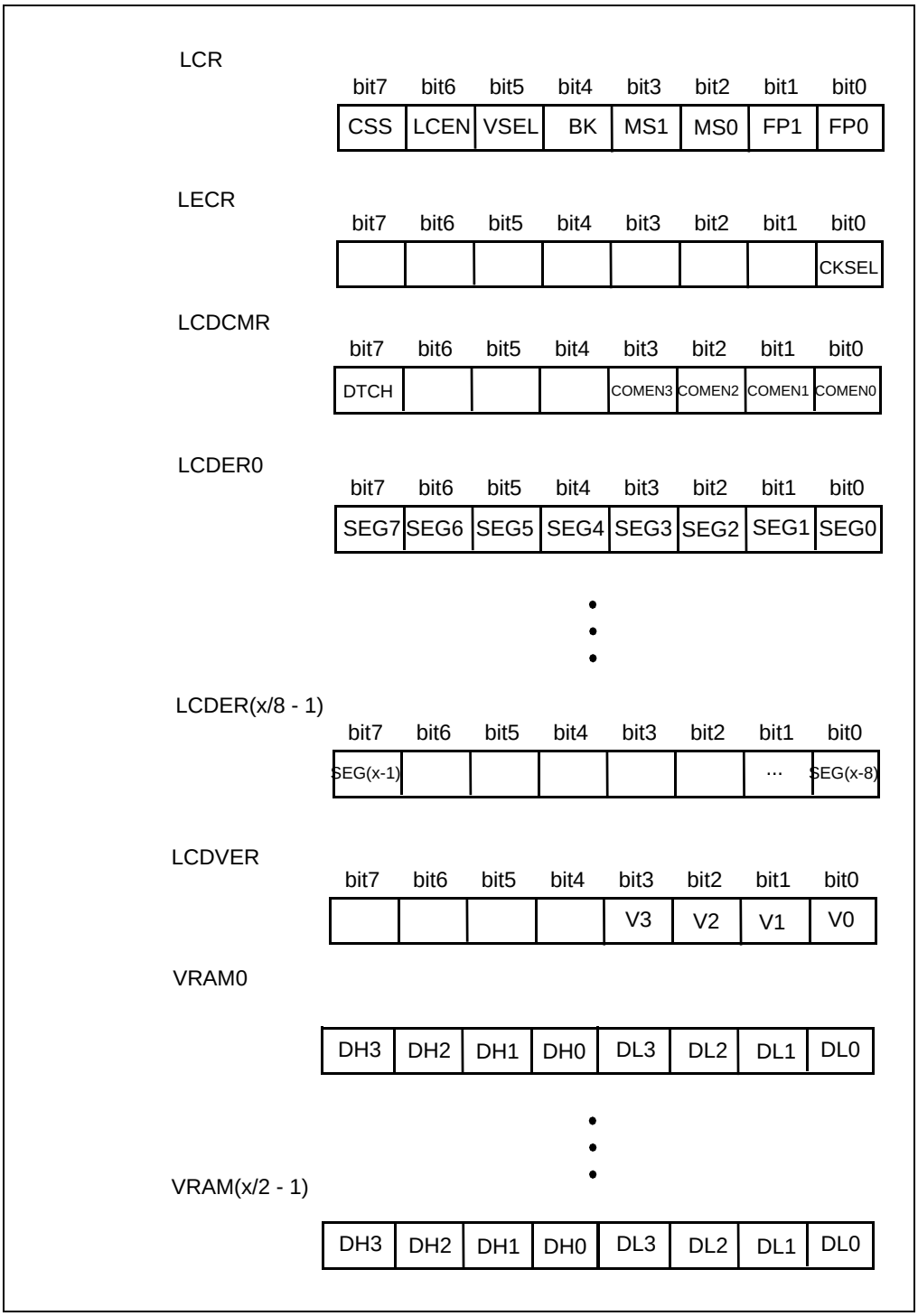
Figure 26-1. Configuration Diagram




x = Number of available segments

## Register List

Figure 26-2. Register List



### 26.1.1 LCD Controller/Driver's divide resistors

The LCD drive voltage is generated either by external voltage divide resistors or internal voltage divide resistors.

The brightness can be controlled by connecting a variable resister between the $V_{CC}$ and V3 pins.

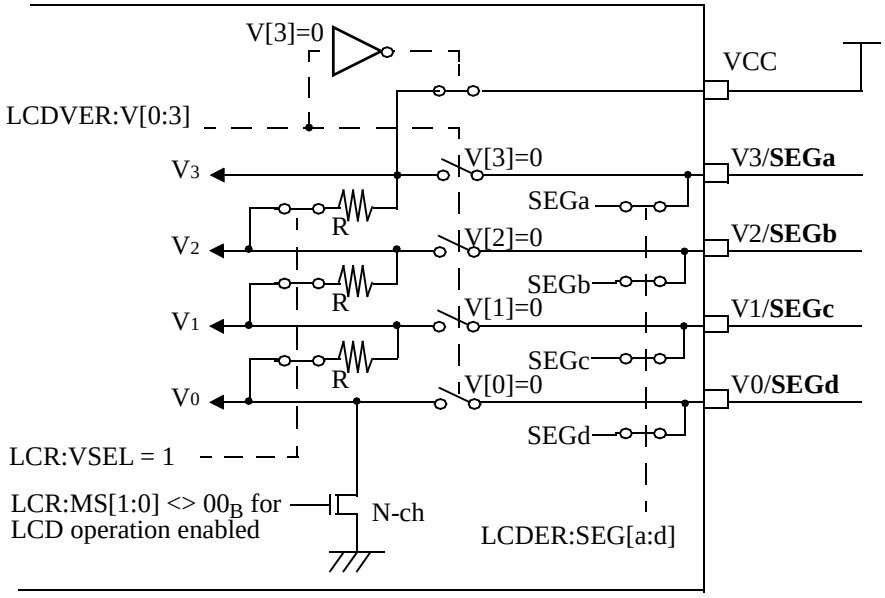
When internal divide resistors are used, the corresponding pins can be used as segment drivers.

#### Selection of internal or external divide resistors

Use the LCD drive power supply control bit (LCR:VSEL)

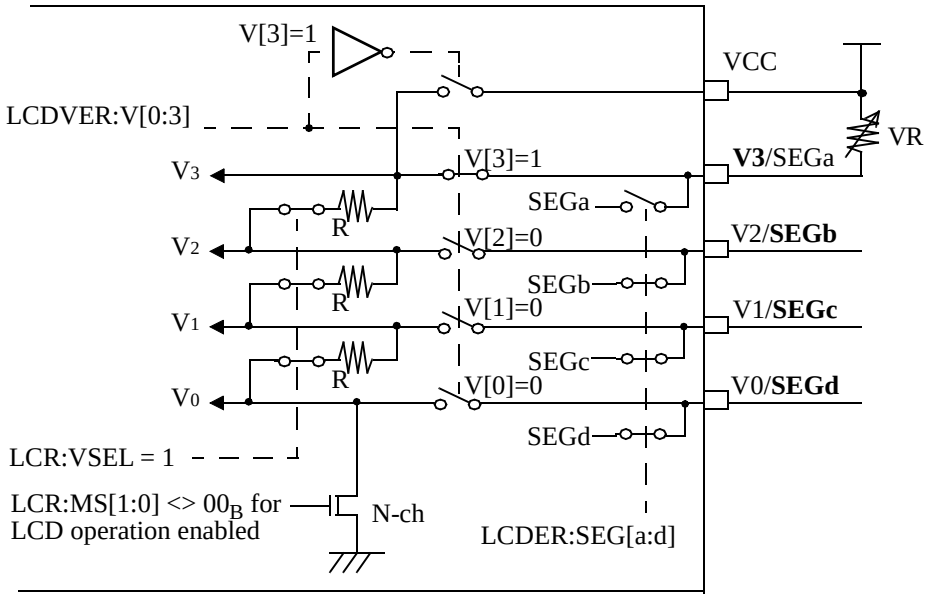
| Controlled operation | LCD drive power supply control bit (VSEL). |
|---|---|
| To use external divide resistors<br>(Internal divide resistors disconnected) | Set to "0". |
| To use internal divide resistors<br>(Internal divide resistors connected) | Set to "1". |

Figure 26-3. Using internal divide resistors:



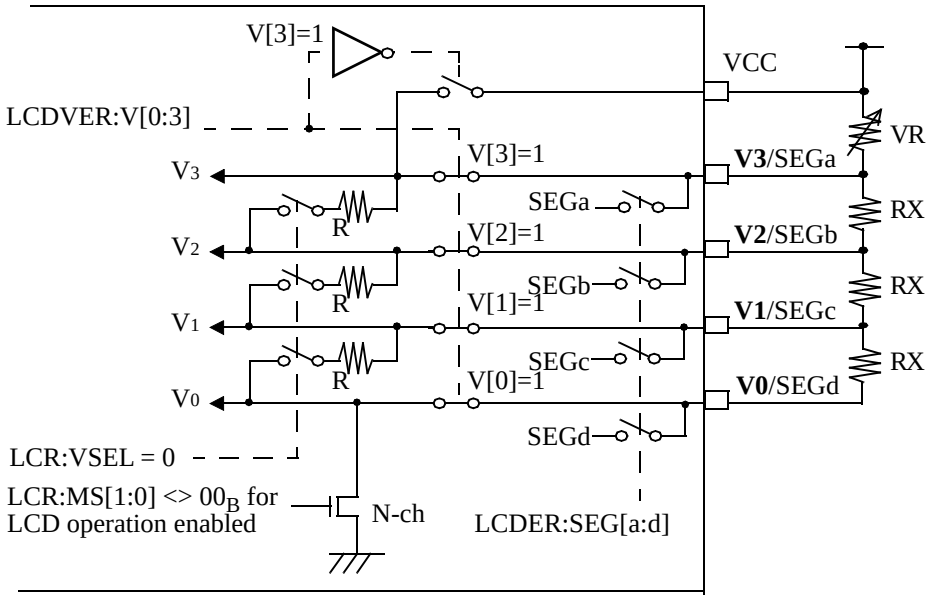

The LCD driving voltage is generated by internal divide resistors. When pins V0..V3 are disabled by setting the corresponding bit in LCDVER:V[3:0] to zero, additional segment pins become available. Since LCDVER:V3 = 0, the internal resistor chain is connected to VCC.

Figure 26-4. Using internal divide resistors and an external resistor for dimming:




Dimming with an external resistor when using the internal voltage divide resistors is possible by setting LCDVER:V3 = 1. The internal resistor chain is connected to pin V3. An external variable resistor connected to V3 is used for dimming. The setting of LCDER:SEGa has no effect.

Figure 26-5. Using external divide resistors:




The LCD driving voltage is generated by connecting external divide resistors to the LCD drive power supply pins (V0 to V3).

Each V pin must be enabled by setting LCDVER:V[3:0] = $1111_B$. For each enabled V pin, the corresponding segment enable bit LCDVER:SEG[x] has no effect.

To avoid the effect of internal divide resistors, the resistors must be disconnected by setting LCR:VSEL = 0.

### Usage of external divide resistors to shut off the current when LCD is deactivated

The V0 pin is internally connected to Vss (GND) via a transistor. For this reason, the current generated on deactivating LCD controller can be shut off by connecting external divide resistors to the V0 pin on the Vss side. To shut off the current, use the display mode select bit (LCR:MS[1:0] = “00”).

## 26.2 LCD Controller Registers

This section describes the registers related to the configuration of LCD Controller/Driver.

### 26.2.1 LCD Control register (LCR)

Figure 26-6 shows the bit configuration of the registers related to the LCD Controller/Driver.

Figure 26-6. Bit configuration of LCD Control Register (LCR)

LCR (LCD control register)

| | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| | CSS | LCEN | VSEL | BK | MS1 | MS0 | FP1 | FP0 | $00010000_B$ |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W: Reading/writing permitted

### bit7 CSS: Select the frame period generation clock

| | |
|---|---|
| 0 | Peripheral clock (CLKP1) |
| 1 | clock selected by LECR:CKSEL |

**Notes:**

- When the peripheral clock CLKP1 is selected and the MCU is in STOP mode or in Timer mode, the LCD stops operation, because CLKP1 is stopped.

### bit6 LCEN: Enable operation in Timer mode

| | |
|---|---|
| 0 | Disable LCD display in the Timer mode |
| 1 | Enable LCD display in the Timer mode |

**Notes:**

- To enable LCD display in Timer mode, the frame period generation clock select bit (CSS) must be also set to “1” and the selected clock (sub clock CLKSC or RC clock CLKSC) must be enabled.

### bit5 VSEL: Control LCD drive power supply

| | |
|---|---|
| 0 | Disconnect internal divider resistors |
| 1 | Connect internal divider resistors |

**Remark:**

To connect external divider resistors, the LCD drive power supply control bit (VSEL) must be set to “0”

When external divide resistors are selected, the corresponding segments can not be enabled. The setting of the corresponding LCDER register bits is ignored.

When internal divide resistors are selected the external voltage pins can not be connected. The setting of LCDVER:V[0:3] is ignored

### bit4 BK: Select blanking

| | |
|---|---|
| 0 | Enable LCD display |
| 1 | Disable (blank) LCD display |

### bit3-2: Select a display mode

Table 26-1. Function Settings

| MS1 | MS0 | Display mode |
|---|---|---|
| 0 | 0 | Deactivate LCD |
| 0 | 1 | 1/2 duty cycle output mode (Time division number: N=2, COM0-COM1) |
| 1 | 0 | 1/3 duty cycle output mode (Time division number: N=3, COM0-COM2) |
| 1 | 1 | 1/4 duty cycle output mode (Time division number: N=4, COM0-COM3) |

### Notes:

- If the display mode select bit (MS[1:0]) is set to “00”, the LCD Controller is disabled, i. e. a “L” level is output to the common/segment pins.

### bit1-0: Frame period

Table 26-2. Frame period

| FP1 | FP0 | When peripheral clock CLKP1 is selected: | When sub clock CLKSC or RC-Clock CLKRC is selected |
|---|---|---|---|
| 0 | 0 | $F_{CLKP1}/(2^{13} \times N)$ | $F_{CL}/(2^{8} \times N)$ |
| 0 | 1 | $F_{CLKP1}/(2^{15} \times N)$ | $F_{CL}/(2^{9} \times N)$ |
| 1 | 0 | $F_{CLKP1}/(2^{17} \times N)$ | $F_{CL}/(2^{10} \times N)$ |
| 1 | 1 | $F_{CLKP1}/(2^{19} \times N)$ | $F_{CL}/(2^{11} \times N)$ |

- $F_{CLKP1}$: Peripheral clock (CLKP1) frequency
- $F_{CL}$: Subclock (CLKSC) or RC-Clock (CLKRC) frequency
- N: Time division number (Selected with the display mode select bits, MS1 and MS0.)
- Select an appropriate value in accordance with the frame frequency of your LCD panel.

## 26.2.2 LCD Common Pin switching Register (LCDCMR)

Figure 26-7. Bit configuration of LCD Common Pin Switching register (LCDCMR)

LCDCMR (LCD Common Pin Switching Register)

| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | initial value |
|---|---|---|---|---|---|---|---|---|
| DTCH | - | - | - | COMEN3 | COMEN2 | COMEN1 | COMEN0 | $0XXX0000_B$ |
| (R/W) | (-) | (-) | (-) | (R/W) | (R/W) | (R/W) | (R/W) | |

R/W read and write permission
- write always 0, read value is undefined, read-modify-write is not affected

### bit7 DTCH: Bias control

| | |
|---|---|
| 0 | 1/3 bias |
| 1 | Reserved |

Set LCDCMR:DTCH to ’0’ to use the LCD Controller

### bit6-4 unused bits:

Write always 0 to these bits.

Read value is undefined.

Read-Modify-Write Instructions are not affected.

### bit3-0 COMEN[3:0]: Common driver enable

| | |
|---|---|
| 0 | corresponding common driver (COMx) is disabled |
| 1 | corresponding common driver (COMx) is enabled |

If COMENx is set to ’1’, the digital function (input/output) of the corresponding pin is disabled.

## 26.2.3 LCD Extended Control Register (LECR)

Figure 26-8. Bit configuration of LCD Extended Control register (LECR)

LECR (LCD extended Control Register)

| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | initial value |
|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | CKSEL | $XXXXXXX0_B$ |
| (-) | (-) | (-) | (-) | (-) | (-) | (-) | (R/W) | |

R/W read and write permission
- write always 0, read value is undefined, read-modify-write is not affected

### bit0 CKSEL: Clock selection bit

| | |
|---|---|
| 0 | use Sub clock CLKSC |
| 1 | use RC-Clock CLKRC |

If Sub clock or RC-clock shall be used as LCD clock, the LCR:CSS bit must be set.

Devices with sub clock: If the Clock setting shall be changed, the clock which is currently selected must be active for at lease 3 cycles if it was enabled at any time before. Otherwise the selected clock does not change to the new clock.

### bit7-1 unused bits

Write always 0 to this bits.

Read value is undefined.

Read-Modify-Write Instructions are not affected.

## 26.3 LCD Enable Registers (LCDER, LCDVER)

This section describes the LCD enable registers

### LCD Segment Enable Registers LCDER and Voltage Line Enable Register LCDVER

Figure 26-9. Bit configuration of LCD Segment Enable and Reference Voltage Enable registers

LCD SEG enable register 1/0

LCDER1/0
Initial value
00000000 00000000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEG15 | SEG14 | SEG13 | SEG12 | SEG11 | SEG10 | SEG9 | SEG8 | SEG7 | SEG6 | SEG5 | SEG4 | SEG3 | SEG2 | SEG1 | SEG0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

LCD SEG enable register 3/2

LCDER3/2
Initial value
00000000 00000000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEG31 | SEG30 | SEG29 | SEG28 | SEG27 | SEG26 | SEG25 | SEG24 | SEG23 | SEG22 | SEG21 | SEG20 | SEG19 | SEG18 | SEG17 | SEG16 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

LCD SEG enable register 5/4

LCDER5/4
Initial value
00000000 00000000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEG47 | SEG46 | SEG45 | SEG44 | SEG43 | SEG42 | SEG41 | SEG40 | SEG39 | SEG38 | SEG37 | SEG36 | SEG35 | SEG34 | SEG33 | SEG32 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

LCD SEG enable register 7/6

LCDER7/6
Initial value
00000000 00000000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEG63 | SEG62 | SEG61 | SEG60 | SEG59 | SEG58 | SEG57 | SEG56 | SEG55 | SEG54 | SEG53 | SEG52 | SEG51 | SEG50 | SEG49 | SEG48 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

LCD SEG enable register 9/8

LCDER9/8
Initial value
XXXXXXXX 00000000

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | - | - | SEG71 | SEG70 | SEG69 | SEG68 | SEG67 | SEG66 | SEG65 | SEG64 |
| - | - | - | - | - | - | - | - | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

LCD Voltage enable register

LCDVER
Initial value
XXXX0000

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | V3 | V2 | V1 | V0 |
| - | - | - | - | R/W | R/W | R/W | R/W |

0 - LCD output (SEGxx) or input (Vx) is disabled, other resource function can be used or pin is General purpose IO

1 - LCD output (SEGxx) or input (Vx) is enabled, other pin functions cannot be used (digital function is disabled)

'-' - reserved bit: write always '0', read value is undefined, RMW access is not affected

**Remark:**

When an internal divide resistor x is activated by LCDVER:Vx = 1, the setting in the LCD enable register LCDERy:SEGz of the segment sharing the pin of Vx is ignored and the segment is disabled. This is implemented to avoid a shortcut between a segment output and an internal voltage divider.

# 26.4 LCD Controller/Driver Display RAM

The display RAM is a 36 x 8 bit display memory area used to generate a segment output signal.

## 26.4.1 Memory area (VRAM) for setting display data

**VRAM0 (SEG1, SEG0): (Access: Byte)**

.

.

**VRAM(x/2 - 1) (SEGx-1, SEGx-2): (Access: Byte)**

**Notes:**

- x: Maximum number of segments

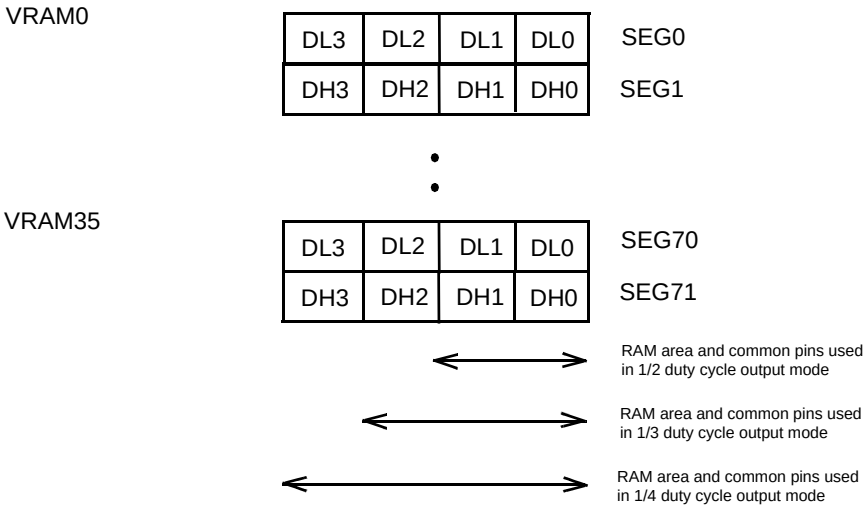
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | bit |
|---|---|---|---|---|---|---|---|---|
| DH3 | DH2 | DH1 | DH0 | DL3 | DL2 | DL1 | DL0 | |
| X | X | X | X | X | X | X | X | Initial value |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | Attribute |

**Notes:**

- Regardless of the operation of LCD Controller/Driver, RAM can be read and written any time.

### 26.4.2 Correspondence between VRAM and Common/Segment Pins

Figure 26-10. Correspondence between VRAM and Common/Segment Pins




## 26.5 Operation of LCD Controller/Driver

This chapter describes step by step how to use the LCD Controller/Driver

### 26.5.1 LCD Controller/Driver (LCDC) Operation

- Set values to the display data memory (VRAM) in advance.
- Make necessary settings to each register.
- When the frame period generation clock oscillates, LCD drive waveform is output through common/segment output pins (COM0 - COM3, SEG0 - SEGx-1).
- VRAM contents are automatically read in synchronization with common signals to be output through segment output pins.
  - If the bit is set to “1”, the selected waveform is output through the segment output pins.If the bit is set to “0”, non-selected waveform is output through the segment output pins.
- If the display mode is set to 1/2 duty cycle, non-selected waveform is output through the COM2 and COM3 pins. For the 1/3 duty cycle, the COM3 pin is used to output non-selected waveform.
- The output waveform is a 2-frame AC waveform in accordance with the duty cycle setting, and drives LCD.
- When MS[1:0] = “00” is used to deactivate LCD, a “L” level is output through both common and segment pins.
- If LCD operation is enabled in the Timer mode (LCEN=“1”), the LCD Controller/Driver continues to display even when MCU enters Timer mode.
  - Note that frame period generation clock signals must be supplied at this time
- LCD display can be blanked by selecting “blank” (BK=“1”) in blanking selection.
  - Note that non-selected waveform continues to be output.
- When LCD deactivation is selected (MS[1:0]=“00”) with the display mode, LCD stops operating.

## 26.5.2 Output waveform during LCD Controller/Driver operation (1/2 duty cycle)

Only COM0 and COM1 outputs are used for LCD display. Neither COM2 nor COM3 output is used.

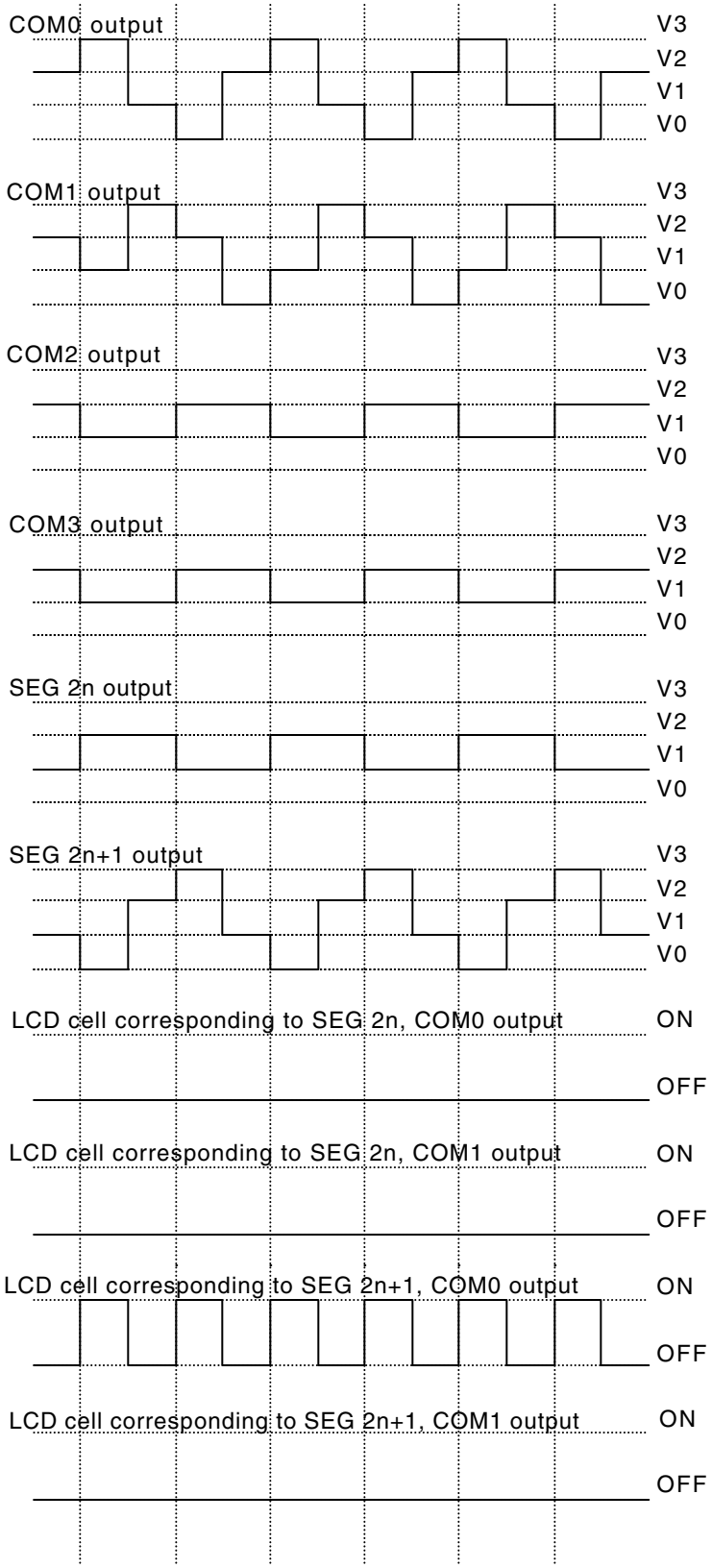
### Example of 1/3 Bias Output Waveform

LCD cells with the maximum voltage difference between common and segment outputs are lit.

Table 26-3. Example of Data Memory Contents for display

| Segment | Contents of data memory for display | | | |
|---|---|---|---|---|
| | COM3 output | COM2 output | COM1 output | COM0 output |
| SEG 2n output | - | - | 0 | 0 |
| SEG2n+1 output | - | - | 0 | 1 |

Figure 26-11. Output Waveform (1/2 duty cycle)



### 26.5.3 Output waveform during LCD controller/driver operation (1/3 duty cycle)

In the 1/3 duty cycle output mode, COM0, COM1 and COM2 outputs are used for LCD display. COM3 output is not used

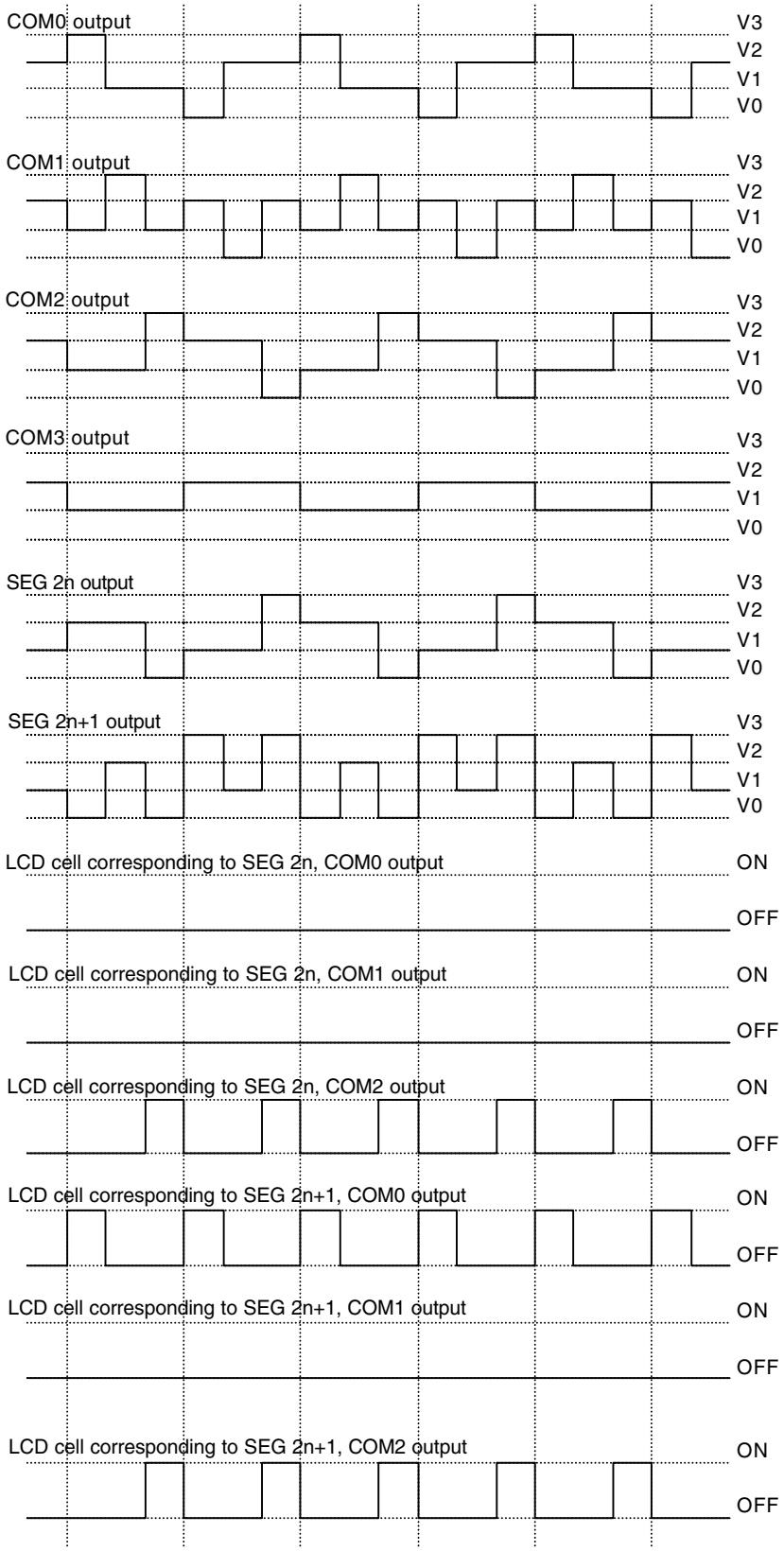
**Example of 1/3 Bias Output Waveform**

LCD cells with the maximum voltage difference between common and segment outputs are lit.

Table 26-4. Example of Data Memory Contents for display

| Segment | Contents of data memory for display | | | |
|---|---|---|---|---|
| | COM3 output | COM2 output | COM1 output | COM0 output |
| SEG 2n output | - | 1 | 0 | 0 |
| SEG2n+1 output | - | 1 | 0 | 1 |

Figure 26-12. Output Waveform (1/3 duty cycle)



## 26.5.4 Output waveform during LCD controller/driver operation (1/4 duty cycle)

In the 1/4 duty cycle output mode, COM0, COM1, COM2, and COM3 outputs are all used for LCD display

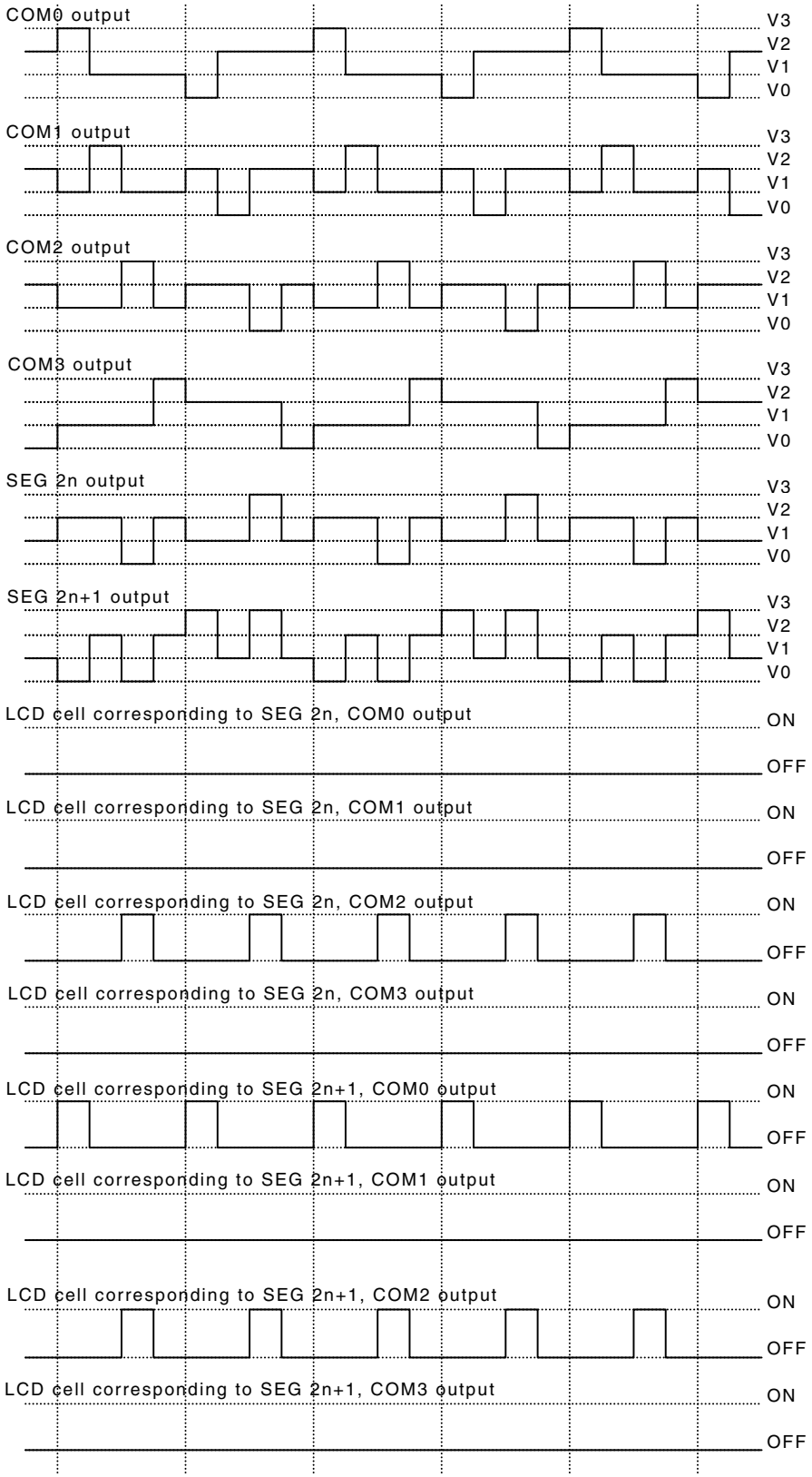
### Example of 1/3 Bias Output Waveform

LCD cells with the maximum voltage difference between common and segment outputs are lit.

Table 26-5. Example of Data Memory Contents for display

| Segment | Contents of data memory for display | | | |
|---|---|---|---|---|
| | COM3 output | COM2 output | COM1 output | COM0 output |
| SEG 2n output | 0 | 1 | 0 | 0 |
| SEG2n+1 output | 0 | 1 | 0 | 1 |

Figure 26-13. Output Waveform (1/4 duty cycle)



## 26.6 Cautions

Cautions covering the usage of LCD controller/driver

- Switching the frame period generation clocks:
  - Frame period generation clocks (LCR:CSS) can be switched even during LCD display.
  - However, switching may cause some screen flicker. To avoid such flicker, be sure to set the blanking select bit (LCR:BK) to “1” (blank display) before switching.
- Depending on your LCD, different external divider resistors are used. Use appropriate resistor values.
- When the display mode is set to 1/2 duty cycle, non-selected waveform is output through the COM2 and COM3 pins. For 1/3 duty cycle, the COM3 pin is used to output non-selected waveform.
- Inappropriate selection or setting of frame period generation clock (CSS), LCD drive power supply control (VSEL), duty cycle (MS[1:0]) and frame period (FP[1:0]) results in inappropriate LCD display
- By default, LCD display is disabled in MCU Timer modes. To enable LCD display in MCU Timer modes, use LCR:LCEN and make sure the required clock is operating.
- LCD display is disabled in MCU STOP mode
- Take care that the LCD display is not enabled if the required clock is not operating. Otherwise the LCD panel might be damaged by DC voltage. Refer to the following tables.

Table 26-6. LCD operation except in MCU Timer modes or MCU STOP mode

| CSS | CKSEL | Sub clock CLKSC running | RC clock CLKRC running | Operation for MS[1:0] ≠ "00" |
|---|---|---|---|---|
| 0 | X | X | X | LCD controller is running with Peripheral Clock CLKP1. |
| 1 | 0 | No | X | **Do not use this setting! DC voltage is applied to the LCD panel.** |
| 1 | 0 | Yes | X | LCD controller is running with Sub Clock CLKSC. |
| 1 | 1 | X | No | **Do not use this setting! DC voltage is applied to the LCD panel.** |
| 1 | 1 | X | Yes | LCD controller is running with RC Clock CLKRC. |

Table 26-7. LCD operation in MCU Timer modes

| CSS | LCEN | CKSEL | Sub clock CLKSC running | RC clock CLKRC running | Operation for MS[1:0] ≠ "00" |
|---|---|---|---|---|---|
| 0 | 0 | X | X | X | LCD display is deactivated. |
| 0 | 1 | X | X | X | **Do not use this setting! DC voltage is applied to the LCD panel.** |
| 1 | 0 | X | X | X | LCD display is deactivated. |
| 1 | 1 | 0 | No | X | **Do not use this setting! DC voltage is applied to the LCD panel.** |
| 1 | 1 | 0 | Yes | X | LCD controller is running with Sub Clock CLKSC. |
| 1 | 1 | 1 | X | No | **Do not use this setting! DC voltage is applied to the LCD panel.** |
| 1 | 1 | 1 | X | Yes | LCD controller is running with RC Clock CLKRC. |

For MS[1:0] = "00" and in MCU STOP mode, the LCD display is deactivated.

# 27. Stepper Motor Controller

This chapter explains the functions and operations of the stepper motor controller.


27.1 Outline of Stepper Motor Controller

27.2 Stepper Motor Controller Registers

27.3 PWM Control register (PWCn)

27.4 PWM Extended Control register (PWECn)

27.5 PWM1 and PWM2 Compare Registers (PWC1n, PWC2n)

27.6 PWM1 and PWM2 Selection registers (PWS1n, PWS2n)

27.7 Operation of Stepper Motor Controller

27.8 Notes on Using Stepper Motor Controller


## 27.1 Outline of Stepper Motor Controller

The Stepper Motor Controller consists of PWM pulse generators, motor drivers and selector logic circuits.

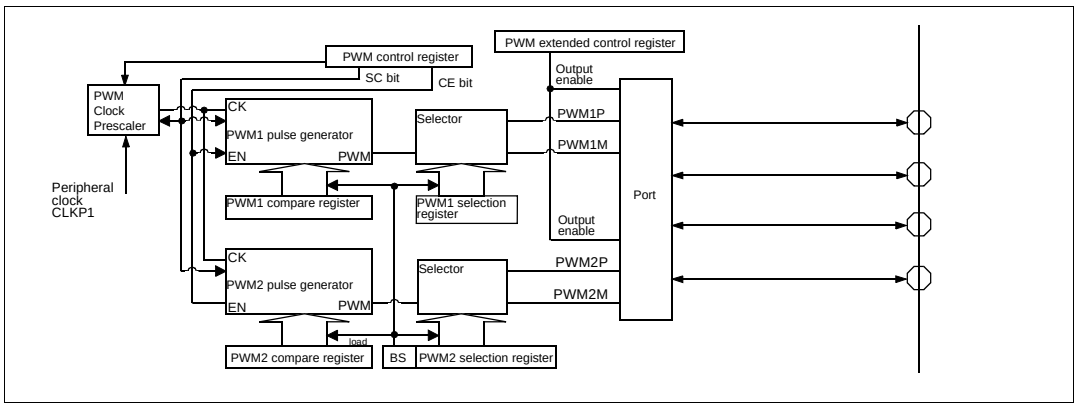
### Outline

- The four motor drivers have a high-output driving capability, and two motor coils can be connected directly to four pins. The motor rotation is designed to be controlled by a combination of the PWM pulse generators and selector logic circuits. The synchronization mechanism enables synchronous operation of two PWM pulse generators to operate surely.

### Block diagram of Stepper Motor Controller

Figure 27-1 shows a block diagram of the Stepper Motor Controller.

Figure 27-1. Block diagram of Stepper Motor Controller



## 27.2 Stepper Motor Controller Registers

There are seven registers for the Stepper Motor Controller:

- PWM Control Register
- PWM Extended Control Register
- PWM1 Compare Register
- PWM2 Compare Register
- PWM1 Selection Register
- PWM2 Selection Register

**Stepper Motor Controller registers**

Figure 27-2. Overview of the Stepper Motor Controller registers

PWM Control register (PWCn)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | P2 | P1 | P0 | CE | SC | - | - |
| - | R/W | R/W | R/W | R/W | R/W | - | - |

Initial value: - 0 0 0 0 0 - - $_B$

PWM Extended control register (PWECn)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | OE2 | OE1 |
| - | - | - | - | - | - | R/W | R/W |

Initial value: x x x x x x 0 0 $_B$

PWM1 Compare register (PWC1n)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | D9 | D8 |
| - | - | - | - | - | - | R/W | R/W |

Initial value: - - - - - - x x $_B$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value: x x x x x x x x $_B$

PWM2 Compare register (PWC2n)

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | D9 | D8 |
| - | - | - | - | - | - | R/W | R/W |

Initial value: - - - - - - x x $_B$

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value: x x x x x x x x $_B$

PWM1 Selection register (PWS1n)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | P2 | P1 | P0 | M2 | M1 | M0 |
| - | - | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value: - - 0 0 0 0 0 0 $_B$

PWM2 Selection register (PWS2n)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | BS | P2 | P1 | P0 | M2 | M1 | M0 |
| - | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value: x x 0 0 0 0 0 0 $_B$

# 27.3 PWM Control register (PWCn)

The PWM Control register (PWCn) starts and stops the Stepper Motor Controller, controls interrupts, and sets the external output pins.

## 27.3.1 PWM Control register (PWCn)

Figure 27-3. PWM Control Register (PWCn)

PWM Control register (PWCn)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | P2 | P1 | P0 | CE | SC | - | - |
| - | R/W | R/W | R/W | R/W | R/W | - | - |

Initial value:
- 0 0 0 0 0 - - $_B$

### [bit 7] Reserved bit

### [bit 6 to 4] P2, P1, P0: Operating clock select bits (bits to select operating clock)

The P2, P1 and P0 bits specify the clock input signal for the PWM pulse generator.

| P2 | P1 | P0 | Clock input | PWM cycle (at $f_{CLKP1}$ = 32 MHz) | | PWM cycle (at $f_{CLKP1}$ = 40 MHz) | |
|---|---|---|---|---|---|---|---|
| | | | | SC=0 | SC=1 | SC=0 | SC=1 |
| 0 | 0 | 0 | $f_{CLKP1}$ | 8.0 us | 32.0 us | 6.4 us | 25.6 us |
| 0 | 0 | 1 | $f_{CLKP1}/4$ | 32.0 us | 128.0 us | 25.6 us | 102.4 us |
| 0 | 1 | 0 | $f_{CLKP1}/5$ | 40.0 us | 160.0 us | 32.0 us | 128.0 us |
| 0 | 1 | 1 | $f_{CLKP1}/6$ | 48.0 us | 192.0 us | 38.4 us | 153.6 us |
| 1 | 0 | 0 | $f_{CLKP1}/8$ | 64.0 us | 256.0 us | 51.2 us | 204.8 us |
| 1 | 0 | 1 | $f_{CLKP1}/10$ | 80.0 us | 320.0 us | 64.0 us | 256.0 us |
| 1 | 1 | 0 | $f_{CLKP1}/12$ | 96.0 us | 384.0 us | 76.8 us | 307.2 us |
| 1 | 1 | 1 | $f_{CLKP1}/16$ | 128.0 us | 512.0 us | 102.4 us | 409.6 us |

**Note:**

After operation clock selection, set 1 in CE.

### [bit 3] CE: Count Enable bit

The CE bit enables operation of the PWM pulse generator. When "1" is set to the CE bit, the PWM pulse generator starts its operating. The PWM2 pulse generator starts after the PWM1 pulse generator starts in order to reduce the switching noise generated by the output driver.

When the CE bit is cleared to 0 during operation of the PWM pulse generator, the generator is initialized to stop operating.

### [bit 2] SC: 8/10 bits switching bit

When "1" is set to the SC bit, the PWM pulse generator operates at 10 bits. When "0" is set to the SC bit, the PWM pulse generator operates at 8 bits.

### [bit 1 to 0] Reserved bit

Always write "0". Read value is undefined. Read-modify-write is not affected.

## 27.4 PWM Extended Control register (PWECn)

The PWM Extended Control register (PWECn) enables the PWM output.

### 27.4.1 PWM Extended Control register (PWECn)

Figure 27-4. PWM Extended Control register (PWECn)

PWM Extended Control register (PWECn)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | OE2 | OE1 |
| - | - | - | - | - | - | R/W | R/W |

Initial value:
x x x x x x 0 0 $_B$

**[bits 7 to 2] Unused bits**

Writing to these bits has no effect.

**[bit 1 to 0] OE2, OE1: Output enable bits for PWM output**

These bits enable or disable the PWM output.

OE1 enables output of PWM1Pn/PWM1Mn

OE2 enables output of PWM2Pn/PWM2Mn

Table 27-1. Operation

| OE2, OE1 | Operation |
|---|---|
| 0 | PWM1Pn/PWM1Mn outputs disabled |
| 1 | PWM1Pn/PWM1Mn outputs enabled |

## 27.5 PWM1 and PWM2 Compare Registers (PWC1n, PWC2n)

The value of the two 8 (10) bits compare register of PWM1 and PWM2 (PWC1n, PWC2n) determine the width of the PWM pulse. The stored "$00_H$ ($000_H$)" value indicates that the PWM duty is 0%, and the stored "$FF_H$" ("$3FF_H$") value indicates that the PWM duty is 99.6% (99.9%).

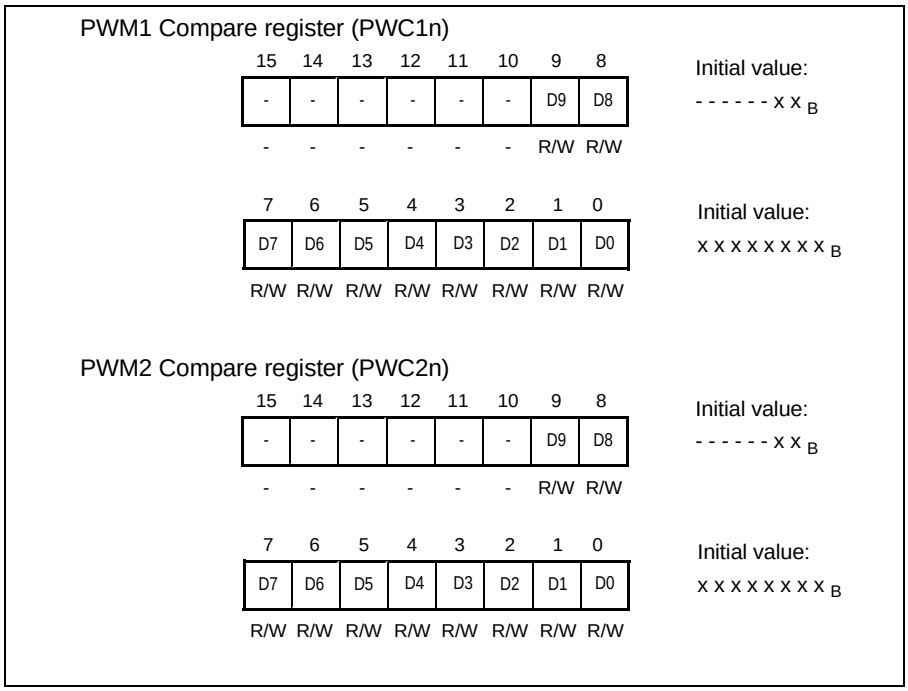
### 27.5.1 PWM1 and PWM2 Compare registers (PWC1n, PWC2n)

The PWM1 and PWM2 Compare registers can be accessed at any time, but the changed value is reflected in the pulse width at the end of the current PWM cycle after "1" is set to the BS bit of the PWM2 selection register.

When "0" is set to SC bit of the PWM Control register, and PWM performs 8-bit operation, the D9 and D8 bits are undefined value.

PWM1&2 Compare registers must be read or written 16-bit wide.

Figure 27-5. PWM1&2 compare registers




### [bit 15 to 10] Reserved bit

Always set the reserved bit to "0".

### [bits 9 to 0] D9 to D0: Compare data

These bits are used to set the PWM pulse width.

Figure 27-6. Relationship between the Compare Register setting value and PWM pulse width

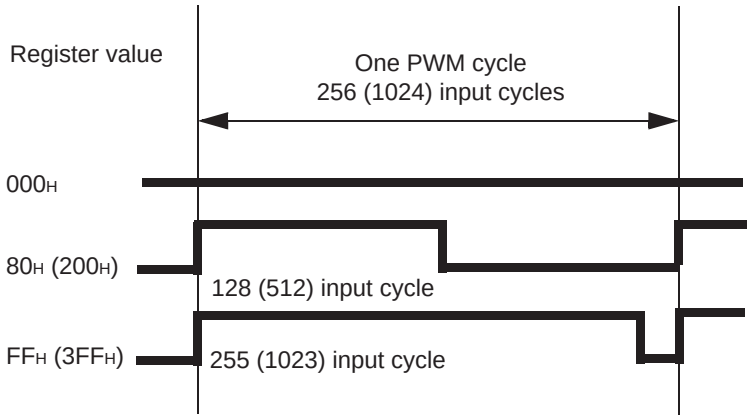
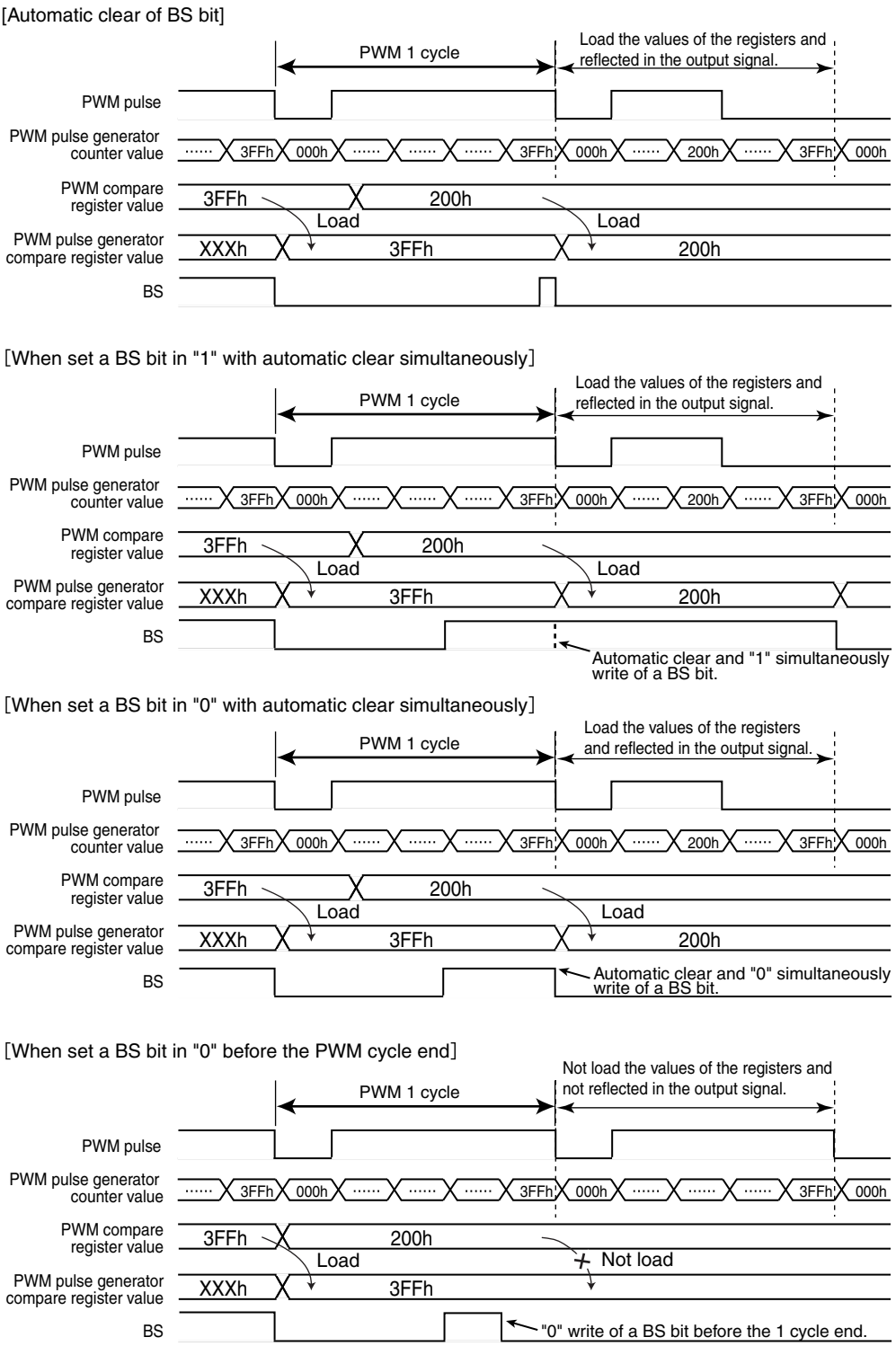


Figure 27-7. Load timing of PWM compare register value



# 27.6 PWM1 and PWM2 Selection registers (PWS1n, PWS2n)

The PWM1 and PWM2 Selection registers (PWS1n, PWS2n) determine whether to set the output of the external pin of the Stepper Motor Controller to "0", "1", PWM pulse or high impedance.

## 27.6.1 PWM1 and PWM2 Selection registers (PWS1n, PWS2n)

Figure 27-8. PWM1 and PWM2 selection registers (PWS1n, PWS2n)

PWM1 Selection register (PWS1n)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | - | P2 | P1 | P0 | M2 | M1 | M0 |
| - | - | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value: $0\,0\,0\,0\,0\,0\,0\,0_B$

PWM2 Selection register (PWS2n)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | BS | P2 | P1 | P0 | M2 | M1 | M0 |
| - | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

Initial value: $x\,0\,0\,0\,0\,0\,0\,0_B$

## 27.6.2 PWM1 Selection register (PWS1n)

### [bit 7 to 6] Reserved bits

Always set the reserved bit to "0".

### [bit 5 to 3] P2 to P0: Output select bits

These bits are used to select the output signal for SMCmPn.

### [bit 2 to 0] M2 to M0: Output select bits

These bits are used to select the output signal for SMCmMn.

The table below shows the relationship between the output levels and the select bits.

| P2 | P1 | P0 | PWMmPn | M2 | M1 | M0 | PWMmMn |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | L | 0 | 0 | 0 | L |
| 0 | 0 | 1 | H | 0 | 0 | 1 | H |
| 0 | 1 | X | PWM Pulse | 0 | 1 | X | PWM Pulse |
| 1 | X | X | High impedance | 1 | X | X | High impedance |

m = 1 to 2 (motor coils).

n = 0 to 5 (stepper motor channels).

## 27.6.3 PWM2 Selection register (PWS2n)

### [bit 7] Reserved bit:

Always set the reserved bit to "0".

### [bit 6] BS: Rewrite bit

The BS bit makes the setting for the PWM output match another setting. Until the BS bit is set, changes made to the two Compare registers and two Selection registers are not reflected in the output signal.

When "1" is set to the BS bit, the PWM pulse generator and the selector load the values of the registers at the end of the current PWM cycle. The BS bit is cleared to 0 automatically at the beginning of the next PWM cycle.

When "1" is set to the BS bit by software at the same time of an automatic clear, the BS bit is set to 1 (no change is made to the BS bit) and automatic clearing is cancelled.

When "0" is set to the BS bit by software at the same time of an automatic clear, the BS bit is set to 0, however the PWM pulse generator and the selector do not load the values of the registers at the end of the current PWM cycle.

**Note:**

When BS = 1, a read-modify-write instruction read this bit as "1" and writes the value "1" back to this bit.

When a BS bit is automatically cleared by the starting PWM cycle in between read and write, the read-modify-write instruction writes "1" to the cleared BS bit again.

Therefore, values of the registers are loaded in PWM pulse generator and selector even when the BS bit was not set to "1" by the end of the next PWM cycle.

### [bit 5 to 3] P2 to P0: Output select bits

These bits are used to select the output signal for SMCmPn.

### [bit 2 to 0] M2 to M0: Output select bits

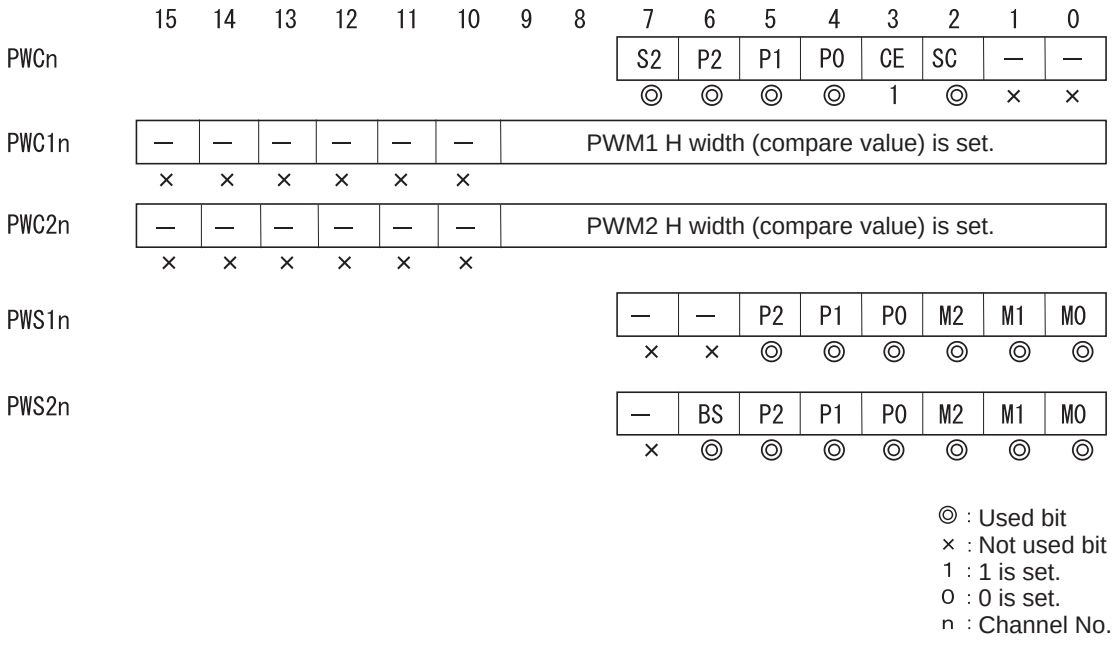
These bits are used to select the output signal for SMCmMn.

## 27.7 Operation of Stepper Motor Controller

Description of the Stepper Motor Controller operation.

### Setting Operation of Stepper Motor Controller

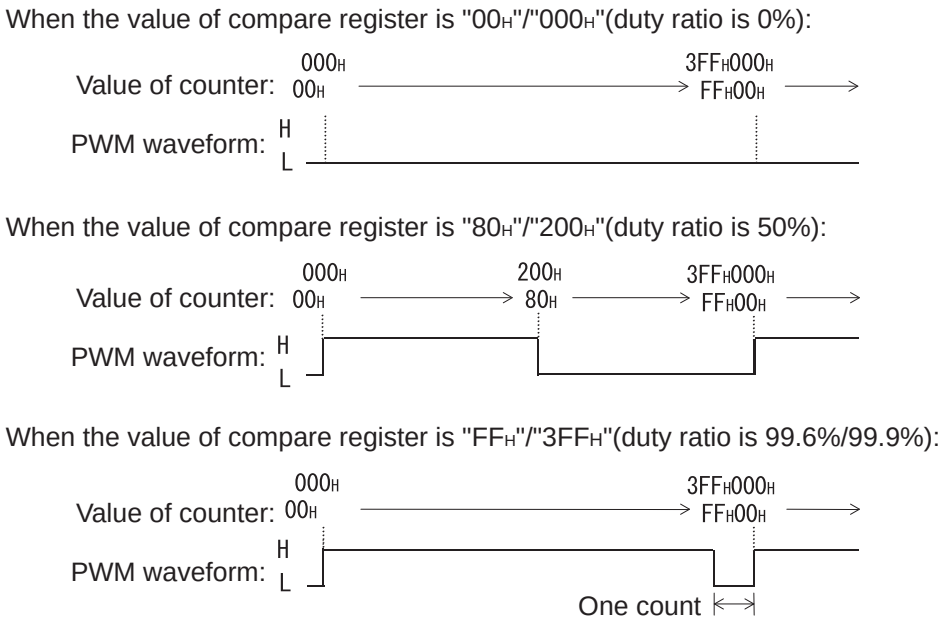
Figure 27-9. Setting of Stepper Motor Controller



### Operation of PWM-pulse generator

When the counter is started (PWC: CE = 1), the counter starts incrementing from $00_H$ on the selected count clock rising. The PWM output pulse wave remains "H" until the value of the counter matches the value set to PWM compare register, and then changes to and remains "L" until the value of the counter overflows ($FF_H$ --> $00_H$). Figure 27-10 shows the PWM waveform generated by the PWM generator.

Figure 27-10. Examples of PWM1 and PWM2 output waveforms




### Selection of motor drive signals

Motor drive signals that are output to each pin related to the Stepper Motor Controller can be selected among four types of signals for each pin by setting the PWM selection register. Table 27-2 lists the selection of the motor drive signals and the settings of PWM selection registers 1 and 2.

When these registers are set and "1" is written to the BS bit of the PWM selection register 2, the setting of these registers is enabled at the end of the current PWM cycle. The BS bit is to be cleared automatically to 0 at the beginning of the next PWM cycle. When "1" is written to the BS bit, and the BS bit is cleared to 0 simultaneously at the beginning of the next PWM cycle, "1" is written to the BS bit and BS bit clearing is cancelled.

Table 27-2. Motor drive signals selection and PWM Selection registers 1 and 2 setting

| P2, P1, P0 Bits | PWM2P Output<br>PWM1P Output | M2, M1, M0 Bits | PWM1M Output<br>PWM2M Output |
|---|---|---|---|
| 000B | L | $000_B$ | L |
| 001B | H | $001_B$ | H |
| 01XB | PWM Pulse | $01X_B$ | PWM Pulse |
| 1XXB | High impedance | $1XX_B$ | High impedance |

## 27.8 Notes on Using Stepper Motor Controller

The cautions when using the Stepper Motor Controller are described below.

### Cautions when Changing the PWM Setting

PWM Compare register 1 (PWC1x), PWM Compare register 2 (PWC2x), PWM Selection register 1 (PWS1x), and PWM Selection register 2 (PWS2x) can always be accessed. To change the setting of the PWM's "H" width or PWM output, write the setting values to these registers, then set the BS bit of PWM Selection register 2 to "1" (or do this simultaneously).

If the BS bit is set to "1", the new setting value will become effective at the end of the current PWM cycle, and the BS bit is automatically cleared.

If setting the BS bit to "1" and resetting the BS bit at the end of the PWM cycle both occur at the same time, writing "1" has priority and resetting the BS bit will be cancelled.

The PWM compare registers 1 and 2 (PWC1x, PWC2x) and the PWM selection registers 1 and 2 (PWS1x, PWS2x) can be accessed at any time. However, to change setting of the "H" width of PWM or to change the PWM output, "1" must be written to the BS bit of the PWM2 selection register after (or and at the same time) a setting is written to those registers (the PWM compare register 1 and 2 and the PWM selection register 1 and 2).

When "1" is set to the BS bit, the new setting is enabled at the end of the current PWM cycle and the BS bit is cleared automatically.

Also, when "1" is written to the BS bit and the BS bit is reset at the end of the PWM cycle simultaneously, "1" is written to the BS and resetting of the BS bit is cancelled.

# 28. Sound Generator

This chapter explains the functions and operations of the sound generator.

28.1 Outline of Sound Generator

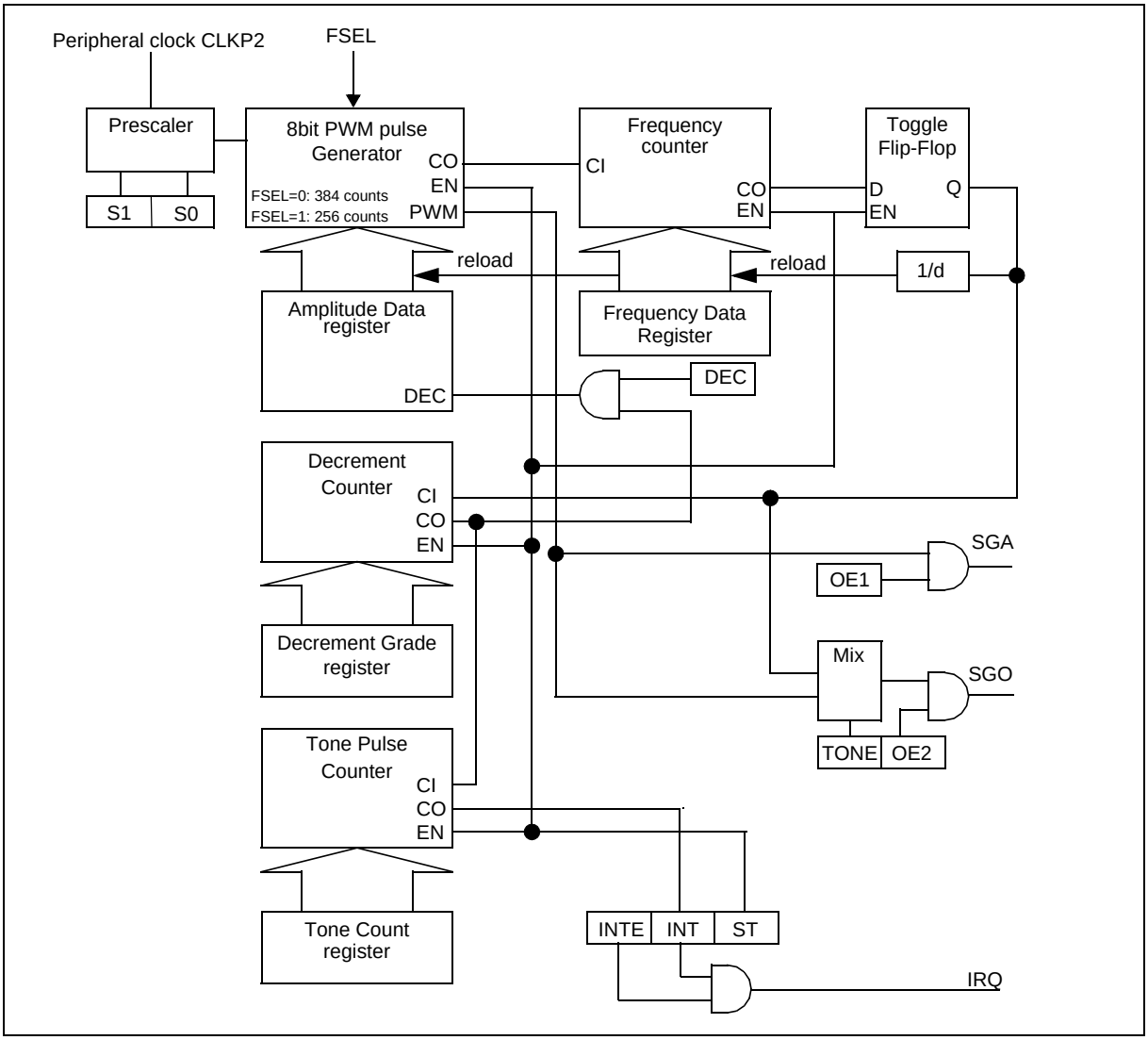
28.2 Sound Generator Registers

## 28.1 Outline of Sound Generator

The Sound Generator consists of the Sound Control Register, Frequency Data Register, Amplitude Data Register, Decrement Grade Register, Tone Count Register, PWM Pulse Generator, Frequency Counter, Decrement Counter and Tone Pulse Counter.

**Block diagram of Sound Generator**

Figure 28-1. Block diagram of Sound Generator



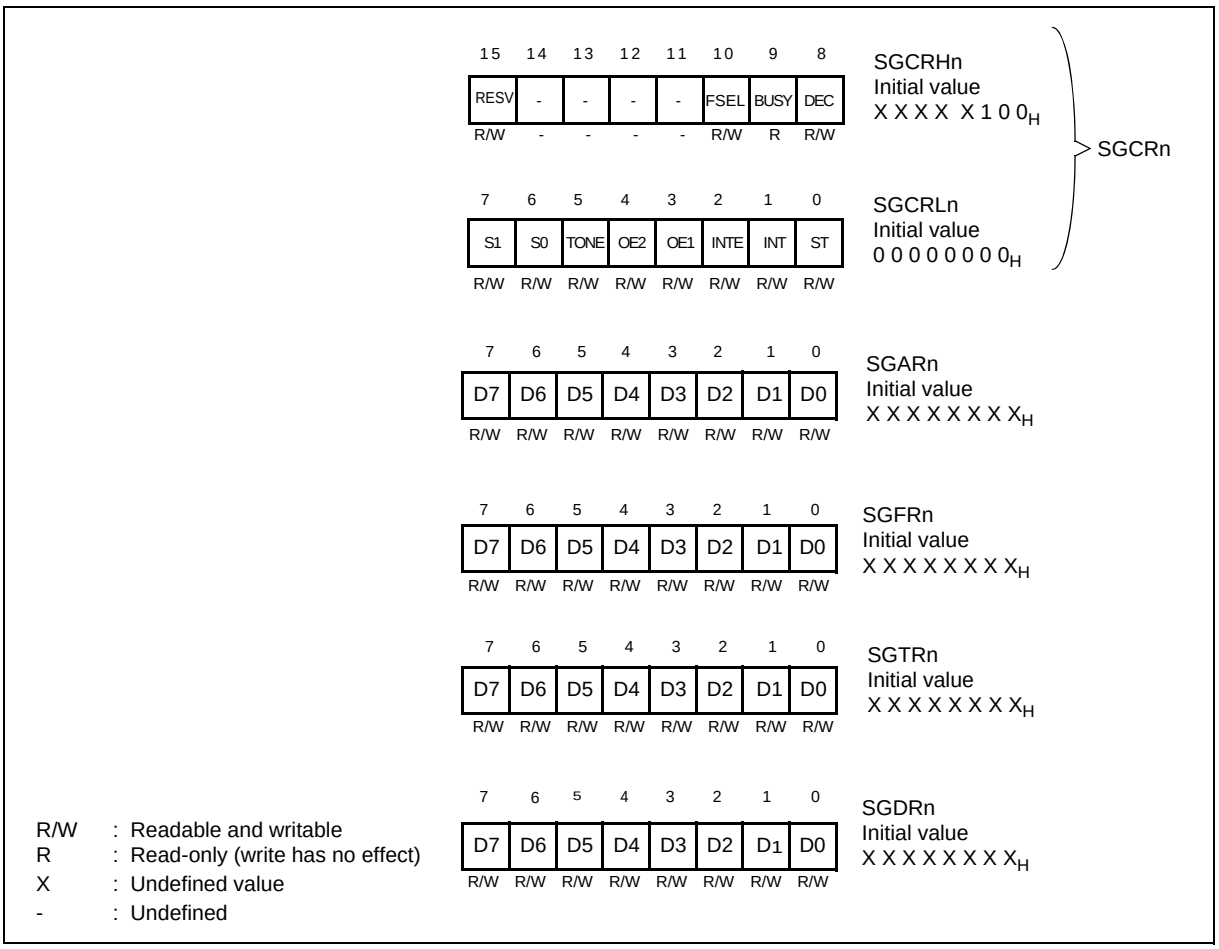
## 28.2 Sound Generator Registers

The Sound Generator has the following registers:

- Sound Control Register (SGCRn)
- Frequency Data Register (SGFRn)
- Amplitude Data Register (SGARn)
- Decrement Grade Register (SGDRn)
- Tone Count Register (SGTRn)

### Sound Generator registers

Figure 28-2. Overview of Sound Generator registers




**Note:**

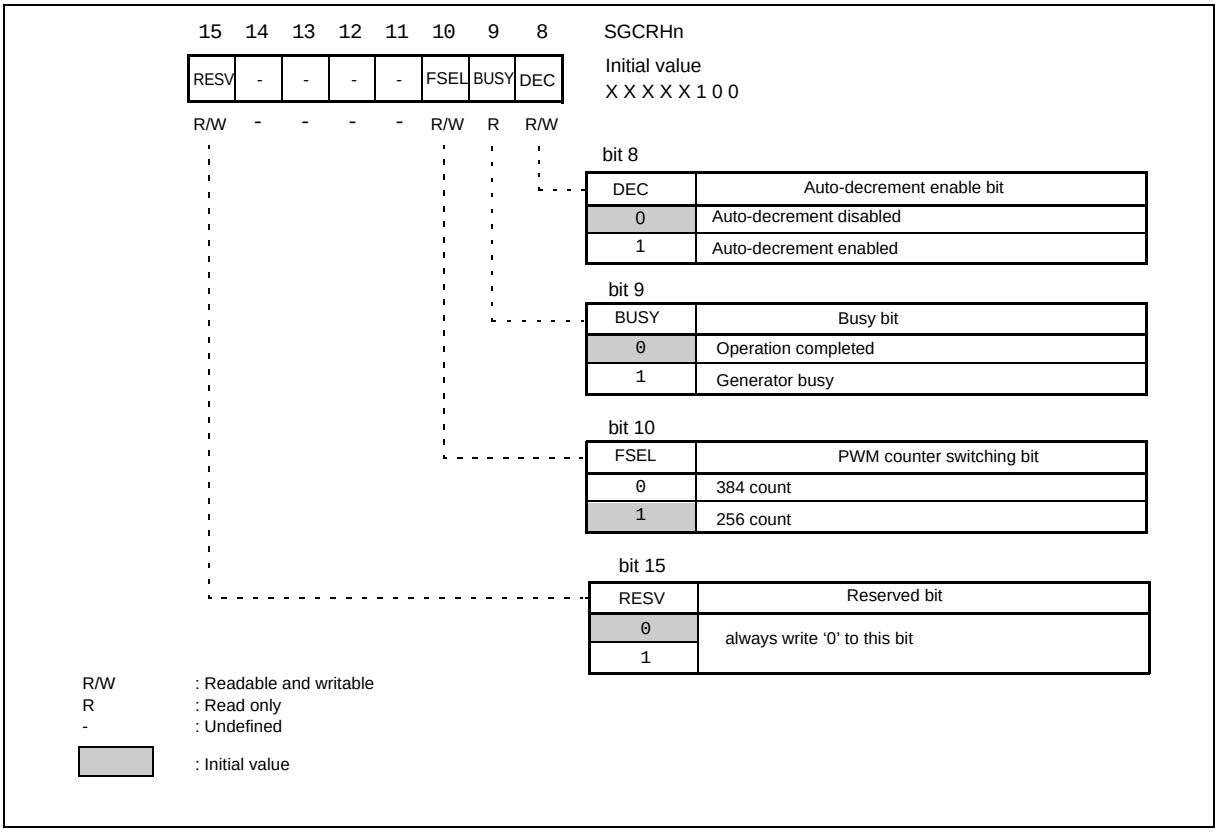
The Sound Generator Control Register SGCRn can be accessed as a 16-bit register or as two 8-bit registers SGCRHn and SGCRLn.

### 28.2.1 Sound Generator Control Register (SGCRn)

The Sound Control Register (SGCRn) controls the operation status of the Sound Generator by controlling interrupts and setting the external output pins.

## Sound Generator Control Register (SGCRHn)

Figure 28-3. Configuration of the Sound Generator Control Register (SGCRHn)



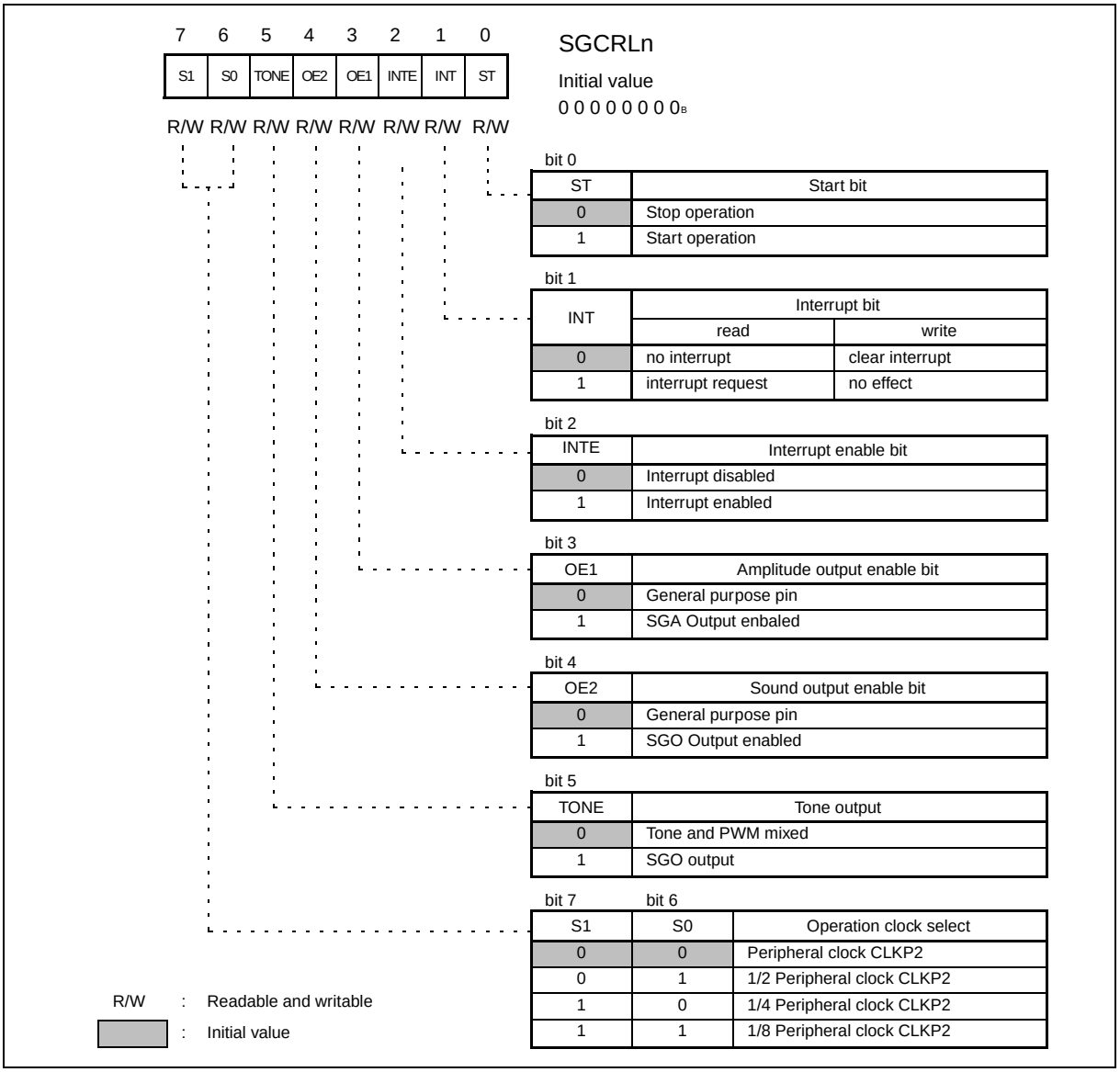

## Sound Generator Control Register (SGCRHn) contents

Table 28-1. Function of each bit of the Sound Generator Control Register (SGCRHn)

| Name | | Function |
|---|---|---|
| bit 15 | Reserved bit | This is a reserved bit. Always write "0" to this bit. Read value is undefined. Read-Modify-Write is not affected. |
| bit 14 to 11 | Undefined | |
| bit 10 | FSEL: PWM counter switching bit | This bit switches the count value of the PWM counter between 256 and 384. Stop operation when switching the count value. After FSEL is switched, set the amplitide data, frequency data, decrement grade and tone register again.<br>0: 384 count<br>1: 256 count |
| bit 9 | BUSY:<br>Busy bit | This bit indicates whether the Sound Generator is in operation. This bit is set to "1" upon the ST bit is set to "1". It is reset to "0" when the ST bit is reset to "0" and the operation is completed at the end of one tone cycle. Any write instructions performed on this bit has no effect. |
| bit 8 | DEC:<br>Auto-decrement enable bit | The DEC bit is prepared for an automatic degradation of the sound in conjunction with the Decrement Grade Register.<br>If this bit is set to "1", the stored value in the Amplitude Data Register is decremented by 1(one), every time when the Decrement counter counts the number of tone pulses from the toggle flip-flop specified by the Decrement Grade register. |

## Sound Generator Control Register (SGCRLn)

Figure 28-4. Configuration of the Sound Generator Control Register (SGCRLn)




bit 0

| ST | Start bit |
|---|---|
| 0 | Stop operation |
| 1 | Start operation |

bit 1

| INT | Interrupt bit | |
|---|---|---|
| | read | write |
| 0 | no interrupt | clear interrupt |
| 1 | interrupt request | no effect |

bit 2

| INTE | Interrupt enable bit |
|---|---|
| 0 | Interrupt disabled |
| 1 | Interrupt enabled |

bit 3

| OE1 | Amplitude output enable bit |
|---|---|
| 0 | General purpose pin |
| 1 | SGA Output enbaled |

bit 4

| OE2 | Sound output enable bit |
|---|---|
| 0 | General purpose pin |
| 1 | SGO Output enabled |

bit 5

| TONE | Tone output |
|---|---|
| 0 | Tone and PWM mixed |
| 1 | SGO output |

bit 7 bit 6

| S1 | S0 | Operation clock select |
|---|---|---|
| 0 | 0 | Peripheral clock CLKP2 |
| 0 | 1 | 1/2 Peripheral clock CLKP2 |
| 1 | 0 | 1/4 Peripheral clock CLKP2 |
| 1 | 1 | 1/8 Peripheral clock CLKP2 |

### Sound Generator Control Register (SGCRLn) contents

Table 28-2. Function of each bit of the Sound Generator Control Register (SGCRLn)

| Name | | Function |
|---|---|---|
| bit7, 6 | S1, S0:<br>Operation clock select bits | These bits specify the clock input signal for the Sound Generator.<br><br>S1 / S0 / Clock input<br>0 / 0 / Peripheral clock CLKP2<br>0 / 1 / 1/2 Peripheral clock CLKP2<br>1 / 0 / 1/4 Peripheral clock CLKP2<br>1 / 1 / 1/8 Peripheral clock CLKP2 |
| bit 5 | TONE:<br>Tone output bit | When this bit is set to "1", the SGO signal becomes a simple square-waveform (tone pulses) from the toggle flip-flop. Otherwise it is the mixed (AND logic) signal of the tone and PWM pulses. |
| bit 4 | OE2:<br>Sound output enable bit | When this bit is set to "1", the external pin is assigned as the SGO output. Otherwise the pin can be used as a general purpose IO. To enable the SGO output, the corresponding bit of the Port Direction register should also be set to "1". |
| bit 3 | OE1:<br>Amplitude output enable bit | When this bit is set to "1", the external pin is assigned as the SGA output. Otherwise the pin can be used as a general purpose IO. To enable the SGA output, the corresponding bit of the Port Direction register should also be set to "1".<br>The SGA signal is the PWM pulses from the PWM pulse generator representing the amplitude of the sound. |
| bit 2 | INTE:<br>Interrupt enable bit | This bit enables the interrupt signal of the Sound Generator. When this bit is "1" and the INT bit is set to "1", the Sound Generator signals an interrupt. |
| bit 1 | INT:<br>Interrupt bit | This bit is set to "1" when the Tone Pulse counter counts the number of the tone pulses specified by the Tone Count register and Decrement Grade register.<br>This bit is reset to "0" by writing "0". Writing "1" has no effect and Read-Modify-Write instructions always result in reading "1". |
| bit 0 | ST:<br>Start bit | This bit is for starting the operation of the Sound Generator. While this bit is "1", the Sound Generator perform its operation.<br>When this bit is reset to "0", the Sound Generator stops its operation at the end of the current tone cycle. The BUSY bit indicates whether the Sound Generator is fully stopped.<br>When this bit is changed from "0" to "1", the value of Frequency Data register, Amplitude Data register, Decrement Grade register, and Tone Count register is loaded into each counter. |

## 28.2.2 Amplitude Data Register (SGARn)

The Amplitude Data Register (SGARn) stores the reload value for the PWM pulse generator. The register value represents the amplitude of the sound. The register value is reloaded into the PWM pulse generator at falling edge of tone signal.

### Amplitude Data Register (SGARn)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

SGAR

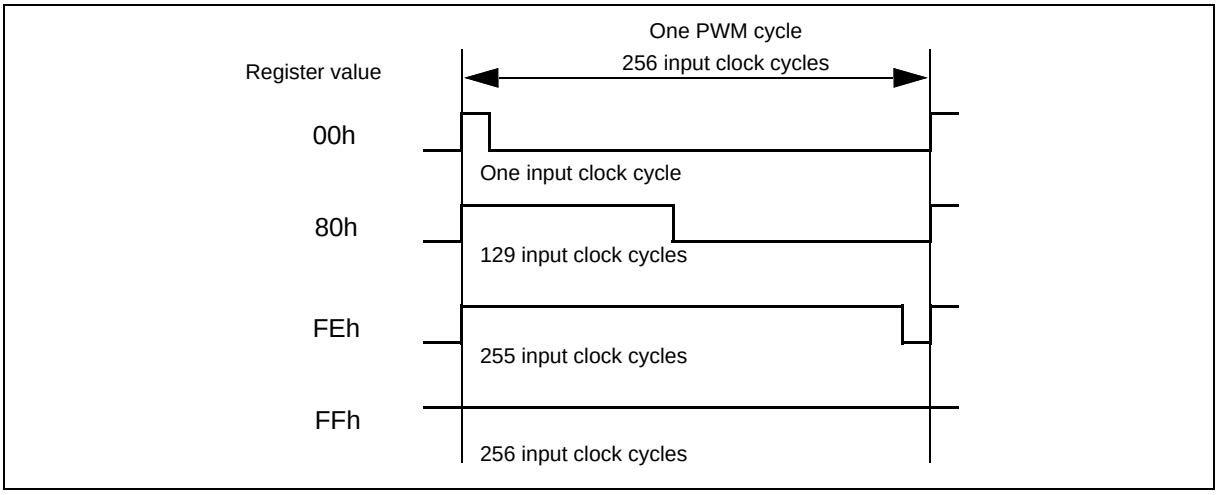
Initial value

X X X X X X X X B

When the DEC bit is "1" and the Decrement counter reaches its reload value, this register value is decremented by 1(one). When the register value reaches "00", further decrements are not performed. However the Sound Generator continues its operation until the ST bit is cleared.

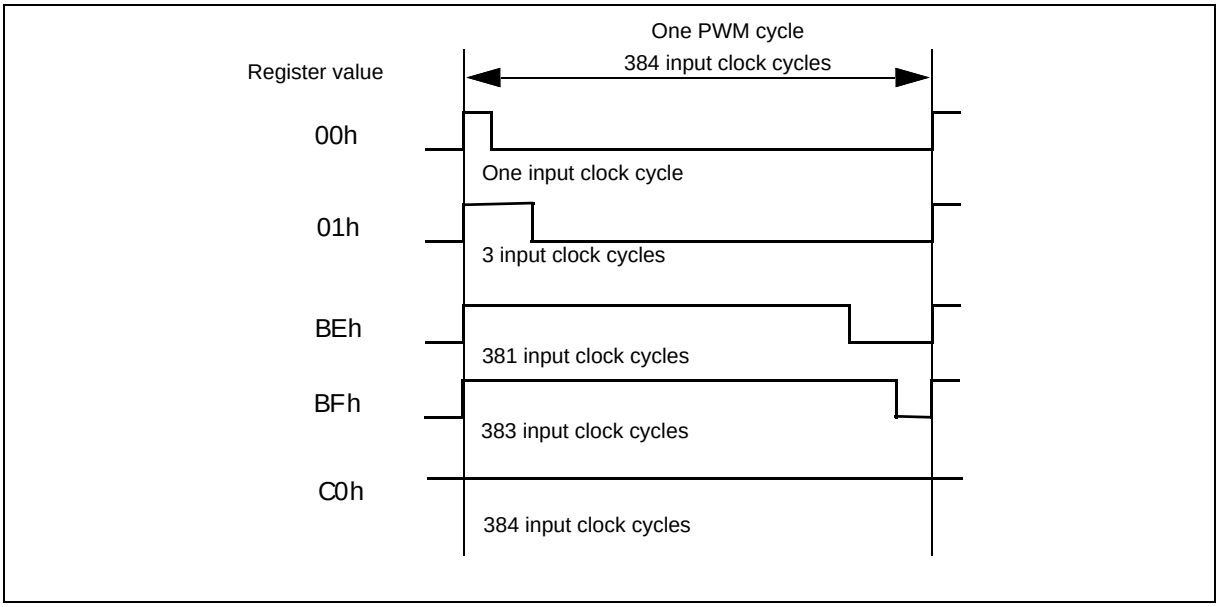
Figure 28-5 and Figure 28-6 show the relationship between the register value and the PWM pulse.

Figure 28-5. Relationship between SGARn value and PWM Pulse (FSEL=1)




When the register value is set to "FF", the PWM signal is always "1".

Figure 28-6. Relationship between SGARn value and PWM Pulse (FSEL=0)




**Note:**

When SGCRHn:FSEL is "0", the PWM signal is always set to "1" for $C0_H$ to $FF_H$.

When SGCRHn:DEC is "1", the PWM signal will fluctuate due to signal decrementation if the SGARn value is set to value greater than $C0_H$.

## 28.2.3 Frequency Data Register (SGFRn)

The Frequency Data Register (SGFRn) stores the reload value for the Frequency counter. The stored value represents the frequency of the sound (or the tone signal from the toggle flip-flop). The register value is reloaded into the counter at Frequency counter underflow and PWM pulse generator underflow.

The following figure shows the relationship between the tone signal and the register value.

**Frequency data register (SGFRn)**

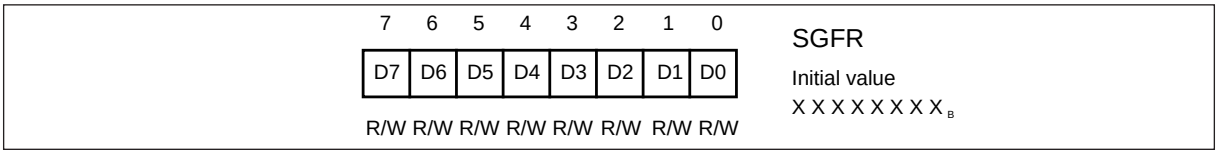
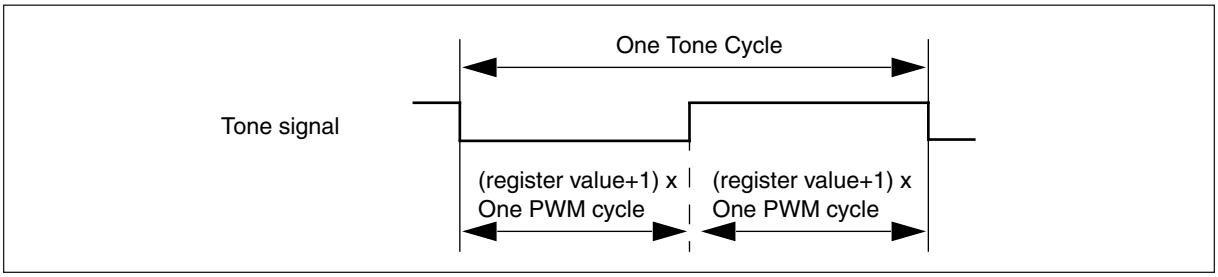



Figure 28-7 shows the relationship between a tone signal and a register value.

Figure 28-7. Relationship between tone signal and SGFRn value

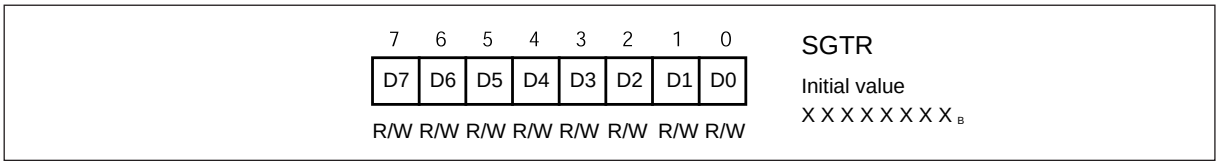



It should be noted that modifications of the register value during operation may alter the duty cycle of 50% depending on the timing of the modification.

## 28.2.4 Tone Count Register (SGTRn)

The Tone Count Register (SGTRn) stores the reload value for the Tone Pulse counter. The Tone Pulse counter accumulates the number of tone pulses (or number of decrement operations) and when it reaches the reload value it sets the INT bit, reducing the frequency of interrupts by the number of programmed tones. The register value is reloaded into the counter at Tone Pulse counter underflow, Decrement counter underflow, and falling edge of tone signal.

**Tone Count Register (SGTRn)**




The count input of the Tone Pulse counter is connected to the carry-out signal from the Decrement counter. When the Tone count register is set to "00", the Tone Pulse counter sets the INT bit every carry-out from the Decrement counter. Thus the number of accumulated tone pulses is:

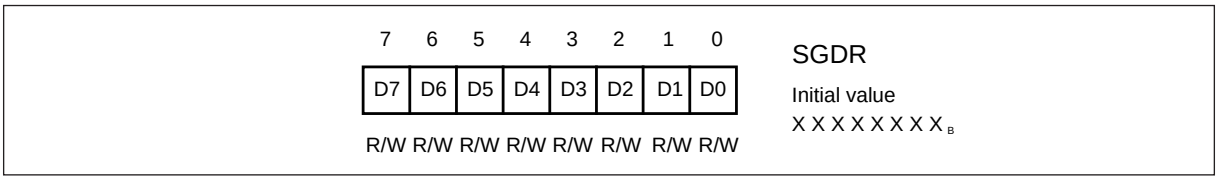
((Decrement Grade register) + 1) x ((Tone Count register) + 1)

i.e. when both registers are set to "00", the INT bit is set every tone cycle.

## 28.2.5 Decrement Grade Register (SGDRn)

The Decrement Grade Register (SGDRn) stores the reload value for the Decrement counter. It automatically decrements the reload value of the Amplitude Data register to reduce the frequency of interrupts. The register value is reloaded into the counter at Decrement counter underflow and falling edge of tone signal.

**Decrement Grade Register (SGDRn)**




When the DEC bit is "1" and the Decrement counter counts the number of tone pulses up to the reload value, the stored value in the Amplitude Data register is decremented by 1(one) at the end of the tone cycle.

This operation realizes automatic degradation of the sound with fewer number of CPU interventions.

It should be noted that the number of the tone pulses specified by this register equals to "register value + 1". When the Decrement Grade register is set to "00", the decrement operation is performed every tone cycle.

# 29. USB Function

This chapter explains the functions and operation of the USB Function.

29.1 USB Function Overview

29.2 USB Function Block Diagram

29.3 USB Function Register Description

29.4 Description of the USB function operation

## 29.1 USB Function Overview

The USB Function is an interface that supports the USB (Universal Serial Bus) 1.1 communication protocol. Only Full speed transfer (12 Mbps) is supported.

### Features of USB Function

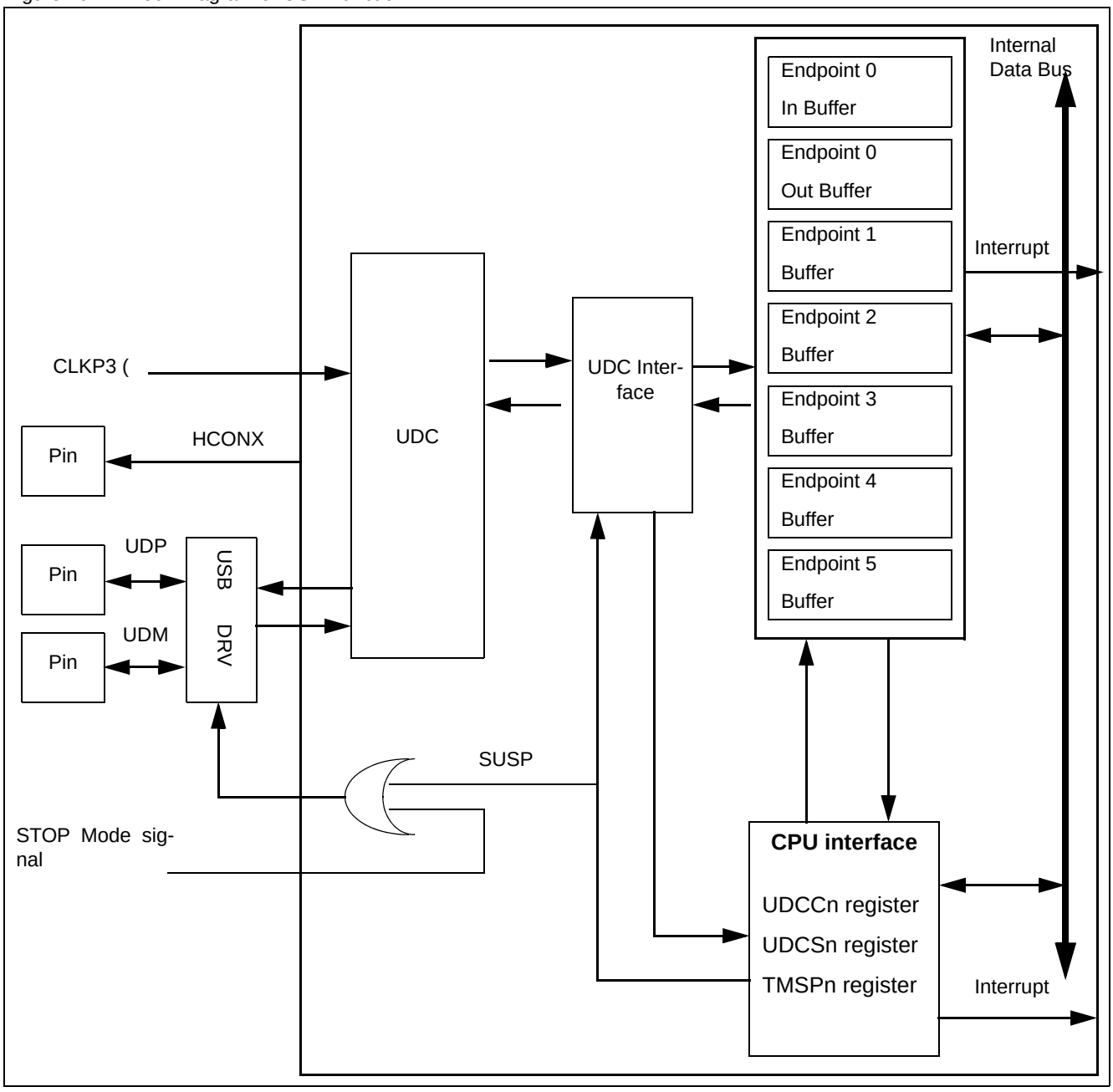
- Full speed (12 Mbps) is supported, corresponds to USB Full Speed.
- Automatically responds to the Device Status commands..
- Bit stripping, bit stuffing and automatic generation and check of CRC5 and CRC16.
- Toggle check by data synchronization bit.
- Automatic response to all standard commands except GetDescriptor, SetDescriptor and SynchFrame commands. These commands can be processed the same way as the class vendor commands.
- The class vendor commands can be received as data and handled by firmware.
- Supports up to six endpoints (endpoint 0 is fixed for control transfers).
- Two built-in transfer data buffers for each endpoint. Each of the two built-in buffers dedicated to IN and OUT, respectively for endpoint 0.
- Supports automatic transfer mode for DMA data transfer (except buffers for endpoint 0).

## 29.2 USB Function Block Diagram

Description of the USB Function.

### Block Diagram of USB Function

Figure 29-1. Block Diagram of USB Function



## 29.3 USB Function Register Description

Description of the USB function registers.

### Register List of the USB function

Table 29-1. Register List

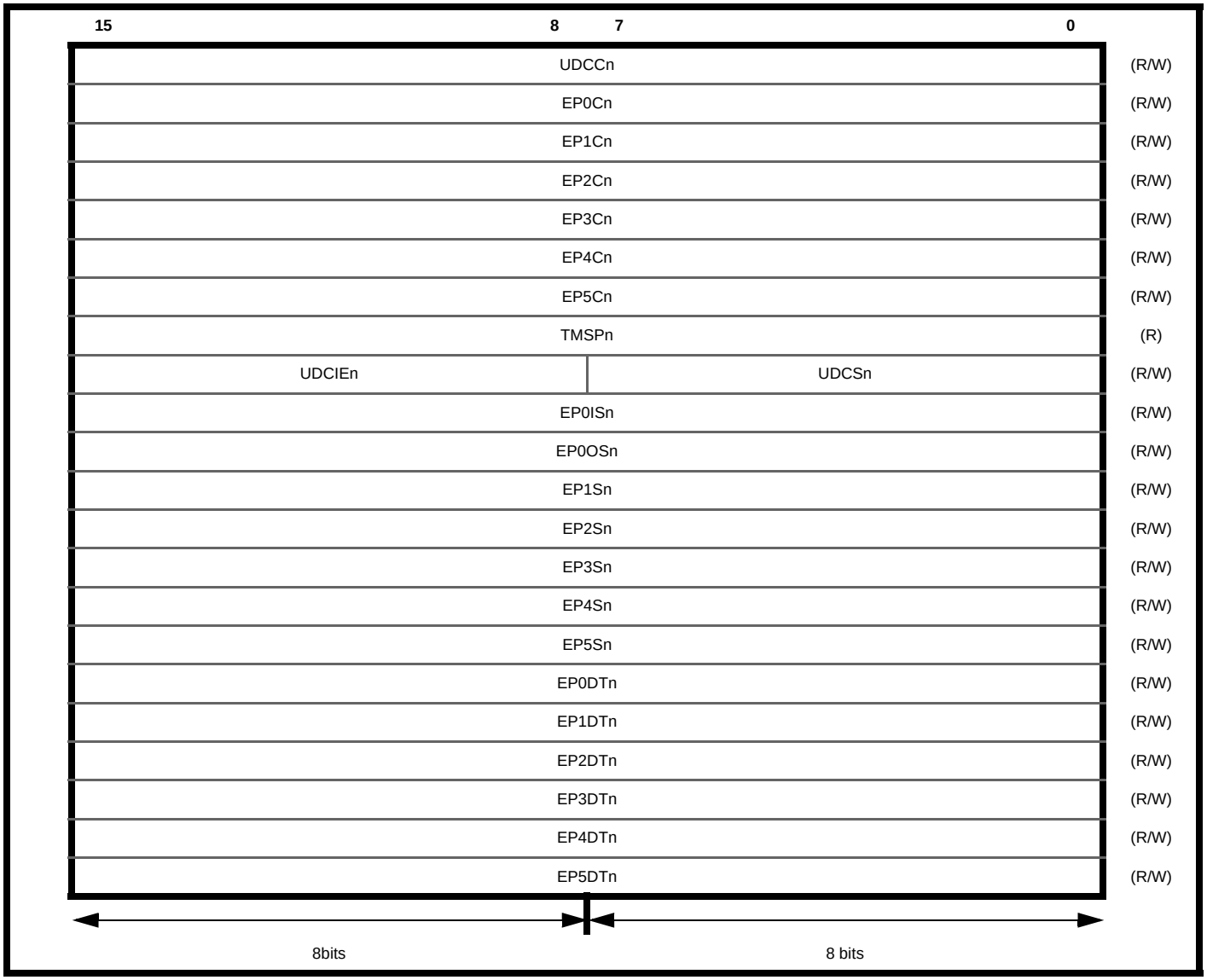


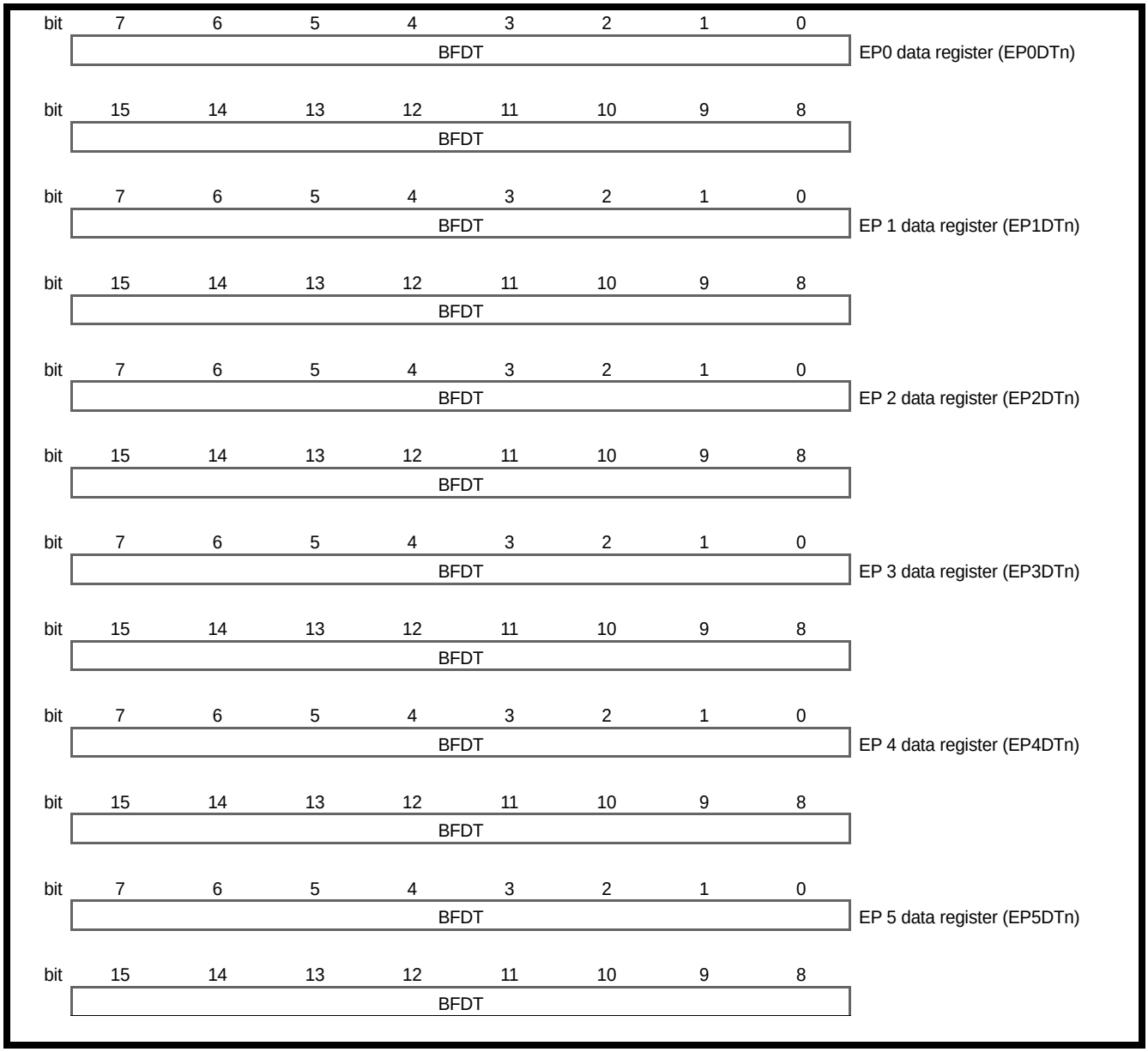
Table 29-2. Registers of USB function

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RST | RESUM | HCONX | USTP | Reserved | Reserved | RFBK | PWC | UDC control register (UDCCn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | PKS0 | | | | | | | EP0 control register (EP0Cn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | - | - | - | - | Reserved | Reserved | STAL | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | PKS 1 | | | | | | | | EP1 control register (EP1Cn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | EPEN | TYPE | | DIR | DMAE | NULE | STAL | PKS1 | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | PKS 2 | | | | | | | EP2 control register (EP2Cn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | EPEN | TYPE | | DIR | DMAE | NULE | STAL | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | PKS 3 | | | | | | | EP3 control register (EP3Cn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | EPEN | TYPE | | DIR | DMAE | NULE | STAL | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | PKS 4 | | | | | | | EP4 control register (EP4Cn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | EPEN | TYPE | | DIR | DMAE | NULE | STAL | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | PKS 5 | | | | | | | EP5 control register (EP5Cn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | EPEN | TYPE | | DIR | DMAE | NULE | STAL | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | TMSP | | | | | | | | Time stamp register (TMSPn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | - | - | - | - | - | TMSP | | | |

Table 29-2. Registers of USB function

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | SUSP | SOF | BRST | WKUP | SETP | CONF | UDC status register (UDCSn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | Reserved | Reserved | SUSPIE | SOFIE | BRSTIE | WKUPIE | CONFN | CONFIE | UDC interruption permission register (UDCIEn) |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | - | - | - | - | - | - | - | - | EP0I status register (EP0ISn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQIIE | - | - | - | DRQI | - | - | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | SIZE | | | | | | | EP0O status register (EP0OSn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQOIE | SPKIE | - | - | DRQO | SPK | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | SIZE | | | | | | | | EP1 status register (EP1Sn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQIE | SPKIE | Reserved | BUSY | DRQ | SPK | SIZE | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | SIZE | | | | | | | EP2 status register (EP2Sn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQIE | SPKIE | Reserved | BUSY | DRQ | SPK | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | SIZE | | | | | | | EP3 status register (EP3Sn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQIE | SPKIE | Reserved | BUSY | DRQ | SPK | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | SIZE | | | | | | | EP4 status register (EP4Sn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQIE | SPKIE | Reserved | BUSY | DRQ | SPK | Reserved | |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | Reserved | SIZE | | | | | | | EP5 status register (EP5Sn) |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| | BFINI | DRQIE | SPKIE | Reserved | BUSY | DRQ | SPK | Reserved | |

Table 29-2. Registers of USB function

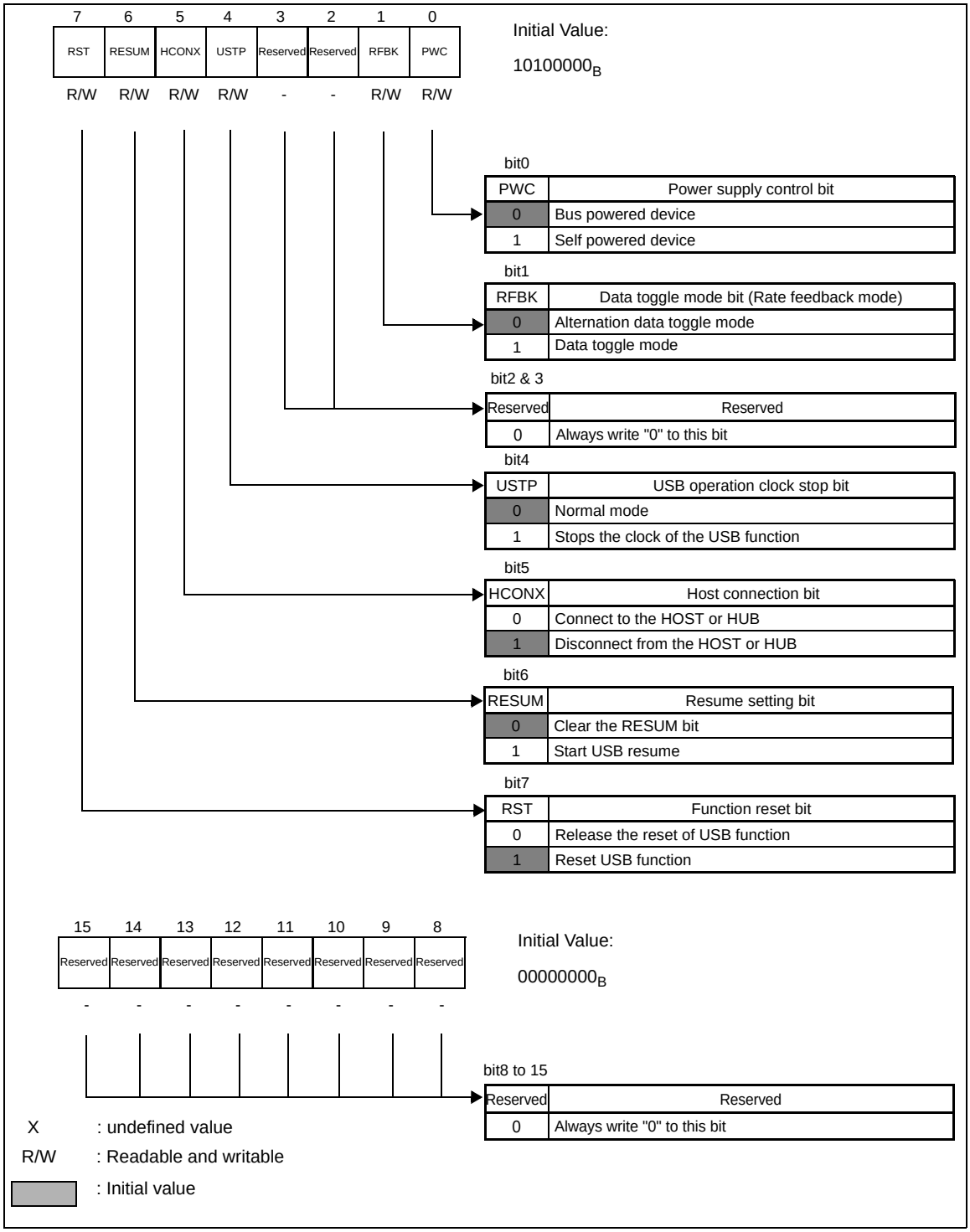


## 29.3.1 UDC Control Register (UDCCn)

UDC control register (UDCCn) controls the UDC (USB Device Controller Core) circuit.

### UDC Control Register (UDCCn)

Figure 29-2. UDC Control Register (UDCCn)



The following describes the function of each bit in the UDC control register (UDCC).

| | Bit names | Function |
|---|---|---|
| Bit7 | RST | • Reset the whole USB function.<br>• Apply a reset to the USB Function by setting the RST bit to "1" when connecting a cable to the HOST.<br>• As the initial value of this bit is "1" i.e the USB function is reset, write "0" to this bit to release the USB function from its reset status.<br>• RST bit initializes the corresponding bits of the timestamp register, UDC status register, and interrupt enable register at once. In addition, since it initialises BFINI bit of EP0I, EP0O, and EP1 to EP5 status registers at the same time, first clear the RST bit (which does not clear the BFINI bits) after initialization and then clear the BFINI bit of the endpoint used. |
| Bit6 | RESUM | • When the USB function is in remote Wake-up enabled status (or DEVICE_REMOTE_WAKEUP feature is set with the SET_FEATURE command by the host) and is in suspend state, the resume operation is started by writing 1 to the RESUM bit. For resume of the USB function, please set the RESUM bit to "1" and then clear the RESUM bit to "0". |
| Bit5 | HCONX | • Controls a switch between an external pull-up resistor and the USB data line to make the HOST or HUB recognize that USB function is connected.<br>• Even if the external pull-up resistor is ON and the Host or Hub recognizes that USB function is connected, USB Function module ignores the bus reset and the command on the USB bus when the HCONX bit is set to "1". |
| Bit4 | USTP | • Stops the clock to the USB function.<br>• Setting the USTP bit to "1" stops the clock to the USB module to reduce power consumption.<br>• When you set USTP bit to "1" but are not going to select STOP mode after it, at first set RST bit to "1" and wait for 3 cycles to make sure that the USB module is reset, and then set USTP bit to "1". Clearing the USTP bit and the RST bit at once is OK. |
| Bit3 | Reserved | • Please write "0". The bit always reads "0". |
| Bit2 | Reserved | • Please write "0". The bit always reads "0". |
| Bit1 | RFBK | • This bits selects the Data Toggle mode.<br>• If set to "0" it toggles the data PID between DATA0 and DATA1 only when the transfer has been successfully completed.<br>• If set to "1" it toggles the data PID between DATA0 and DATA1 unconditionally. |
| Bit0 | PWC | • This bit specifies the power supply mode (self or bus powered) for the USB function. (The setting of the PWC bit is reflected in the standard usb command: GetStatus command). |

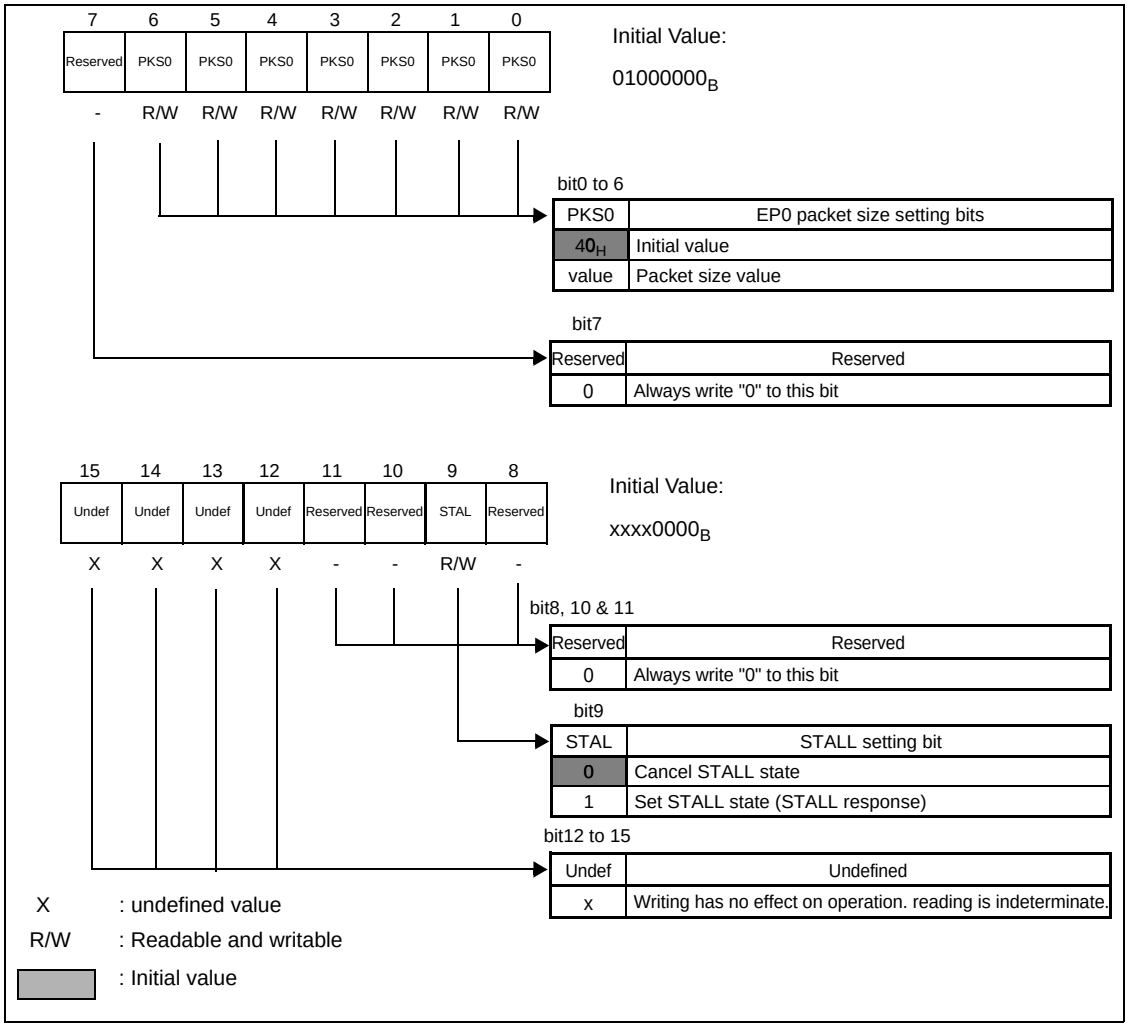
**Note:** Bits of the UDC control register (UDCCn) must not be changed when RST ='1' except RESUM and USTP bit. Do not change them while the USB is operating. Set and reset RESUM bit only when USB is in suspend mode and Remote Wake-up is allowed (DEVICE_REMOTE_WAKEUP bit is set by the SET_FEATURE command from the Host). When stop mode is selected, set USTP bit before entering the stop mode. After the stop mode is cancelled, clear the SUSP bit to "0" and then clear the USTP bit to "0".

## 29.3.2 EP0 Control Register (EP0Cn)

EP0 control register (EP0Cn) is used to control the operation of endpoint 0.

### EP0 Control Register (EP0Cn)

Figure 29-3. EP0 Control Register (EP0Cn)




**Note:**

Ensure that you must set the EP0 control register (EP0Cn), except bit 9 STAL, when both RST of the UDC control register (UDCCn) and BFINI bit of the EP0I/EP0O status register (EP0ISn / EP0OSn) are "1" . Changing EP0Cn is not allowed while the USB is operating.

| Bit names | | Function |
|---|---|---|
| Bit15-12 | Undefined | • Writing has no effect on operation. Reading is indeterminate. |
| Bit11-10 | Reserved | • Always write "0". These bits always read "0". |
| Bit9 | STAL | • Setting the STAL bit to "1" can put endpoint 0 in STALL state (STALL response).<br>• The STALL response is sent to the host till the STAL bit is set to "1". The USB Function returns from STALL state when it receives a normal SETUP packet after the STAL bit is cleared by writing "0" to it. |
| Bit8-7 | Reserved | • Always write "0". These bits always read "0". |
| Bit6-0 | PKS0 | • It specifies the maximum number of bytes that can be transferred per packet. The maximum number of transfer bytes per packet for EndPoint0 is 64 bytes, which is a common setting for both IN and OUT packet transfers. Example: "$08_H$"→8 Byte, "$40_H$"→64 Byte (maximum specified value)<br>• Value should always be greater than "$00_H$" and less than or equal to the maximum number of transfer bytes ($40_H$). |

### 29.3.3 EP1 to EP5 Control Register (EP1Cn to EP5Cn)

EP1 to EP5 Control Register (EP1Cn to EP5Cn) are used to control the operation of endpoints 1 to 5.

**EP1 to EP5 Control Register (EP1Cn to EP5Cn)**

Figure 29-4. EP1 Control Register (EP1Cn)

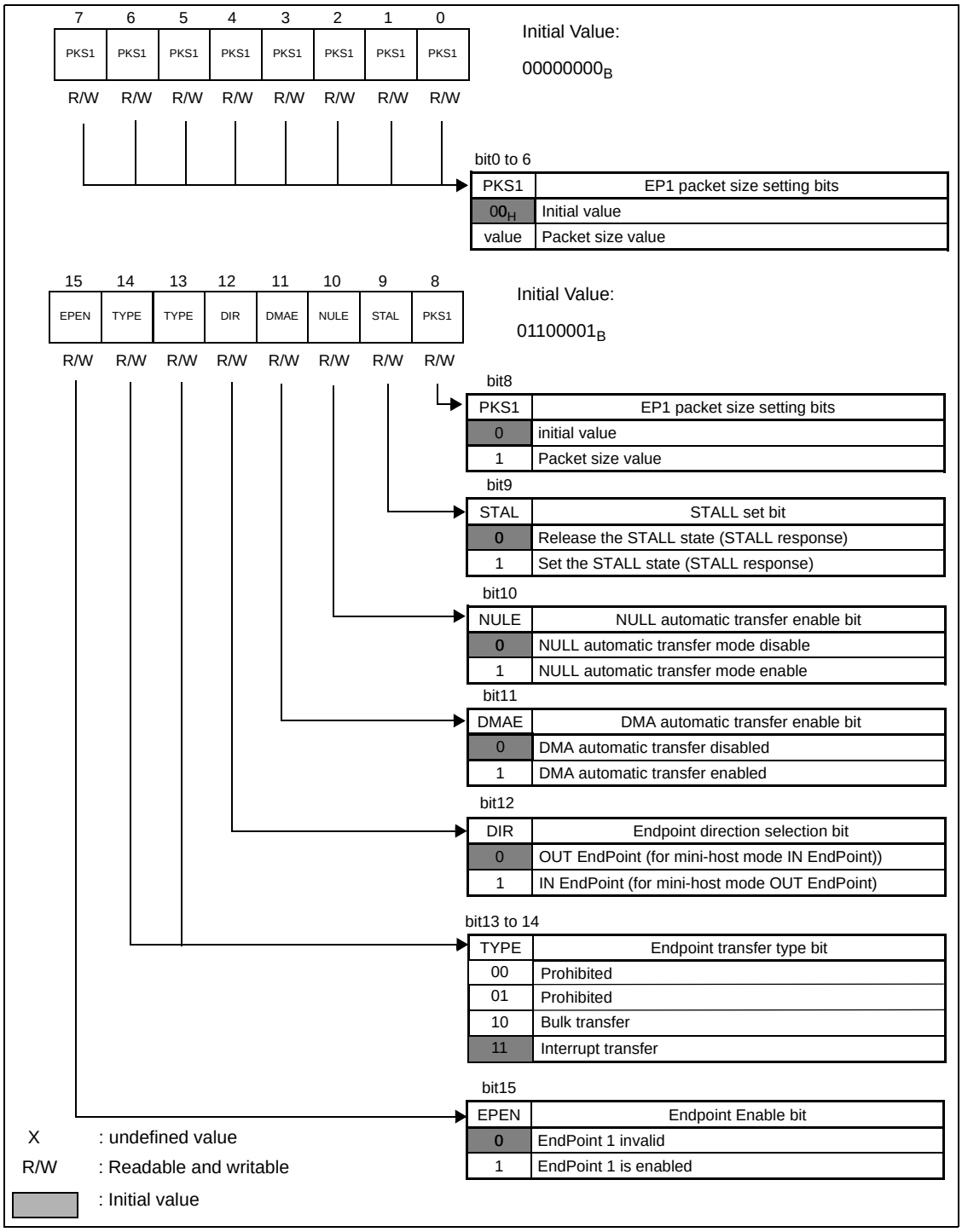
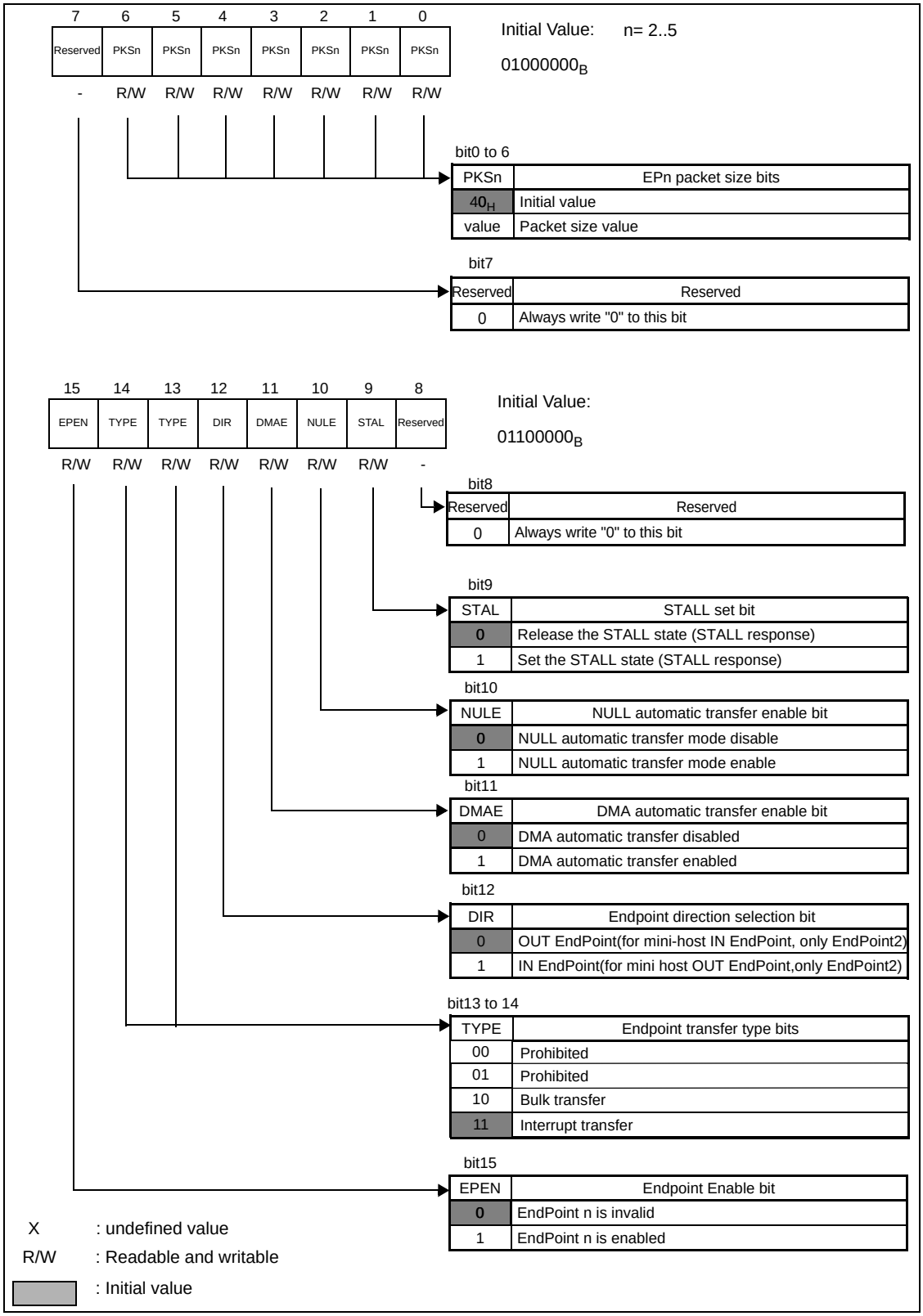


Figure 29-5. EP2 to EP5 Control register (EP2Cn to EP5Cn)



**Note:**

EP1 to EP5 control registers (EP1Cn to EP5Cn) except DMAE, NULE, and STAL bits must be configured only when both UDCCn:RST bit and EPxSn:BFINI are "1". Do not write to these registers while the USB is operating.

Below a description of the function of each bit in the EP1 to EP5 control register (EP1Cn to EP5Cn).

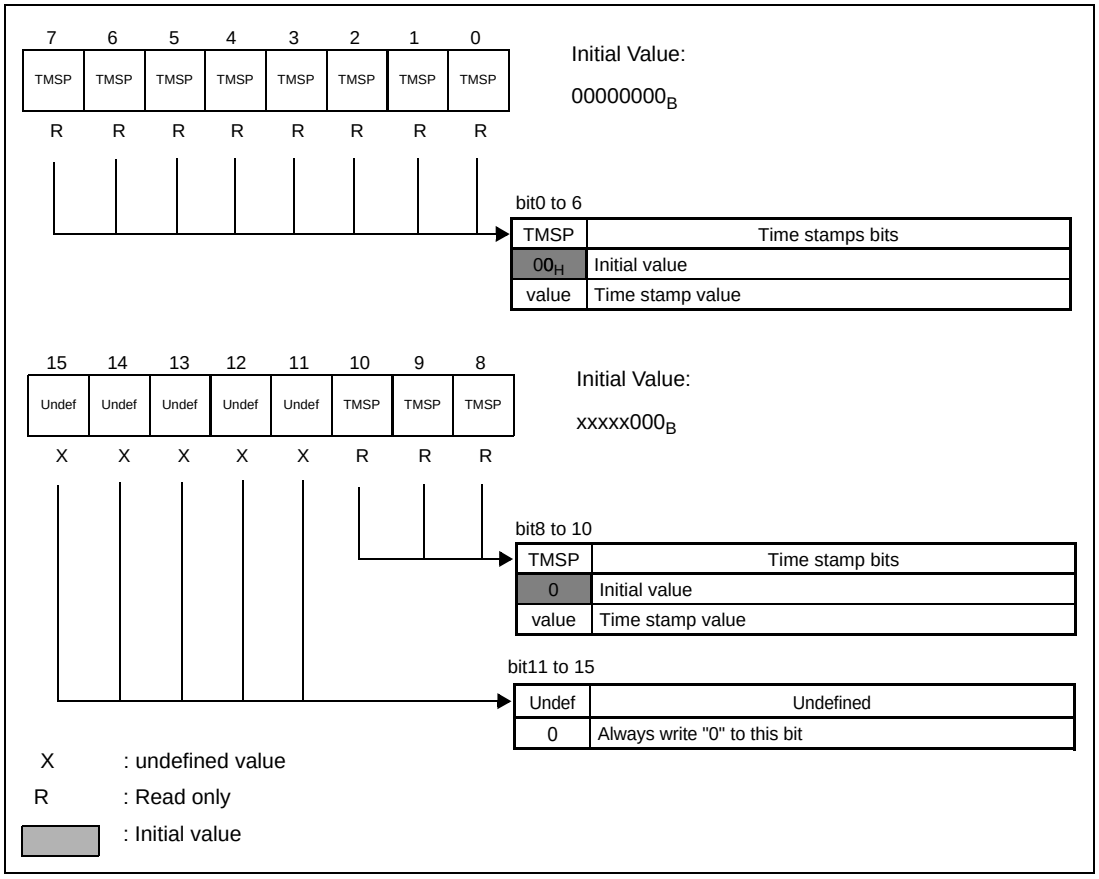
<table>
<tr><th colspan="2">Bit names</th><th>Function</th></tr>
<tr><td>Bit15</td><td>EPEN</td><td>• Endpoint enable. Setting the EPEN bit allows an endpoint to be configured by the host as an end point for use in the USB Function. When this bit is set to "1", TYPE, DIR, and PKS of the EP1 to EP5 control registers become valid for configuration.</td></tr>
<tr><td>Bit14-13</td><td>TYPE</td><td>• These two bits specify the transfer type for the corresponding endpoint, i.e Bulk or Interrupt transfer.</td></tr>
<tr><td>Bit12</td><td>DIR</td><td>• This bit specifies the transfer direction of the endpoint<br>• For USB function, writing "1" to this bit configures the corresponding End Point as IN endpoint. Writing "0" configures the endpoint as OUT endpoint.<br>• In USB Host mode, only End Point 1 and 2 are valid. Writing "0" to this bit in host mode configures the corresponding End Point as IN endpoint and writing "1" to this bit configures the End Point as OUT endpoint.</td></tr>
<tr><td>Bit11</td><td>DMAE</td><td>• DMA automatic transfer mode enable bit.<br>• If enabled, the DMA handles automatically the transmission and reception of data in sync with IN and OUT data requests from the HOST until the data bytes equal to the number set in DCT register of DMA is transferred. See chapter 29.4.6 DMA Transfer Function for details.<br>• Access to transmit and receive buffers by CPU is forbidden while the DMAE bit is set. Set the number of DMA transfer data to a multiple of the value set in the PKS bits of EP1 to EP5 control registers (EP1Cn to EP5Cn) when the endpoint is used for OUT transfer.</td></tr>
<tr><td>Bit10</td><td>NULE</td><td>• Enable NULL transfer. See chapter 29.4.7 NULL Transfer Function for details.<br>• During IN transfer , the last packet transfer will be detected and a 0-byte data packet will be automatically sent when IN data transfer request arrives from the host and the automatic buffer transfer mode is set (DMAE=1).<br>• NULE bit has no effect on communications when transferring data to OUT direction or when the automatic buffer transfer mode is disabled.</td></tr>
<tr><td>Bit9</td><td>STAL</td><td>• Set the endpoint in STALL status (STALL response).<br>• When the STAL bit is set, the USB function will respond to any requests from the HOST with a STALL handshake packet . The USB Function can return from STALL status after receiving a ClearFeature command from the host after the STAL bit is cleared by writing "0" to this bit.</td></tr>
<tr><td>Bit8</td><td>Reserved</td><td>• Reserved for EP2 to EP5<br>• Always write "0" to this bit.</td></tr>
<tr><td>Bit7</td><td>Reserved</td><td>• Reserved for EP2 to EP5<br>• Always write "0" to this bit.</td></tr>
<tr><td>Bit0-8:<br>EPICn<br><br>Bit0-6:<br>EP2Cn-EP5Cn</td><td>PKS</td><td>• This bit specifies the maximum number of bytes that can be transferred per packet. The following table shows the maximum number of bytes per packet to be transferred which can be specified for each endpoint:
<table>
<tr><th>Endpoint</th><th>Max number of transfer</th><th>Possible values</th></tr>
<tr><td>1</td><td>256 bytes</td><td>001h to 100h</td></tr>
<tr><td>2 to 5</td><td>64 bytes</td><td>01h to 40h</td></tr>
</table>
• Value should always be greater than "00H" and less than or equal to the maximum number of transfer bytes (100H or 40H).<br>• "00" must be written in bits 7 and 8 for endpoints 2 to 5.<br>• When the Data number automatic transfer mode is selected (DMAE=1), setting value less than 3 in the corresponding end point is forbidden.</td></tr>
</table>

## 29.3.4 Time Stamp Register (TMSPn)

The time stamp register (TMSPn) displays a frame number when a SOF packet is received.

### Time Stamp register (TMSPn)

Figure 29-6. Time Stamp register (TMSPn)




The functions of each bit of the time stamp register (TMSPn) are described below.

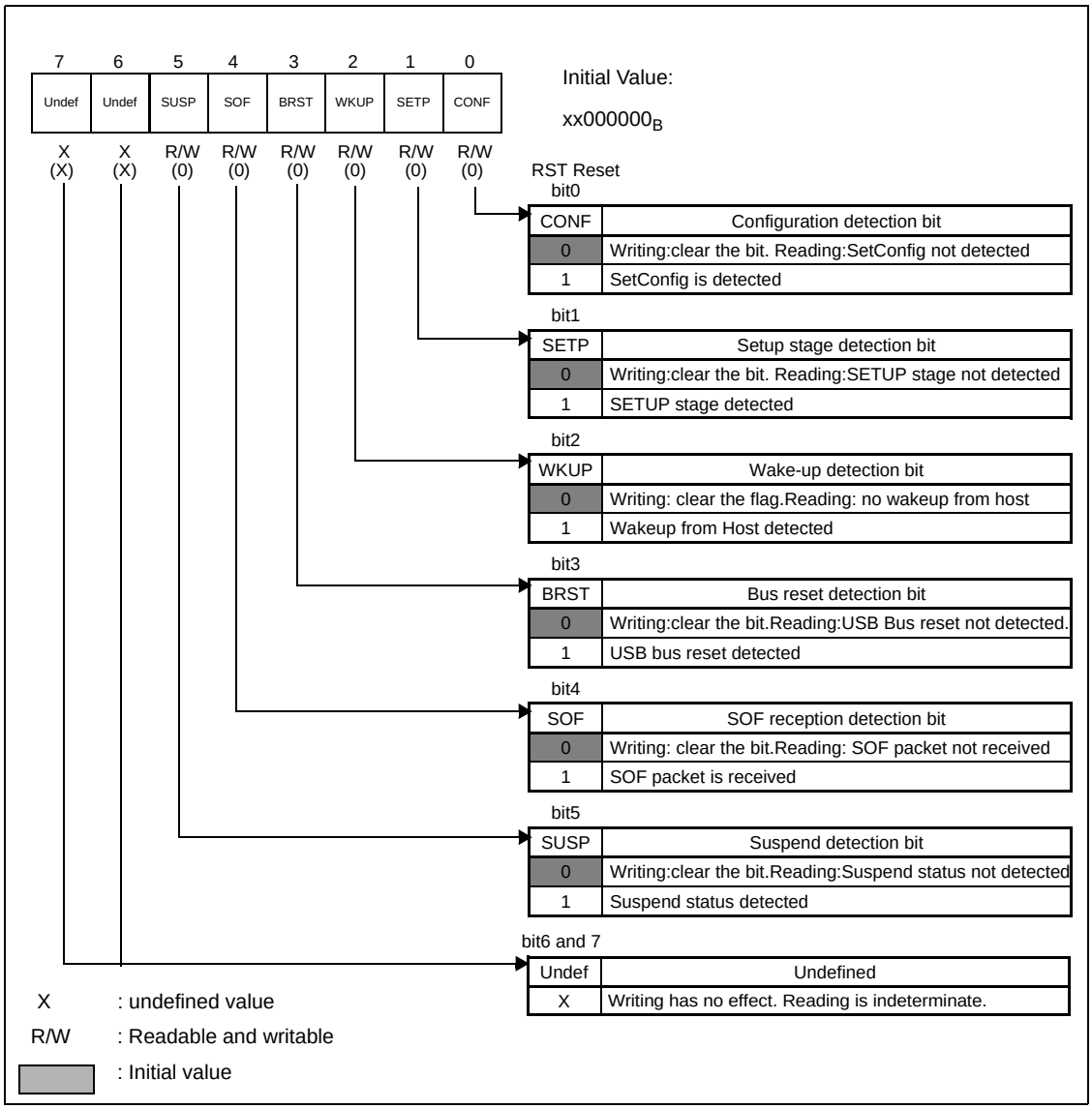
| Bit names | | Function |
|---|---|---|
| Bit15-11 | Undefined | • No functionality |
| Bit10-0 | TMSP | • It contains the Frame number of the current SOF packet. These bits are used when a SOF packet is received. |

## 29.3.5 UDC Status register (UDCSn)

The UDC Status Register (UDCSn) indicates the status of the USB communication. Each bit in the register, except SETP indicates an interrupt factor and raises an interrupt to the CPU if the corresponding interrupt enable bit in the UDC Interruption Enable register (UDCIEn) is set.

### UDC Status Register (UDCSn)

Figure 29-7. UDC Status register



The following describes the function of each bit in the UDC status register (UDCSn):

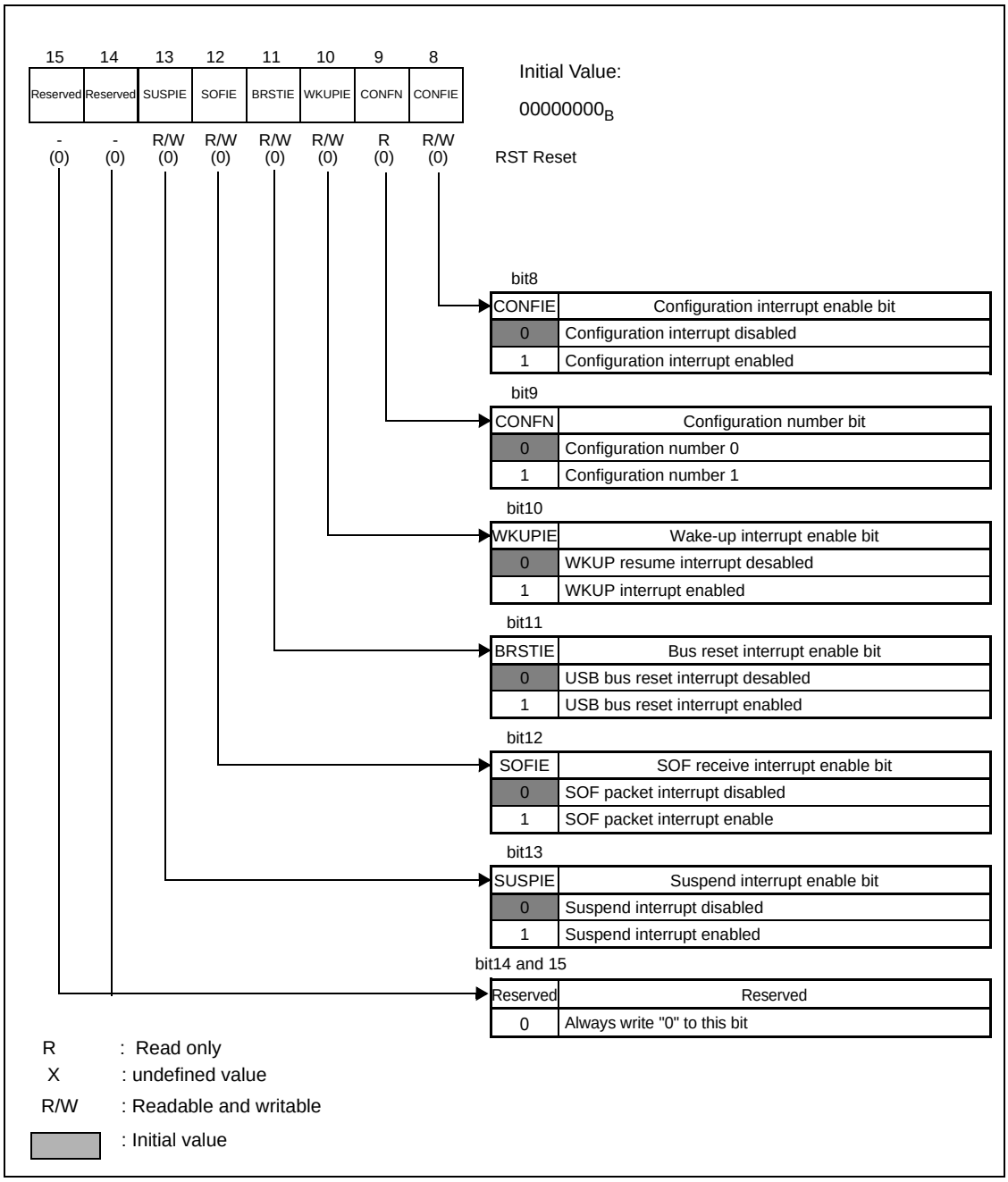
| Bit names | | Function |
|---|---|---|
| Bit7-6 | Undefined | • No functionality |
| Bit5 | SUSP | • This flag indicates that the USB function is in suspend mode. SUSP bit triggers the USB_F10 interrupt.<br>• Writing "1" is ignored.<br>• Clear by writing "0" to the bit. "1" is read at the read modification write access. |
| Bit4 | SOF | • Flag indicating the reception of a SOF packet.<br>• The value of the time stamp register (TMSPn) is updated when SOF is set to "1".<br>• The SOF flag triggers the USB_F10 interrupt.<br>• Writing "1" is ignored.<br>• Clear by writing "0" to this bit. "1" is read at the read modification write access. |
| Bit3 | BRST | • This flag indicates the reset of USB bus. It triggers the USB_F10 interrupt.<br>• Set all USB registers again after initializing the USB function with UDCCn:RST when bus reset is detected (BRST = "1").<br>• Writing "1" to this bit is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access. |
| Bit2 | WKUP | • This flag indicates that the USB function has returned from the suspend mode. This flag triggers the USB_F10 interrupt.<br>• A remote wake-up (by setting the RESUM bit) or a wake-up request from the HOST causes the USB function to return from suspend state. WKUP bit is automatically set only by a resume request from the HOST.<br>• If the RESUM bit in UDCCn register is set to "1", WKUP flag is not set even if wakeup is caused by the host.<br>• Writing "1" is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access. |
| Bit1 | SETP | • This bit indicates that the received data belongs to the SETUP stage of USB control transfer.<br>• This bit is not set when the USB function automatically responds to any standard commands.<br>• SETP bit does not trigger any interrupt.<br>• Writing "1" has no effect.<br>• Clear by writing "0". "1" is read at the read modification write access. |
| Bit0 | CONF | • It is set when the USB command SetConfig has been received and the USB Function is configured.<br>• The CONF bit trigger the USB_F10 interrupt.<br>• Writing "1" is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access. |

## 29.3.6 UDC Interruption Enable Register (UDCIEn)

The UDC Interrupt Enable Register (UDCIEn) is a register that enables each flag in the UDC status register (UDCSn) to trigger an interrupt. (except the CONFN bit).

### UDC Interrupt Enable register (UDCIEn)

Figure 29-8. UDC Interrupt Enable Register (UDCIEn)




The function of each bit in the UDC interrupt register (UDCIEn) is described in the following table.

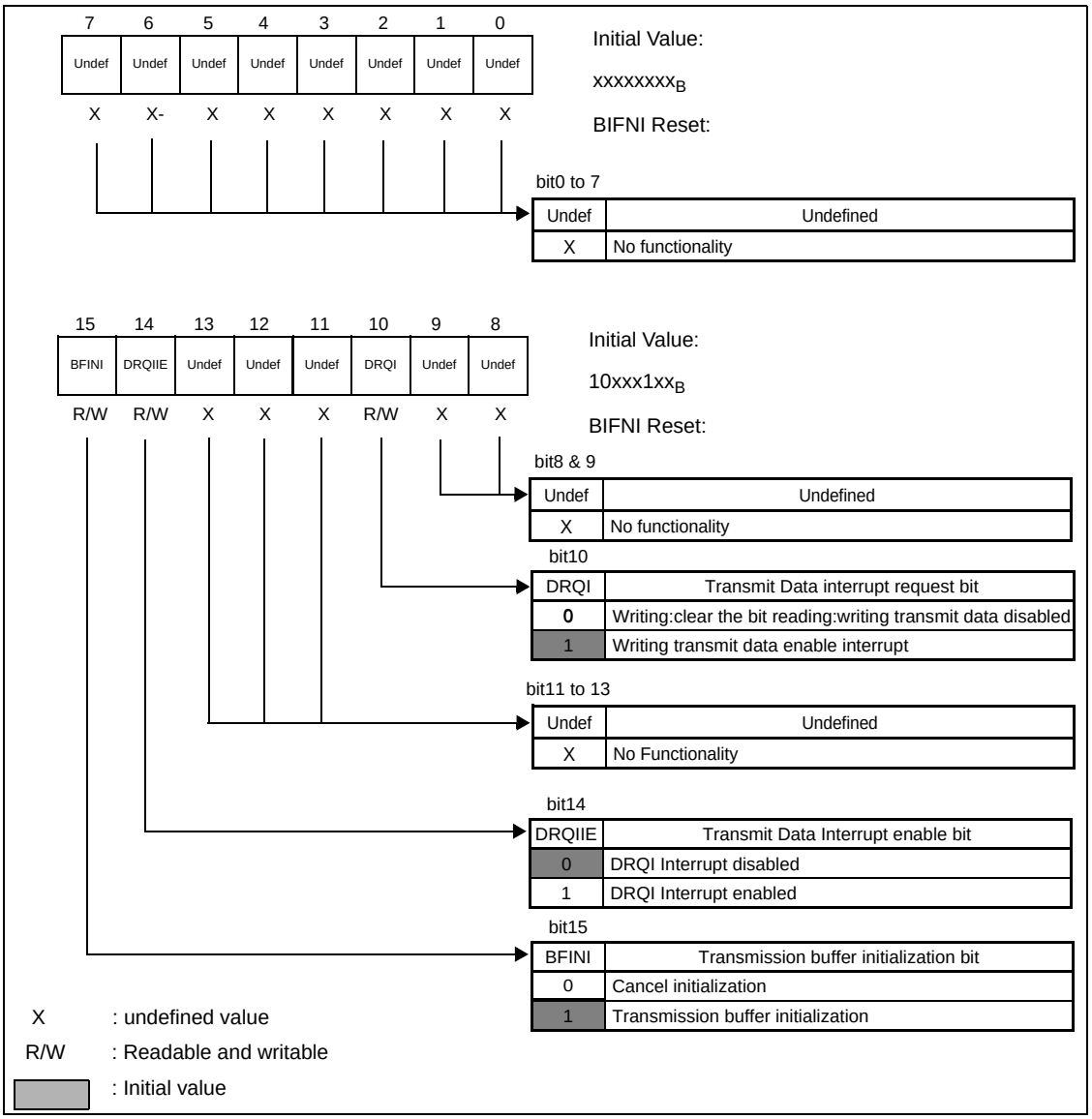
| Bit names | | Function |
|---|---|---|
| Bit15-14 | Reserved | • Always write "0" to this bit. Read value of this bit is always "0" |
| Bit13 | SUSPIE | • Enable interrupt caused by SUSP flag. |
| Bit12 | SOFIE | • Enable interrupt caused by SOF flag. |
| Bit11 | BRSTIE | • Enable interrupt caused by BRST flag. |
| Bit10 | WKUPIE | • Enable interrupt caused by WKUP flag. |
| Bit9 | CONFN | • This bit displays the configuration number.<br>• It is updated when CONF flag of UDCSn is set by SET_CONFIGURATION command from the host. |
| Bit8 | CONFIE | • Enable interrupt caused by CONF flag of UDCSn. |

### 29.3.7 EP0I Status Register (EP0ISn)

The EP0I Status Register (EP0ISn) displays the status of IN direction transfer for EndPoint 0.

**EP0I Status Register (EP0ISn)**

Figure 29-9. EP0I Status register (EP0ISn)




The function of each bit in the EP0I status register (EP0ISn) is described in the following.

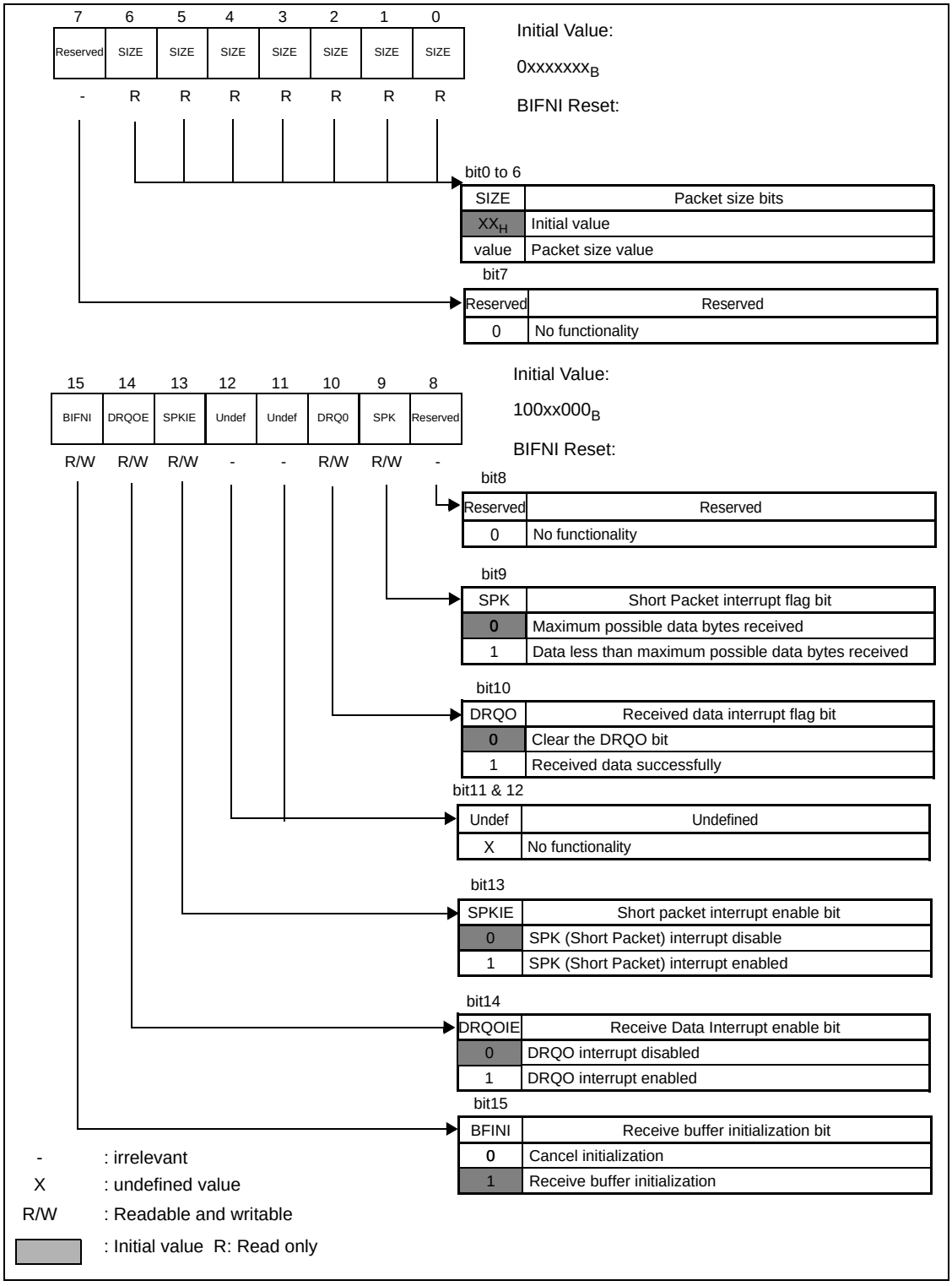
| Bit names | | Function |
|---|---|---|
| Bit15 | BFNI | • This bit initializes the transmission buffer.<br>• The BFINI bit is automatically set when the RST flag in the UDC Control Register (UDCCn) is set to "0".<br>• When the reset operation has been performed with the UDCCn:RST bit, clear the UDCCn:RST bit before clearing the BFINI bit.<br>• The BFINI bit initializes a buffer and the DRQI bit. Before initializing the buffer, please ensure that the DRQI or DRQO bit is set to "1" that means there is no access from the HOST, and set the STAL bit (if it is necessary). |
| Bit14 | DRQIIE | • DRQI interrupt enable.<br>• It enables an interrupt to be triggered when DRQI flag is set. |
| Bit13-11 | Undefined | • No functionality. |
| Bit10 | DRQI | • This bit indicates that an IN packet has been successfully transferred from the EP0 to the host. The data has been read out from the transmission buffer, and the next data to be transmitted can be written into the buffer.<br>• After the data is written to the transmission buffer, the DRQI must be cleared.<br>• When DRQI = "0", writing "0" is prohibited.<br>• When DRQI = "1", writing data to the transmission buffer is<br>• enabled. Clearing the DRQI bit indicates that writing data to the transmit buffer is completed. If IN packet request is performed when DRQI = "1" , NAK response is sent to the HOST automatically. So this bit is to be cleared to process IN transfer request.<br>• The DRQI bit triggers USB_EP0IN0 interrupt.<br>• Writing "1" to this bit is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access. |
| Bit9-0 | Undefined | • No functionality |

## 29.3.8 EP0O Status Register (EP0OSn)

The EP0O Status Register (EP0OSn) displays status of OUT direction transfer of the endpoint 0.

**EP0O Status Register (EP0OSn)**

Figure 29-10. EP0O Status Register (EP0OSn)




The function of each bit in the EP0O status register (EP0OSn) is described in the following.

| Bit names | | Function |
|---|---|---|
| Bit15 | BFINI | • This bit initializes the receive buffer.<br>• The BFINI bit is automatically set when the RST bit in the UDC control register (UDCCn) is set to "1".<br>• When the reset operation has been performed with the UDCCn:RST bit, clear the UDCCn:RST bit before clearing the BFINI bit.<br>• The BFINI bit initializes a buffer and the DRQO bit and SPK bit. Before initializing the buffer please ensure that the DRQI or DRQO bit is set to "1" that means there is no access from the HOST, and set the STAL bit (if it is necessary). |
| Bit14 | DRQOIE | • DRQO interrupt enable. It enables an interrupt to be triggered when DRQO flag is set. |
| Bit13 | SPKIE | • This bit enables interrupt to be triggered when SPK flag is set. |
| Bit12-11 | Undefined | • No functionality. |
| Bit10 | DRQO | • It indicates that an OUT packet has been successfully received from the host, data has been written into the receive buffer and the received data can be read from the receive buffer.<br>• After the receive buffer is read, DRQO must be cleared.<br>• When DRQO = "0", writing "0" is prohibited.<br>• When DRQO= "1", receive buffer is not updated. The buffer is enabled to be updated when DRQO is cleared. When OUT packet request is performed by the host when DRQO bit set to "1", NAK response is sent to the HOST automatically.<br>• The DRQO bit trigger an interrupt USB_EP0OUT0.<br>• Writing "1" is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access. |
| Bit9 | SPK | • This bit indicates that the number of bytes of the data packet successfully received from the HOST is less than the maximum packet size value set in EP0Cn:PKS. (zero length packet will also set this bit).<br>• This bit triggers an interrupt USB_F20.<br>• Writing "1" is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access. |
| Bit8-7 | Undefined | • No functionality |
| Bit6-0 | SIZE | • When OUT packets have been transferred from host to EP0, the number of data bytes that have been written into the reception buffer is displayed. The SIZE bits are updated to a valid value when EP0OSn:DRQO flag is set.<br>• Example: 8 bytes → "$08_H$", 64 bytes → "$40_H$" (maximum value) |

## 29.3.9 EP1 to EP5 Status Register (EP1Sn to EP5Sn)

The EP1 to EP5 Status Registers (EP1Sn to EP5Sn) display status of endpoint 1 to endpoint 5.

**EP1 to EP5 Status Register (EP1Sn to EP5Sn)**

Figure 29-11. EP1 Status Register (EP1Sn)

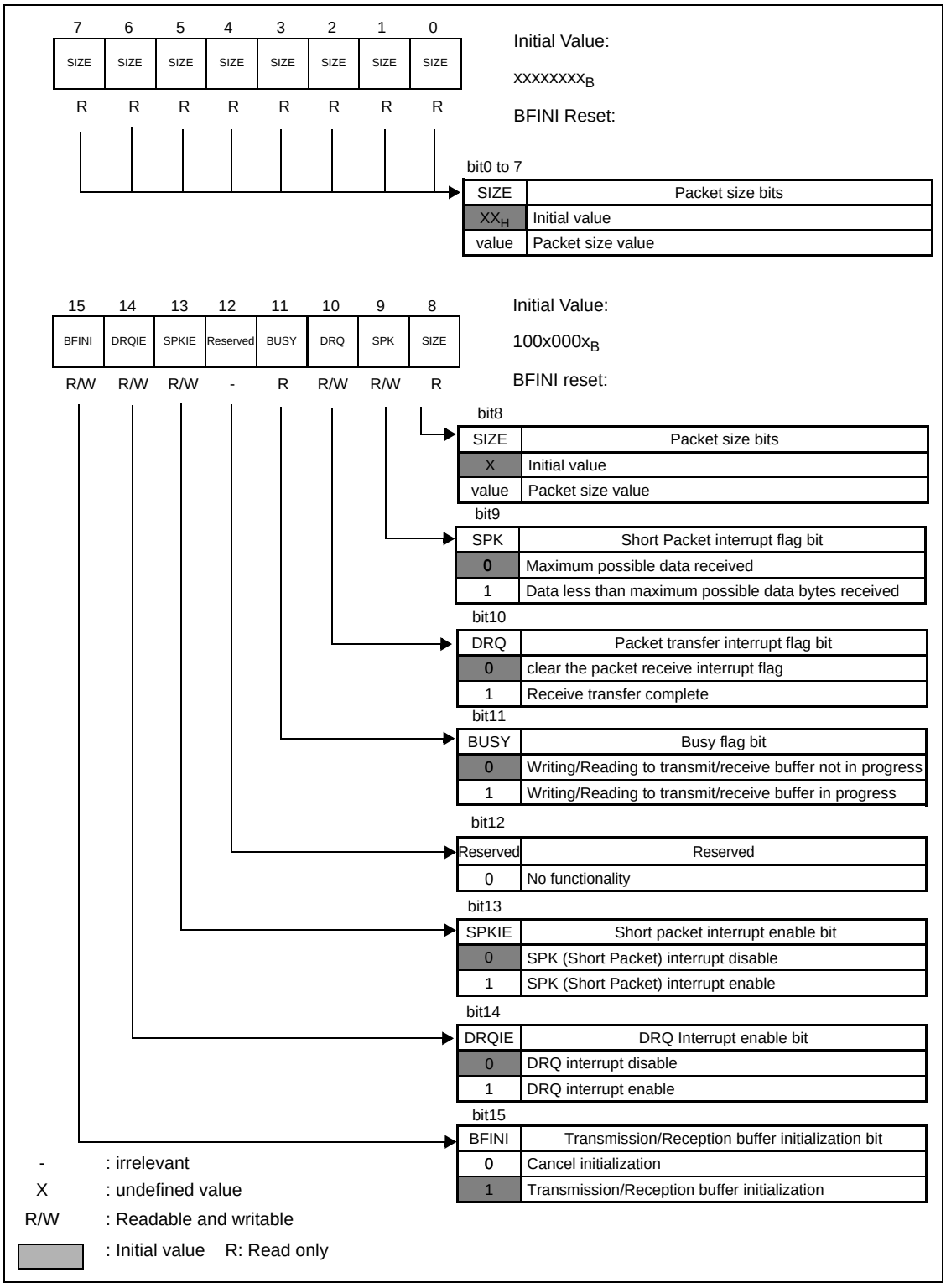
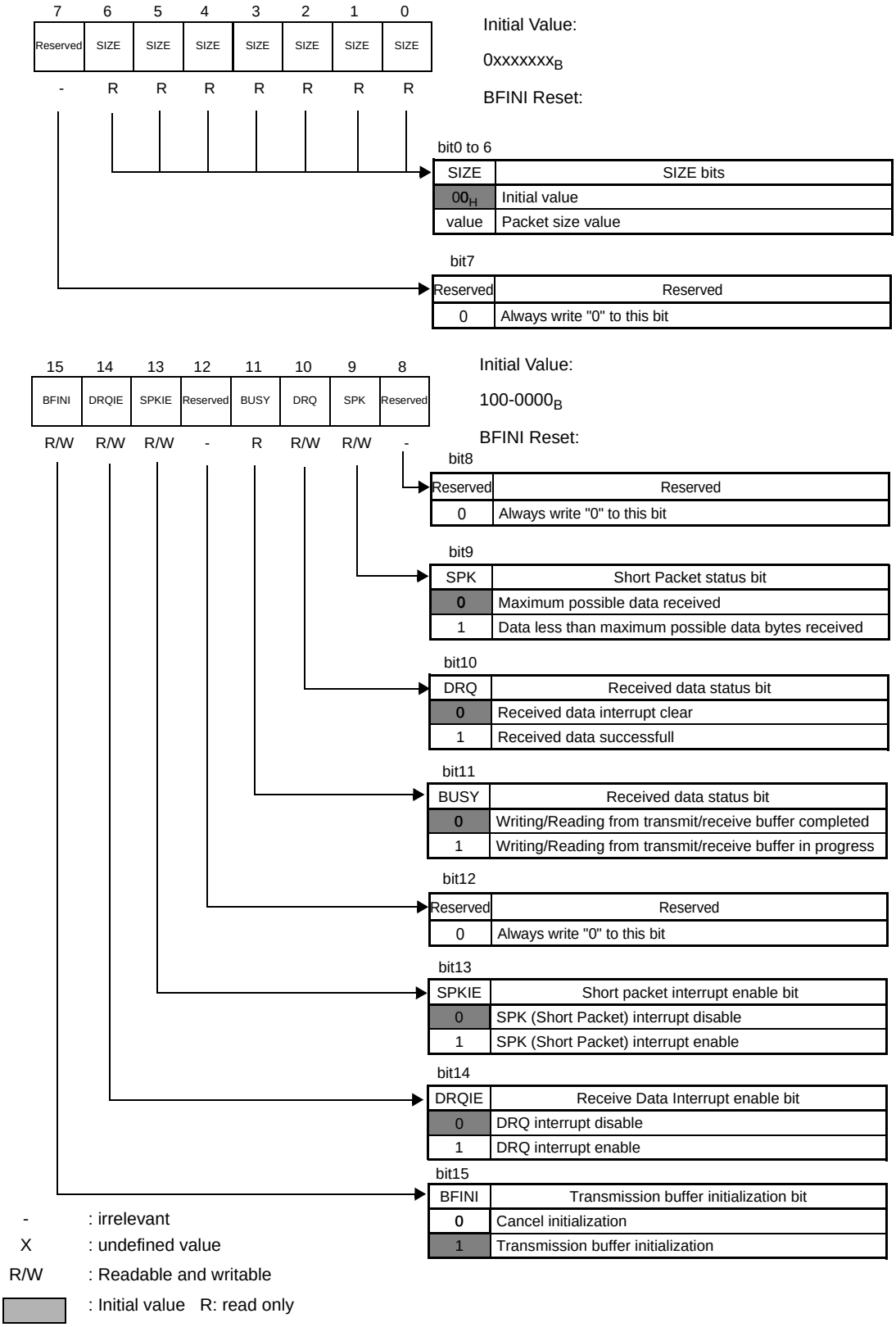


Figure 29-12. EP2 to EP5 Status Register (EP2Sn to EP5Sn)



The function of each bit in the EP1 to EP5 status register (EP1Sn to EP5Sn) is described in the following.

| Bit names | | Function |
|---|---|---|
| Bit15 | BFINI | • This bit initializes the transmission/reception buffer.<br>• The BFINI bit is automatically set when the RST flag in the UDC control register (UDCCn) is set to 1.<br>• When the reset operation has been performed with the RST bit, clear the RST bit before clearing the BFINI bit.<br>• The transmission/receive buffer for EP1 to EP5 has a configuration of double buffers. Initialization by BFINI bit initializes the double buffers, the DRQ and SPK bits all at once.<br>• Before initializing the buffer please ensure that the DRQ bit is set to "1", and ensure that BUSY="0" that means there is no access from the HOST, and set the STAL bit. |
| Bit14 | DRQIE | • DRQ interrupt enable. It enables an interrupt to be triggered when the DRQ flag of one of the EP1Sn to EP5Sn is set.<br>• In automatic buffer transfer mode (DMAE = "1"), DMA settings must be enabled before setting DRQIE. |
| Bit13 | SPKIE | • This bit enables an interrupt to be triggered by the SPK flag of each EP1Sn to EP5Sn register. |
| Bit12 | Undefined | • No functionality. |
| Bit11 | BUSY | • This bit indicates that writing into the transmission/receive buffer or reading it for sending data to/from the host is in progress.<br>• It is set/reset automatically.<br>• When the DRQ bit is set and also the BUSY flag is set, it means that the host is accessing one of the double buffers that is different from the buffer accessed by the CPU or DMA.<br>• Normally threre is no need to refer to the BUSY bit when controlling the buffers. But before initializing the buffer, please ensure that the DRQ bit is set to "1", and ensure that the BUSY="0" that means there is no access from the HOST, and set the STAL bit. |
| Bit10 | DRQ | • This bit indicates that packet transfer for EPx has been successfully done and data processing is needed.<br>• The EPxSn:DRQ bit triggers corresponding USB_EPx0 interrupt .<br>• Writing "1" to this bit is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access.<br>• After the data to be sent to the host is written to the transmission buffer (in response to the IN packet request from the host), the DRQ bit must be cleared. When the OUT packet transfer is completed DRQ bit is automatically set to "1". It should be cleared when the data from the receive buffer is completely read by the CPU/DMA.<br>• When DRQ = "0", writing "0" is prohibited.<br>• When the automatic buffer transfer mode (DMAE=1) is not used after the data read or write of transmission and reception buffers is completed, "0" must be written to the DRQ bit. When DRQ bit is cleared, access buffer is switched. If the transfer direction is set to IN direction and the DRQ bit is "1" and the DRQ bit is cleared without writing data to the transmission buffer, 0-byte data packet is sent to the host.<br>• In the initial setting, when the DIR of the EP1 to EP5 control register (EP1Cn to EP5Cn) is set to "1", DRQ bit of the corresponding end point is set at the same time. |
| Bit9 | SPK | • This bit indicates that the number of data bytes received successfully from the HOST is less than the maximum packet size value set in EPxCn:PKS (including 0 byte data packets).<br>• The SPK bit triggers USB_F20 interrupt.<br>• Writing "1" to this bit is ignored.<br>• Clear by writing "0". "1" is read at the read modification write access.<br>• The SPK bit is not set during IN data transfer. |
| Bit8-7 | Reserved/SIZE (EP1Sn) | • Do not use for EP2Sn to EP5Sn.<br>• For EP1Sn these are the SIZE bits (see below) |

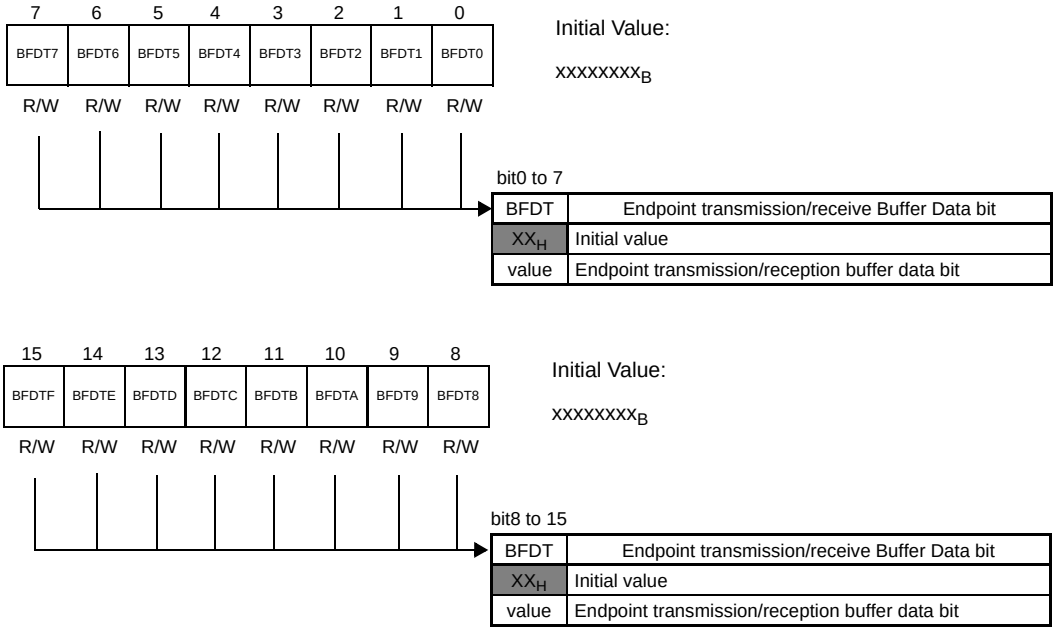
| Bit names | | Function |
|---|---|---|
| Bit8-0: EP1Sn<br>Bit6-0:<br>EP2Sn - EP5Sn | SIZE | • When OUT packets have been transferred to EPx, the number of data bytes that has been written into the reception buffer is displayed by SIZE bits. The SIZE bit is updated to a valid value when EPxSn:DRQ flag is set. The following table displays the maximum transferable number of data for each endpoint 1 to 5:<br>Endpoint / Max number of transfer / Possible values<br>1 / 256 bytes / 000h to 100h<br>2 to 5 / 64 bytes / 00h to 40h<br>• Since SIZE displays data packet size written in the buffer by the host during OUT packet transfer, any SIZE value read when an IN direction transfer is performed, is meaningless. |

## 29.3.10 EP0 to EP5 Data Register (EP0DTn to EP5DTn)

The EP0 to EP5 Data Registers (EP0DTn to EP5DTn) are access registers used for read or write access into the transmission/receive buffer related to endpoint 0 to endpoint 5.

### EP0 to EP5 Data Register (EP0DTn to EP5DTn)

Figure 29-13. EP0 to EP5 Data Register (EP0DTn to EP5DTn)



The following describes the function of each bit in the EP0 to EP5 data registers (EP0DTn to EP5DTn):

| Bit names | | Function |
|---|---|---|
| Bit15-0 | BFDT | • It is a data read/write register for the transmission/receive buffer for each endpoint.<br>• Access to the BFDT register via DMA transfer is supported by word access only. If you transfer odd number of data bytes through DMA transfer, you have to set a byte transfer for the last data transfer. If you perform word transfer (for odd number of data bytes) via CPU access, the last transfer must be a byte transfer, the same as in case of DMA transfer.<br>• CPU access to the EP0DTn to EP5DTn registers are possible both byte and word. If byte access is needed for any of the registers, access lower byte (bit 7 to bit 0) first and then access upper byte (bit 15 to bit 8). Subsequently access the lower byte and upper byte alternately.<br>• Bit access to the EP0DTn to EP5DTn registers is prohibited |

# 29.4 Description of the USB function operation

This chapter describes the basics of the USB function.

## 29.4.1 USB Function operation

The USB function performs a both way packet transfer with a host controller that supports USB protocol. A host and its devices are connected and configured by the enumeration process. Then, communication based on various types of transfers using device drivers can be performed.

This section describes the operation of the USB communication between a HOST and the USB device by using enumeration as an example. It illustrates the operation of registers and USB packet transfers to give an overview of the USB communication.

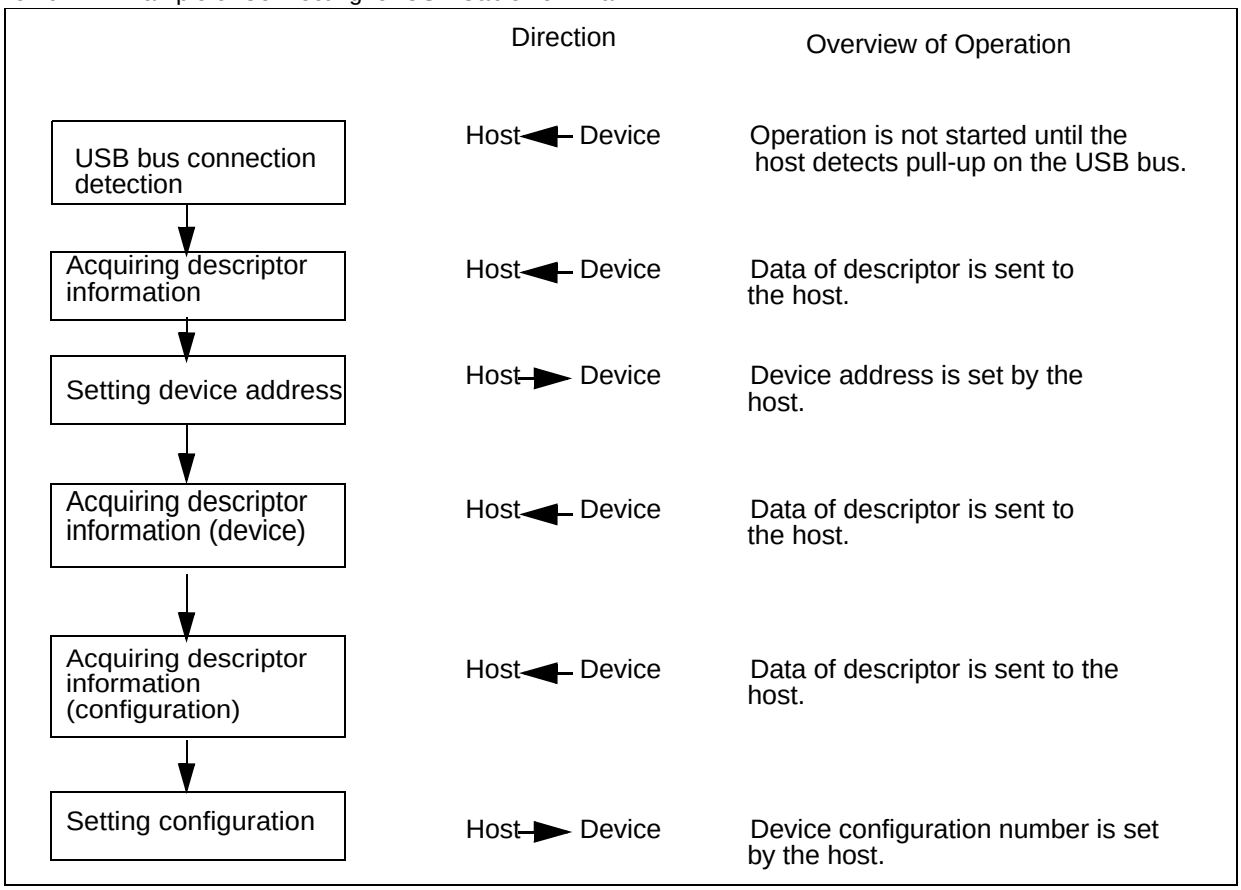
### Enumeration process

The enumeration process is the initial communication process between a host and the USB device when the device is attached to the bus before the USB operates. The host examines which device is connected to the USB bus and what parameters does it require, the number of EndPoints it has etc., by using the USB Control transfers (one of the types of USB transfers). USB Control Transfers use the EP0 (endpoint 0) out of the available six endpoints (as defined in the USB specification).

Before EP1 to EP5 are used, the followings must be received on the USB bus (i.e basic enumeration process should be completed).

1. USB bus reset (UDP is "0" and UDM is "0" for at least 40 USB clock cycles).
2. Address set by SET_ADDRESS.
3. Configuration set by SET_CONFIG.

Figure 29-14. Example of Connecting for USB Cable Terminal




### Detecting a connection

The HOST monitors the two signals (D+ and D-) on the USB bus and detects a device connection if a signal goes to the "H" level.

For the detailed procedure for the case of a self-powered device, see 29.4.2 Detecting Connection and Disconnection.

### Example of Register Initialization and Operation Startup Procedure

The following example describes how to initialize the registers and start operation.

1. Set EP0 in the EP0Cn register (packet size, etc.).
2. Set the EPEN, DIR, TYPE, etc. settings for each endpoint (see EP1Cn to EP5Cn registers).
3. Clear the RST bit in the UDCCn register.
4. Clear BFINI in the EP0ISn, EP0OSn, and EP1Sn to EP5Sn registers.
5. Clear the HCONX bit in the UDCCn register.

### USB bus reset

A bus reset is issued by the HOST to initialize the USB device. The USB device must then perform the following steps: (The first bus reset after USB has been connected does not need any processing.)

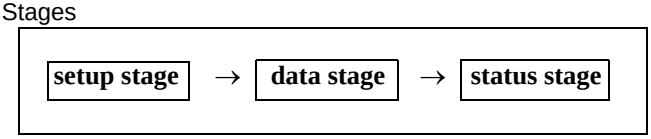
1. The USB function is initialized by setting the RST bit of the UDCCn register.
2. Set again the transmission/reception buffers in use and related control registers
3. Set the firmware control to the pre-enumeration state.

### Getting descriptor

The USB device sends data to the host when it receives a request from the HOST.

In more detail, communications are performed in the following three stages:

Figure 29-15. Communication Stages




The setup stage ensures that the device receives normal packets (specific request e.g. GET_DESCRIPTOR) from the HOST and identifies the command by decoding it and prepares information of a descriptor in the transmit buffer that is to be sent back to the host in Status stage . The data stage simply confirms that normal data is sent or received from the HOST (depending on whether it is a SET request or GET request). The status stage reports the status of the overall request and performs the end processing when the HOST sends (in response to the DESCRIPTOR sent in the data stage) a zero length data packet.

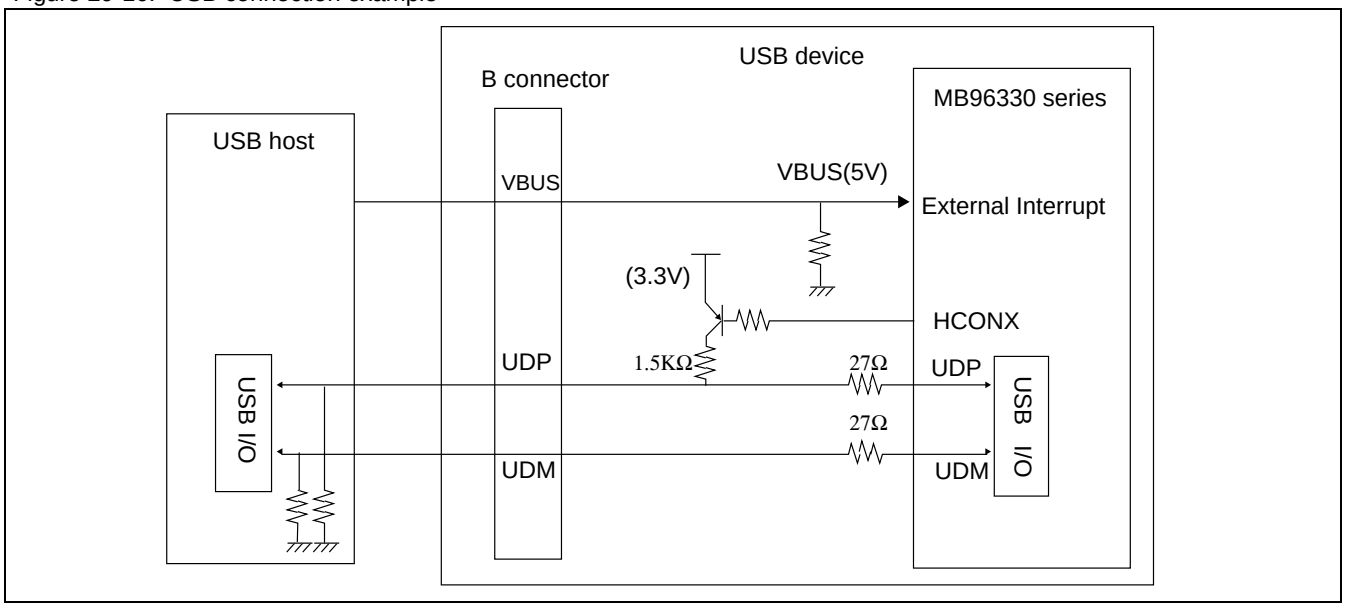
## 29.4.2 Detecting Connection and Disconnection

This section describes how to detect connection and disconnection from the USB host.

### *29.4.2.1 USB connection example*

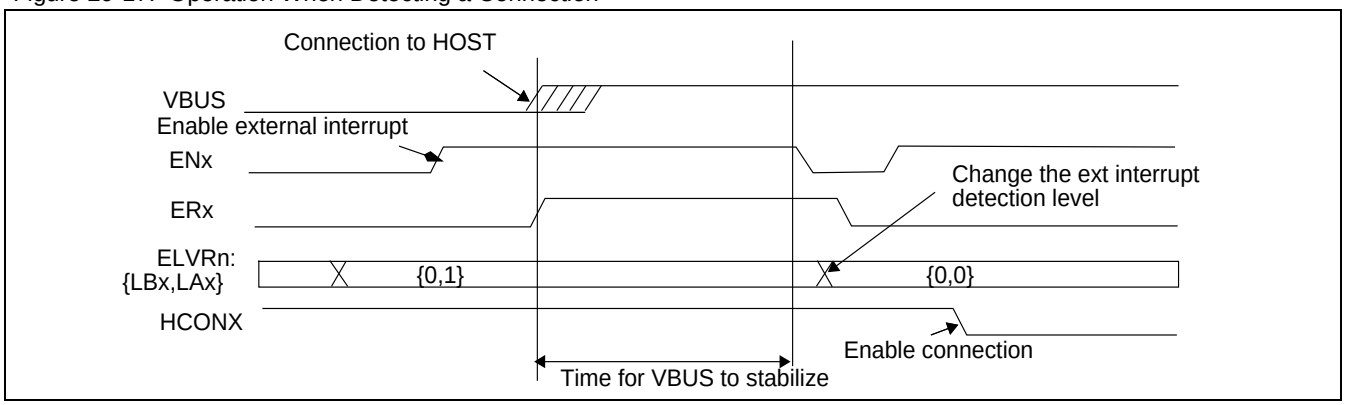
Connection and disconnection from the USB host can be detected by connecting an external interrupt pin to the VBUS pin on the USB connector and connecting a pull-down resistor. Figure 29-16 shows an connection example for the UDP, UDM, and VBUS pins on the USB connector.

Figure 29-16. USB connection example



### Detecting connection

Figure 29-17. Operation When Detecting a Connection




The device uses the following sequence to detect a connection with the HOST.

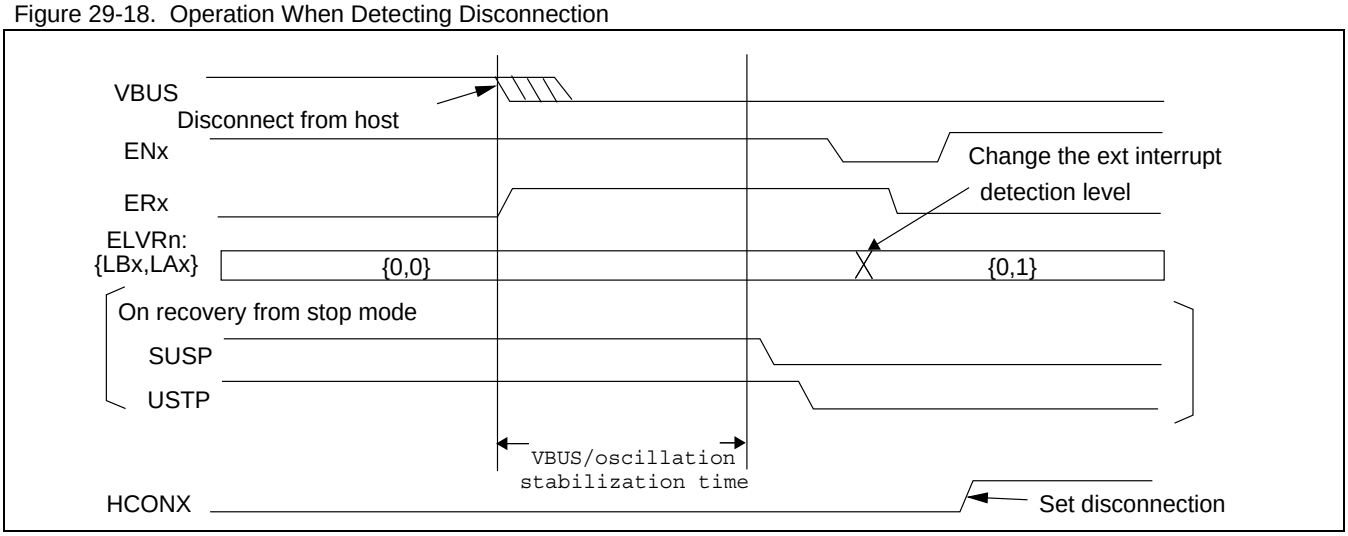
1. Set the external interrupt connected to the VBUS to detect "H" level input and then enable the interrupt.
2. Detection of an "H" level on the external interrupt pin indicates a connection to a USB host. When this occurs, please wait for the VBUS stabilization time before proceeding further.
3. Disable the external interrupt temporarily. Set the external interrupt to detect "L" level inputs to the external interrupt pin ( for detecting disconnection from the host) and clear the external interrupt flag and enabled it again.
4. Initialization (complete initialization including the USB Function Register) can then be performed. See "Example of Register Initialization and Operation Startup Procedure" in section 29.4.
5. Clear UDCCn:HCONX bit by writing "0" to it to enable the D+ pull-up resistor. (Clear the HCONX bit even if it is not used for controlling the pull-up resistor).

#### Note:

You need not wait for the VBUS stabilization time in your program if an external noise filter is used on the external interrupt pin.

### Detecting disconnection

Figure 29-18. Operation When Detecting Disconnection




The USB device uses the following sequence to detect a disconnection from the HOST.

1. Detection of an "L" level on the external interrupt pin connected to VBUS indicates the disconnection from the USB host.

2. On recovery from STOP mode:

   After waiting for the oscillation stabilization time, clear UDCSn:SUSP followed by UDCCn:USTP.

   When not recovering from stop mode:

   Wait for the VBUS stabilization time.

3. Disable the external interrupt temporarily. Change the external interrupt setting to detect "H" level inputs to the external interrupt pin, clear the external interrupt flag, and then re-enable the external interrupt.
4. Set the UDCCn:HCONX bit to "1" to disconnect the D+ pull-up resistor. (Set the HCONX bit to "1" even if it is not used for control of the pull-up resistor).

**Note:**

You need not wait for the VBUS stabilization time in your program if an external noise filter is used on the external interrupt pin.

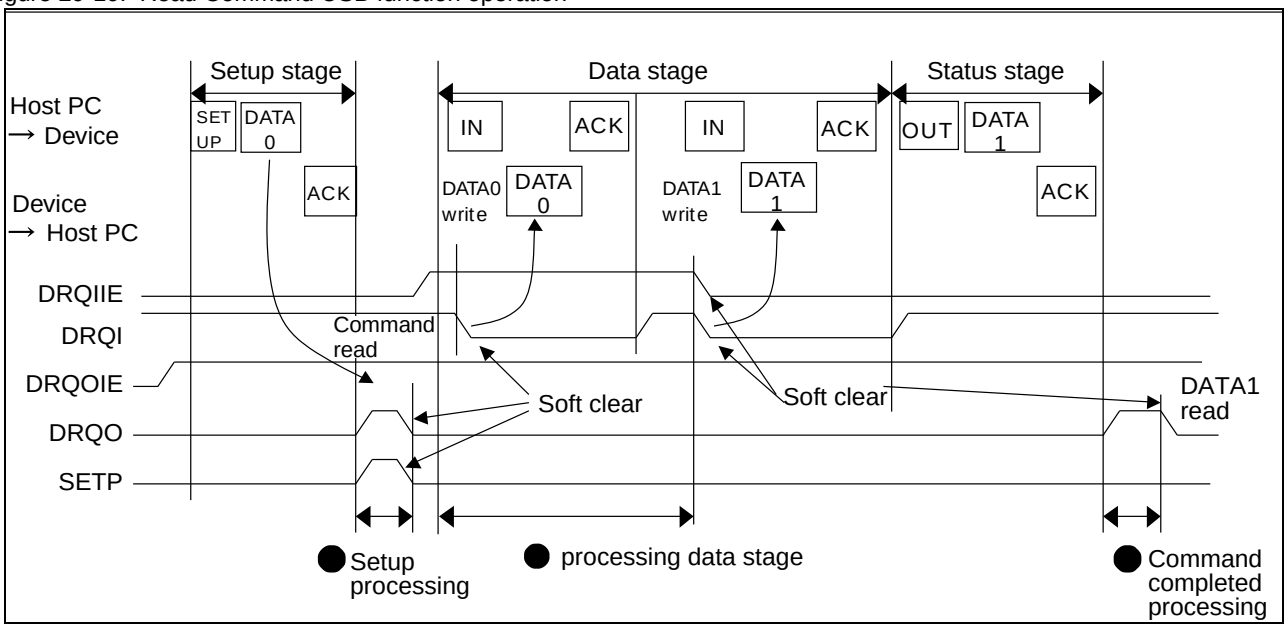
## 29.4.3 Each Register Operation with Command Response

This section describes the basic operation of USB registers and processing of the USB packets. The processing of firmware tasks triggered via CPU interrupts are also described.

### *29.4.3.1 Read Command USB function operation*

For GetDescriptor, SynchFrame, and the class vendor command

Figure 29-19. Read Command USB function operation




**Set up operation**

When the setup packet is received by the USB device, DRQO and SETP are set to "1". When DRQO is set, CPU interrupt is raised and the SETP flag is conformed in the interrupt service routine. The USB device then reads as many commands as necessary from the receive buffer if the SETP is set (which does not mean all eight bytes need to be read. The number of bytes to be read depends on the SIZE value in the EP0OSn register), decodes the commands, performs setting tasks, clears the SETP flag and DRQO interrupt cause (EP0OSn:DRQO) by writing "0" to them and returns from the interrupt service routine.

### Data stage operation

If the result of decoding the command says that the data stage is an IN transfer, enable the DRQIIE bit ( no need to set DRQI bit because its initial value is "1") . The CPU interrupt service routine for DRQI interrupt transfers data to be sent to the host to the transmit buffer. Once the transfer is complete, clear the DRQI interrupt cause (EP0ISn:DRQI) before returning from the interrupt.

The DRQI is set again when the data packet transfer to the host is complete. This generates a CPU interrupt and the corresponding interrupt service routine for DRQI is entered again. The data is transferred to the transmit buffer to prepare for the next data packet to be sent to the host. Once the transfer is complete, clear the DRQI interrupt cause (EP0ISn:DRQI) before returning from the interrupt service routine.

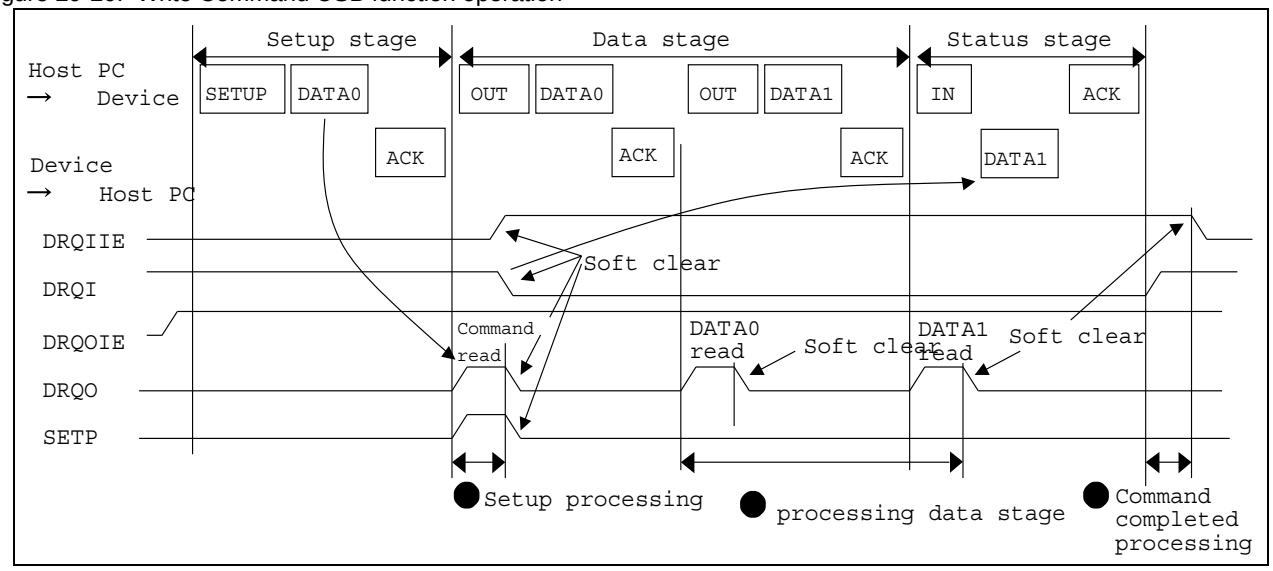
### Status stage operation

The DRQI is set when the data stage is complete. Since the DRQI interrupt is not enabled (DRQIIE ="0") the CPU does not execute the interrupt service routine for DRQI. When a zero length OUT packet is received from the host the DRQO is set to "1". The CPU processes the DRQO interrupt and the number of received bytes is checked in the SIZE bits of the EP0OSn. If EP0Sn:PKS is equal to zero (indicating that a zero byte data packet is received) clear the DRQO interrupt cause to prepare for the next setup stage.

## *29.4.3.2 Write Command USB function operation*

For GetDescriptor and the class vendor command

Figure 29-20. Write Command USB function operation




### Set up operation

When the setup packet is received, DRQO and SETP is set. When DRQO is set the CPU interrupt corrosponding to DRQ0 is raised and the SETP flag is conformed in the interrupt service routine. The USB device then reads as many commands bytes as necessary from the receive buffer if the SETP is set (which does not mean all eight bytes need to be read. The number of bytes to be read depends on the SIZE value in the EP0OSn register), decodes the commands, performs setting tasks, and clear the DRQI bit (because initial value of DRQI is "1") without writing data to the transmission buffer to prepare for a 0-byte response in the status stage. Then set DRQIIE to enable the interrupt cause due to DRQI to check the normal completion of Status stage. The routine also clears the SETP flag and DRQO by writing "0" to it and then returns from the interrupt routine.

### Data stage operation

The DRQO is set when the data stage toward OUT is complete. The interrupt is processed when the DRQO is set. Then check the SIZE bits of the EP0OSn register and transfer the data from the receive buffer by DMA or read the data from the receive buffer directly by CPU. Then clear the DRQO bit before returning from the interrupt.

**Status stage operation**

When the Data stage is completed the host sends an IN token request. A zero length packet is sent by the device in response to this request. When the status stage is completed, a CPU interrupt USB_EP0IN is triggered due to DRQI and the program enters in an interrupt service routine and confirms if the status stage has been successfully completed. Then clear the EP0ISn:DRQIIE before returning to the interrupted point.

## 29.4.4 Suspend Mode Function

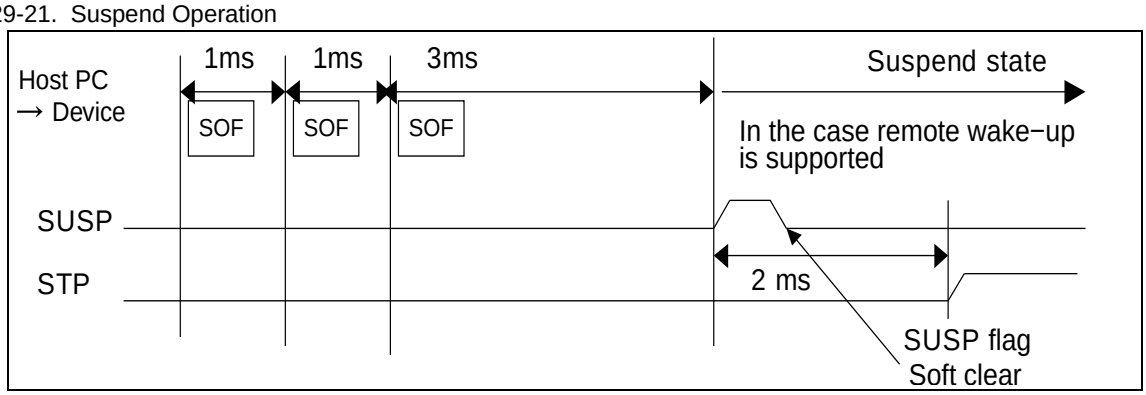
A USB device must have bus power supply configuration with a power consumption less than 500μA in SUSPEND mode. This section describes the USB device procedure to enter SUSPEND mode and then STOP mode.

### *29.4.4.1 USB function SUSPEND mode*

When the USB device core detects a SUSPEND mode request, the SUSP flag of the UDCSn register is set.

The figure below shows an example of the suspend operation:

Figure 29-21. Suspend Operation




**SUSPEND mode operation**

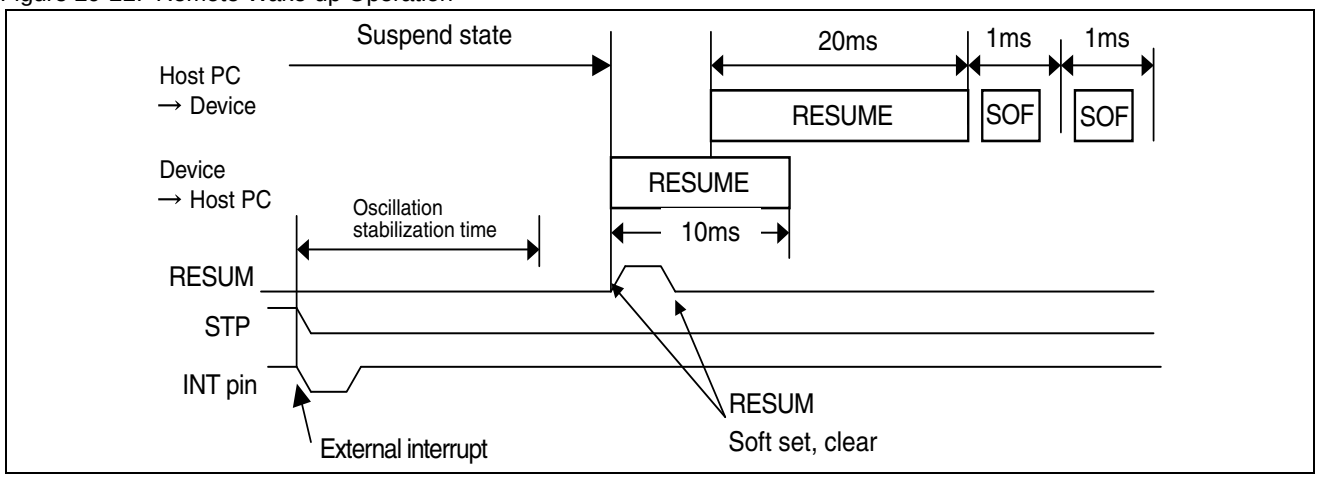
The USB function detects a SUSPEND mode when there is no activity for at least 3 ms on the bus. If so, the SUSP flag in the UDCSn register is set which causes an interrupt. If a USB device supports remote wake-up, wait for another 2 ms (which blocks remote wake-up during this time period), and then set the usb device to STOP mode.

## 29.4.5 Wake-up Function

To change a USB device from SUSPEND mode to WAKE-UP mode, the USB protocol provides two different options: Remote wake-up from device and wake-up from HOST.

### Remote wake-up by device

Figure 29-22. Remote Wake-up Operation

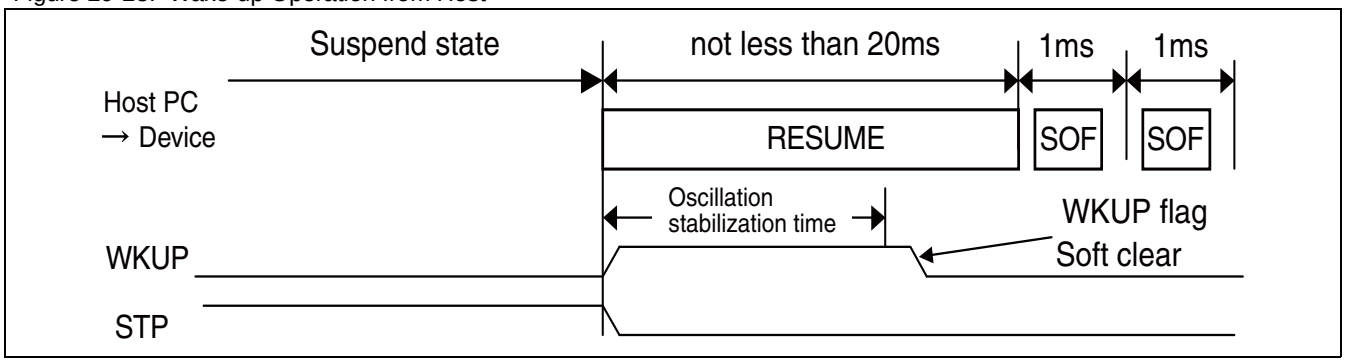



A USB device must performed the following steps:

1. Recover the USB device from STOP mode through an external interrupt.
2. Set the UDCCn:RESUM bit.
3. Clear the UDCCn:RESUM bit.

### Wake-up from Host

Figure 29-23. Wake-up Operation from Host




For the devices, the following steps are required:

1. Oscillation stabilizing time setting must not exceed 10 ms.
2. WKUP interrupt triggers the entry into an interrupt routine that clears the WKUP flag of UDCSn (interrupt cause) and then returns to main program.

## 29.4.6 DMA Transfer Function

Data transfers between USB function buffers and internal RAM is possible with two different DMA transfer modes: Packet Transfer mode and Data Number Automatic Transfer mode. In Packet Transfer mode the DMA transfer data size is to be set

for each packet one by one, on the other hands, in Data Number Automatic Transfer mode the DMA transfer data size for all packets is to be set once.

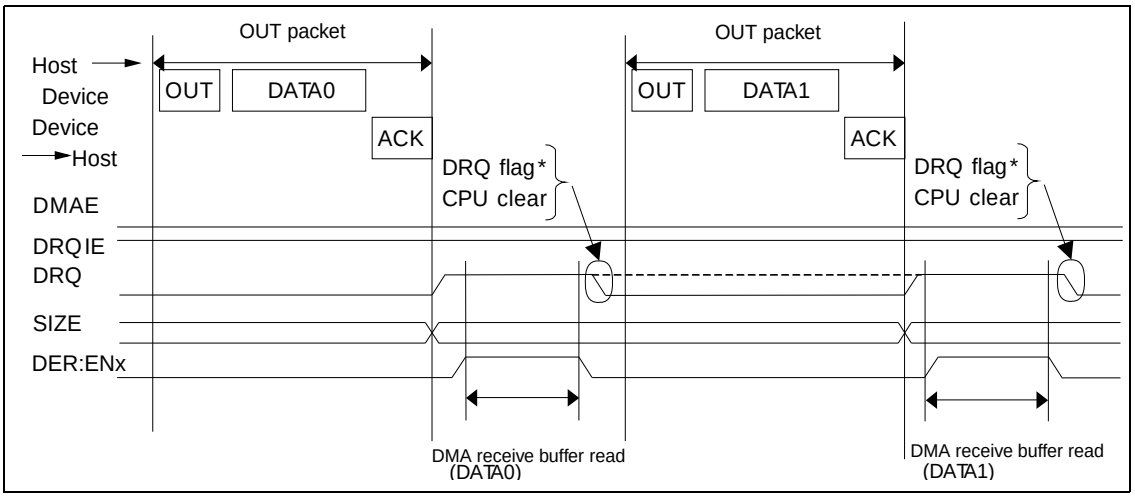
### *29.4.6.1 Packet Transfer Mode*

The packet transfer mode performs transfer by setting the number of data per packet to be transferred by the DMA and clearing the interrupt cause when the transfer is complete. The transfer mode can access any endpoint buffer.

Buffer access timings in OUT and IN directions are described below.

**OUT direction (HOST → device) transfer**

Figure 29-24. OUT Packet Transfer




In OUT- direction transfer, the USB device performs the following steps:

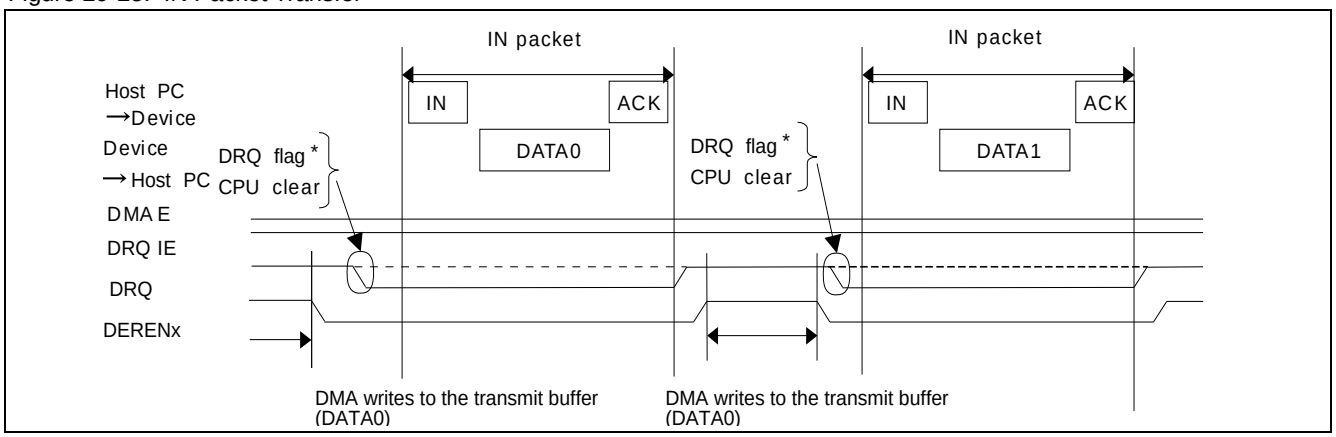
1. When the DRQ flag is set, the interrupt routine is called and the number of received data bytes must be checked (in EPxSn:PKS)
2. Set the number of data to be transferred in the data counter register DCT of DMA and the DMA is enabled by setting the DER register.
3. Once the transfer is complete, clear the corresponding DRQ flag in the EPxSn registers and the corresponding flag in the DSR register of the DMA.

**Note*:**

Each of the endpoints EP1 to EP5 consists of double buffers. DRQ can be cleared only when one buffer that is currently not being accessed is empty and data has been read from the other buffer. It cannot be cleared (even if "0" is written to it) when the buffer which is not being accessed has data left to be read (Dotted line status). In this case it continuously enters DRQ interrupt process.

**IN direction (device → host) transfer.**

Figure 29-25. IN Packet Transfer




In IN- direction transfer, a USB device performs the following steps:

1. Once the DRQ flag is set, the device enters the interrupt routine in which the number of data to be transferred in an IN packet is set in the data counter register DCT of the DMA. It enables the DMA by setting the DER register and data transfer from RAM to EndPoint buffer is triggered.
2. Once the DMA transfer is complete, clear the corresponding DRQ flag in the EP1Sn to EP5Sn registers and the corresponding interrupt flag in the DSR register of the DMA . The device then returns from the interrupt process.

**Note*:**

Each of endpoints EP1 to EP5 consists of double buffers. DRQ can be cleared only when one buffer that is currently not being accessed has already data written into it and data has been written to the other buffer. It cannot be cleared (even if "0" is written to it) when the buffer which is not being accessed is empty (dotted line status). In this case it continuously enters DRQ interrupt process.

### *29.4.6.2 Data Number automatic Transfer Mode*

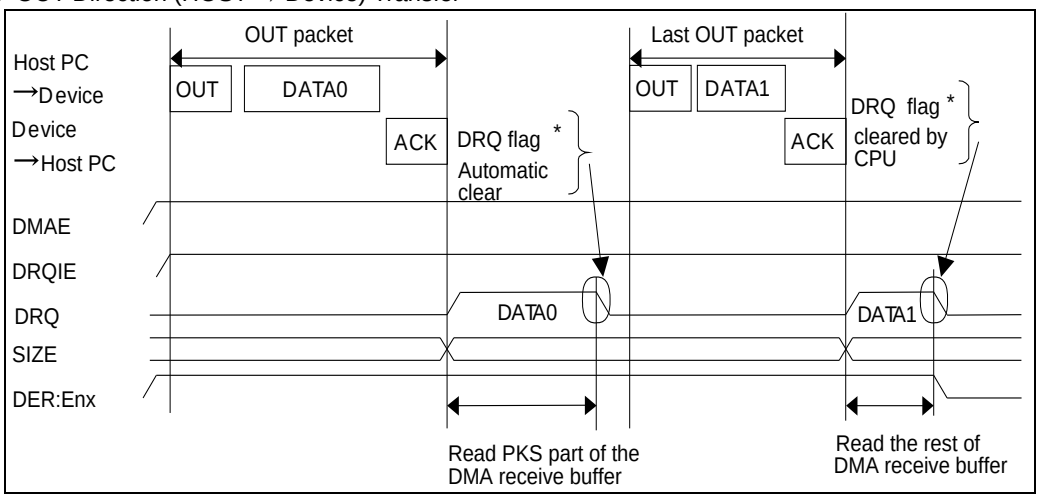
In this mode, the total number of data to be transferred is set in DMA and the transfer enable bit is set in advance. When EPxCn:DMAE bit is enabled and the DRQ is set after data from the Host is received successfully (data is ready to be read from the receive buffer), DRQ is automatically cleared after the data equal to the number of bytes indicated in the EPxSn:PKS is transferred. (Whether the DRQ flag is actually cleared depends on the fact that both buffers in the double buffer are empty or full). Similar process is repeated until the exact number of data to be transferred as defined in the DMA have been effectively transferred except for the last packet transfer. For the last packet transfer under the Data Number Automatic Transfer mode, the DRQ interrupt cause is not cleared automatically but is cleared by CPU in the interrupt service routine.

If the device performs the next transfer, it sets DMA again and enables DMA when a CPU interrupt is raised after the last data has been transferred, and returns from the CPU interrupt. Since the data number automatic transfer mode is used for DMAE=1, only buffer access to endpoint 1 to 5 is enabled.

Buffer access timings in OUT/IN directions are described below:

### OUT direction (HOST → device) transfer

Figure 29-26. OUT Direction (HOST → Device) Transfer




In OUT transfer directions, a USB device performs the following steps:

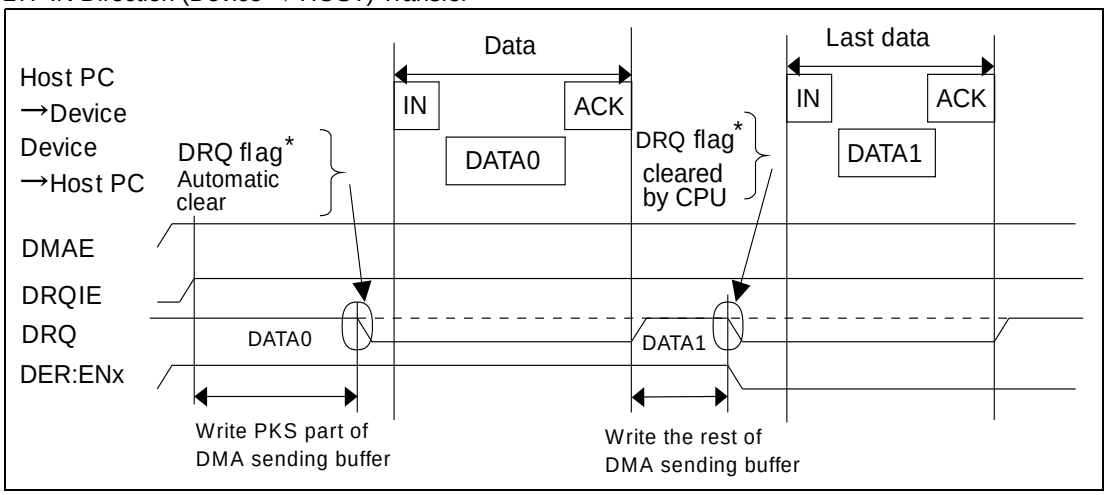
1. Setl the total number of data to be transferred in the data counter register DCT in DMA, and enable the DMA by setting the DER register.
2. Set EPxCn:DMAE and EPxSn:DRQIE.
3. Once the transfer is complete, a DMA interrupt routine is called and clear the flag (DRQ and DTEx) and to set the DMA for next transfer, and then return from the inturrupt routine.

### Note*:

Each of the endpoints EP1 to EP5 consists of double buffers. DRQ can be cleared only when one buffer that is currently not being accessed is empty and data has been read from the other buffer (automatic clear). It cannot be cleared (even if "0" is written to it) when the buffer which is not being accessed has data left to be read (Dotted line status). In this case it continuously enters DRQ interrupt process.

### IN direction (HOST → device) transfer

Figure 29-27. IN Direction (Device → HOST) Transfer




In IN transfer direction, a USB device performs the following steps:

1. Set the total number of data to be transferred in the data counter register DCT in DMA, and enable DMA by setting the DER register.

2. Set DMAE and DRQIE.
3. Once the transfer is complete, a DMA interrupt routine is called, then set DMA aagin and clear the flags, then return from the interrupt routine.

**Note*:**

Each of endpoints EP1 to EP5 consists of double buffers. DRQ can be cleared only when one buffer that is currently not being accessed has already data written into it and data has been written to the other buffer. It cannot be cleared (even if "0" is written to it) when the buffer which is not being accessed is empty (dotted line status). In this case it continuously enters DRQ interrupt process.

## 29.4.7 NULL Transfer Function

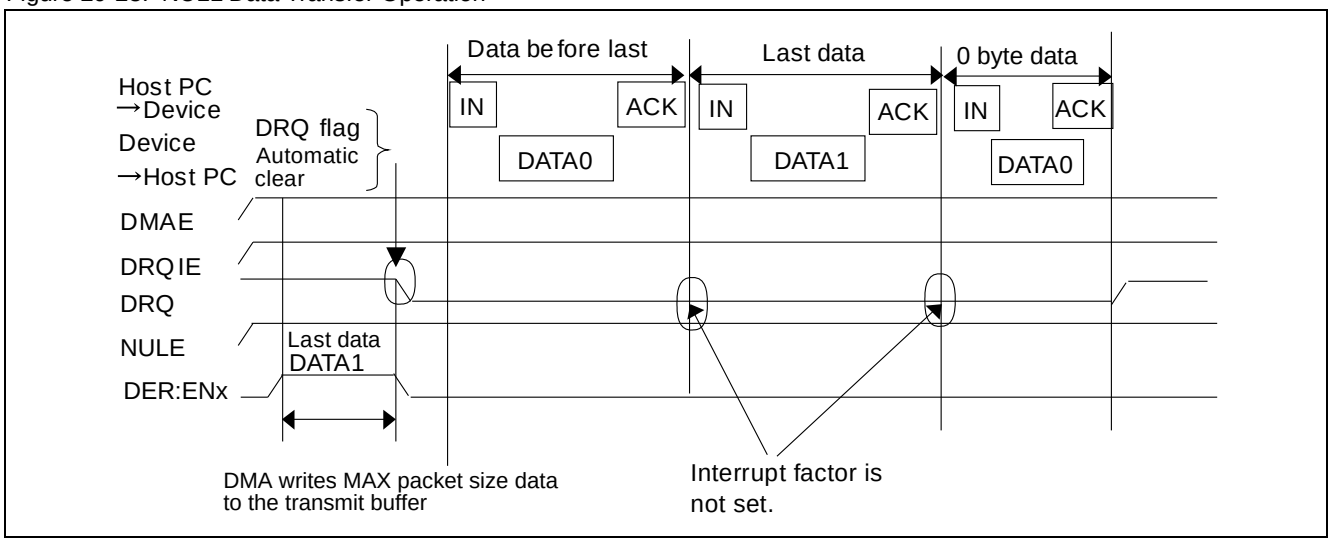
In case of Data Number Automatic IN type transfer if the total number of data bytes as set in the DCT register of DMA is already sent, its possible to automatically send a 0-byte data packet in response to the next IN transfer request from the host by enabling the NULL transfer mode.

### *29.4.7.1 NULL Transfer Mode*

In this mode the 0 byte data transfer is set automatically when the last IN transfer request from the HOST is detected in the case the Data number automatic transfer mode is set for IN transfers (DMAE=1) and the maximum size data set in PKS bits is written in to the transfer buffer. If the DCT register of DMA is decremented to 0 at the last data writing, a 0 byte data is transferred when the next IN transfer request from the host is detected. The DRQ interrupt flag is not set until 0-byte data is received by the host i.e. after receiving the ACK for the 0-byte transfer from the host. Buffer access timings are shown in the figure below.

**Only IN direction (device → host) transfer**

Figure 29-28. NULL Data Transfer Operation




For the device, DMAE, DRQIE, and NULE bits must be set.

# 30. USB Mini-Host

This chapter describes the functions and operations of USB Mini-host.

30.1 USB Mini-host features

30.2 Differences in USB Mini-host

30.3 USB Mini-host Block Diagram

30.4 USB Mini-host Registers

30.5 USB Mini-host operation

30.6 Token Flow Chart

## 30.1 USB Mini-host features

USB Mini-host provides minimum host functionalities required. It enables data to be transferred to and from a device without PC intervention.

### USB Mini-host features

USB Mini-host has the following features:

- Automatic detection of Full Speed / Low Speed devices.
- Support Full Speed transfer.
- Automatic detection of device connection and disconnection.
- Support of reset sending function to USB bus.
- IN/OUT/SETUP/SOF token support.
- Automatic transmission of handshake packet for IN token (excluding STALL).
- Handshake packet automatic detection at OUT token.
- Support a maximum data packet length of 256 bytes.
- Supports various error handling (CRC error/toggle error/time-out).
- Wake-up function support.

## 30.2 Differences in USB Mini-host

Description of the differences between the standard USB host and USB Mini-host.

### USB Host versus USB Mini-host

| | | Host | Mini-host |
|---|---|---|---|
| Support Hub | | ❍ | × |
| Transfer | Bulk transfer | ❍ | ❍ |
| | Control transfer | ❍ | ❍ |
| | Interrupt transfer | ❍ | ❍ |
| | Isochronous transfer | ❍ | × |
| Transfer speed | Low Speed | ❍ | × |
| | Full Speed | ❍ | ❍ |
| PRE packet support | | ❍ | × |
| SOF packet support | | ❍ | ❍ |
| Error | CRC error | ❍ | ❍ |
| | Toggle error | ❍ | ❍ |
| | Time-out | ❍ | ❍ |
| | Max. packet < Receive Data | ❍ | ❍ |
| Detection of connection and disconnection of devices | | ❍ | ❍ |
| Transfer speed detection | | ❍ | ❍ |

❍: Supported

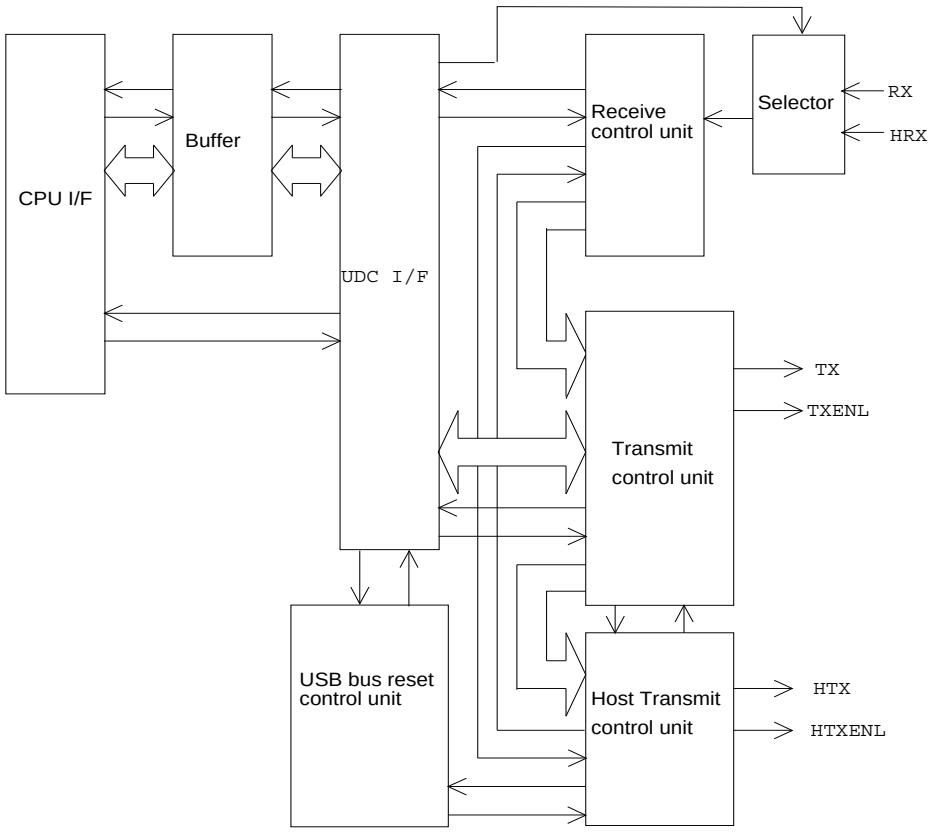
×: Not supported

## 30.3 USB Mini-host Block Diagram

Block diagram of USB Mini-host.

### Block Diagram of USB Mini-host

Figure 30-1. Block Diagram of USB Mini-host




Explanation of blocks:

| | |
|---|---|
| CPU I/F | : Interface circuit block with CPU |
| Buffer | : Buffer and buffer control circuit block |
| USB I/F | : Interface circuit block with USB |
| USB bus reset control part | : USB bus reset and connection control block |
| RX control part | : Receive data serial to parellel conversion circuit and RX control block |
| TX control part | : Transmission data parellel to serail conversion circuit and TX control circuit block for the USB Function |
| Host TX control part | : Transmission data parallel to serial conversion circuit and TX control circuit block for Mini Host |
| TX | : USB Function transmission data signal |
| RX | : USB Function received data signal |
| TXENL | : Function transmit and receive direction signal |
| HTX | : Mini−Host transmit data signal |
| HRX | : Mini_Host receive data signal |
| HTXENL | : Min−Host transmit and receive direction signal |

## 30.4 USB Mini-host Registers

This chapter describes all the registers of the USB Mini-host function.

### Register of USB Mini-host

Table 30-1. Registers of USB Mini-host

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RWKIRE | URIRE | CMPIRE | CNNIRE | DIRE | SOFIRE | URST | HOST | HCNTLn |

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | Reserved | | | | | SOFSTEP | CANCEL | RETRY | HCNTHn |

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | TCAN | Reserved | RWKIRQ | URIRQ | CMPIRQ | CNNIRQ | DIRQ | SOFIRQ | HIRQn |

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | LSTSOF | RERR | TOUT | CRC | TGERR | STUFF | HS | | HERRn |

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | Reserved | | | | SOFBUSY | SUSP | TMODE | CSTAT | HSTATEn |

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | FCMP | | | | | | | | HFCOMPn |

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RTIM0 | | | | | | | | HRTIMER0n |

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | RTIM1 | | | | | | | | HRTIMER1n |

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | Reserved | | | | | | RTIM2 | | HRTIMER2n |

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | Reserved | HADR | | | | | | | HADRn |

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | EOF0 | | | | | | | | HEOFLn |

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | Reserved | | EOF1 | | | | | | HEOFHn |

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | FRAME0 | | | | | | | | HFRAMELn |

Table 30-1. Registers of USB Mini-host

| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | Reserved | | | | | FRAME1 | | | HFRAMEHn |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | TGGL | TKEN | | | EDPT | | | | HTOKENn |

## 30.4.1 Host Control Register (HCNTn)

Host control register (HCNTn) specifies the USB operation mode and the interrupt settings.

### Host Control Register Low (HCNTLn)

Figure 30-2. Host Control Register Low (HCNTLn)

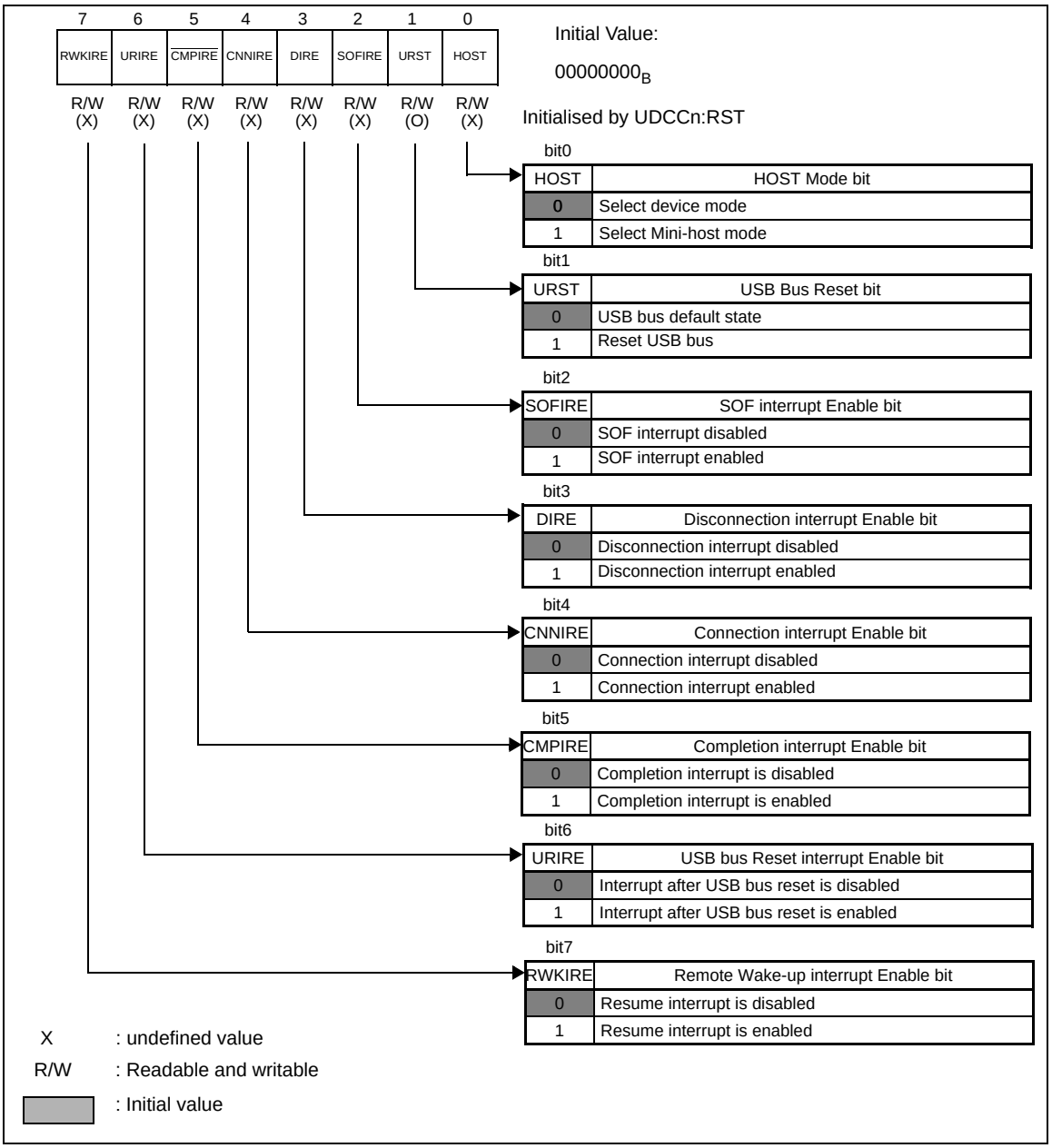
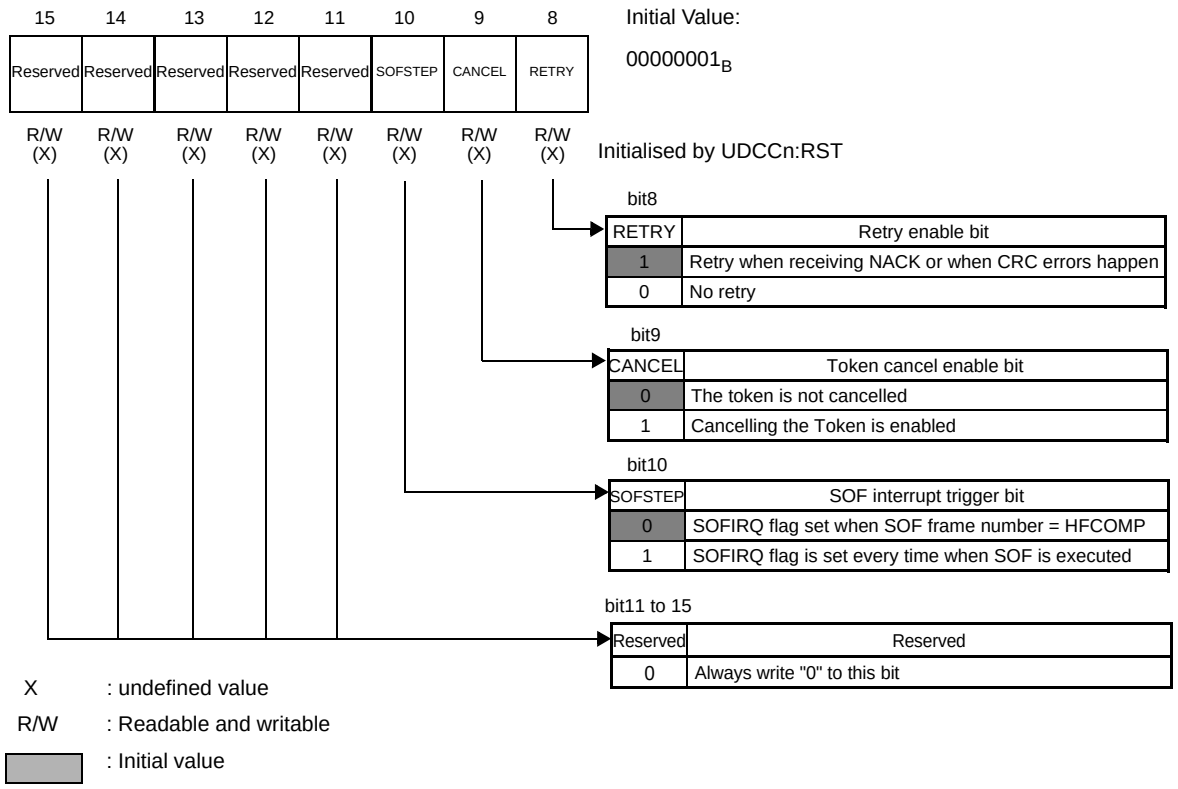


Figure 30-3. Host Control Register High (HCNTHn)




The following describes the function of each bit in the HCNTn registers:

| | Bit names | Function |
|---|---|---|
| Bit15-11 | Reserved | Please write "0" to these bits |
| Bit10 | SOFSTEP | This bit sets whether an interrupt due to SOF is generated every time SOF is executed (when set to "1").<br>The interrupt is enabled if HCNTLn:SOFIRE bit is set.<br>If SOFSTEP is "0", SOF Interrupt request flag HIRQn:SOFIRQ is set when the lower 8 bits of the SOF Frame number are equal to the value set in the SOF Inturruption Comparison Register (HFCOMPn). If SOFSTEP is "1", SOF Interrutpion Request Flag HIRQn:SOFIRQ is set whenever SOF is executed. But HIRQn:SOFIRQ bit is not set by the SOF token executed by setting HTOKENn:TKEN bits to "001B". This bit is not initialized with the RST bit in the UDC control register (UDCCn). |
| Bit9 | CANCEL | This bit is used to cancel a token.<br>If this bit is set to "1", when HTOKENn:TKEN bits is set to execute a token in EOF area (it is specified by the HEOFn register), the token is cancelled.<br>If this bit is set to "0", the token indicated by HTOKENn:TKEN bits is not cancelled.<br>To confirm if any token was cancelled, read HIRQn:TCAN bit.<br>UDCCn:RST does not initialize this bit. |
| Bit8 | RETRY | This bit is used to enable the retry of the token.<br>If this bit is set to "1", when a NAK is received from the usb-device or various errors occur (HERRn:RERR="1", HERRn:TOUT="1", HERRn:CRC="1", HERRn:TGERR="1", HERRn:STUFF="1"), USB mini-host retries the token.<br>The retry is executed for the time period specified in Retry Timer Setting Registers (HRTIMERHn, HRTIMERMn, HRTIMERLn).<br>This bit is not initialized with the RST bit in the UDC control register (UDCCn). |

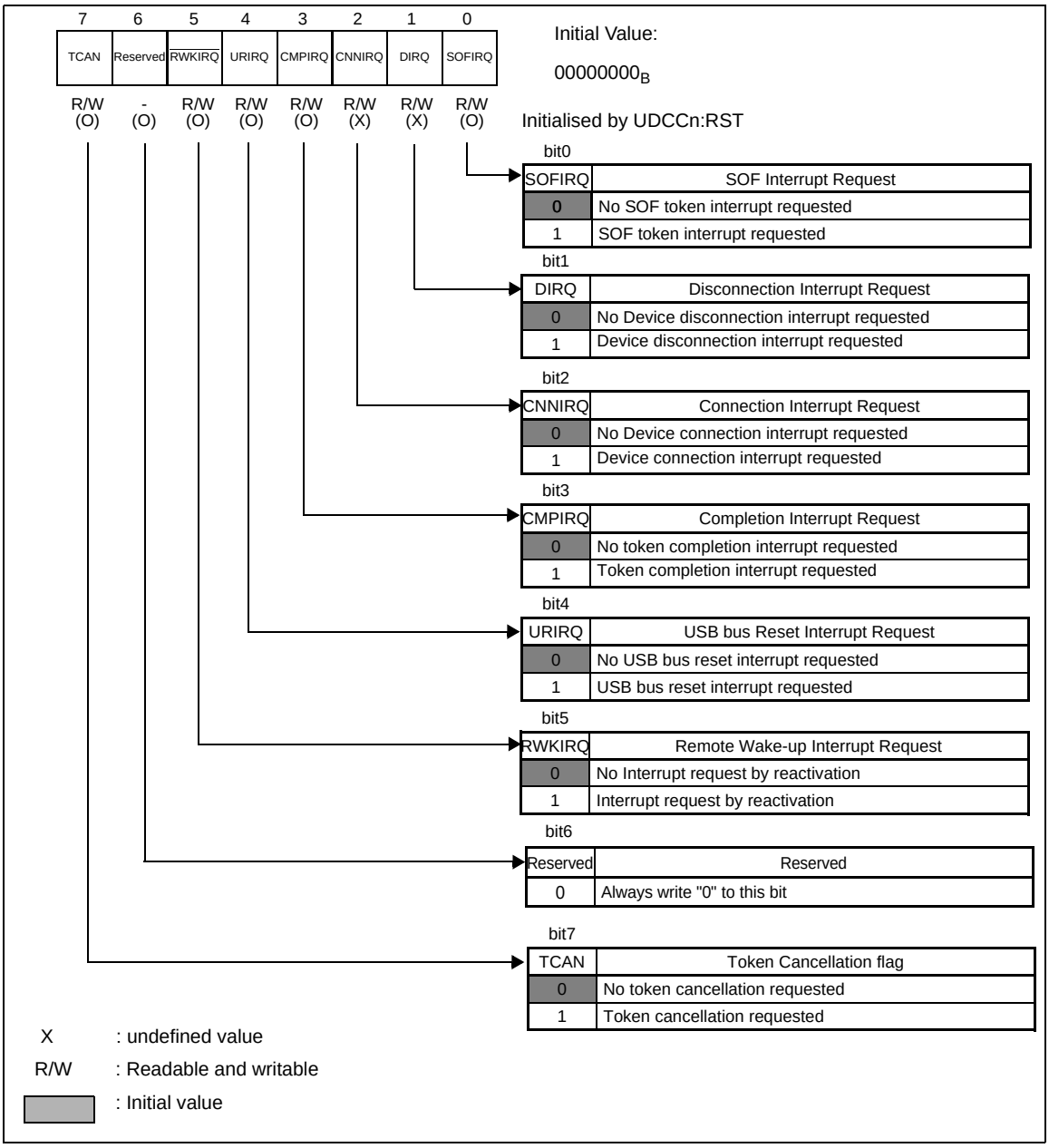
| | Bit names | Function |
|---|---|---|
| Bit7 | RWKIRE | This bit is used to enable Remote Wake-up Interrupt.<br>If this bit is set to "1", when the HIRQn:RWKIRQ bit is set to "1", the related interrupt is triggered<br>If this bit is set to "0", no interrupt is generated when the HIRQn:RWKIRQ bit is set to "1",<br>UDCCn:RST does not initialize this bit. |
| Bit6 | URIRE | This bit is used to enable USB bus Reset Interrupt.<br>If this bit is set to "1", when the HIRQn:URIRQ bit is set to "1", the related interrupt is triggered<br>If this bit is set to "0", no interrupt is generated when the HIRQn:URIRQ bit is set to "1",<br>UDCCn:RST does not initialize this bit. |
| Bit5 | CMPIRE | This bit is used to enable Completion Interrupt Request.<br>If this bit is set to "1", when the HIRQn:CMPIRQ bit is set to "1", the related interrupt is triggered<br>If this bit is set to "0", no interrupt is generated when the HIRQn:CMPIRQ bit is set to "1",<br>UDCCn:RST does not initialize this bit. |
| Bit4 | CNNIRE | This bit is used to enable Connection Interrupt<br>If this bit is set to "1", when the HIRQn:CNNIRQ bit is set to "1", the related interrupt is triggered<br>If this bit is set to "0", no interrupt is generated when the HIRQn:CNNIRQ bit is set to "1",<br>UDCCn:RST does not initialize this bit. |
| Bit3 | DIRE | This bit is used to enable Disconnection Interrupt.<br>If this bit is set to "1", when the HIRQn:DIRQ bit is set to "1", the related interrupt is triggered<br>If this bit is set to "0", no interrupt is generated when the HIRQn:DIRQ bit is set to "1",<br>UDCCn:RST does not initialize this bit. |
| Bit2 | SOFIRE | This bit is used to enable SOF Interrupt.<br>If this bit is set to "1", when the HIRQn:SOFIRQ bit is set to "1", the related interrupt is triggered<br>If this bit is set to "0", no interrupt is generated when the HIRQn:SOFIRQ bit is set to "1",<br>UDCCn:RST does not initialize this bit. |
| Bit1 | URST | Writing "1" to this bit resets the USB bus. It is cleared when USB bus reset is completed.<br>Reading "1" indicates that the USB bus is still being reset.<br>Writing "1" to this bit during UDCCn:RST="1" is ignored.<br>It is forbidden to set this bit when the HSTATEn:SUSP bit is set or while a token is being executed.<br>It is also forbidden to update HCNTLn or HCNTHn while this bit is set. |
| Bit0 | HOST | This bit controls the USB function to act as a Device or as a Mini-Host. Setting "1" to this bit selects the Mini-Host mode and setting "0" selects Function mode. Because it takes time to transit mode after updating this bit, please read this bit to confirm if the mode transition is completed.<br>This bit must be changed when RST bit is set to "1".<br>To change the function mode from device to host function, a disconnection of the HOST PC by setting the UDCCn:HCONX bit must be performed before updating the HOST bit.<br>To change the function mode from Host mode to Function mode, clear the HSTATEn:SOFBUSY bit to "0", set the HTOKENn:TKEN bits to "000B", and clear the HSTATEn:SUSP bit to "0", before updating the HOST bit.<br>UDCCn:RST does not initialize this bit. |

## 30.4.2 Host Interrupt Register (HIRQn)

The host interrupt register (HIRQn) displays all the interrupt request flags of USB mini-Host. The interrupt is enabled only when the corresponding interrupt enable bit in the host control register (HCNTLn/HCNTHn) is enabled except for the TCAN bit.

### Host Interrupt Register (HIRQn)

Figure 30-4. Bit Configuration of Host Interrupt Register (HIRQn)



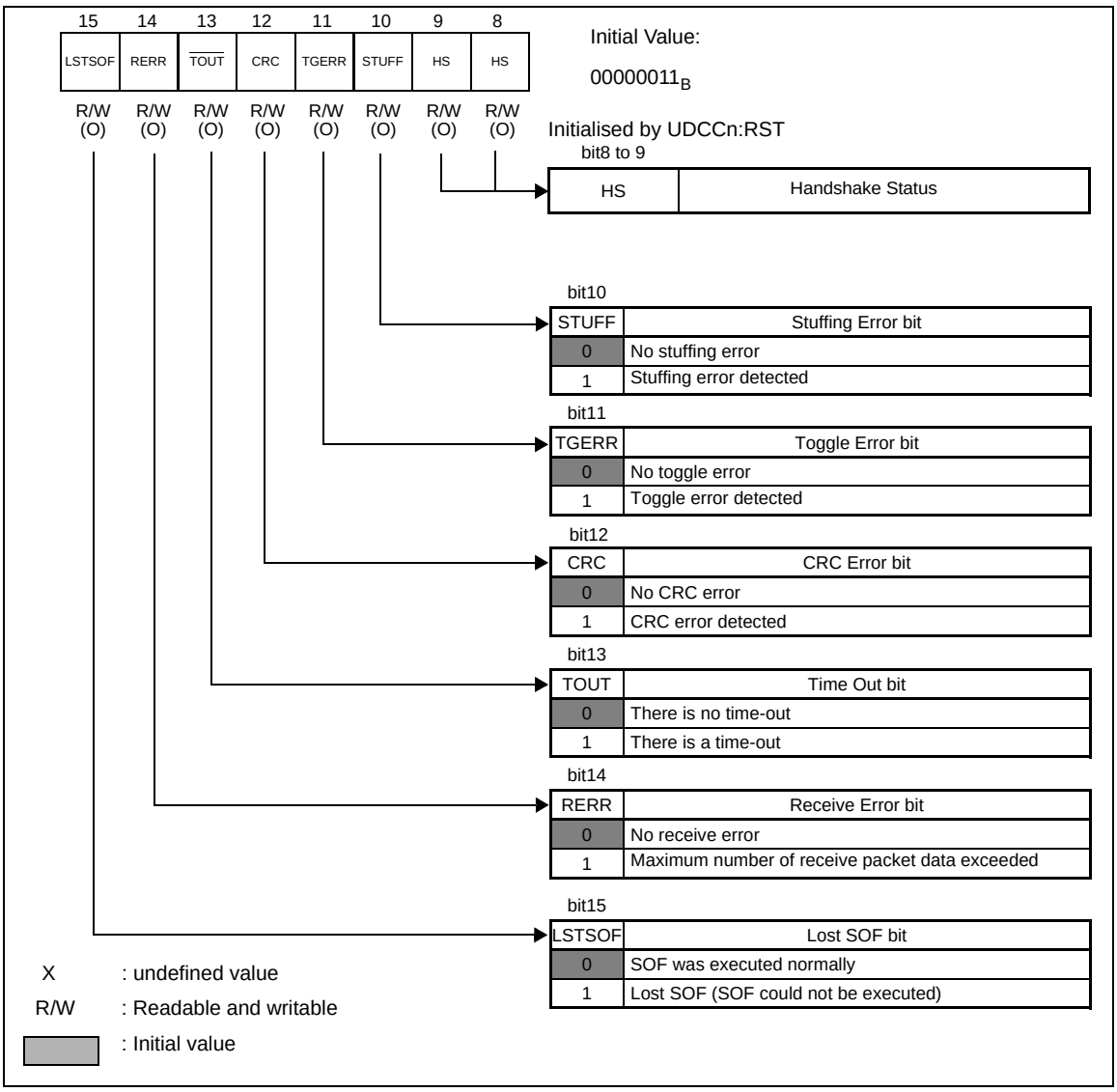
| | Bit names | Function |
|---|---|---|
| Bit7 | TCAN | When a token is cancelled by the setting of HCNTHn:CANCEL bit, this bit is set to "1".<br>If this bit is "0", it indicates that no token has been cancelled.<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>No interrupt is raised when this bit is set to "1". If you want to handle this bit in an intrrupt routine, please check this bit in the interrupt routine called by SOF interrupt.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized. |
| Bit6 | Reserved | Please write "0" to this bit. |
| Bit5 | RWKIRQ | This bit is set to "1" when the resume operation is completed, and an interrupt is triggered if HCNTLn:RWKIRE bit is "1".<br>If this bit is "0", it doesn't indicate anything.<br>This bit is cleared by writing "0".<br>Writing "1" preserves the current state.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized. |
| Bit4 | URIRQ | This bit is set to "1" when the reset operation of USB bus has been completed, and an interrupt is triggered if HCNTLn:URIRE bit is "1".<br>If this bit is "0", it doesn't indicate anything.<br>This bit is cleared by writing "0".<br>Writing "1" preserves the current state.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized. |
| Bit3 | CMPIRQ | This bit is set to "1" when a token has been completed, and an interrupt is triggered if HCNTLn:CMPIRE bit is "1".<br>If this bit is "0", it doesn't indicate anything.<br>This bit is cleared by writing "0".<br>Writing "1" preserves the current state.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.<br>This flag is not set to "1" when the HIRQn:TCAN bit is set<br>to "1". |
| Bit2 | CNNIRQ | This bit is set to "1" when a device connection has been detected, and an interrupt is triggered if HCNTLn:CNNIRE bit is "1".<br>If this bit is "0", it doesn't indicate anything.<br>This bit is cleared by writing "0".<br>Writing "1" preserves the current state.<br>It is not initialized with the UDCCn:RST bit.<br>This bit works also in the USB device mode, not only in the USB mini-host mode. |
| Bit1 | DIRQ | This bit is set to "1" when the device dissconnection has been detected, and an interrupt is triggered if HCNTLn:DIRE bit is "1".<br>If this bit is "0", it doesn't indicate anything.<br>This bit is cleared by writing "0".<br>Writing "1" preserves the current state.<br>It is not initialized with the UDCCn:RST bit.<br>This bit works also in the USB device mode, not only in the USB mini-host mode. |
| Bit0 | SOFIRQ | When a SOF transmission starts, in the case HCNTHn:SOFSTEP is "0", the FRAME Number in least significant 8 bits of the SOF token and HFCOMP are compared. And if a match is detected, this bit is set to "1", In the case the HCNTHn:SOFSTEP bit is "1", this bit is set to "1" every time a SOF transmission satarts.<br>When this bit is set to "1" and If the HCNTLn:SOFIRE bit is "1", an interrupt is generated<br>If this bit is "0", it doesn't indicate anything.<br>This bit is cleared by writing "0".<br>Writing "1" preserves the current state.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized. |

## 30.4.3 Host Error Status Register (HERRn)

The host error status register (HERRn) indicates whether an error has occurred while sending or receiving data in host mode.

### Host Error Status Register (HERRn)

Figure 30-5. Bit Description of the Host Error Status Register (HERRn)



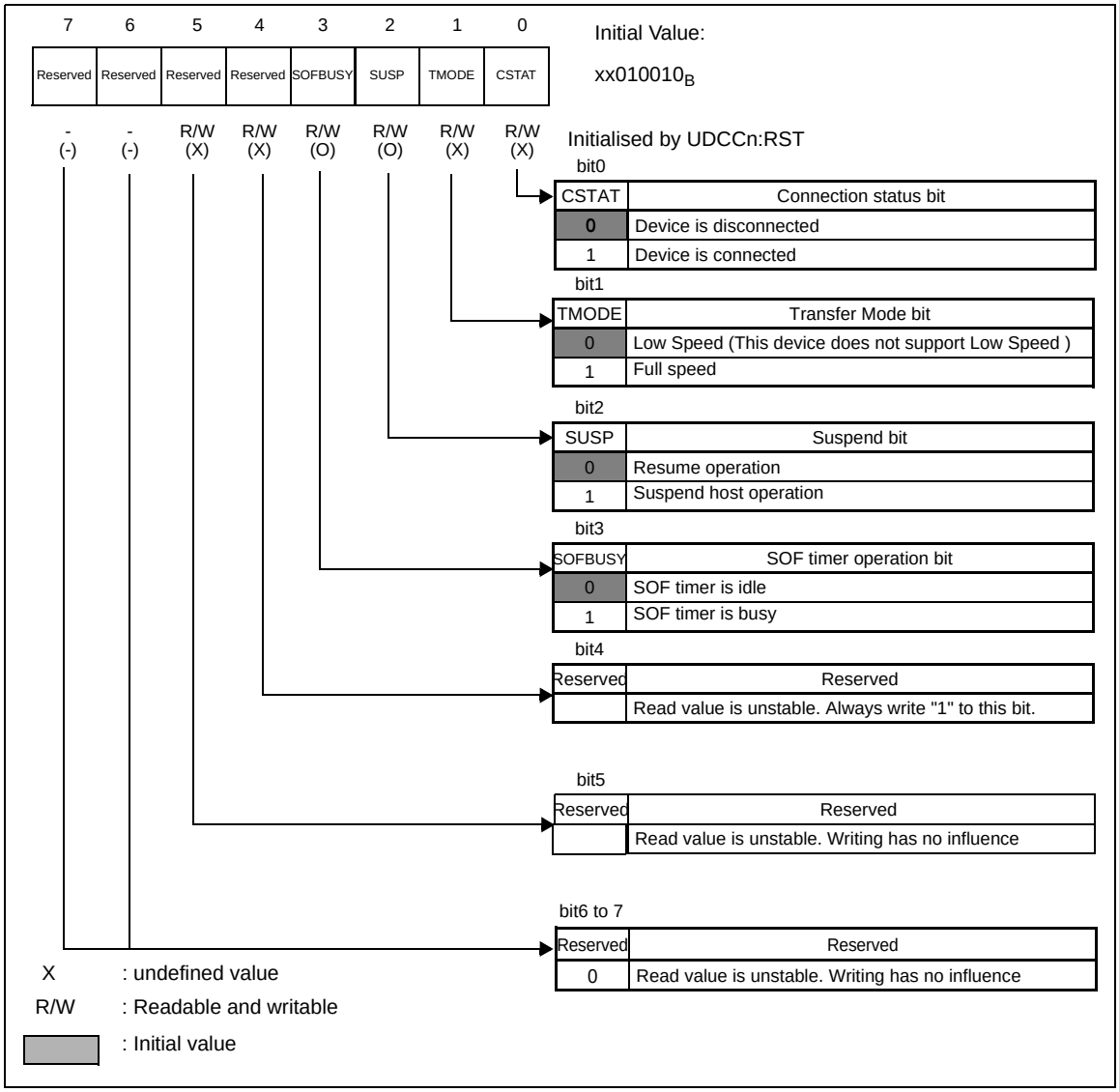
<table>
<tr><th></th><th>Bit names</th><th>Function</th></tr>
<tr><td>Bit15</td><td>LSTSOF</td><td>In the case the read value of this bit is "1", it indicates that a SOF token could not be executed because another token was running when trying to execute it in host mode. In the case the read value of this bit is "0", it indicates that no Lost SOF Error has been detected.<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
<tr><td>Bit14</td><td>RERR</td><td>In the case the read value of this bit is "1", it indicates that the number of packet received has exceeded a maximum Packet Size setting in host mode. When this bit is set to "1", the HERRn:TOUT bit is also set to "1" at the same time,  In the case the read value of this bit is "0", it indicates that no such a error has occured..<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
<tr><td>Bit13</td><td>TOUT</td><td>In the case the read value of this bit is "1", it indicates that no response from the device has been received for a certain time, in Host mode. In the case the read value of this bit is "0", it indicates that no time out has detected.<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
<tr><td>Bit12</td><td>CRC</td><td>In the case the read value of this bit is "1", it indicates that a CRC error has occurred in Host mode. When this bit is set to "1", the HERRn:TOUT bit is also set to "1" at the same time,  In the case the read value of this bit is "0", it indicates that no such a error has occured..<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
<tr><td>Bit11</td><td>TGERR</td><td>In the case the read value of this bit is "1", it indicates that the received toggle has not matched with the setting in Host mode. In the case the read value of this bit is "0", it indicates that such a error has occured.<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
<tr><td>Bit10</td><td>STUFF</td><td>In the case the read value of this bit is "1", it indicates that a CRC error has occurred in Host mode. When this bit is set to "1", the HERRn:TOUT bit is also set to "1" at the same time,  In the case the read value of this bit is "0", it indicates that no such a error has occured..<br>This bit is cleared by writing "0". Writing "1" to this bit is ignored.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
<tr><td>Bit9-8</td><td>HS</td><td>These bits indicate NULL when handshake operation is not performed due to any error, or when the SOF token triggered by settign the HTOKENn:TKEN bits is completed.<br>These bits are updated when transmission or reception is completed.<br>Handshake values are described below:<br><table><tr><th>Bit9</th><th>Bit8</th><th>Handshake</th></tr><tr><td>0</td><td>0</td><td>ACK</td></tr><tr><td>0</td><td>1</td><td>NAK</td></tr><tr><td>1</td><td>0</td><td>STALL</td></tr><tr><td>1</td><td>1</td><td>NULL</td></tr></table>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized.</td></tr>
</table>

## 30.4.4 Host State Status Register (HSTATEn)

The Host State Status Register (HSTATEn) indicates the status of the USB circuit such as connections to devices and transfer mode.

### Host State Status Register (HSTATEn)

Figure 30-6. Bit Configuration of the Host State Status Register (HSTATEn)



| | Bit names | Function |
|---|---|---|
| Bit7-5 | Reserved | Read value is unstable. Writing has no influence. |
| Bit4 | Reserved | Always write "1" to this bit. Read value is unstable. |
| Bit3 | SOFBUSY | This bit is set to "1" when a SOF token is executed by setting HTOKENn register. When the read value of this bit is "1", it means that the SOF timer is operating. When the read value of this bit is "0", it means that the SOF timer is not operating.<br>Writing "0" to this bit stops SOF timer. Writing "1" to this bit is ignored.<br>It takes time to stop the SOF timer after writing "0" to this bit. To confirm if the SOF timer has been stoped, read this bit.<br>UDCCn:RST bit must be set to "0" to update this bit. When UDCCn:RST bit is "1", this bit is initilaized. |
| Bit2 | SUSP | This bit sets the suspend status in the host mode.<br>Writing "1" to the bit sets suspend status.<br>Writing "0" to this bit when it is "1" or a change of USB bus to the k-state cancels the suspend status. In this case HIRQn:RWKIRQ is set.<br>It is forbidden to set the bit to "1" while the USB is operating (when resetting the USB bus or sending/receiving data or the SOF timer is operating).<br>In host mode, it is forbidden to stop the USB clock even in suspend status.<br>To update this bit the UDCCn:RST bit must be set to "0".<br>It is prohibited to set this bit to "1" in USB function mode.<br>If it is set to "1" before changing the mode from Host to Function mode, you must get out of suspend status by writing "0" into it before changing the mode. (please refer to Table 30.4-2). |
| Bit1 | TMODE | This bit indicates the transfer mode, i.e when "1" Full Speed and when "0" Low Speed (This device does not support Low Speed mode).<br>It is not initialized with the UDCCn:RST bit. |
| Bit0 | CSTAT | This bit indicates whether a device is connected or not, in Host mode. When the read value of this bit is "1", it means that the USB device is connected. When the read value of this bit is "0", it means that the USB device is not connected.<br>This bit is not initialized with the UDCCn:RST bit. |

Table 30-2. Suspend Setting

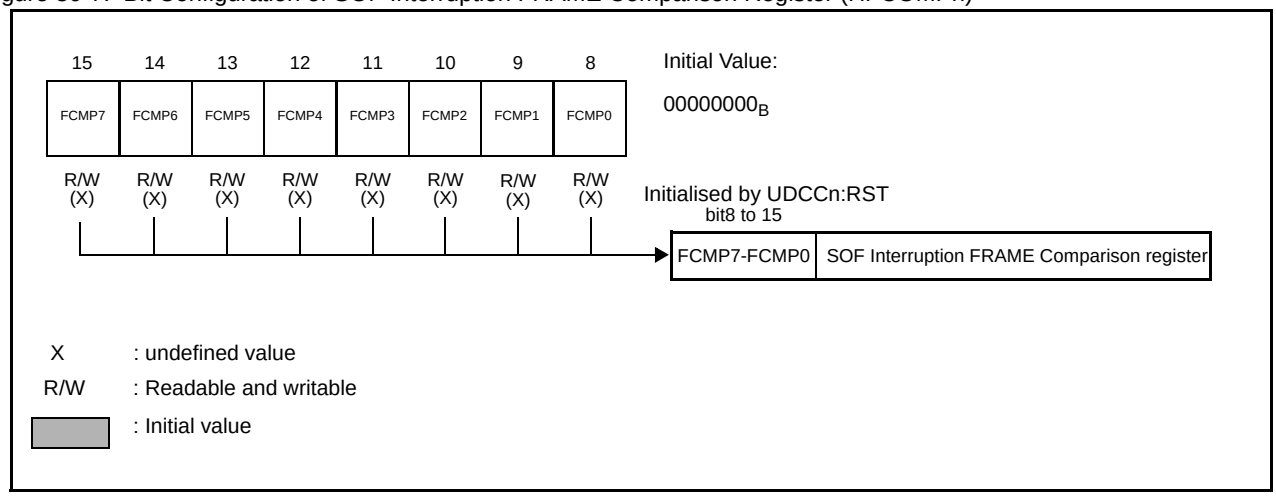
| SUSP | Operation |
|---|---|
| Writing "1" | Suspend |
| Writing "0" when it is set to "1" | Resume |
| Other | State maintenance |

## 30.4.5 SOF Interruption FRAME Comparison Register (HFCOMPn)

The SOF interrupt FRAME comparison register (HFCOMPn) is used to set data that is compared with the lower 8 bits of FRAME Number of SOF token.

### SOF Interruption FRAME Comparison Register (HFCOMPn)

Figure 30-7. Bit Configuration of SOF Interruption FRAME Comparison Register (HFCOMPn)



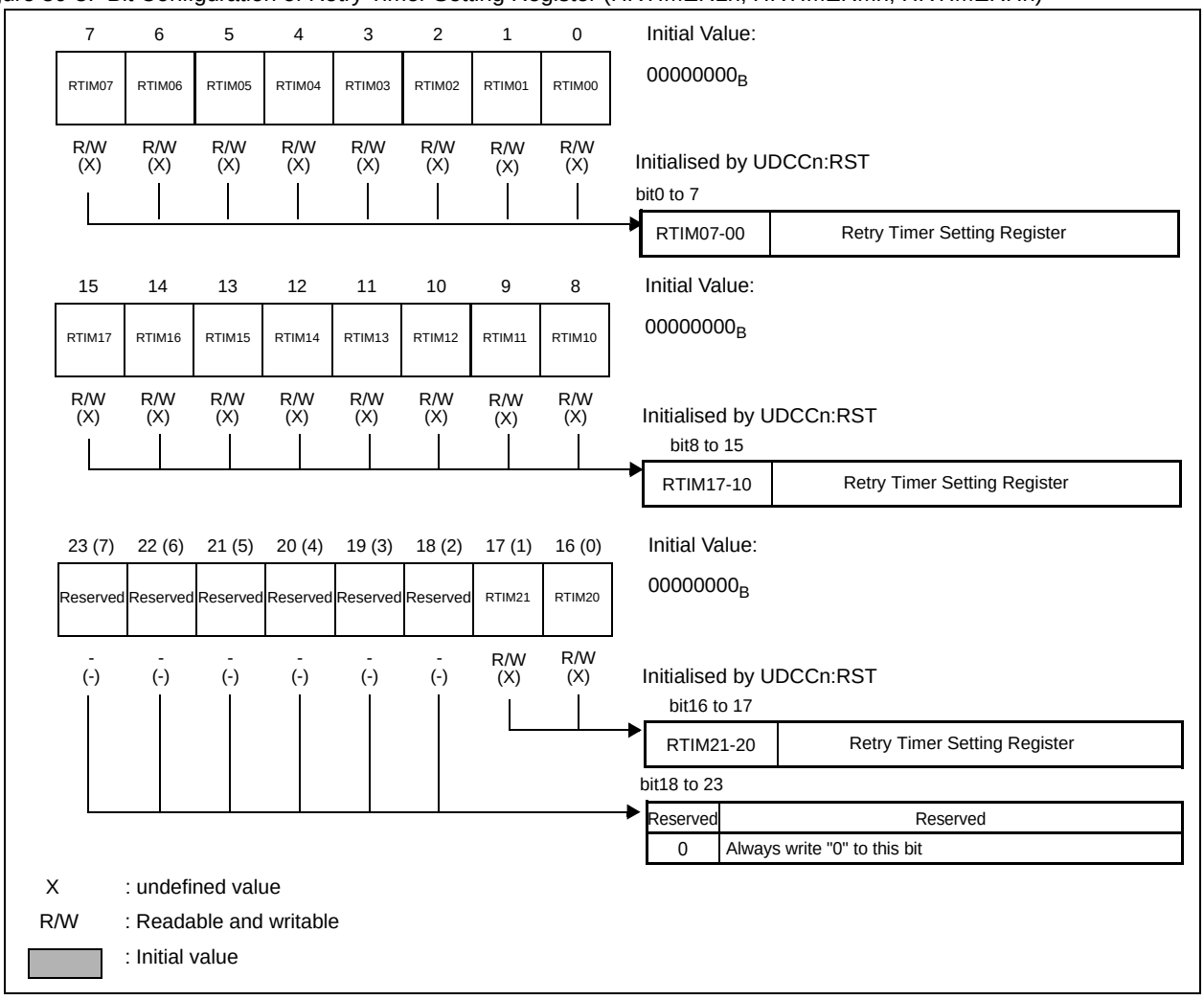

| | Bit names | Function |
|---|---|---|
| Bit15-8 | FCMP7-FCMP0 | This register holds the value to be compared with the lower 8 bits of the Frame Number of SOF token.<br>It is not initialized with the UDCCn:RST bit.<br>When a SOF transmission starts, in the case HCNTHn:SOFSTEP is "0", the FRAME Number in least significant 8 bits of the SOF token and HFCOMP are compared. And if a match is detected, HIRQn:SOFIRQ bit is set. In this case, if HCNTLn:SOFIRE bit is "1", an interrupt is generated.<br>In the case HCNTHn:SOFSTEP is "1", this register is ignored.<br>This register is not initialized with the UDCCn:RST bit. |

## 30.4.6 Retry Timer Setting Registers (HRTIMERLn, HRTIMERMn, HRTIMERHn)

The retry timer setting register (HRTIMERLn, HRTIMERMn, HRTIMERHn) is used to set a retry time period for a token.

### Retry Timer Setting Register (HRTIMERLn, HRTIMERMn, HRTIMERHn)

Figure 30-8. Bit Configuration of Retry Timer Setting Register (HRTIMERLn, HRTIMERMn, HRTIMERHn)



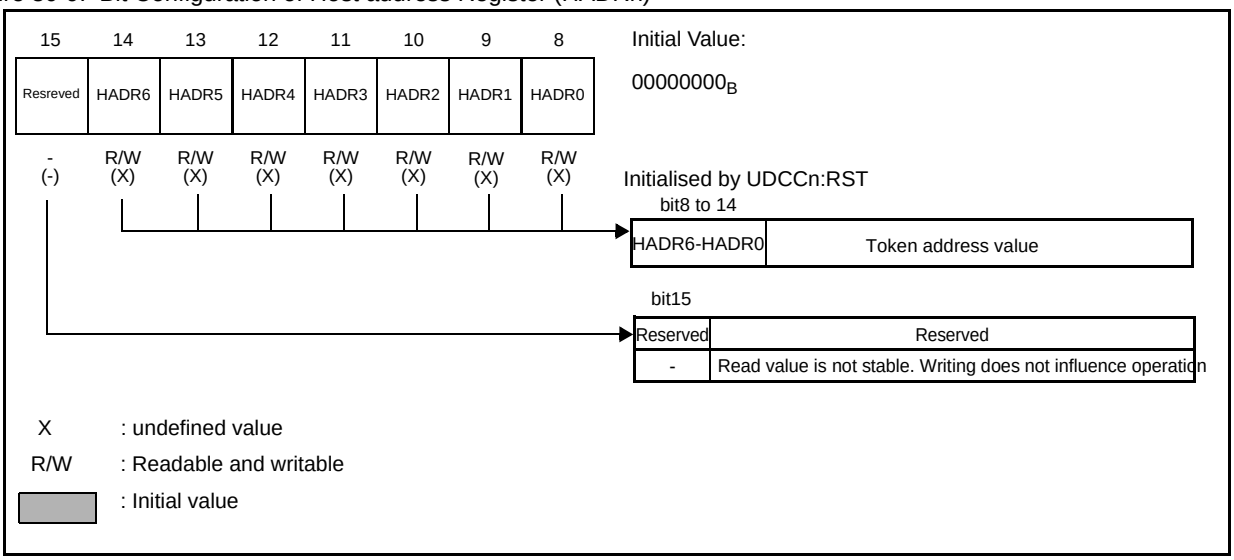

| | Bit names | Function |
|---|---|---|
| Bit17-0 | RTIM21-RTIM20<br>RTIM17-RTIM10<br>RTIM07-RTIM00 | These bits set the time to retry a token (after receiving a NACK). When the HCNTHn:RETRY bit is "1", a retry timer is activated after the token is started, and the timer is decremented by "1" at the transfer clock rate (12 MHz at Full Speed) until it reaches zero. When the counter reaches zero, the retry operation is finished after the current token is executed.<br>When a token retry occurs in an EOF area, the retry timer stops until the SOF is complete. The down counting is resumed after the SOF is executed<br>This register is not initialized with the UDCCn:RST bit |

## 30.4.7 Host Address Register (HADRn)

The host address register (HADRn) is a register used to set the address field when a token is to be sent.

### Host Address Register (HADRn)

Figure 30-9. Bit Configuration of Host address Register (HADRn)



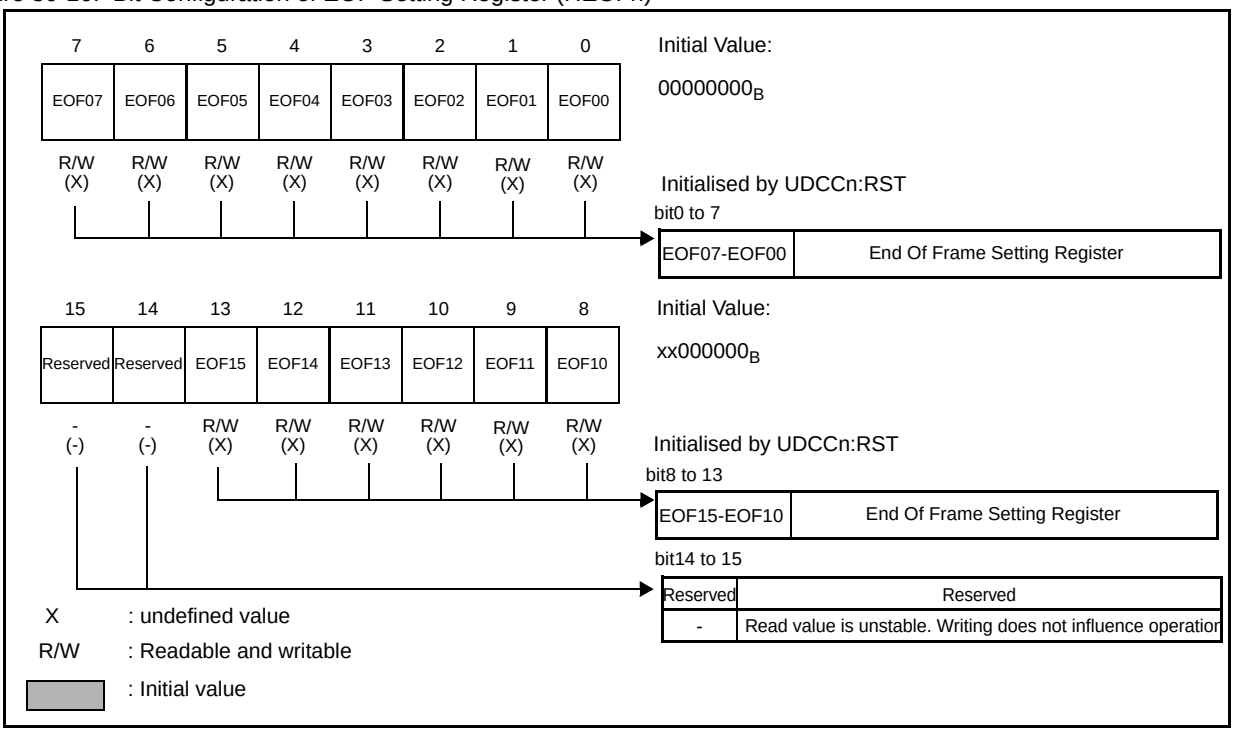

| | Bit names | Function |
|---|---|---|
| Bit15 | Reserved | Read value is not stable. Writing does not have any influence. |
| Bit14-8 | HADR6-HADR0 | Set token address in this register.<br>It is not initialized with the UDCCn:RST bit.<br>This register is not initialized with the UDCCn:RST bit. |

## 30.4.8 EOF Setting Register (HEOFn)

The EOF setting register (HEOFn) defines a time period for which a token is inhibited before the execution of the SOF token.

### EOF Setting Register (HEOFn)

Figure 30-10. Bit Configuration of EOF Setting Register (HEOFn)



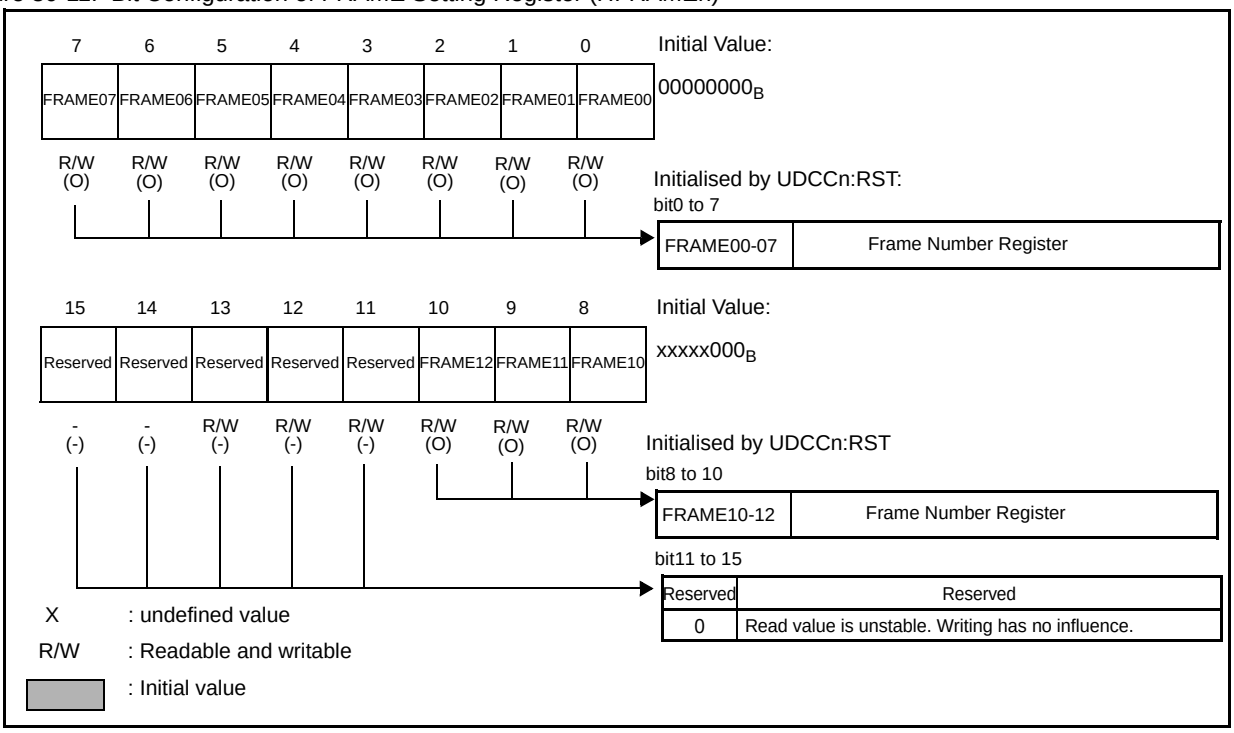

| | Bit names | Function |
|---|---|---|
| Bit15-14 | Reserved | Read value is unstable. Writing does not have any influence. |
| Bit13-0 | EOF15-EOF10<br>EOF07-EOF00 | The EOF registers hold a time period during which the execution of a token is inhibited before the execution of SOF. If the value stored in the SOF timer turns out to be lower than value stored in the HEOF register, for any of an IN, OUT or SETUP token execution requests made, they will be run only after the SOF token is executed. This prevents simultaneous execution of any of those tokens with an SOF token.<br>This register is not initialized by setting the UDCCn:RST bit to "1".<br>The time unit of the HEOF register is one bit time. A margin longer than one packet length must be set.<br>e.g. for the MAXPKT=64 byte and Full Speed: (Token_length + packet_length + header + CRC)×7/6 + Turn_around_time = (34bit + 546bit)×7/6 + 36bit = 712.7 bits. As a result, $(2C9)_H$ is to be set in this register. |

## 30.4.9 FRAME Setting Register (HFRAMEn)

The FRAME setting register (HFRAMEn) is used to set the FRAME Number of SOF tokens. SOF is transmitted automatically every 1ms after HTOKENn:TKEN bits are set to SOF transfer (TKEN[2:0] = 100B).

### FRAME Setting Register (HFRAMEn)

Figure 30-11. Bit Configuration of FRAME Setting Register (HFRAMEn)




| | Bit names | Function |
|---|---|---|
| Bit15-11 | Reserved | Read value is unstable. Writing has no influence |
| Bit10-0 | FRAME00 - FRAME07<br>FRAME10 - FRAME12 | These bits are used to set the frame number of SOF token.<br>Before setting the HTOKENn:TKNEN bits to SOF (TKNEN[2:0] = 100B), the Frame Number must be set to these bits.<br>When the HSTATEn:SOFBUSY bit is "1" or an SOF token is being executed, write operation is inhibited.<br>The UDCCn:RST bit must be set to "0" to update the register. This bit is initialized when UDCCn:RST bit is "1". |

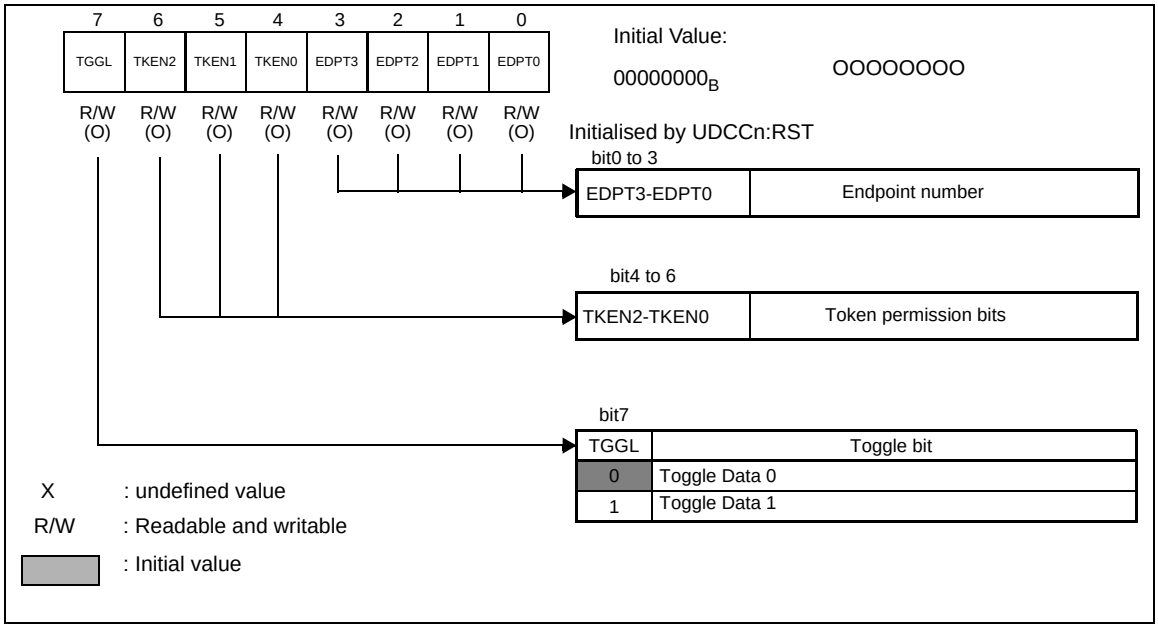
When the HTOKENn:TKEN bits to SOF activation are set, the SOF timer starts and, afterwards, an SOF is automatically sent out every 1 ms. The FRAME setting register is automatically incremented by 1 every time a SOF is completed.

## 30.4.10 Host Token Endpoint Register (HTOKENn)

The host token endpoint register (HTOKENn) is a register that sets a toggle, endpoint, and token.

### Host Token Endpoint Register (HTOKENn)

Figure 30-12. Bit Configuration of Host Token Endpoint Register (HTOKENn)




| | Bit names | Function |
|---|---|---|
| Bit7 | TGGL | The bit setting defines, at transmission, the toggle value to be sent.<br>At reception, the received toggle data is compared to the toggle data which this bit shows and used for error detection.<br>When updating this bit confirm that UDCCn:RST bit is set to "0" and the TKEN bits are set to "000". |
| Bit6-4 | TKEN2-TKEN0 | Setting these bits sends a token corresponding to its value. Once the operation is completed, these bits are changed to "000B", and the HIRQn:CMPIRQ bit is set to "1". If HCNTLn:CMPIRE bit is set an interrupt is generated.<br>When update TKNEN bits, the UDCCn:RST bit must be "0". These bits are initialized when UDCCn:RST bit is "1". and operation mode is to be Mini-Host mode.<br>If a token is issued again after a interrupt due to token is generated, you should wait for three cycles or longer in terms of USB transfer clock (12 MHz) before writing to the TKNEN bits.<br>Writing to the TKNEN bits will not transmit any token in disconnection status (HSTATEn:CSTAT = "0")<br>In the case that the SOF token is set to these bits (100B), setting of TGGL bit and ENDPT bit are ignored.<br>If the HSTATEn:SOFBUSY bit is set, writing "100" to these bits is forbidden. (Please refer to table 30.4-3 for token settings) |
| Bit3-0 | EDPT3-EDPT0 | Those bits define the transmit and receive endpoint of the device.<br>The UDCCn:RST bit must be set to "0" to update the register. |

**Note:**

The TGGL and EDPT bits are ignored when a SOF token is executed.

Table 30-3. Token Setting

| Bit6 | Bit5 | Bit4 | Operation |
|---|---|---|---|
| 0 | 0 | 0 | No token sent out |
| 0 | 0 | 1 | SETUP token sent out |
| 0 | 1 | 0 | IN token sent out |
| 0 | 1 | 1 | OUT token sent out |
| 1 | 0 | 0 | SOF token sent out |

# 30.5 USB Mini-host operation

This section describes the operation of USB Mini-Host.

30.5.1 Device Connection

30.5.2 USB Bus Reset

30.5.3 Token Packet

30.5.4 Data Packet

30.5.5 Handshake Packet

30.5.6 Retry Function

30.5.7 SOF Interrupt

30.5.8 Error Status

30.5.9 Packet End

30.5.10 Suspend Resume

30.5.11 Device disconnection

## 30.5.1 Device Connection

The method of detecting the connection of an external USB device by software is described.

### Setting of Mini-host Function

The HCNTLn:HOST bit must be set to "1" to turn the USB function as a host.

### Disconnection Status, Connection Status of an External USB Device

When the external USB device is disconnected, both pins UDP and UDM, are "L" because of the pull-down resistors. Then the HSTATEn:CSTAT bit is "0", and the TMODE bit is undefined. The HSTATEn:CSTAT bit becomes "1" when the external USB device is connected.

### Detection the connection of an external USB device

When an external USB device is connected, the HIRQn:CNNIRQ bit becomes "1". In the case that the HCNTLn:CNNIRE bit is "1", a device connection interrupt is generated. Clearing this bit to clear the interrupt. To detect the connection of the external USB device not through interrupt but through polling, a program so that it ensures that the HCNTLn:CNNIRE bit is set to "0" and then HIRQn:CNNIRQ bit is set to "1".

### Acquiring transfer speed of destination USB device and selecting the clock

HSTATEn:TMODE bit holds the information regarding the transfer speed of the connected USB device.

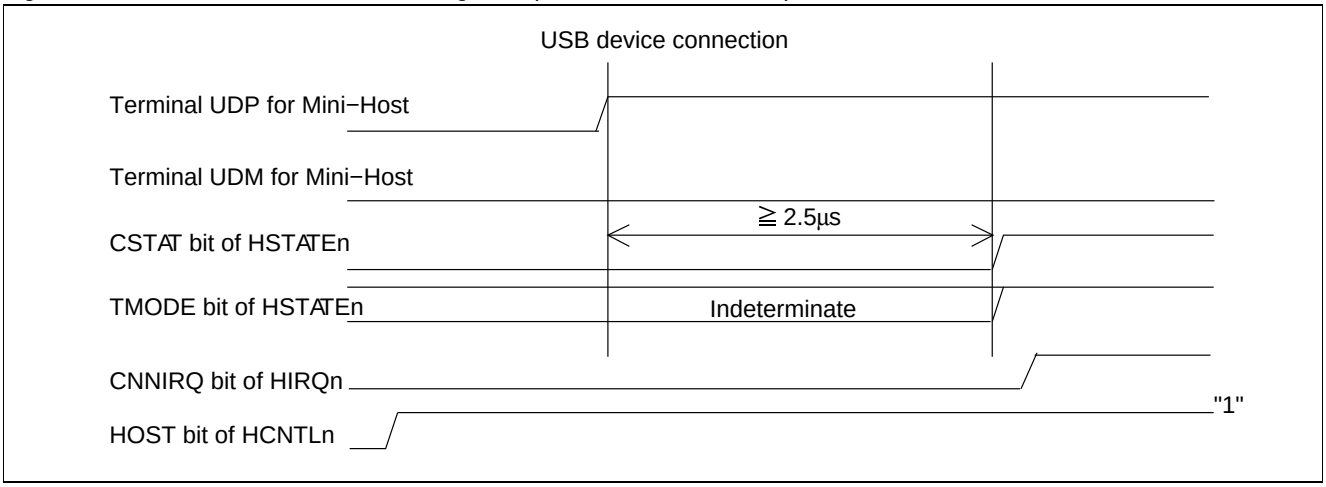
If the connected device is a Full Speed device then TMODE = "1" .

If the connected device is a Low Speed device then TMODE = "0" .

**Note:**

Low Speed mode is not supported by this device.

Figure 30-13. ² Connection detection timing example of the USB device speed




**Note:**

The HSTATEn:CSTAT bit is set to "1" in 2.5 us after the external USB device is connected. The HSTATEn:TMODE and HSTATEn:CSTAT bits are updated regardless of the setting of the HCNTLn:HOST bit.

## 30.5.2 USB Bus Reset

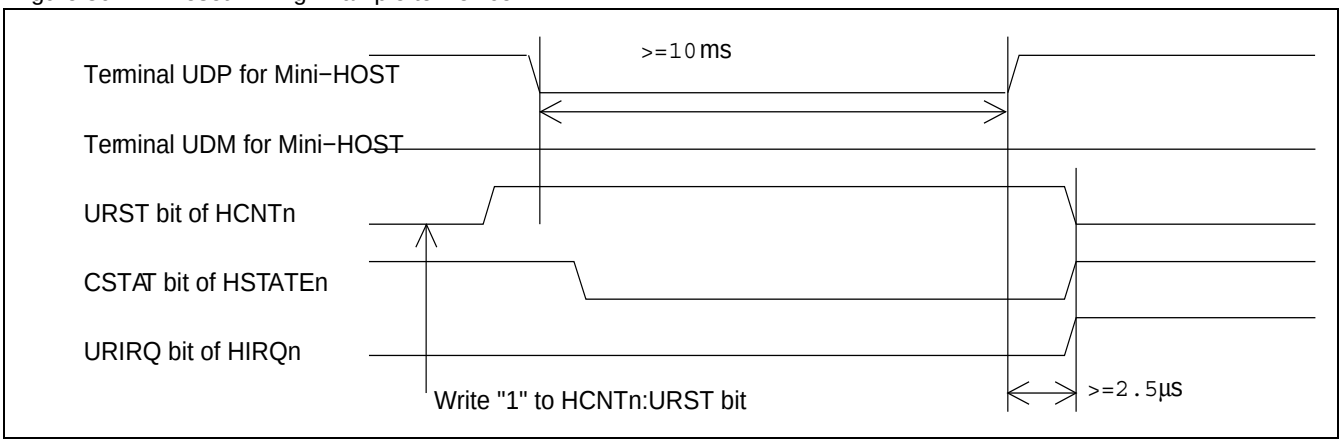
When the HCNTLn:URST bit is set to "1" in the host mode, USB Mini-Host drives SE0 for not less than 10 ms to indicate a reset condition to the device. When the USB bus reset is completed, the URST bit is cleared to "0" and HIRQn:URIRQ bit is set to "1". In this case, if the HCNTL: URIRE bit is enabled, an interrupt is generated. Clearing HIRQn:URIRQ bit clears the interrupt.

### Precautions to be taken when resetting the USB bus

Please note the following when resetting the USB bus:

1. Before the USB bus is reset, HSTATEn:CSTAT bit must be checked to confirm if the device is connected.
2. When the USB bus is being reset, the HSTATEn:CSTAT bit is automatically cleared to "0" and the USB device status turns "disconnected". In this case the HIRQn:DIRQ bit is not set to "1".
3. After the USB bus reset is completed, the HSTATEn:TMODE bit should be checked to confirm that the connected device is "Full Speed" device.

Figure 30-14. Reset Timing Example to Device




## 30.5.3 Token Packet

To execute any of IN, OUT or SETUP token, the token packet is started when the necessary data are set in the Host Token Register (HTOKENn) after settjng the HADRn register, DIR bit of EP1Cn register (or EP2Cn register) and PKS bit of EP1Cn register (or EP2Cn register). In case you want to send a SOF token, you must set necessary data in the Host Token Endpoint Register (HTOKEN) after configuring the FRAME Setting Register (HFRAME) and EOF Setting Register (HEOF). If registers (HADRn, EP1Cn, EP2Cn, HFRAMEn and HEOFn) have not been changed, it is not required to set them.

### Setting of Token Packet

In the host mode endpoint 1 and endpoint 2 are used as transmission buffer and reception buffer.

For the IN, OUT or SETUP token, set the destination address in the host address register (HADRn), and set the maximum transferable number of bytes in one packet is set in the EP1Cn:PKS bits (or the EP2Cn:PKS bits), and the the direction of the token transfer is to be set in the EP1Cn:DIR bit (or the EP2Cn:DIR bit). If the EP1Cn:DIR is set to "1", the buffer for endpoint 1 is used for OUT-direction transfer (transmit buffer) and the buffer for endpoint 2 is used for IN-direction (receive buffer). In this condition, the EP2Cn:DIR bit must be set to "0" (The EP1Cn:DIR and EP2Cn:DIR bits should be complement of each other i.e, if one is IN buffer the other should be OUT buffer). On similar lines, when EP1Cn:DIR bit is "0", the buffer for end-point 1 is used for IN-direction transfers and the buffer for endpoint 2 is used for OUT direction transfers. In this condition, the EP2Cn:DIR bit must be set to "1".

To excute a TOKEN, the setting is to be done in following sequence.

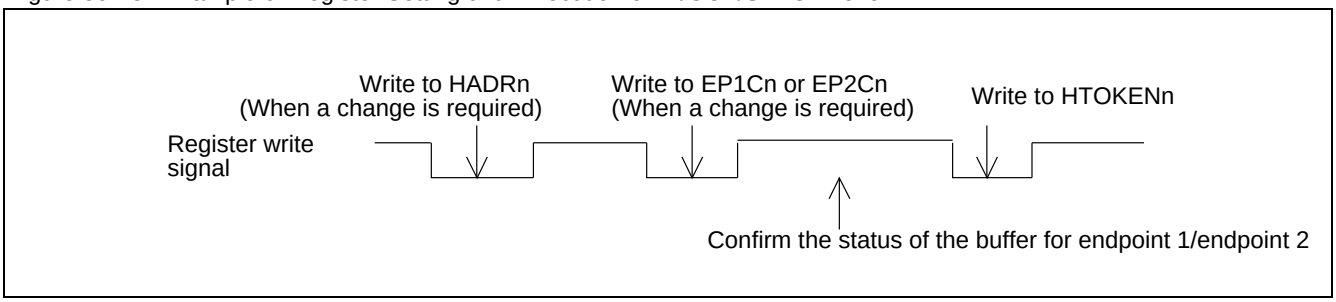
1) The direction is to be specified in to the DIR bits of EP1Cn and EP2Cn.

2) In the case that the target endpoint m (m=1 or 2) is for OUT transfer, write the transmition data in to the buffer of endpoint m, and clear the EPmSn:DRQ bit to "0". In the case the direction is IN transfer, readn the EPmSn:DRQ bit to confirm that it is "0".

3) Set the target endpoint, token and toggle data in to the host token endpoint register (HTOKENn).rget endpoint token and toggle data in the Host Token endpoint register (HTOKENn).

The USB circuit sends out a token packet in the following order: Sync, token, address, endpoint, CRC5, and EOP based on the specified token (a Sync, CRC5, and EOP are automatically sent). Once one packet is complete, the HIRQn:CMPIRQ bit is set to "1", and the HTOKENn:TKEN bit is set to $000_B$ (See 30.5.7 SOF Interrupt). If the HCNTLn:CMPIRE bit is "1" at this time, an interrupt occurs. Clearing HIRQn:CMPIRQ clears the interrupt.

Figure 30-15. Example of Register Setting until Execution of IN/OUT/SETUP Token




In the case of an SOF token, when an EOF time and FRAME number are set in the EOF setting register (HEOFn) and FRAME setting register (HFRAMEn) respectively, and the SOF token code to the HTOKENn:TKEN bits is written, a Sync, SOF token, FRAME number, CRC5, and EOP are sent out and the HSTATEn:SOFBUSY bit is set and the HFRAMEn is incremented. The HIRQn:CMPIRQ is also set (triggering an interrupt if enabled in the HCNTn:CMPIRE bit) and the HTOKENn:TKEN bits are cleared. Next or later SOF executed automatically does not cause an interrupt by the HIRQn:CMPIRQ.

SOF is automatically sent out every 1 ms as long as the HSTATEn:SOFBUSY bit is set to "1".

The conditions (SOF stop conditions) that set the HSTATEn:SOFBUSY bit to "0" are described below:

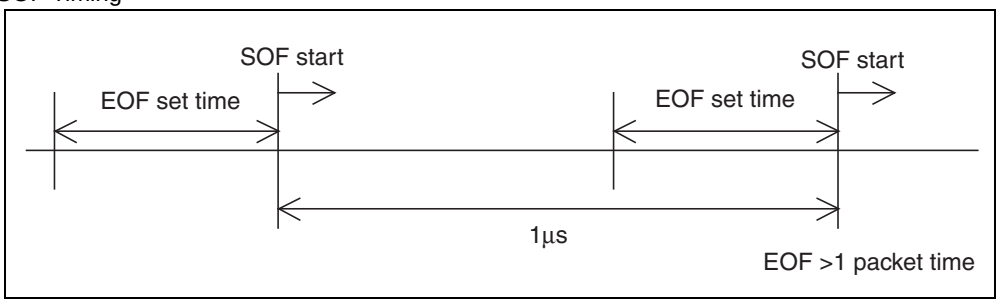
- Writing "0" to HSTATEn:SOFBUSY bit directly.
- A USB bus Reset (Writing "1" to HCNTn:URST).
- A HSTATEn:SUSP bit is set (by a direct write access).
- Disconnection of the device (HSTATEn:CSTAT bit gets cleared to "0").

To switch from host mode to device mode, at first clear the HSTATEn:SOFBUSY bit to "0", confirm that the read value of HSTATEn:SOFBUSY is "0" and read value of the HTOKENn:TKEN bits are "000B" and read value of the HSTATEn:SUSP is "0", and then clear HCNTLn:HOST bit to "0" to change the mode to device mode.

If you want to set back the HSTATEn:SOFBUSY bit to "1" again, you need to run an SOF token once again.

To prevent the simultaneous executions of a SOF token with other tokens, the EOF setting register is used to send a token (which is already set) after waiting for SOF completion, if the HTOKENn:TKEN bits are written between the EOF setting time and the SOF start time. The unit of time for the EOF setting register is one bit time. For example, when setting $10_H$ to the EOF setting register, the following time is required: 16x1/12 MHz = 1333.3 ns in Full speed mode. If the EOF set time is shorter than one packet time, the SOF execution may overlap other token execution. In this case, the HERRn:LSTSOF bit is set to "1" and the SOF is not executed. When the HERRn:LSTOF bit is set to "1", the data of the EOF setting register must be increased (See 30.4.8 EOF Setting Register (HEOFn)).

Figure 30-16. SOF Timing




## 30.5.4 Data Packet

In case a data packet is transmitted after a token packet has been sent, a toggle data is transmitted based on the HTOKENn:TGGL bit, and then the buffer data (from the transmit buffer of EndPoint1 or EndPoint2 that is selected according to the EP1Cn:DIR bit setting), the CRC16 data, and EOP is sent. In case of receiving the data packet, the HTOKENn:TGGL bit and the received toggle data are compared and in case of match, the received data is stored in the buffer of endpoint 1 or endpoint 2 depending on the EP1Cn:DIR bit setting and the CRC16 error is checked.

#### *30.5.4.1 Data Packet*

After sending a token packet, the data packet is executed in the following procedure:

**At Transmission**

- ²Sync is sent automatically.
- ²DATA0 is transmitted when the HTOKENn:TGGL bit is "0" and DATA1 is transmitted when the HTOKENn:TGGL bit is "1".
- ²When the EP1Cn:DIR is set to "1", the buffer for endpoint 1 is selected, otherwise endpoint 2 buffer is selected and all data is transmitted.
- EP1Cn:DIR bit is to be set to inverted value of EP2Cn:DIR bit, when the HCNTLn:HOST bit is "1". For example, when the EP1Cn:DIR bit is "0", the EP2Cn:DIR bit is set to "1".
- ²The CRC16-bits are sent.
- ²The EOP 2-bits are sent.
- ²The J State 1-bit is sent.

**At Reception**

- Sync is received
- ²The toggle data is received and is compared with the HTOKENn:TGGL bit.
- ²In the case they match, if EP1C:DIR bit ="0" the received data is stored in the buffer for endpoint 1 otherwise it is stored in the buffer of endpoint2.
- ²When the EOF is received, the CRC16 bit is checked.
- EP1Cn:DIR bit is to be set to inverted value of EP2Cn:DIR bit, when the HCNTLn:HOST bit is "1". For example, when the EP1Cn:DIR bit is "0", the EP2Cn:DIR bit is set to "1".

### 30.5.5 Handshake Packet

Handshake packet is used to inform the status of Mini-Host to the USB device.

**Handshake Packet**

The handshake packet is used by the receiver to inform from the receiver to the transmitter about the status of the transmission. The receiver transmits one of ACK, NAK or STALL status in a handshake packet to inform the transmitter whether it is in proper condition and can receive data properly. When the USB circuit receives a handshake packet, the type of received handshake packet is set in the HERRn:HS bits. When the handshake packet is transmitted, the type of transmitted handshake packet is set in the HERRn:HS bits.

### 30.5.6 Retry Function

This function enables the transfer error handling by retransmission.

**Retry Function**

If NAK or an error such as CRC error occurs when the packet  transfer is completed, USB MIni-Host continues to retry the transfer during a time period set in the retry timer register (HRTIMERLn, HRTIMERMn, HRTIMERHn) if the retry function is enabled (i.e. HCNTHn:RETRY is set to "1").

If an error except STALL and disconnected device happens (HERR:HS="01", HERR:RERR="1", HERR:TOUT="1", HERR:CRC="1", HERR:STUFF="1"), USB Mini-Host retries to process the token as long as the HCNTHn:RETRY bit is set to "1".

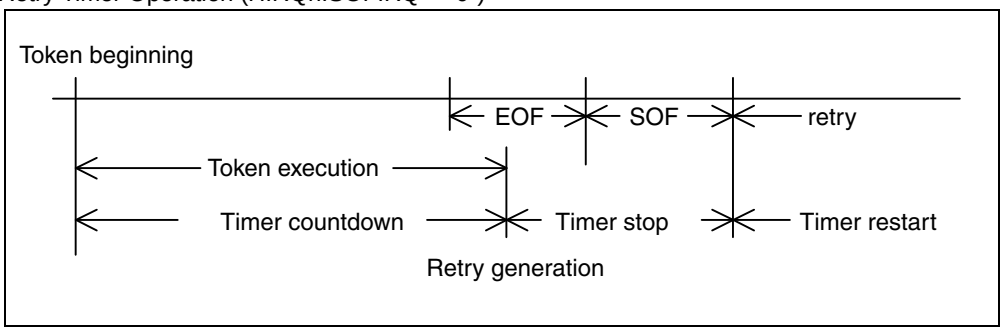
The retry is stopped by the following conditions:

- HCNTHn:RETRY bit is cleared to "0".
- Detecting "0" in the retry timer registers .
- ²The occurrence of an interrupt due to SOF (HIRQn:SOFIRQ = "1")

- ²ACK Detection
- ²Detection of device disconnection

The retry timer is activated when the token processing starts, counts down with one-bit transfer clock and stops counting when a retry happens in an EOF area. If HIRQn:SOFIRQ bit is "0" when a SOF token is completed, the retry timer restarts from the last value hold in the counter when it was stopped. When the SOF token is complete and the retry timer is "0", the packet is terminated and then HIRQn:CMPIRQ bit is set to "1".

Figure 30-17. Retry Timer Operation (HIRQn:SOFIRQ = "0")




When the retry operation is completed, end information on the complete packet is set in related registers.

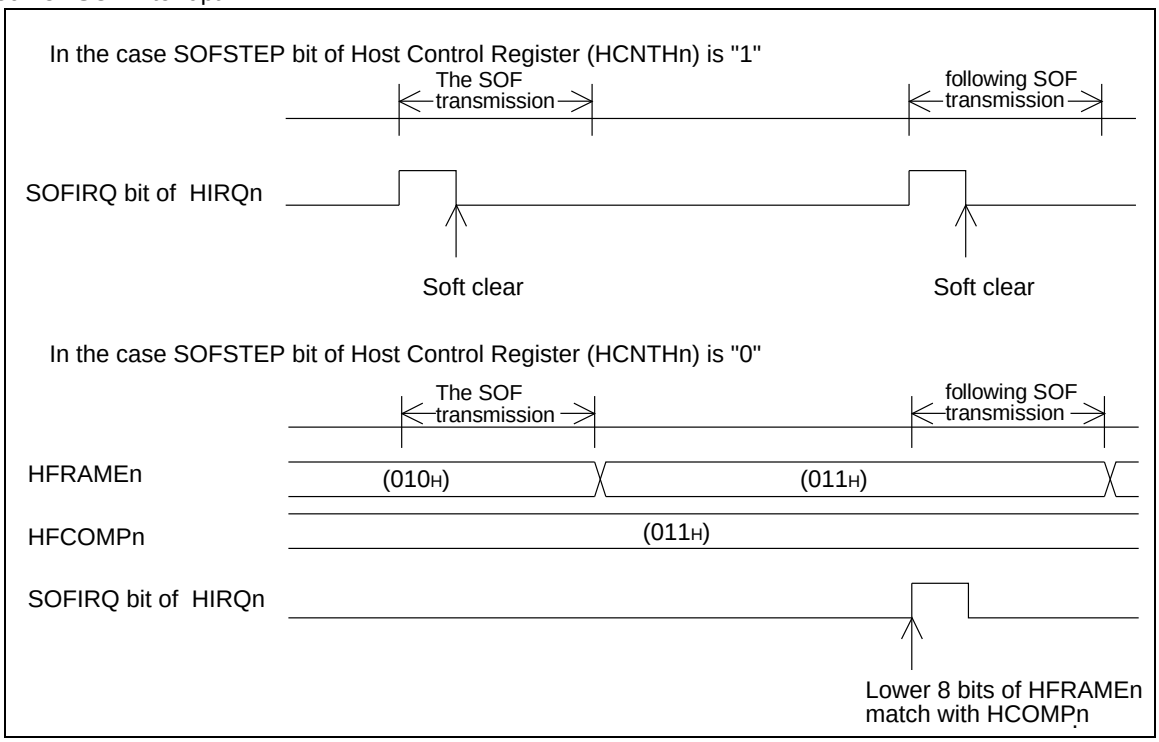
## 30.5.7 SOF Interrupt

This chapter describes the SOF interrupt processing.

### SOF Interrupt

When a SOF starts, HCNTLn:SOFIRQ bit is set to "1" according to HCNTHn:SOFSTEP bit and the SOF interruption FRAME comparison register (HFCOMPn) settings. If the HCNTLn:SOFIRE bit is set to "1", an interrupt is triggered when HIRQn:SOFIRQ bit is set. If SOFSTEP is "0", SOF Interrupt Request Flag HIRQn:SOFIRQ bit is set when the lower 8 bits of SOF Frame number are equal to the value set in the SOF Interruption Frame Comparison Register(HFCOMPn). If SOFSTEP is "1", SOF Interrupt Request Flag HIRQn:SOFIRQ bit is set whenever SOF is executed. But HIRQn:SOFIRQ bit is not set by the first SOF token (the SOF token is set by setting the HTOKEN:TKEN bits). The interrupt is enabled if HCNTLn:SOFIRE bit is set. The first SOF execution with the host token endpoint register (HTOKENn) setting does not set the HIRQn:SOFIRQ bit to "1".

Figure 30-18. SOF Interrupt




If you set the HCNTHn:CANCEL bit and a token other than SOF (in the HTOKENn) is set in the EOF area, and the HIRQn:SOFIRQ bit is set to "1" in the next SOF, the token required by HTOKENn is not executed and the HTOKENn:TKEN bits are set to $000_B$. In this case, the HIRQn:CMPIRQ bit is not raised.

Cancellation of a token can be known by checking the HIRQn:TCAN (Token Cancellation) flag after the HIRQn:SOFIRQ flag is raised. If the token is to be executed again, HIRQn:TCAN flag must be cleared, and the token settings must be specified again in the HTOKENn:TKEN field.

If the HCNTHn:CANCEL bit is set to "0", the token set in the HTOKENn register is executed after the SOF is sent.

Figure 30-19. Example of Token Cancel Operation when HCNTHn:CANCEL bit is "1".

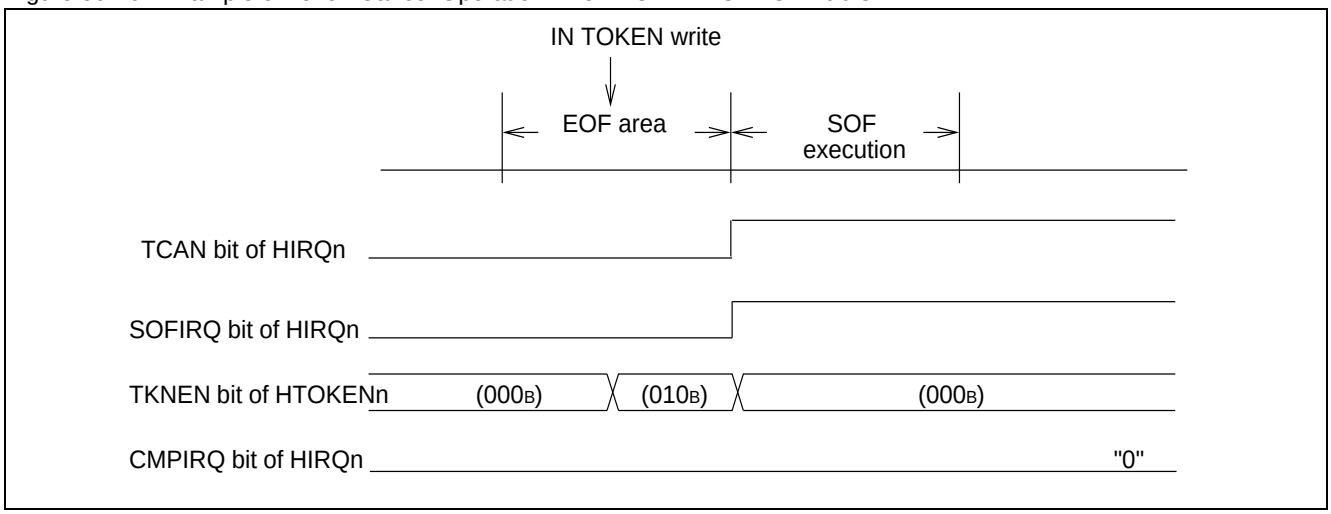
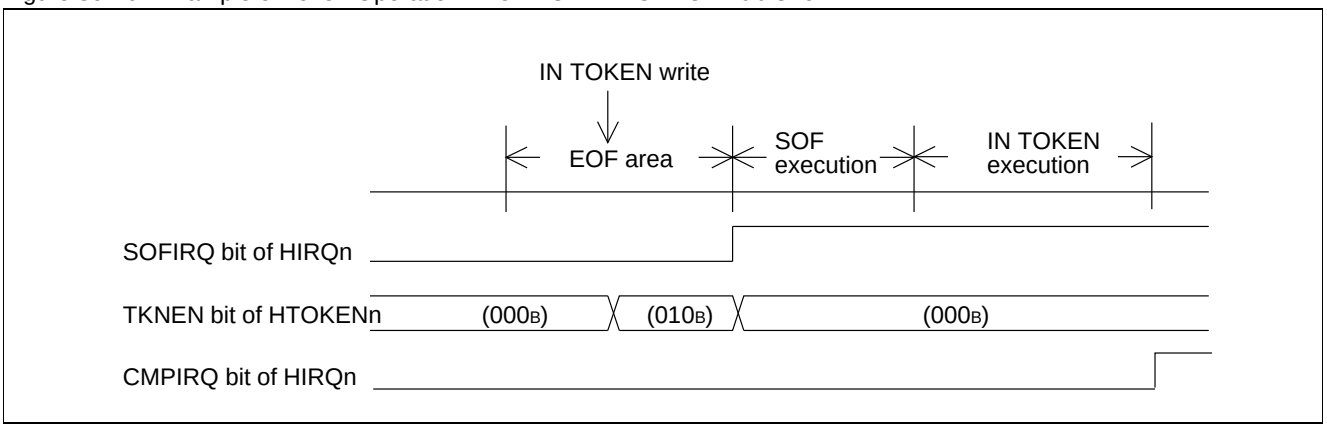


Figure 30-20. Example of Token Operation when HCNTHn:CANCEL bit is "0".




## 30.5.8 Error Status

This chapter describes the various error types that the USB Mini-host can detect.

### *30.5.8.1 Error Status*

#### Stuffing error

If 6 bits are continuously equal to "1", one "0" bit must be stuffed in the sequence to keep the receiver synchronized. If 7 bits in a row are equal to "1" are encountered, the HERRn:STUFF bit is set signaling a stuffing error. Write "0" to clear HERRn:STUFF bit. If the next token is executed without clearing the STUFF bit, it is updated when the next token is completed.

#### Toggle error

When an IN token is received, the HERRn:TGERR bit is set if the toggle data for the data packet and the HTOKENn:TGGL bit do not match. Write "0" to clear HERRn:TGERR bit. If the next token is executed without clearing the TGERR bit, it is updated when the next token is completed.

#### CRC error

When an IN token is received, a CRC calculation is performed on the received data with the CRC polynomial $G(X) = X^{16}+X^{15}+X^{2}+1$. If the remainder is not $800D_H$, then a CRC error is detected and the HERRn:CRC bit is set. Write "0" to clear HERRn:CRC bit. If the next token is executed without clearing the CRC bit, it is updated when the next token is completed.

#### Time-out error

The HERRn:TOUT bit is set if a data packet or handshake is not received within a the specified interval or if SE0 is detected in the received data, or a stuffing error is detected. Write "0" to clear HERRn:TOUT bit. If the next token is executed without clearing the TOUT bit, it is updated when the next token is completed.

#### Receive error

The HERRn:RERR bit is set if the received data size is greater than the packet size limit. The packet size limit specified in the EP1Cn:PKS field is referred when endpoint 1 is used and EP2Cn:PKS is referred when endpoint 2 is used. Write "0" to clear HERRn:RERR bit. If the next token is executed without clearing the RERR bit, it is updated when the next token is completed.

## 30.5.9 Packet End

This chapter describes the Packet End process.

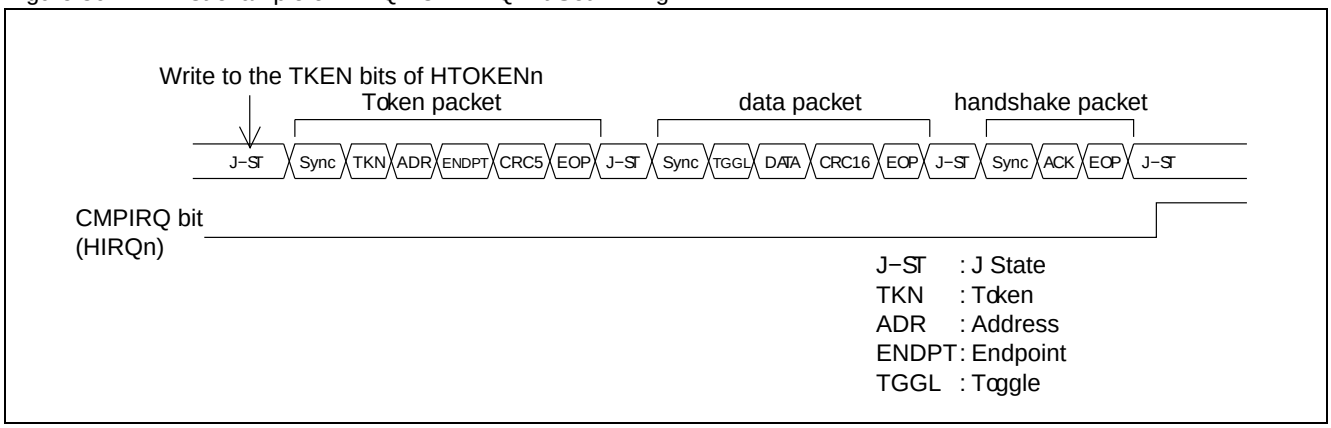
### Packet End Timing

When one packet transfer is completed in USB Mini-host, an interrupt is triggered by the HIRQn:CMPIRQ bit (if the interrupt is enabled by setting HCNTLn:CMPIRE).

The interrupt is generated in the following timing:

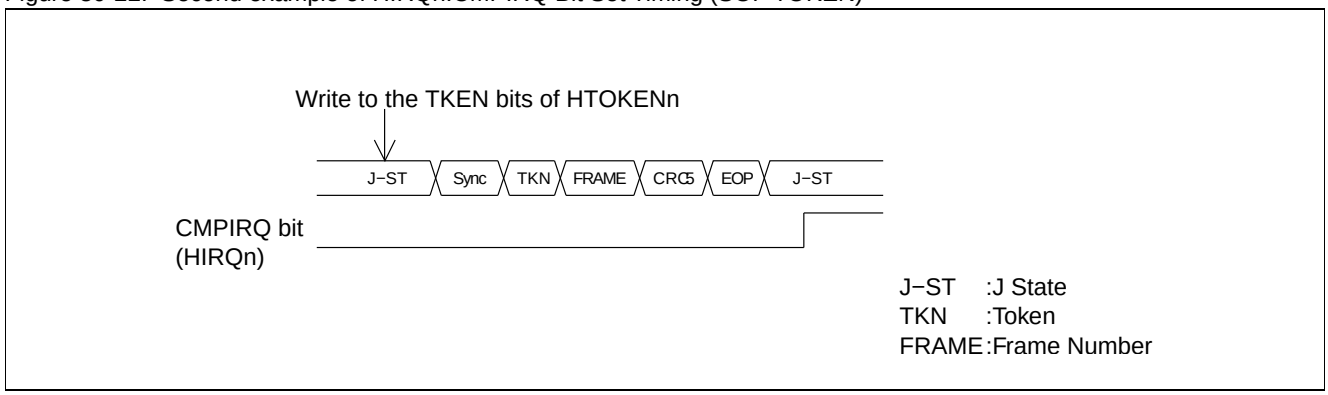
- When the HTOKENn:TKEN bits are $001_B$, $010_B$ or $011_B$ (SETUP token, IN token, and OUT token)

Figure 30-21. First example of HIRQn:CMPIRQ Bit Set Timing




- When the HTOKENn:TKEN is $100_B$ (SOF token)

Figure 30-22. Second example of HIRQn:CMPIRQ Bit Set Timing (SOF TOKEN)



## 30.5.10 Suspend Resume

USB Mini-host supports suspend and resume operations.

### Suspend Operation

When "1" is written to the HSTATEn:SUSP bit, USB Mini-Host follows steps below and transits to suspend status.

- USB bus is set in a high impedance state.
- Circuit blocks where clock is not necessary is stopped.

When the USB circuit is put in suspend status, it sets the SUSP bit of the host status register (HSTATEn:SUSP) to "1". It is prohibited to set HSTATEn:SUSP to "1" when USB bus is being reset or the HSTATEn:SOFBUSY bit is "1" or data is being sent or received. Stopping clock supplied to the USB circuit is also forbidden in such case.

### Resume Operation

Resume operation is started when one of the following conditions is fulfilled:

1. Writing "0" to HSTATEn:SUSP bit.
2. The pins UDP and UDM for Mini-host are detected to be in k-state.
3. Detecting device disconnection.
4. Detecting device connection.

After the HIRQn:RWKIRQ bit is set, token is allowed again. The followings diagram s show the operation timing for each condition:

Figure 30-23. Resume Operation by Register (Full Speed Mode)

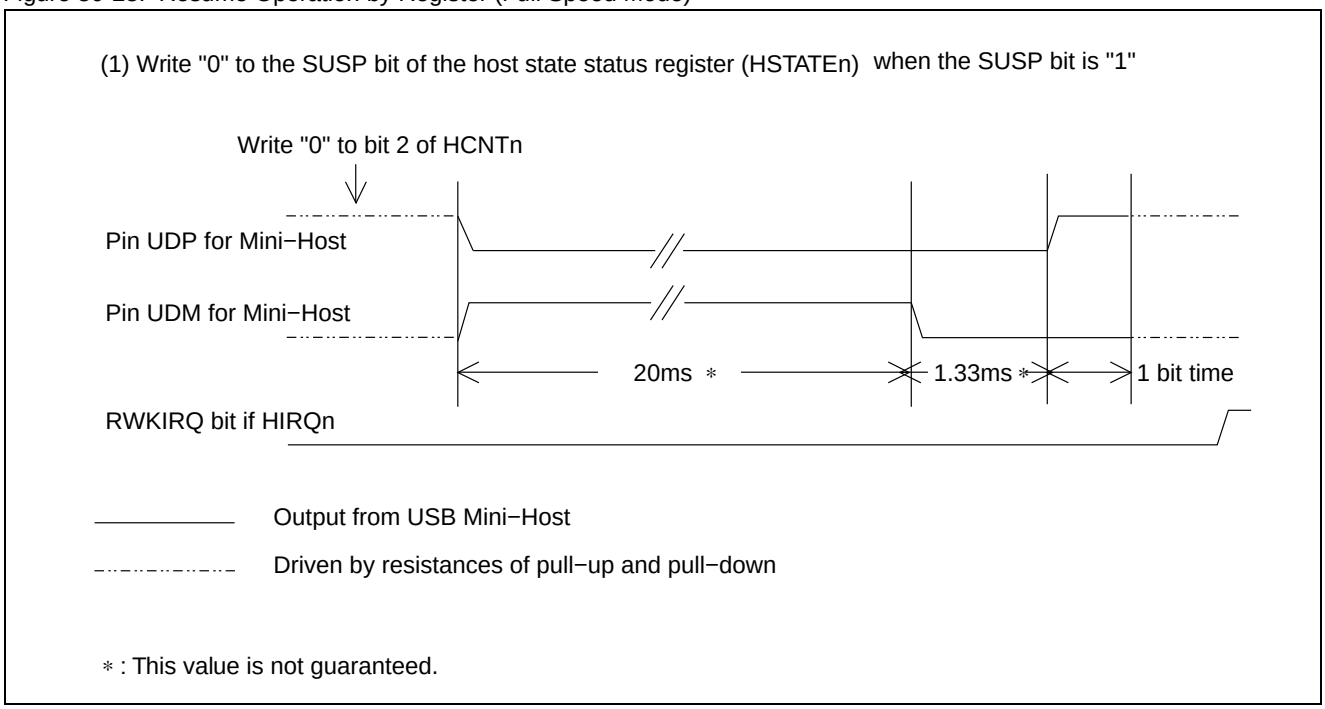


Figure 30-24. Resume Operation by detecting K-state on UDP and UDM (Full Speed Mode)

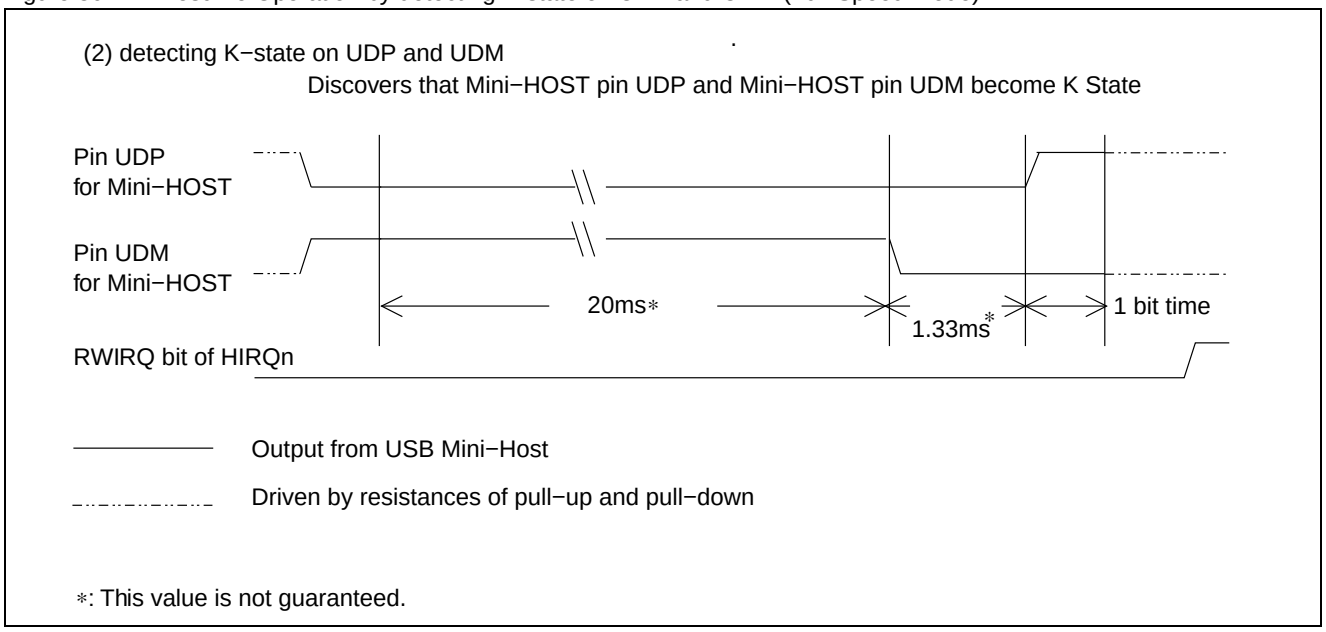



Figure 30-25. Resume Operation by Device disconnection

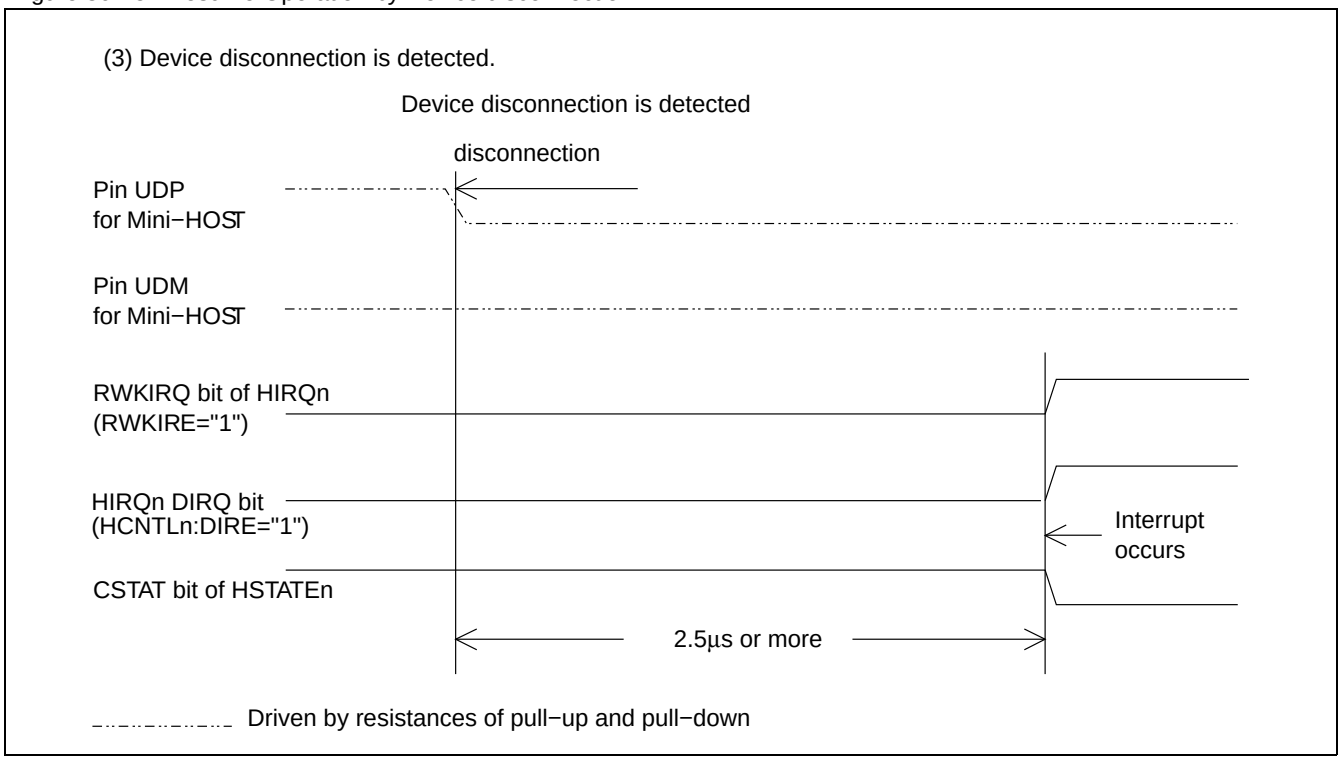
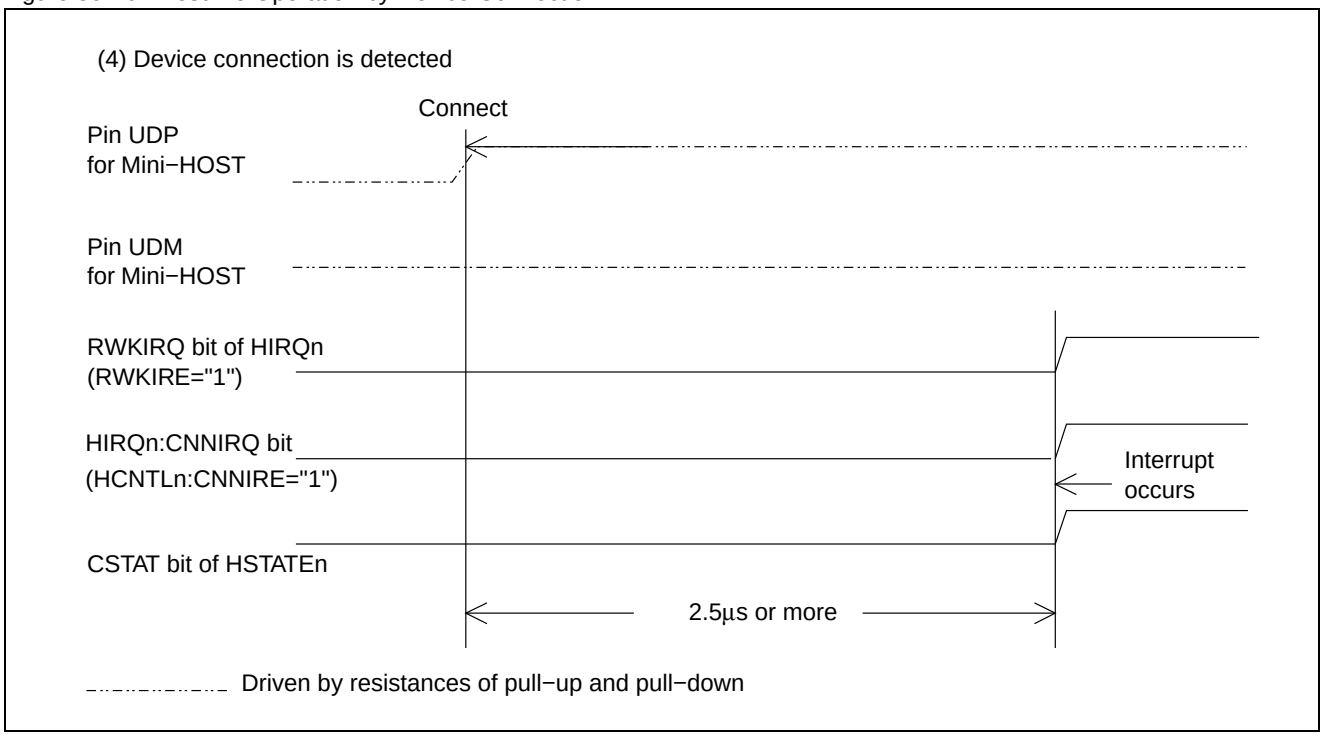


Figure 30-26. Resume Operation by Device Connection




## 30.5.11 Device disconnection

Once both Mini-host pins UDP and UDM become "L", the disconnection timer starts, and sets the HSTATEn:CSTAT bit to "0" when both pins detect "L" for 2.5 µs or longer.

### Device disconnection

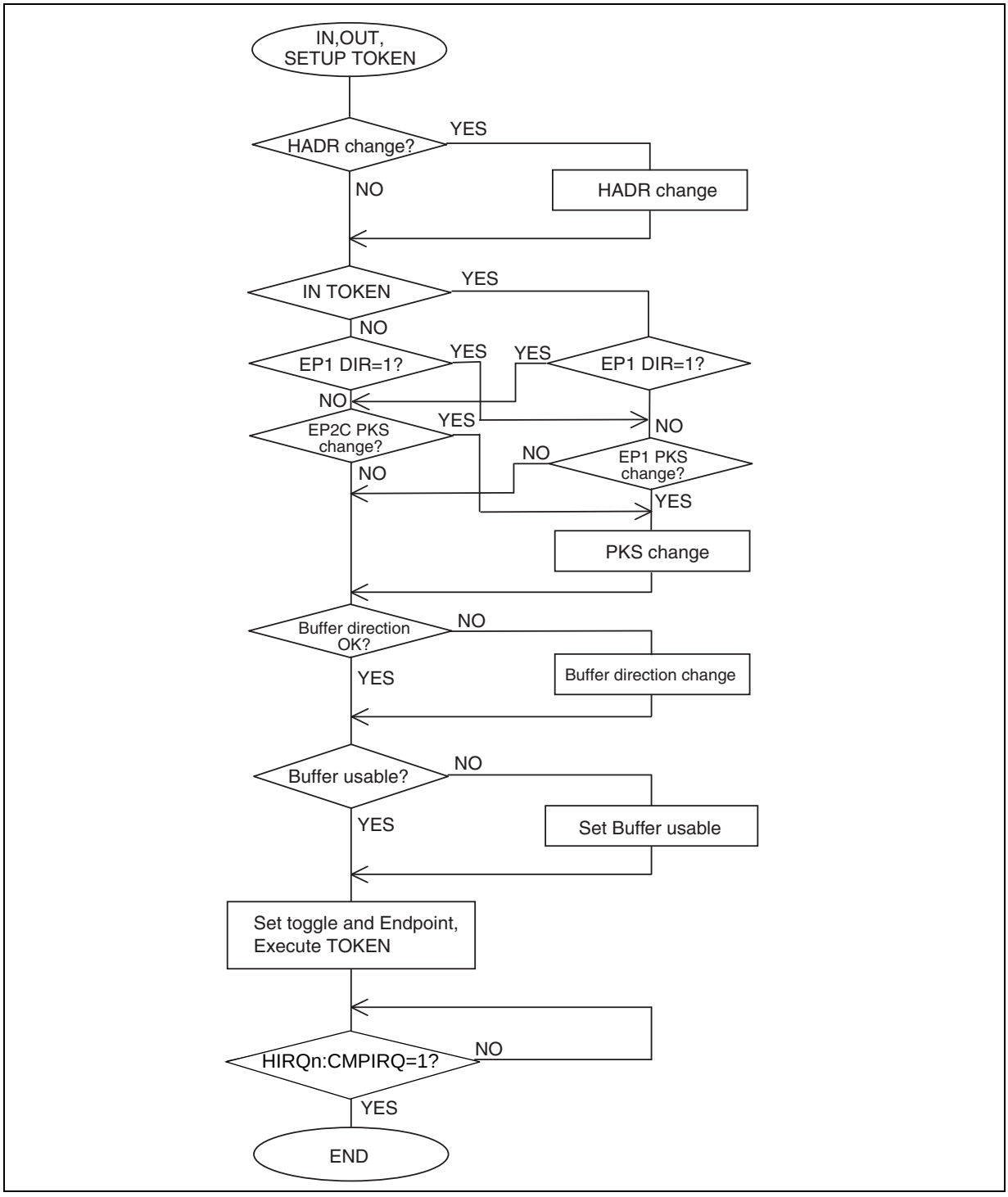
When both mini-host pins UDP and UDM detect "L" for 2.5 µs or longer, it determines that the device is disconnected. Therefore, the HSTATEn:CSTAT bit becomes "0" and the HIRQn:DIRQ flag is also set to "1". If the HCNTLn:DIRE bit is "1", an interrupt is triggered. Clearing the HIRQn:DIRQ flag clears the interrupt. When the USB bus is reset, it determines that the device is disconnected and sets the HSTATEn:CSTAT bit to "0", but the HIRQn:DIRQ bit is not set to "1".

## 30.6 Token Flow Chart

The flow chart of each USB Mini-host token.

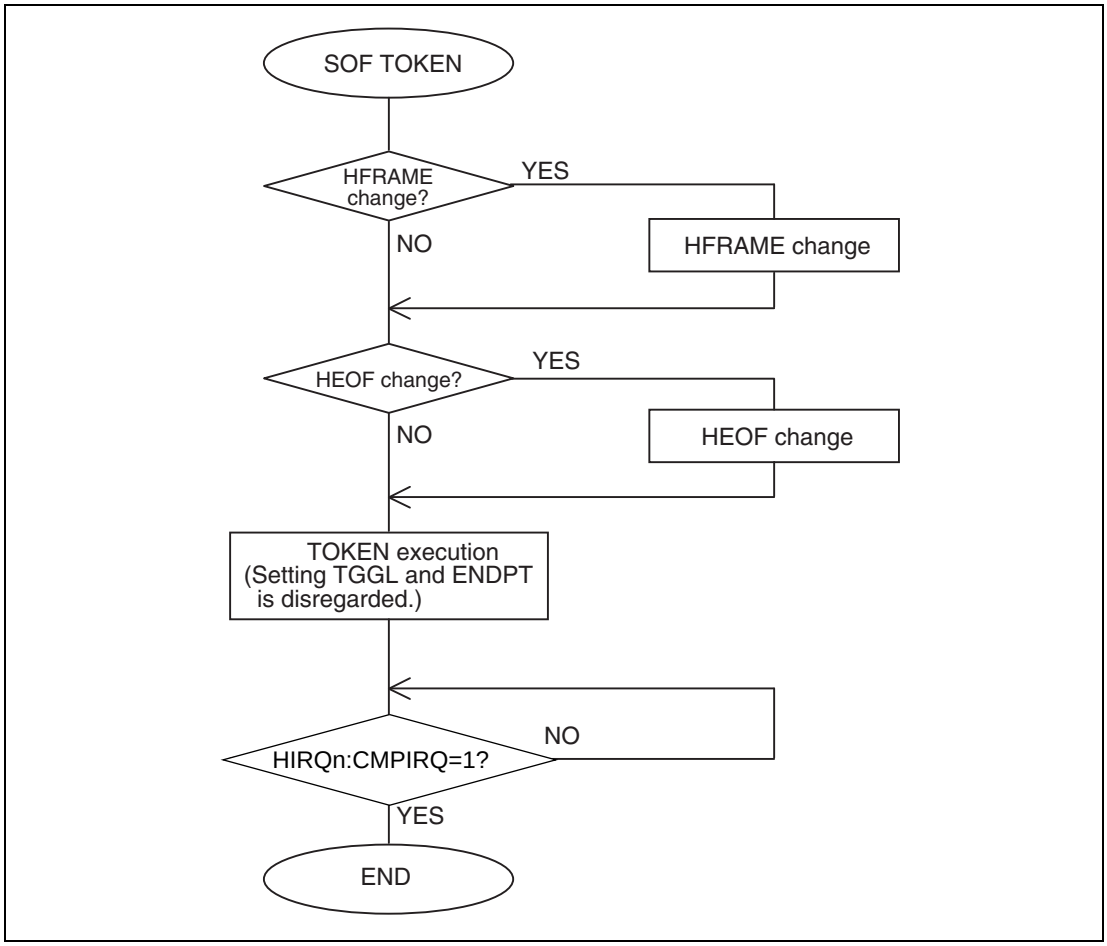
### IN, OUT, SETUP Token

Figure 30-27. Flow Chart at IN, OUT, SETUP Token



### SOF Token

Figure 30-28. Flow Chart of SOF Token



# 31. Memory Patch Function

This chapter explains the memory patch function and how the data patch or a debug function can be realised with that.

31.1 Outline of the Memory Patch Function

31.2 Registers of the Memory Patch Function

31.3 Operation of the Memory Patch Function

## 31.1 Outline of the Memory Patch Function

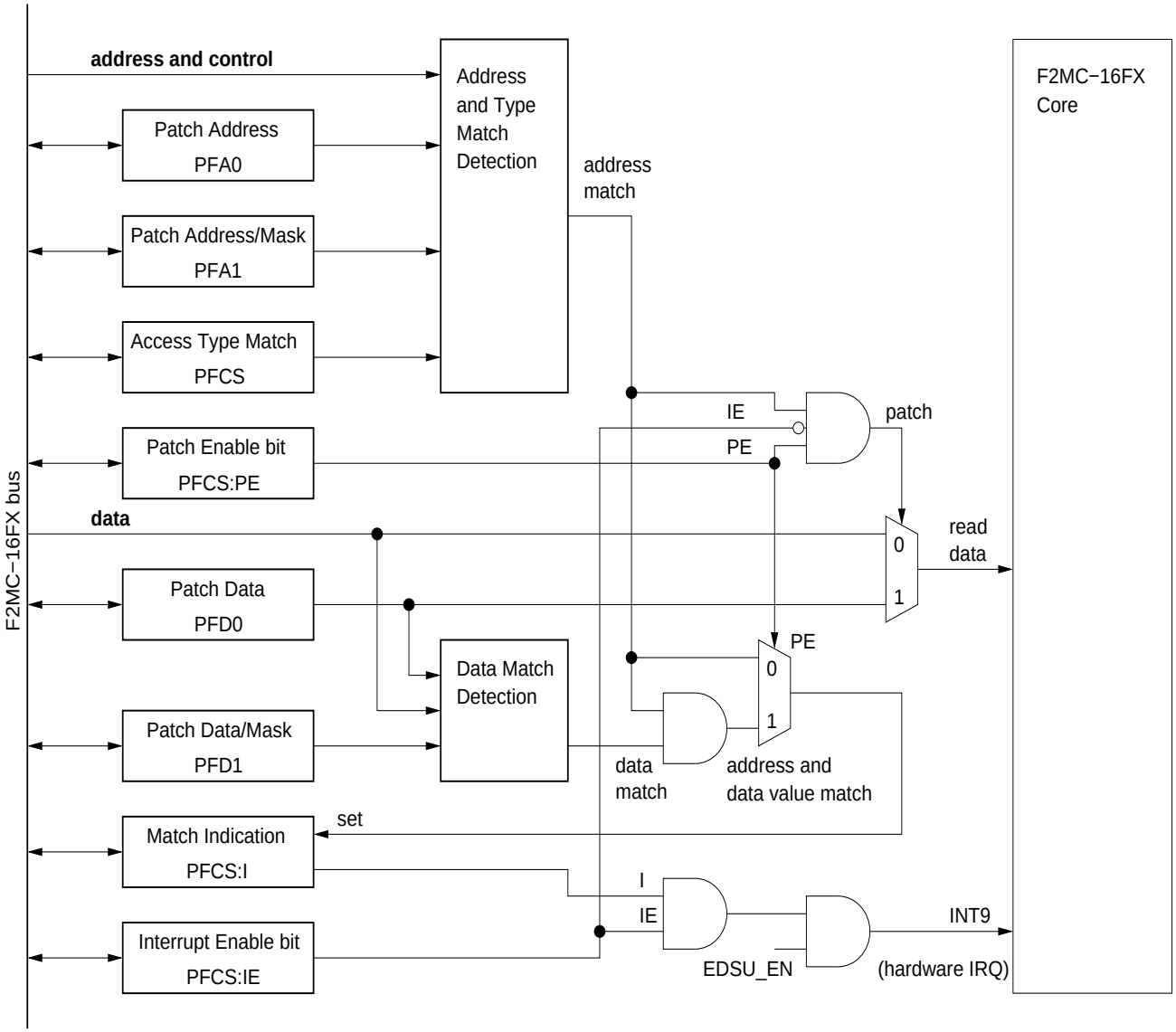
The Memory Patch Function can detect addresses for code fetch or data access on the 16FX core bus. If the address matches the value set in a patch function address register, the code or data read is replaced with the value given by the according data patch register or an hardware interrupt is asserted.

This data replacement function can be used to insert a software interrupt/break point, branch instruction or to patch the code or data words directly. Consequently, the CPU executes the instruction or reads the data specified in the patch register instead of the value addressed in the memory.

The hardware interrupt can be used to implement a data break point.

### Structure of the Memory Patch Function

Figure 31-1. Block diagram of the Memory Patch Function




The Memory Patch Function (MPF) has 8 channels, which can be used independently for 8 single address points or grouped for 4 address ranges. Figure 31-1 shows the structure of the MPF.

Each channel of the MPF has a patch function address register (PFAx), a patch data register (PFDx) and a common patch function control/status register (PFCSx) for a group of 2 channels. Each channel has an address match indication flag (I), an interrupt enable bit (IE) and a patch enable bit (PE) in the PFCSx. Each group of two channels has a common address mask (AM) and a common address range (AR) control bit. In addition there is one byte configuration data to match a specific access type for each group of two channels (READ, WRITE, BYTE, WORD, CODE, DATA, CPU, DMA).

## 31.2 Registers of the Memory Patch Function

The Memory Patch Function has following registers:

- Patch function address registers (PFA0 to PFA7)
- Patch function data registers (PFD0 to PFD7)
- Patch function control/status registers (PFCS0 to PFCS3)
- EDSU extension register (EDSU)

### Patch function address registers (PFA0 to PFA7)

Figure 31-2. Patch function address registers (PFA0 to PFA7)

| | byte+2 | byte+1 | byte+0 | Access | Initial value |
|---|---|---|---|---|---|
| PFA0 $03BA_H$ - $03B8_H$ | PFAH0 | PFAM0 | PFAL0 | R/W | undefined |
| PFA1 $03BD_H$ - $03BB_H$ | PFAH1 | PFAM1 | PFAL1 | R/W | undefined |
| PFA2 $03C0_H$ - $03BE_H$ | PFAH2 | PFAM2 | PFAL2 | R/W | undefined |
| PFA3 $03C3_H$ - $03C1_H$ | PFAH3 | PFAM3 | PFAL3 | R/W | undefined |
| PFA4 $03C6_H$ - $03C4_H$ | PFAH4 | PFAM4 | PFAL4 | R/W | undefined |
| PFA5 $03C9_H$ - $03C7_H$ | PFAH5 | PFAM5 | PFAL5 | R/W | undefined |
| PFA6 $03CC_H$ - $03CA_H$ | PFAH6 | PFAM6 | PFAL6 | R/W | undefined |
| PFA7 $03CF_H$ - $03CD_H$ | PFAH7 | PFAM7 | PFAL7 | R/W | undefined |

The address of each bus access is compared with the contents of the patch function address registers (PFA0 to PFA7). If the address of the bus access matches one of the patch function address registers and the access type matches one of the upper 8 configuration bits in the according PFCSx, the address match indication bit (PFCSx:I) is set for this channel. Further action depends on the configuration of the IE or PE bits in the patch function control/status register.

Within each group of two channels, an address mask and an address range function can be used. Following description uses the group of channel 0 and 1 as example:

In case of the mask option (PFCS0:AM=1), PFA0 defines the address and PFA1 defines the mask for the address. Bits containing ’1’ in the mask are not compared and are always matched. In the case of a matching access type (upper 8 bits of PFCSx), the match indication bit PFCS0:I0 is set, if following equation is true:

PFA0 | PFA1 == AD | PFA1.

AD reflects the address on the bus.

For the range option (PFCS0:AR=1) PFA0 defines the starting point of the address range and PFA1 defines the end point of the address range. In the case of a matching access type (upper 8 bits of PFCSx), the PFCS0:I0 bit is set for all addresses matching the range from PFA0 to PFA1.

PFA0 <= AD <= PFA1.

For using the memory patch function (PFCSx:PE=1 and PFCSx:IE=0), bit 0 of the PFAx is ignored. The whole word accessed will be patched.

The patch function address registers PFA0 to PFA7 are undefined after reset. The patch function address registers can be read and written.

For reference of the patch function control/status bits see Table 31-2. It lists the correspondence between the patch function address registers PFAx, the patch function data registers PFDx and the patch function control/status register PFCSx.

### Patch function data register (PFD0 to PFD7)

Figure 31-3. Patch function data registers (PFD0 to PFD7)

| | byte+1 | byte+0 | Access | Initial value |
|---|---|---|---|---|
| PFD0 $03D1_H$ - $03D0_H$ | PFDH0 | PFDL0 | R/W | undefined |
| PFD1 $03D3_H$ - $03D2_H$ | PFDH1 | PFDL1 | R/W | undefined |
| PFD2 $03D5_H$ - $03D4_H$ | PFDH2 | PFDL2 | R/W | undefined |
| PFD3 $03D7_H$ - $03D6_H$ | PFDH3 | PFDL3 | R/W | undefined |
| PFD4 $03D9_H$ - $03D8_H$ | PFDH4 | PFDL4 | R/W | undefined |
| PFD5 $03DB_H$ - $03DA_H$ | PFDH5 | PFDL5 | R/W | undefined |
| PFD6 $03DD_H$ - $03DC_H$ | PFDH6 | PFDL6 | R/W | undefined |
| PFD7 $03DF_H$ - $03DE_H$ | PFDH7 | PFDL7 | R/W | undefined |

The patch function data registers (PFD0 to PFD7) define the replacement value of the data read in the case of a match. If a match on the access type (upper 8 bits of PFCSx) and on the patch function address register (PFAx) occurs and PEx=1 and IEx=0, the data defined by the PFDx register is read.

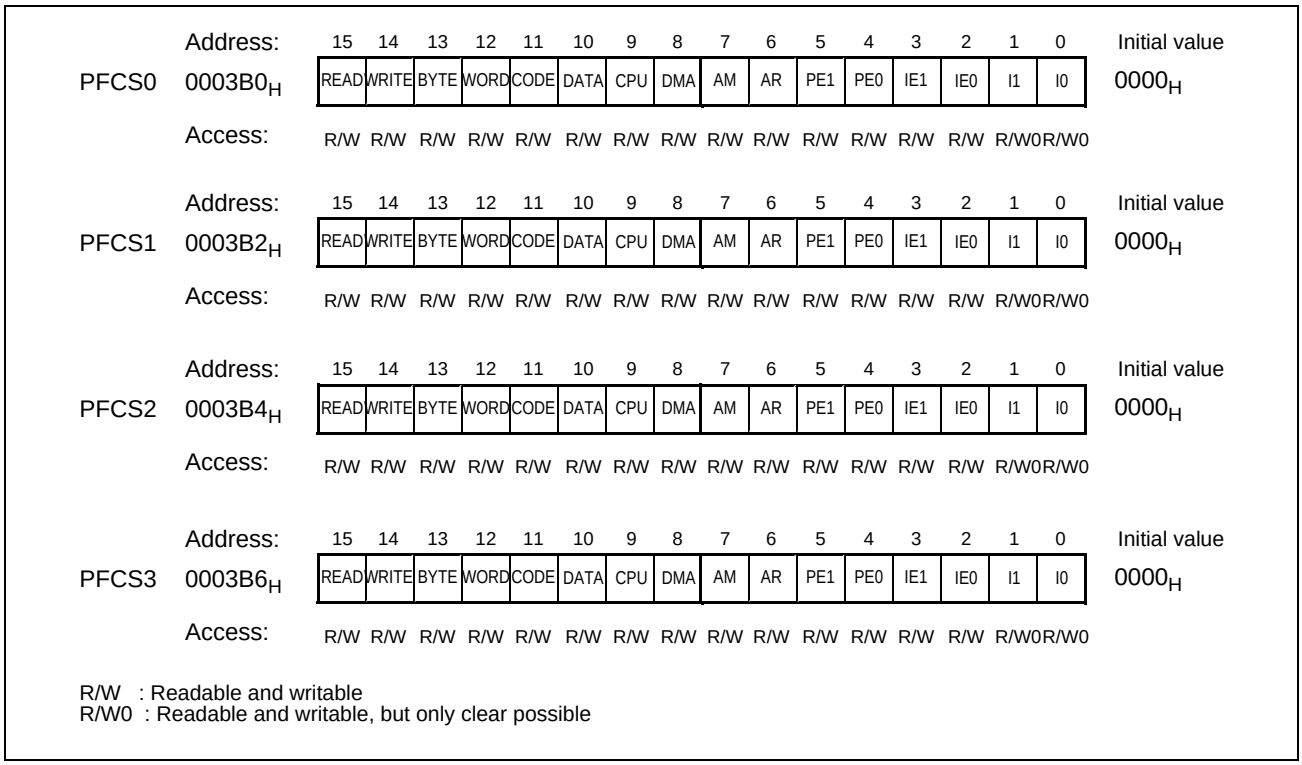
If PFCSx:AM or PFCSx:AR is enabled, only matches on channels 0/2/4/6 are generated and only PFD0/2/4/6 is used for data replacement.

The value of the PFDx register is only replaced on the bus, when the PE bit is set and the IE bit is cleared. However, PFDx can be used by the embedded debug support unit (EDSU) for defining a data value or data mask for operand break detection. For that purpose (data value break) both the PE bit and the IE bit need to be set to ’1’.

The patch function data registers PFD0 to PFD7 are undefined after reset. The patch function data registers can be read and written.

### Patch function control/status register (PFCS0 to PFCS3)

Figure 31-4. Patch function control/status register (PFCS0/1/2/3)




The patch function control/status register (PFCSx) controls the operation of the memory patch function and reflects the status of each MPF channel. Each register word PFCSx belongs to a group of two channels.

For a detailed description of the control and status bits refer to Table 31-1. The functions controlled by each bit in the PFCS1 to PFCS3 registers are identical to PFCS0, only the channel index of the PFA and PFD registers have to be incremented by 2/4/6 accordingly.

Table 31-1. Function of each bit of the PFCS0 (MPF channel 0 and 1)

| Name | | Function |
|---|---|---|
| bit 15 | READ | This bit controls, if read accesses are considered for the address match detection. This includes the read part of read-modify-write.<br>• Writing '0' - Read accesses do not match.<br>• Writing '1' - Read accesses match.<br>• The bit can be read and written.<br>The READ bit is cleared after reset. |
| bit 14 | WRITE | This bit controls, if write accesses are considered for the address match detection.<br>• Writing '0' - Write accesses do not match.<br>• Writing '1' - Write accesses match.<br>• The bit can be read and written.<br>The WRITE bit is cleared after reset. |
| bit 13 | BYTE | This bit controls, if byte accesses are considered for the address match detection.<br>• Writing '0' - Byte accesses do not match.<br>• Writing '1' - Byte accesses match.<br>• The bit can be read and written.<br>The BYTE bit is cleared after reset. |

Table 31-1. Function of each bit of the PFCS0 (MPF channel 0 and 1)

| Name | | Function |
|---|---|---|
| bit 12 | WORD | This bit controls, if word accesses are considered for the address match detection.<br>• Writing '0' - Word accesses do not match.<br>• Writing '1' - Word accesses match.<br>• The bit can be read and written.<br>The WORD bit is cleared after reset. |
| bit 11 | CODE | This bit controls, if accesses for code fetch are considered for the address match detection.<br>• Writing '0' - Code accesses do not match.<br>• Writing '1' - Code accesses match.<br>• The bit can be read and written.<br>The CODE bit is cleared after reset. |
| bit 10 | DATA | This bit controls, if data accesses are considered for the address match detection.<br>• Writing '0' - Data accesses do not match.<br>• Writing '1' - Data accesses match.<br>• The bit can be read and written.<br>The DATA bit is cleared after reset. |
| bit 9 | CPU | This bit controls, if CPU accesses are considered for the address match detection.<br>• Writing '0' - CPU accesses do not match.<br>• Writing '1' - CPU accesses match.<br>• The bit can be read and written.<br>The CPU bit is cleared after reset. |
| bit 8 | DMA | This bit controls, if DMA accesses are considered for the address match detection.<br>• Writing '0' - DMA accesses do not match.<br>• Writing '1' - DMA accesses match.<br>• The bit can be read and written.<br>The DMA bit is cleared after reset. |
| bit 7 | AM | The address mask bit AM controls the mask option of the MPF channel 0.<br>• Writing '0' - The mask option for PFA0 is disabled. Instead PFA1 defines the address for channel 1, if not used as range.<br>• Writing '1' - PFA1 is used as OR-mask for both PFA0 and the address value on the bus at address match evaluation.<br>• If the AR bit is set to '1', the setting of the AM bit is ignored (no mask).<br>• The bit can be read and written.<br>The MPF channel 1 is not available if the mask option is used.<br>The AM bit is cleared after reset. |
| bit 6 | AR | The address range bit AR controls the range option of the MPF channel 0.<br>• Writing '0' - The range option for PFA0/PFA1 is disabled. Instead PFA0 and PFA1 define the address for channel0 and 1 independently, if PFA1 is not used as mask.<br>• Writing '1' - PFA0 is used as start address and PFA1 is used as end address for the address value on the bus at address match evaluation.<br>• The bit can be read and written.<br>The MPF channel 1 is not available if the range option is used.<br>The AR bit is cleared after reset. |
| bit 5 | PE1 | The patch enable bit controls the data replacement function of MPF channel 1.<br>• Writing '0' - The data replacement (patch) function is disabled.<br>• Writing '1' and IE1='0' - The data replacement (patch) function is enabled. In case of an address match with PFA1, the value of PFD1 is read instead of the addressed memory position.<br>• Writing '1' and IE1='1' - The data replacement (patch) function is disabled. However, the channel 1 can not generate a data value break. The PFD1 register is dedicated to define the mask for the data value, in case of channel 0 is used as data value break.<br>• The bit can be read and written.<br>If PE1='1' and IE1='0' (patch), bit 0 of PFA1 is handled as don't care.<br>If either AM or AR is set, no address match is generated on MPF channel 1. Thus the PE1 bit has no meaning for that case.<br>The PE1 bit is cleared after reset. |

Table 31-1. Function of each bit of the PFCS0 (MPF channel 0 and 1)

| Name | | Function |
|---|---|---|
| bit 4 | PE0 | The patch enable bit controls the data replacement function and the data value break of MPF channel 0.<br>• Writing '0' - The data replacement (patch) function is disabled.<br>• Writing '1' and IE0='0' - The data replacement (patch) function is enabled. In case of an address match with PFA0, the value of PFD0 is read instead of the addressed memory position.<br>• Writing '1' and IE0='1' - The data replacement (patch) function is disabled. The combination of PE0='1' and IE0='1' is used to enable the data value match (see Table 31-3).<br>• The bit can be read and written.<br>If PE0='1' and IE0='0' (patch), bit 0 of PFA0 is handled as don't care.<br>The PE0 bit is cleared after reset. |
| bit 3 | IE1 | This is an enable bit for the interrupt flag I1. It controls the hardware interrupt generation of MPF channel 1. This is used to implement an operand break feature.<br>• Writing '0' - The interrupt is disabled.<br>• Writing '1' - The interrupt is enabled. In case of an address or type match, an INT9 hardware interrupt is asserted.<br>• The bit can be read and written.<br>The IE1 bit is cleared after reset.<br>This bit has no meaning if AM or AR is set. |
| bit 2 | IE0 | This is an enable bit for the interrupt flag I0. It controls the hardware interrupt generation of MPF channel 0. This is used to implement a operand break feature.<br>• Writing '0' - The interrupt is disabled.<br>• Writing '1' - The interrupt is enabled. In case of an address, type or data value match, an INT9 hardware interrupt is asserted.<br>• If the PE0 bit is set in addition, the channel 0 operates as data value break. An additional data match is required to assert INT9.<br>• The bit can be read and written.<br>The IE0 bit is cleared after reset. |
| bit 1 | I1 | This is the match indication/interrupt flag for channel 1. The address match condition can be combined with the access type match. Once set, the bit remains set until cleared by software.<br>• Reading '0' - Indicates that there was no match since last clear operation.<br>• Reading '1' - Indicates that at least one match has occurred since last clear operation. If IE1 is set, an interrupt is issued. If PE1 is set and IE1 is cleared, the read data on the bus are replaced by PFD1.<br>• Writing '0' clears the bit<br>• Writing '1' has no effect<br>• A RMW instruction reads always '1'<br>If a write access to the I flag and a hardware event, which sets the I flag, occures at the same time, the hardware event has precedence.<br>If either AM or AR is set, no match is generated on MPF channel 1. Thus the I1 flag will not be set in that case.<br>This bit is cleared after reset.<br>Before enabling the interrupt function (IE1=1), it is recommended to clear the I1 bit. |
| bit 0 | I0 | This is the match indication/interrupt flag for channel 0. The address match condition can be combined with access type and data value matches. Once set, the bit remains set until cleared by software.<br>• Reading '0' - Indicates that there was no match since last clear operation.<br>• Reading '1' - Indicates that at least one match has occurred since last clear operation. If IE0 is set, an interrupt is issued. If PE0 is set and IE0 is cleared, the read data on the bus are replaced by PFD0.<br>• Writing '0' clears the bit<br>• Writing '1' has no effect<br>• A RMW instruction reads always '1'<br>If a write access to the I flag and a hardware event, which sets the I flag, occures at the same time, the hardware event has precedence.<br>This bit is cleared after reset.<br>Before enabling the interrupt function (IE0=1), it is recommended to clear the I0 bit. |

Table 31-2 lists the correspondence between the patch function address registers PFAx, the patch function data registers PFDx and the patch function control/status register PFCSx.

Table 31-2. Correspondence between PFAx, PFDx and PFCSx

| Patch address (PFA) | | | | Patch data (PFD) | Control/status register (PFCS) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | AR=0 AM=0 | AR=0 AM=1 | AR=1 | | Access Type [15:8] | Operation select/status [7:0] | | | | | |
| PFA0 | point | point | start point | PFD0 | PFCS0 | PFCS0 | I0 | IE0 | PE0 | AM | AR |
| PFA1 | point | mask | end point | PFD1 | | | I1 | IE1 | PE1 | | |
| PFA2 | point | point | start point | PFD2 | PFCS1 | PFCS1 | I0 | IE0 | PE0 | AM | AR |
| PFA3 | point | mask | end point | PFD3 | | | I1 | IE1 | PE1 | | |
| PFA4 | point | point | start point | PFD4 | PFCS2 | PFCS2 | I0 | IE0 | PE0 | AM | AR |
| PFA5 | point | mask | end point | PFD5 | | | I1 | IE1 | PE1 | | |
| PFA6 | point | point | start point | PFD6 | PFCS3 | PFCS3 | I0 | IE0 | PE0 | AM | AR |
| PFA7 | point | mask | end point | PFD7 | | | I1 | IE1 | PE1 | | |

The PFCSx:IEx and PFCSx:PEx configuration bits select the main operation of the MPF channel.

Table 31-3. Channel Operation definition by the IE and PE bits in PFCS

| IE | PE | Operation |
|---|---|---|
| 0 | 0 | Off.<br>No interrupt and no data patch enabled. However, an address match is recorded in the I-flag, when one of the enabled access types (PFCS[15:8]) and the given address (PFA, PFCS:AR, PFCS:AM) are matching. |
| 0 | 1 | Memory patch (ROM correction, memory protection).<br>The memory patch function is enabled. In case of an address and type match the data value on the bus is replaced by the contents of the patch function data register (PFD).<br>In addition to the data replacement at read access, the bus transfer is not effective for the accessed address. In case of a read, the memory patch function serves the access. In case of a write, the access is not executed. This can be used to implement a memory protection function, which cancels the access before data were modified.<br>Usually, the memory patch function is not required to distinguish between different access types. Hence, the bits PFCS[15:8] should be all set to "1". However, explicit access type filtering is also possible together with the patch function.<br>If an address containing an opcode is configured to patch this memory location with the opcode of INT9, this function can be used to implement an instruction break. The read value of the original instruction is replaced by the opcode of INT9. |
| 1 | 0 | Operand address break.<br>This function is mainly used to provide an operand address break. In case of an address and access type match, the I-flag is set and an interrupt (INT9) is issued. The access types can be configured for explicit data access (PFCS:DATA=1 and PFCS:CODE=0). The other filter options for the access type are useful to match a more specific type.<br>This function can also match accesses for code fetch (PFCS:DATA=0 and PFCS:CODE=1). Please consider, that fetched code may not be executed (prefetch mechanism of the CPU). Hence, this function is not intended to be used to implement an instruction execution break. The instruction break should be realized by using the memory patch function (IE=0 and PE=1, as described). |
| 1 | 1 | Data value break.<br>This configuration provides a combined operand access break with an explicit data value match. In case of an address, access type and data match with PFD, the I-flag is set and an interrupt (INT9) is issued.<br>The data value break is only available for channels PFD0, PFD2, PFD4 and PFD6. The neighbored PFDs (PFD1, PFD3, PFD5 and PFD7) are used as data mask. However, the neighboured PFAs can still be used for an operand address break, if they are configured for that purpose.<br>The patch function is not enabled in that case. |

### EDSU extension register (EDSU)

Figure 31-5. EDSU extension register (EDSU)

| | Address: | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | Initial value |
|---|---|---|---|---|---|---|---|---|---|---|
| EDSU | $0003AF_H$ | EN | - | TIE | TINT | SEL1 | SEL0 | RIE | RINT | $0X0X000X_B$ |
| | Access: | R/W | - | R/W | R | R/W | R/W | R/W | R | |

R/W : Readable and writable
R : Read only

To avoid the implementation of the debug services directly in the kernel of the application, external event can be handled by the system to trigger the debug services. For further information regarding the debug system, refer to the Boot ROM specifications.

The EDSU extension register (EDSU) controls the embedded debug support, based on the address and data match functions. There are mainly two functions controlled by this register:

- General enable the hardware trigger of INT9 for debug functions (EDSU:EN).
- Select a communication interface; its receive and/or transmit interrupt is able to trigger an INT9 exception to realize a break on request of this interface (EDSU:SEL, EDSU:TIE, EDSU:TINT, EDSU:RIE, EDSU:RINT). For extended capabilities of the break interrupt selection, see the description of the EDSU2 register.

For a detailed description of the control and status bits of EDSU refer to Table 31-4.

The interrupt vector of the INT9 can be configured to be located at a fixed address (0x0FFFD8), to be at a location independent from the value of the table base register (TBR). For details about fixing the vector address of INT9, please refer to the

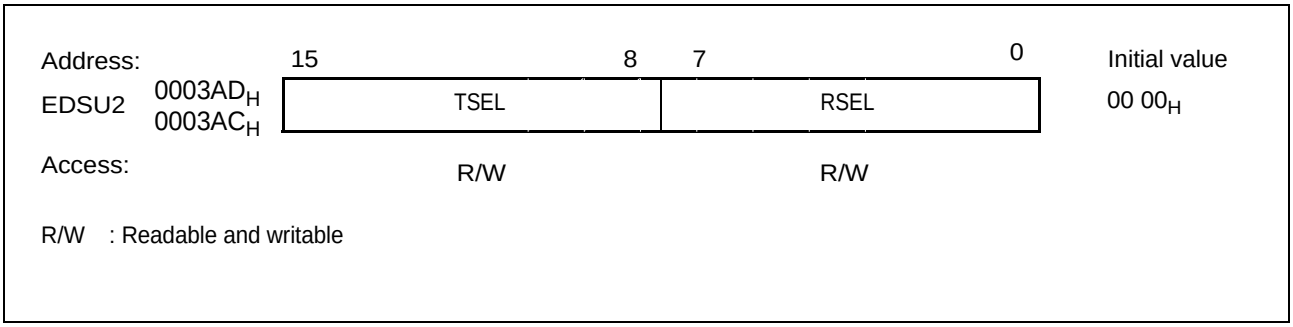
chapter "Interrupts", "NMI control status register (NMI)". This option is used to increase the reliability of a debug system making use of the EDSU functions.

Table 31-4. Function of each bit of the EDSU register

<table>
<tr><th colspan="2">Name</th><th>Function</th></tr>
<tr><td>bit 15</td><td>EN</td><td>The enable bit EN controls the generation of hardware INT9 requests of the embedded debug system. It enables the interrupt generation of the address match detection, data value match detection and of the selected communication device receive interrupt in general. The memory patch function is not affected by this bit.<br>• Writing '0' - Disables embedded debug support.<br>• Writing '1' - Enables embedded debug support.<br>• The bit can be read and written.<br>The EN bit is cleared after reset.</td></tr>
<tr><td>bit 14</td><td>-</td><td>Unused bit.<br>• Read returns undefined value.<br>• Writing to this bit is ignored.</td></tr>
<tr><td>bit 13</td><td>TIE</td><td>This bit enables INT9 for the transmit interrupt of the selected communication device.<br>• Writing '0' - The INT9 generation on transmit IRQ is disabled.<br>• Writing '1' - The INT9 generation on transmit IRQ is enabled.<br>• The bit can be read and written.<br>The TIE bit is cleared after reset.</td></tr>
<tr><td>bit 12</td><td>TINT</td><td>This bit reflects the actual status of the transmit interrupt flag of the selected communication device<br>• The bit is read only.<br>• Writing is ignored.<br>• To clear the interrupt flag, use the according interrupt flag in the selected communication device (peripheral resource).<br>The initial value depends on the selected communication device. See the specification of the according peripheral resource.</td></tr>
<tr><td>bit 11to 10</td><td>SEL1, SEL0</td><td>These bits select the monitor debugger communication device.<br>• USART 0, USART 1, USART 2 and USART 3 can be selected according to the table below.<br>• These bits can be read and written.<br>The SEL bits are initialized with 0 after reset.<br><table><tr><th>SEL1</th><th>SEL0</th><th>USART</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>2</td></tr><tr><td>1</td><td>1</td><td>3</td></tr></table></td></tr>
<tr><td>bit 9</td><td>RIE</td><td>This bit enables INT9 for the receive interrupt of the selected communication device.<br>• Writing '0' - The INT9 generation on receive IRQ is disabled.<br>• Writing '1' - The INT9 generation on receive IRQ is enabled.<br>• The bit can be read and written.<br>The RIE bit is cleared after reset.</td></tr>
<tr><td>bit 8</td><td>RINT</td><td>This bit reflects the actual status of the receive interrupt flag of the selected communication device<br>• The bit is read only.<br>• Writing is ignored.<br>• To clear the interrupt flag, use the according interrupt flag in the selected communication device (peripheral resource).<br>The initial value depends on the selected communication device. See the specification of the according peripheral resource.</td></tr>
</table>

### EDSU extension register 2 (EDSU2)

Figure 31-6. EDSU extension register 2 (EDSU2)




The EDSU extension register 2 (EDSU2) controls the extended break interrupt selection to trigger an INT9 exception. For both transmit and receive interrupt channels an interrupt number can be choosen out of the complete interrupts available in the system. There is no restriction for the selectors, to be bound to a specific transmit or receive function.

For a detailed description of the EDSU2 register refer to Table 31-5.

Table 31-5. Function of the EDSU2 register

| Name | | Function |
|---|---|---|
| bit 15 to 8 | TSEL | Select the interrupt channel number to be reflected by the TINT bit of the EDSU register. The selected interrupt number will be directed to IRQ9 in case of TIE is set in the EDSU register.<br>• Writing a number different from '00' - Enables extended interrupt number selection. An interrupt channel can be selected free of choice.<br>• Writing '00' - The standard selection method by SEL1 and SEL0 in the EDSU register is effective.<br>• The byte can be read and written.<br>TSEL is cleared after reset. |
| bit 7 to 0 | RSEL | Select the interrupt channel number to be reflected by the RINT bit of the EDSU register. The selected interrupt number will be directed to IRQ9 in case of RIE is set in the EDSU register.<br>• Writing a number different from '00' - Enables extended interrupt number selection. An interrupt channel can be selected free of choice.<br>• Writing '00' - The standard selection method by SEL1 and SEL0 in the EDSU register is effective.<br>• The byte can be read and written.<br>RSEL is cleared after reset. |

For the selected interrupt numbers, a possible interrupt level configuration has no effect. The interrupts are taken directly from the peripheral functions or resources connected. However enabling the interrupt within the selected peripheral function or resource is required.

## 31.3 Operation of the Memory Patch Function

This section describes the usage of the Memory Patch Function for different application purposes.

### Memory Patch Function

This is a data replacement function bound to an address match. It can be used to insert a software interrupt/break point, branch instruction or to patch the code or data words directly. Consequently, the CPU executes the instruction or reads the data specified in the patch register instead of the value addressed in the memory.

The memory patch function replaces the read data value on the 16FX core bus with a value loaded to the patch function data register PFDx. The patch function is sensitive on an address given by PFAx.

To use this function, do the following:

- Select a free MPF channel "x" and disable its usage by clearing the PEx and IEx bits.
- Set all bits in PFCS[15:8] to match all access types.
- Program the address into PFAx (16bit - word access, address bit 0 is ignored)
- Program the replacement data into PFDx. If byte data should be corrected, copy the other byte from the original memory location (it is not usable to patch byte variables, where the neighboring byte is not constant).
- Enable the PEx bit

When an address matches the value set in the patch function address register, following operation is performed by the Memory Patch Function:

- The address match indication bit Ix is set.
- The read access is redirected to the patch function data register, consequently the CPU or DMA will read the value programmed into the PFDx register.
- A probable write access has no effect.

For the purpose of the implementation of a memory protection function, the address range of the configuration registers of the address match detection function should be also considered to be protected.

### Instruction address break

Each MPF channel can be used to act as instruction break. The same procedure is used as described for the memory patch function. The special case is, that the INT9 instruction code ($01_H$) is programmed on the according byte position in the patch function data register. PFAx should specify an address where instruction code is located.

The mask and range option can be used to define the address region, which is sensitive for the instruction break.

Be careful with the range option for the instruction address match. Starting from or ending at an odd address requires the use of a separate channel. When the PE bit is set, the address bit 0 is masked.

### Operand address break

With the MPF an operand address break can be realized by using its address match functions.

To use this function, do the following:

- Select a free MPF channel "x" and disable its usage by clearing the PEx and IEx bits.
- Program the address into PFAx (byte access, hence address bit 0 is significant)
- Program the access types to be detected in the PFCSx. At least one item out of each group of preferences need to be selected:
  Direction {READ|WRITE}, Size {BYTE|WORD}, Purpose {CODE|DATA}, Master {CPU|DMA}.
  If both items in a group are enabled, any access matches with regard to the specific preference. Normally DATA=1 and CODE=0 is chosen to detect only operand addresses for the purpose of an operand address break.
- The address mask and range options may be useful for specifying the address match condition.
- Enable the IEx bit
- Set the EDSU_EN bit to ’1’

When an address matches the value set in the patch function address register and the access type matches the given configuration, an interrupt is asserted. The operand address break function will cause an INT9 exception by hardware.

### Data value break

The operand address break feature can be combined with an explicit data value match by using the patch data registers to program the data match rules (data value and data mask).

To use this function, do the following:

- Do the same steps described above for the operand address break.
- Consider that only channels 0/2/4/6 can be used. The odd numbered channels' PFD registers 1/3/5/7 are for specifying the data mask. Do not enable PE of the according odd channel.[1]
- Program the data value into PFD[x] of an even numbered channel.
- Program the data mask into PFD[x+1] (odd numbered channel). Bits set to '1' are not compared. The exact value of PFD[x] is matched when PFD[x+1] is equal to $0000_H$. In case of byte access to variables should be matched, consider the mask usage for the according byte lane (odd address: mask the lower byte, even address: mask the upper byte).
- Enable the IEx bit
- Enable the PEx bit
- Set the EDSU_EN bit to '1'

When an address matches the value set in the patch function address register and the access type matches the given configuration and the data matches the value given in the patch data register, an interrupt is asserted. The data value break function will cause an INT9 exception by hardware.

1. **The odd numbered channel can still be used as operand address break without mask, since it does not need the PFD register.**

# 32. ROM/RAM Mirroring Module

This chapter explains the ROM mirroring module.

32.1 Outline of ROM/RAM Mirroring Module

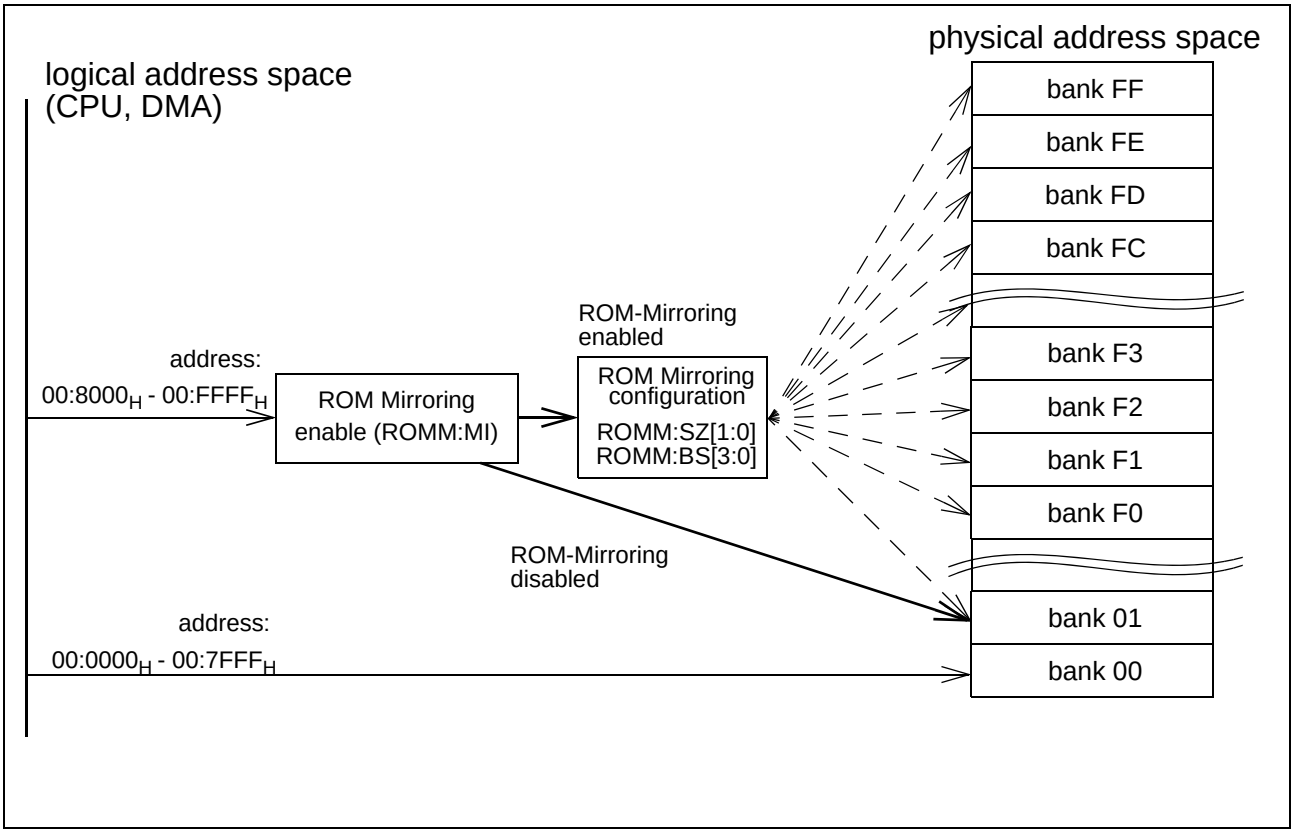
32.2 ROM Mirroring Register (ROMM)

## 32.1 Outline of ROM/RAM Mirroring Module

The ROM/RAM-Mirroring module maps the logical address (e.g. CPU address) of bank 00 to a physical address in a selected ROM bank. The mirrored ROM size is configurable. Unused ROM-Mirror area is used for mirroring RAM from bank 01.

### Block diagram of ROM mirroring module

Figure 32-1. Block diagram of ROM/RAM mirroring module




Only ROM-addresses of the upper half of ROM banks $F0_H$ to $FF_H$ (Fx:$8000_H$ to Fx:$FFFF_H$) can be mirrored to bank $00_H$. The lower half of a ROM bank is never mirrored. If the ROM-Mirroring function is disabled, bank $01_H$ is mirrored to bank $00_H$.

## 32.2 ROM Mirroring Register (ROMM)

The ROM Mirroring register (ROMM) configures all functions of the ROM mirroring module:

- the mirrored ROM bank
- the size of the mirrored ROM
- the ROM mirror enable

**ROM mirroring register (ROMM)**

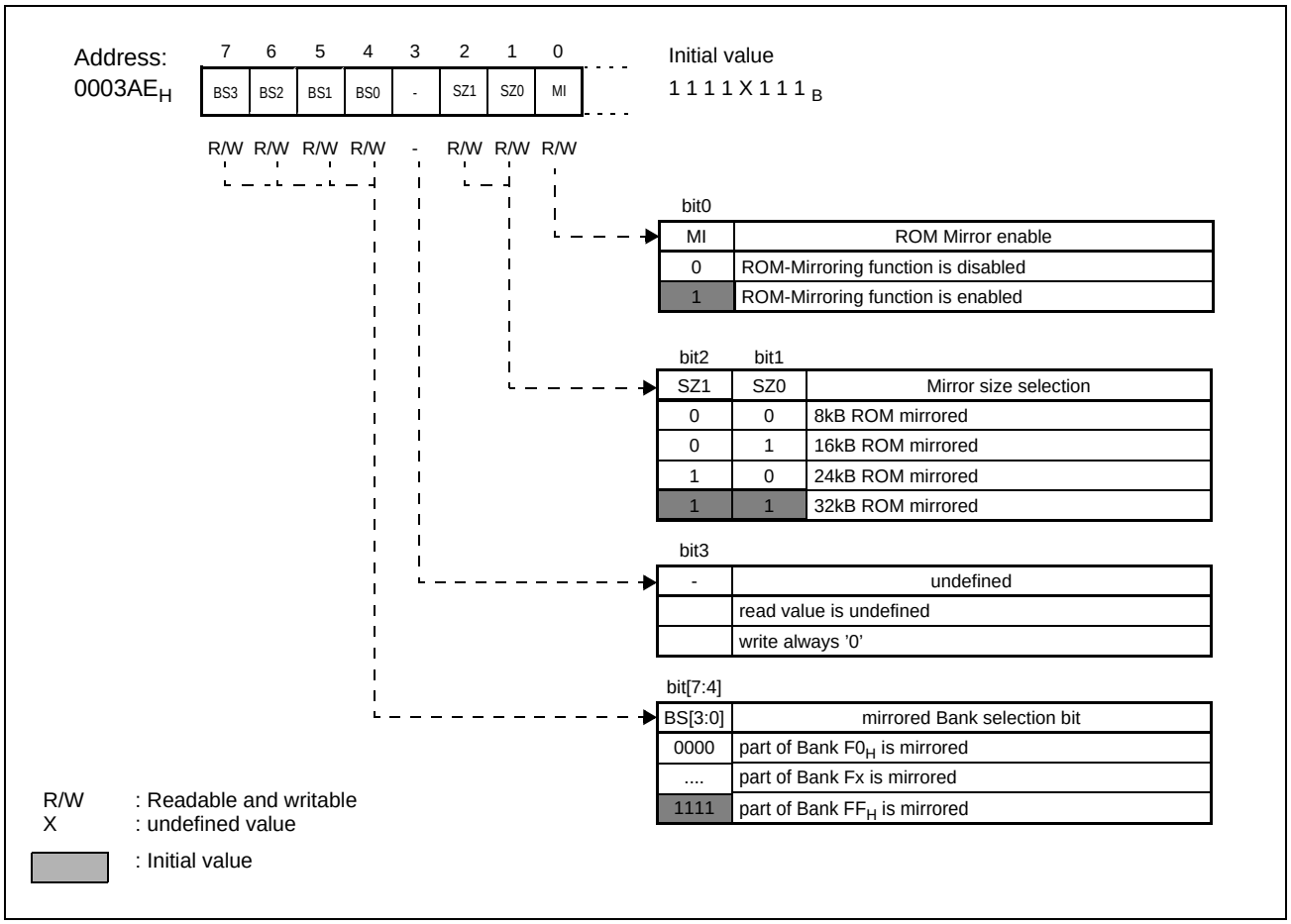


Table 32-1. Description of ROMM

| Bit Name | | Function |
|---|---|---|
| bit [7:4] | BS[3:0]:<br>bank selection bits | These bits select the mirrored ROM-bank. BS[3:0] correspond to bit [3:0] of the memory bank address.<br><br>**ROMM:BS[3:0] / mirrored Memory bank**<br>1111 / FF<br>1110 / FE<br>1101 / FD<br>... / ...<br>0001 / F1<br>0000 / F0<br><br>The initial value is ’1111’ (Bank $FF_H$ selected). |
| bit 3 | undefined | Write always ’0’ to this bits.<br>The read value is undefined.<br>RMW instructions are not affected. |
| bit [2:1] | SZ[1:0]<br>Mirror size select bits | These bits select the size of the mirrored ROM-Bank. See Table 32-2 for detailed description.<br>The initial value is ’11’. |
| bit 0 | MI:<br>Mirror enable bit | This bit enables the ROM mirroring function.<br>• Writing ’0’ disables the ROM-Mirroring function. The addresses $01:8000_H$ to $01:FFFF_H$ are mirrored to bank 00.<br>• Writing ’1’ enables the ROM-Mirroring function. The image of the ROM data in the bank selected by ROMM:BS[3:0] can also be accessed in the 00 bank.<br>• The initial value is ’1’ (ROM-Mirror enabled). |

If the selected ROM-Mirror size is less than 32kB, address area from bank $01_H$ is mirrored to bank $00_H$ instead. If bank $01_H$ contains RAM above address $01:8000_H$ this RAM can be accesses via bank $00_H$.

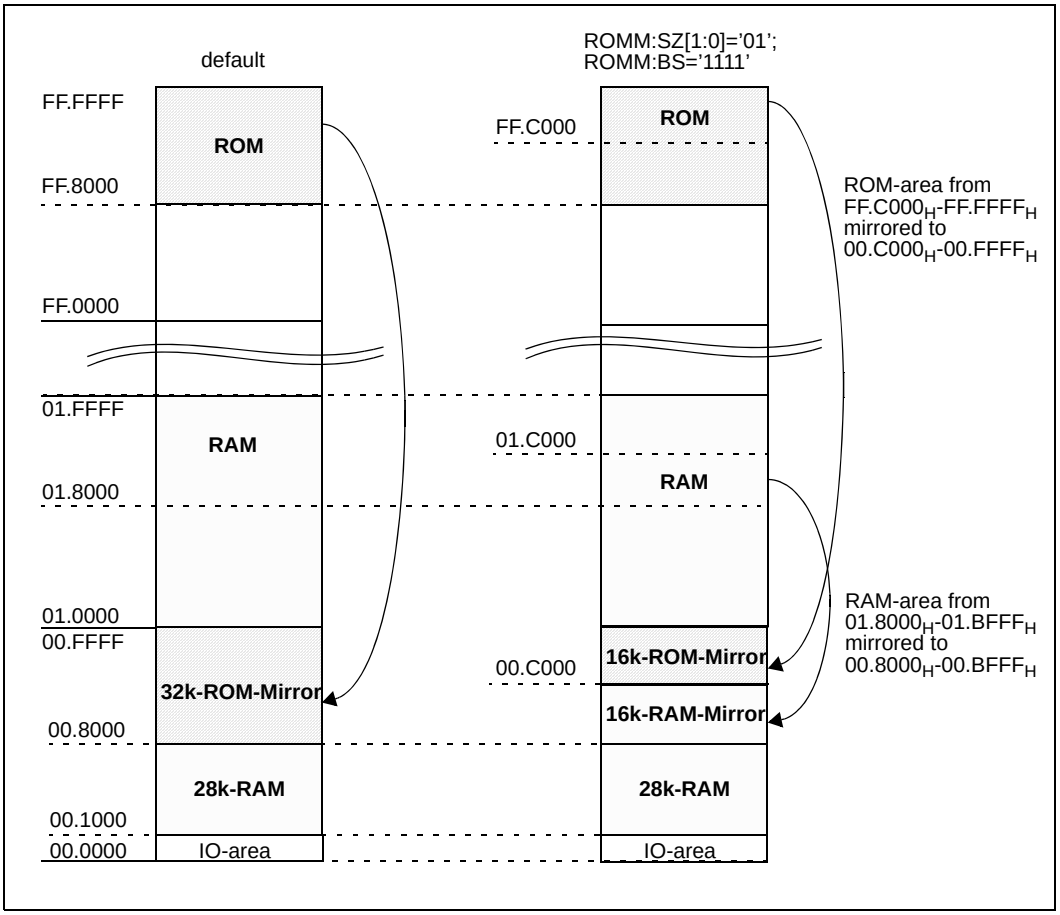
Table 32-2. description of Mirror size select bits MS[1:0]

| MI | SZ1 | SZ0 | | mirrored ROM/RAM-Size | selected ROM/RAM-area for mirroring (physical address) | ROM/RAM-mirrored to area (logical address) |
|---|---|---|---|---|---|---|
| 0 | - | - | ROM | 0k | no ROM mirroring | no ROM mirroring |
| | | | RAM | 32k | $01.8000_H$-$01.FFFF_H$ | $00.8000_H$-$00.FFFF_H$ |
| 1 | 0 | 0 | ROM | 8k | $Fx.E000_H$ - $Fx.FFFF_H$ | $00.E000_H$ - $00.FFFF_H$ |
| | | | RAM | 24k | $01.8000_H$-$01.DFFF_H$ | $00.8000_H$-$00.DFFF_H$ |
| 1 | 0 | 1 | ROM | 16k | $Fx.C000_H$ - $Fx.FFFF_H$ | $00.C000_H$ - $00.FFFF_H$ |
| | | | RAM | 16k | $01.8000_H$-$01.BFFF_H$ | $00.8000_H$-$00.BFFF_H$ |
| 1 | 1 | 0 | ROM | 24k | $Fx.A000_H$ - $Fx.FFFF_H$ | $00.A000_H$ - $00.FFFF_H$ |
| | | | RAM | 8k | $01.8000_H$-$01.9FFF_H$ | $00.8000_H$-$00.9FFF_H$ |
| 1 | 1 | 1 | ROM | 32k | $Fx.8000_H$ - $Fx.FFFF_H$ | $00.8000_H$ - $00.FFFF_H$ |
| | | | RAM | 0k | no RAM-Mirroring | no RAM-Mirroring |

**Note:**

Do not write the ROM mirroring register (ROMM) when code is executed from addresses inside the mirrored memory area.

Figure 32-2. example for ROM-Mirroring function



# 33. Flash Memory

This chapter explains the functions and operation of the flash memory.

33.1 Overview of the Flash Memory

33.2 Block Diagram of the Flash Memory and Sector Configuration of the Flash Memory

33.3 Flash Memory Modes

33.4 Program/Data Flash (Main Flash) Memory Control Registers

33.5 Data Flash Memory Control Registers

33.6 Starting the Flash Memory Automatic Algorithm

33.7 Confirming the Automatic Algorithm Execution State

33.8 Detailed Explanation of Writing to and Erasing Flash Memory

33.9 Notes on using Flash Memory

33.10 Flash memory programming example

## 33.1 Overview of the Flash Memory

The flash memory is used to store the user program and data. Different types of Flash memory modules are available in F$^2$MC-16FX Flash MCU products. See "User ROM Memory map for Flash devices" in the datasheet for a detailed flash configuration of each device.

Instructions from the CPU can be used to write data to and erase data from the flash memory. Internal CPU control therefore enables rewriting of the flash memory while the MCU is mounted on a PCB. As a result, improvements in programs and data can be performed efficiently.

### 33.1.1 Flash memory types

Different types of flash modules are available in F$^2$MC-16FX MCUs. See the datasheet which memory types are available in each device. The datasheet also describes a detailed sector configuration for each device.

**Program/Data Flash (Main Flash)**

This is the standard flash macro which is optimized for high read performance. Up to two modules of this type (Flash A and Flash B) are available on F$^2$MC-16FX MCUs.

The Flash A is mapped in the CPU memory map to the banks DF to FF. Small sectors are located in bank DF. Depending on the size of the Flash A, large sectors are located starting at bank E0 or higher up to bank FF.

On some devices, a second flash memory, called Flash B is available which can be operated independently. The small sectors of this Flash are located in bank DE. They can be used for on-chip EEPROM emulation, boot loader etc.

Devices with a bigger Flash B also have large sectors between bank E0 and FF.

### Data Flash

This memory type is optimized for programming by CPU and on-chip EEPROM emulation. Also up to two modules of this type (Data Flash A and Data Flash B) are available on $F^2$MC-16FX MCUs.

The sectors of the Data Flash A are mapped to bank 0D and 0E. The sectors of the Data Flash B are mapped to bank 0C and 0E.

## 33.1.2 Flash memory features

### Common features of all Flash modules

- Use of automatic program algorithm (Embedded Algorithm: Equivalent to Spansion MBM29LV200TC).
- Erase pause/restart functions provided.
- Detection of completion of writing/erasing using data polling or toggle bit functions.
- Detection of completion of writing/erasing using CPU interrupts.
- Sector erase function (any combination of sectors).
- Fast Mode programming function (equivalent to Spansion MBM29LV200TC)

### Features of Program/Data Flash (Main Flash)

- Program/erase possible by CPU (user program and serial flash writer) and in Flash mode (parallel flash writer).
- Byte and Word programming possible in CPU and Flash mode.
- Flash reading cycle time: Minimum of 1 machine cycles.
- CPU reading is done 32 bit wide. The complete 32 bit word is stored in a read buffer and available without wait cycles in a following CPU read access. Independent buffer for code and data access.
- Flexible programming of the Flash control signals and wait cycles to achieve best access performance at any machine clock frequency.
- Programmable erase/write protection in CPU mode (sector-wise).
- Code security to prevent read-out in a way not intended by the application.

### Features of Data Flash

- Program/erase possible by CPU only (user program and serial flash writer).
- Data width of Flash macro is 8 bit.
- Byte and Word reading supported by internal sequencer hardware.
- 2 CPU write access modes: direct access and command sequencer access.
- Write access data width: 8-bit in direct access mode, 8/16-bit in command sequencer access mode.
- Support for data programming by DMA.

### Note:

The Flash memory manufacturer code and device code are not available. The corresponding read commands specified for MBM29LV200TC are not supported.

## 33.1.3 Writing to/erasing flash memory

While the flash memory is erased or data is written to the flash memory, reading data is not possible. Hence, when it is required to read data or code from the flash memory while the memory is erased or written, the required data or code must be copied to RAM before starting the erase/write operation. This eliminates the need for the program to read the flash memory while the flash memory is erased or written.

If more than one flash module is available, then it is possible to erase/write one Flash module while reading code and/or data from another flash module. Hence, there is no need to copy data or code to RAM.

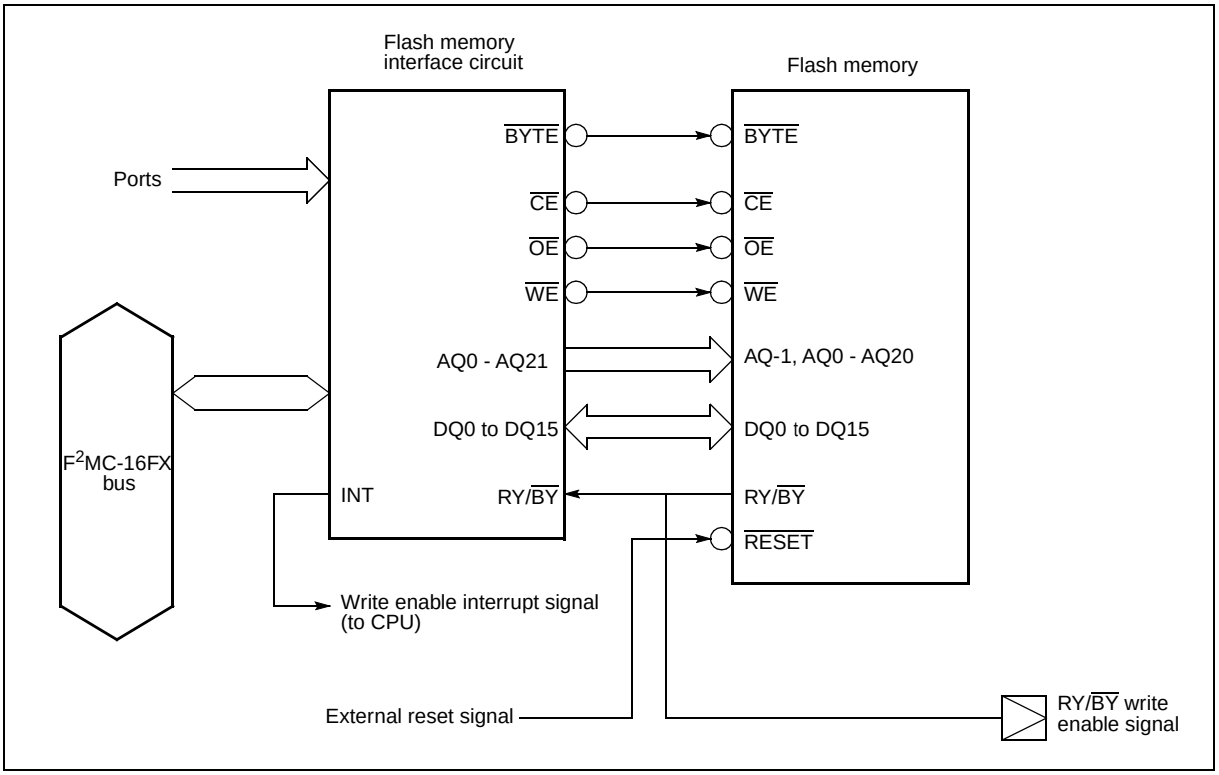
A sector erase operation can be suspended. In this suspend state it is possible to read data from other sectors.

# 33.2 Block Diagram of the Flash Memory and Sector Configuration of the Flash Memory

This chapter shows a basic block diagram of the embedded flash memory and the global sector configuration of the different flash memory models.

### Block diagram of the flash memory

Figure 33-1. Block diagram of the flash memory




### Sector configuration of the flash memory

Figure 33-2 shows the sector configuration of the Program/Data Flash memory (Main Flash memory). The addresses in the figure indicate the high-order and low-order addresses of each sector.

Depending on the size of the Flash A memory, up to 8 small sectors of 8KByte size each are available in SA0 ... SA7, starting at SA0 upwards.

Depending on the size of the Flash A memory, a number of large sectors of 64KByte size are available in S39 ... S8, starting at S39 downwards.

When the Flash B memory is available on the product, depending on the size of the Flash B memory, up to 8 small sectors of 8KByte size each are available in SB0 ... SB7, starting at SB0 upwards.

Depending on the size of the Flash B memory, an additional number of large sectors of 64KByte size are available below the large sectors of Flash A.

See the datasheet for the availability of all sectors in each device.

Figure 33-2. Sector configuration of the Program/Data Flash memory (Main Flash)

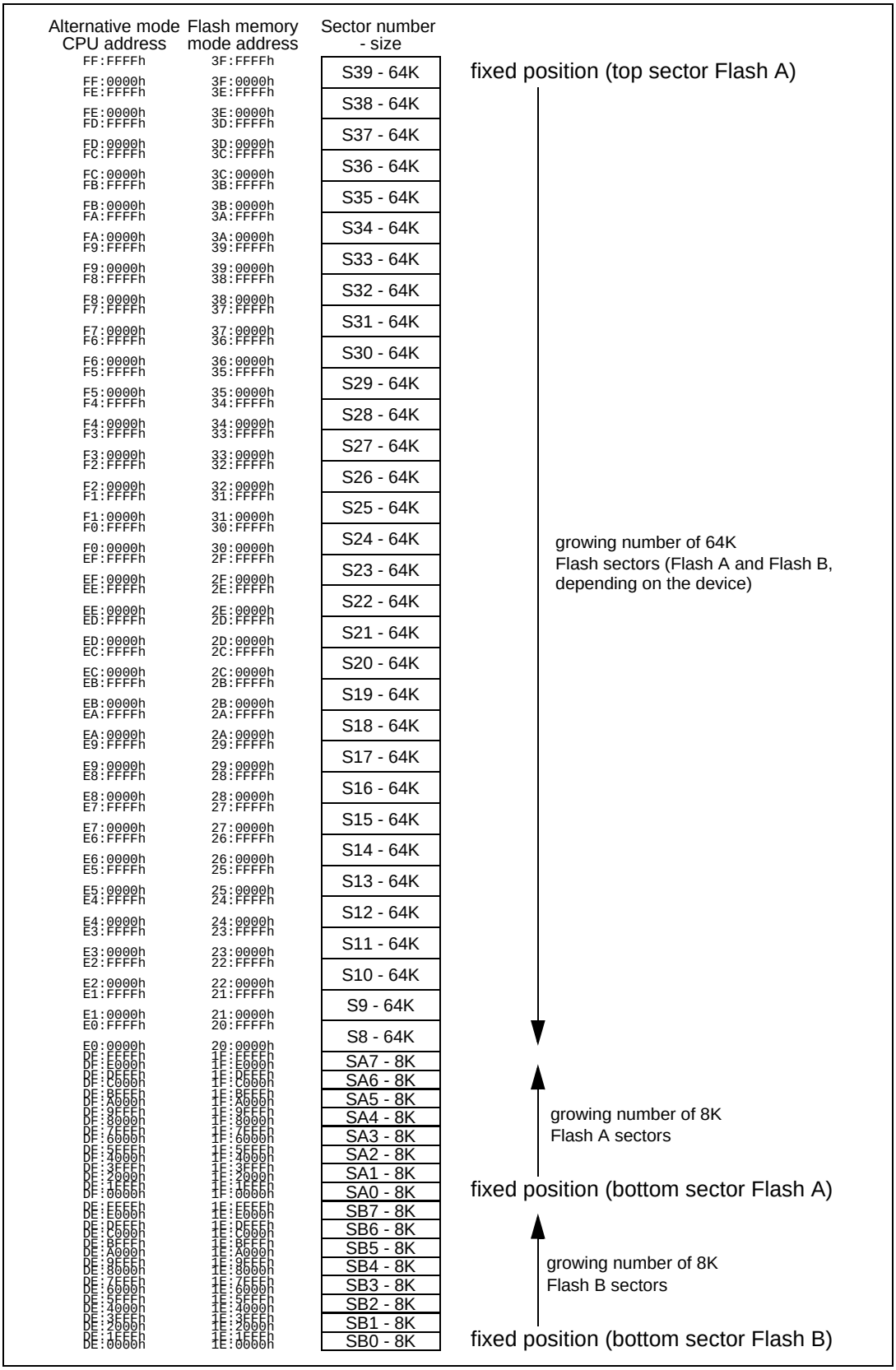


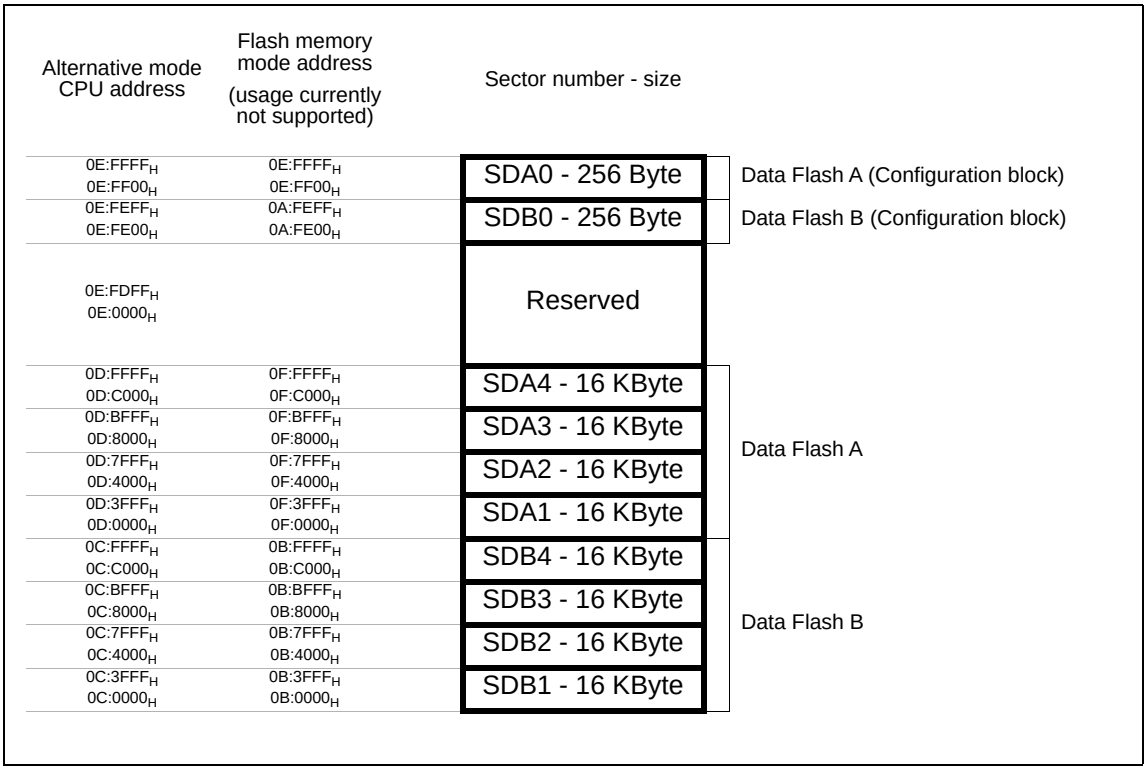
Figure 33-3 shows the sector configuration of the Data Flash memory for products supporting this feature (see the datasheet if and which Data Flash modules are available in each device).

The Data Flash has a small 256 Byte sector for configuration of the Flash security function. This is mapped to CPU bank 0E for both Data Flash memories (sectors SDA0 and SDB0).

Each Data Flash module has 4 Data sectors of 16KByte. Sectors of Data Flash A are mapped to CPU bank 0D while sectors of Data Flash B are mapped to CPU bank 0C.

The Flash memory mode addresses are given for reference only. Please note that the Data Flash modules are currently not supported by parallel flash writer.

Figure 33-3. Sector configuration of the Data Flash memory



## 33.3 Flash Memory Modes

The flash memory can be accessed in two different ways: Flash memory mode and CPU mode (alternative mode). Flash memory mode enables data to be directly written to or erased from the external pins. The CPU mode enables data to be written to or erased from the CPU via the internal bus. Use the mode pins MD[2:0] to select the mode as described in Table 8-3.

### Flash memory mode

The CPU stops when the mode pins are set to MD[2:0] = 111 while the reset signal is asserted. The flash memory interface circuit is connected directly to ports, enabling direct control from the external pins. This mode makes the MCU seem like a standard flash memory to the external pins, and write/erase can be performed using a flash memory programmer.

In flash memory mode, all operations supported by the flash memory automatic algorithm can be used.

For more details about the Flash memory mode, refer to the parallel Flash programming specification.

**Note:**

The sector protect function of MBM29LV200TC via $V_{ID}$ (12 V) pin is not supported. Instead, there is a sector write protection that prevents accidental erase/write in alternative mode. Please refer to 33.4.3 Flash Memory Write Control registers 0-5 (FMWC0-FMWC5) for details.

**Note:**

The Data Flash requires different control signals for parallel programming. Please check with your programming equipment maker for support of this interface. A sector write protection as for the Program/Data Flash (Main Flash) is not supported by the Data Flash.

### CPU mode (Alternative mode)

The Program/Data Flash memory is located in the DE to FF banks in the CPU memory space. The Data Flash memory is located in the 0C to 0E banks. Both can be read-accessed like ordinary mask ROM and also program-accessed from the CPU via the flash memory interface circuit.

Since writing/erasing the flash memory is performed by instructions from the CPU via the flash memory interface circuit, this mode allows rewriting even when the MCU is soldered on the target board.

## 33.4 Program/Data Flash (Main Flash) Memory Control Registers

This chapter describes the control/status registers of the Program/Data Flash (Main Flash) which are needed to control the flash memory erase/write process and to configure the read access timing.

### Memory Control Registers

The Flash A memory is controlled by the following registers:

- Memory Control/Status Register A (MCSRA)
- Memory Timing Configuration Registers A (MTCRAL/MTCRAH)

The Flash B memory is controlled by the following registers:

- Memory Control/Status Register B (MCSRB)
- Memory Timing Configuration Registers B (MTCRBL/MTCRBH)

The permission to erase/write a sector of the Flash memory is stored in the following register:

- Flash Memory Write Control registers FMWC0-5

### 33.4.1 Memory Control Status Register (MCSRA, MCSRB)

The memory control status register (MCSRA for Flash A memory, MCSRB for Flash B memory) controls basic functions of the Flash memory interface circuit for reading and for program/erase.

### Memory Control Status register (MCSRA, MCSRB)

Figure 33-4. Configuration of the Memory Control Status Register (MCSRA, MCSRB)

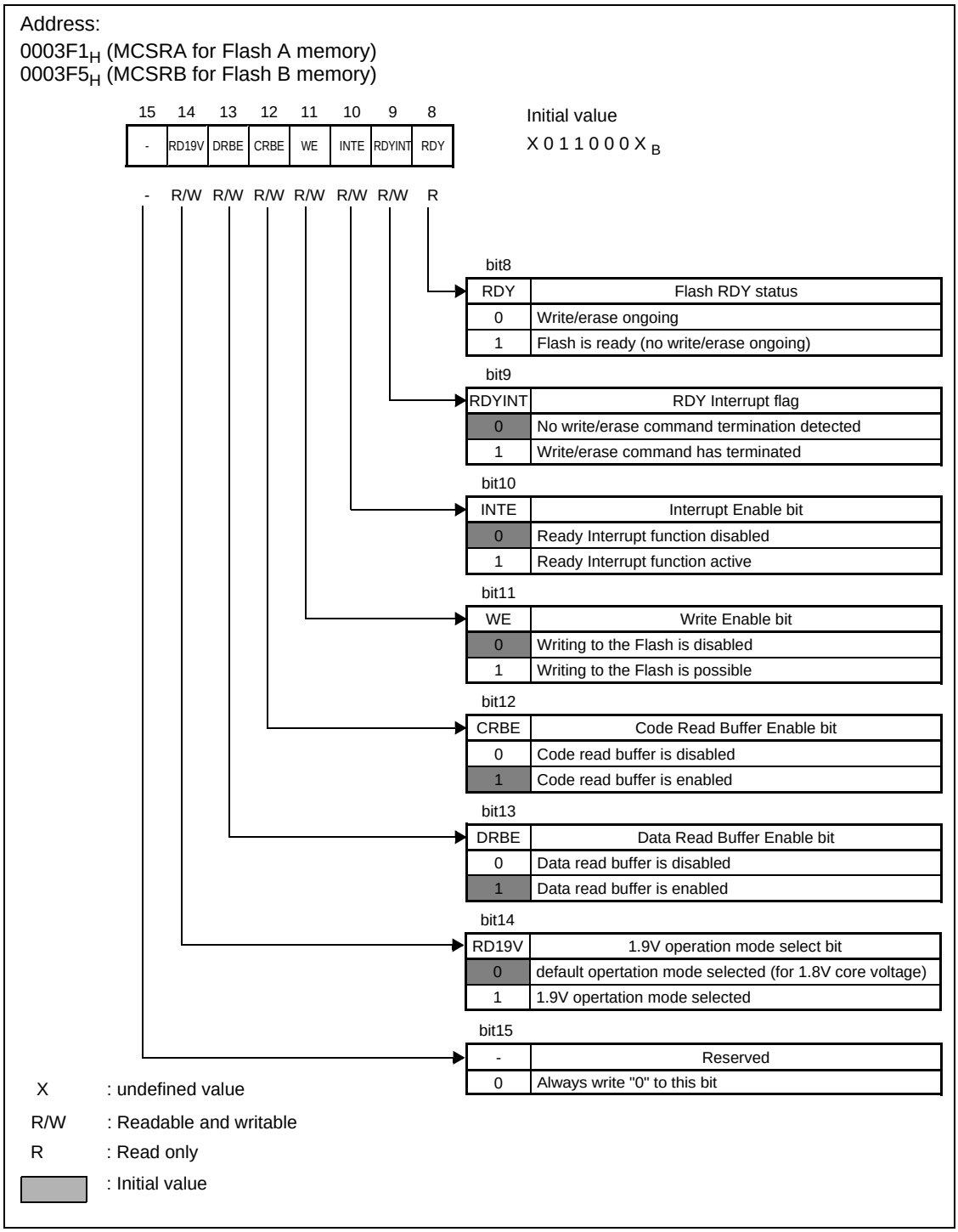


Table 33-1. Function of each bit of the Memory Control Status Register (MCSRA/MCSRB)

| Bit names | | Function |
|---|---|---|
| bit 8 | RDY:<br>Flash RDY status | • This bit shows the status of the (sampled) Flash RDY output.<br>• Writing to this bit has no effect.<br>• A read value of '0' indicates that a program/erase command is currently executed. Only the reset and suspend commands are accepted in this state.<br>• A read value of '1' indicates that no program/erase command is currently executed or that the Flash is in the erase-suspend state. Write/erase commands can be written to the Flash. |
| bit 9 | RDYINT:<br>RDY Interrupt flag | • This bit is the interrupt flag of the Flash ready interrupt function.<br>• This bit is initialized to '0' by reset.<br>• It is set to '1' by a rising edge of the RDY signal.<br>• The RDYINT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>• The read cycle of a read-modify-write access always returns '1'. |
| bit 10 | INTE:<br>Interrupt Enable bit | • This bit enables the interrupt function of the RDYINT flag (write/erase command termination interrupt).<br>• An interrupt is generated when this bit is set to '1' and the RDYINT flag is '1' or gets set by a rising edge of the RDY signal.<br>• When this bit is set to '0', no RDYINT interrupt can be generated.<br>• The initial value of the INTE bit is '0'.<br>• This bit is read- and writable. |
| bit 11 | WE:<br>Write Enable bit | • This bit enables/disables writing to the Flash.<br>• WEX is forced to '1' when this bit is '0'. Hence no write/erase commands can be sent to the Flash.<br>• Setting this bit to '1' enables WEX and command writing.<br>• The initial value of this bit is '0'. |
| bit 12 | CRBE:<br>Code Read Buffer Enable bit | • This bit enables/disables the code read buffer.<br>• Setting this bit to '1' enables the code read buffer. A code read access from any address reads two 16-bit words, the one that is addressed and the neighboring one, such that the resulting two 16-bit words are aligned to an even address. Both words are stored into a read buffer. A later CPU read access to the address of the buffered data will be executed without wait cycles.<br>• Setting this bit to '0' disables the code read buffer. Code is always read directly from the Flash.<br>• The initial value of this bit is '1' (buffer enabled). |
| bit 13 | DRBE:<br>Data Read Buffer Enable bit | • This bit enables/disables the data read buffer.<br>• Setting this bit to '1' enables the data read buffer. A data read access from any address reads two 16-bit words, the one that is addressed and the neighboring one, such that the resulting two 16-bit words are aligned to an even address. Both words are stored into a read buffer. A later CPU read access to the address of the buffered data will be executed without wait cycles.<br>• Setting this bit to '0' disables the data read buffer. Data is always read directly from the Flash.<br>• The initial value of this bit is '1' (buffer enabled). |
| bit 14 | RD19V: 1.9V operation mode select bit | • This bit must be set according to the configuration of the voltage regulator (1.8V or 1.9V core voltage, please refer to Standby Mode and Voltage Regulator Control Circuit on page 187).<br>• For a core voltage of 1.8V, this bit must be set to '0'.<br>• For a core voltage of 1.9V, this bit must be set to '1'.<br>• The maximum allowed frequencies for the internal clocks and Flash timing settings depend on this bit and the selected core voltage. Always set the voltage regulator and this bit to 1.9V mode before changing to a clock frequency which is permitted only in 1.9V mode.<br>• The initial value of this bit is '0'. |
| bit 15 | Reserved | • Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected. |

## 33.4.2 Memory Timing Configuration Register (MTCRA, MTCRB)

The Memory Timing Configuration register (MTCRA, MTCRB) is used to configure the access mode and the number of wait states of the flash memory. Table 33-4 shows recommended settings.

### Memory Timing Configuration Register Low byte (MTCRAL, MTCRBL)

Figure 33-5. Configuration of the Memory Timing Configuration Register (MTCRAL, MTCRBL)

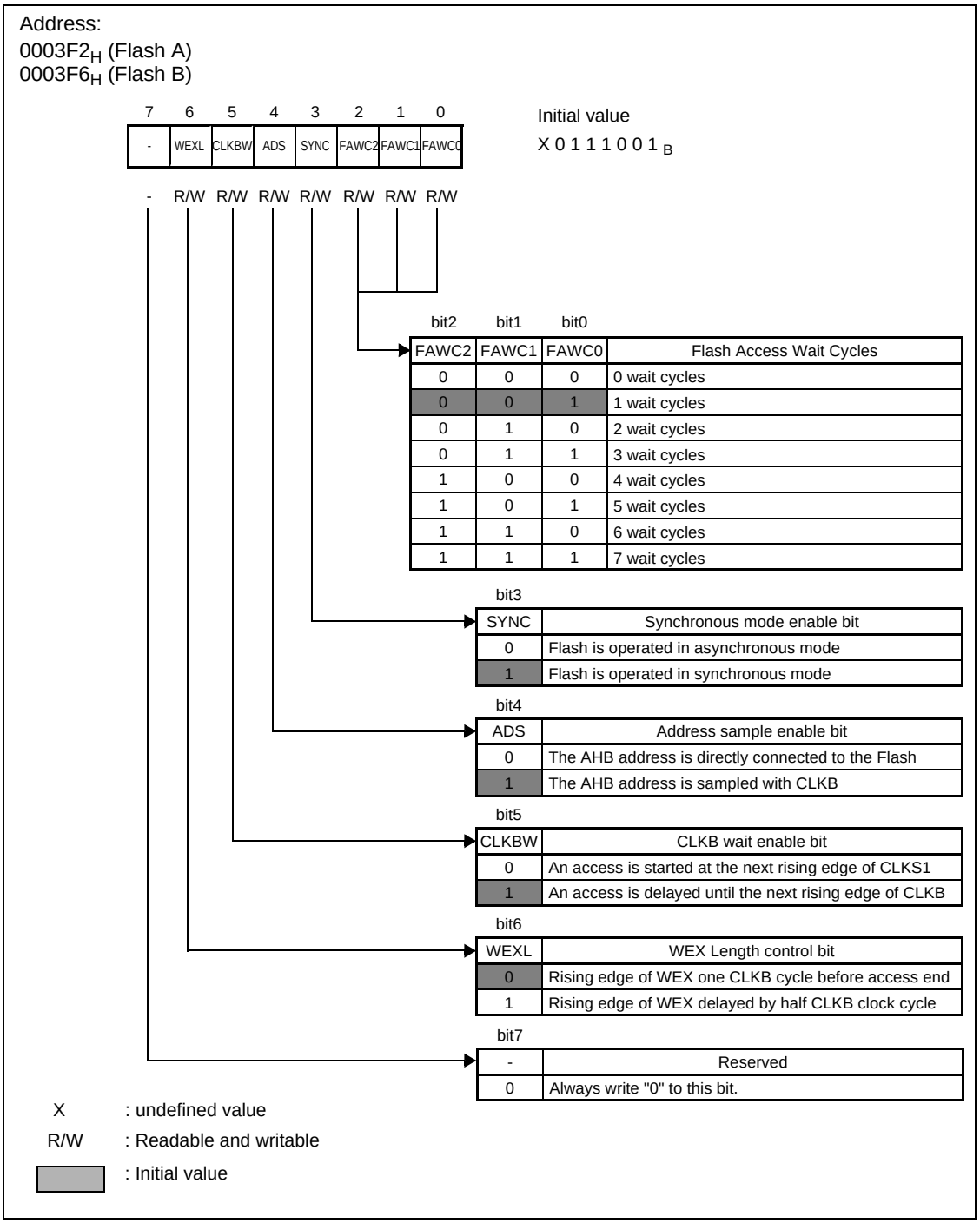


Table 33-2. Function of each bit of the Memory Timing Configuration Register (MTCRAL/MTCRBL)

| Bit name | | Function |
|---|---|---|
| bit 0 - bit 2 | FAWC0 to FAWC2:<br>Flash Access Wait cycles | • These bits define how many CPU wait cycles (CLKB cycles) are added at any read and write access to the Flash.<br>• These bits must be set depending on the machine clock frequency and the other configuration bits.<br>• The initial value of these bits is "001" (1 wait cycle) |
| bit 3 | SYNC:<br>Synchronous mode enable bit | • This bit selects between synchronous or asynchronous operation of the Flash in CPU mode.<br>• Setting this bit to ’1’ selects the synchronous mode. The SYNC input of the Flash is set to ’1’ and the ATDIN and EQIN input signals are used.<br>• Setting this bit to '0’ selects the asynchronous mode. ATDIN and EQIN are not used.<br>• The setting of this bit affects read and write operations.<br>• This bit is initialized to ’1’ by any reset. |
| bit 4 | ADS:<br>Address Sample enable bit | • This bit selects if the direct or sampled address signals are connected to the Flash.<br>• When this bit is set to ’0’, the AHB address signals are connected directly to the Flash.<br>• Setting this bit to ’1’ activates the address sample stage. The address is sampled with CLKB at the beginning of the access and memorized until the end of the access.<br>• Setting this bit to ’1’ is only allowed for timings with odd value of MTCRxH:ATDL (MTCRxH:ATDL0=1). The recommended settings in Table 33-4 are not affected.<br>• In asynchronous mode and for writing to the Flash, this bit must be set to ’1’.<br>• The initial value of this bit is ’1’. |
| bit 5 | CLKBW:<br>CLKB Wait enable bit | • This bit defines the start time of a Flash access.<br>• When this bit is set to ’0’, then the Flash access starts with the next rising edge of CLKS1.<br>• Setting this bit to ’1’ postpones the Flash access start to the next rising edge of CLKB.<br>• Before writing to the Flash, this bit must be set to ’1’.<br>• This bit has no effect when CLKB is set to CLKS1 (divided by ’1’).<br>• The initial value of this bit is ’1’. |
| bit 6 | WEXL:<br>WEX Length control bit | • This bit controls the timing of the WEX rising edge.<br>• When setting this bit to ’0’, then WEX goes to ’1’ at the last rising edge of CLKB before the write access end (one CLKB cycle before the end of the access).<br>• Setting this bit to ’1’ extends the WEX assertion by a half CLKB clock cycle.<br>• The number of wait cycles defined by the FAWC bits must be sufficiently high to generate a valid WEX pulse.<br>• The initial value of this bit is ’0’. |
| bit 7 | Reserved | • Always write "0" to these bits.<br>• The read value of these bits is undefined.<br>• Read modify write operations to this register are not affected. |

### Memory Timing Configuration Register high byte (MTCRAH, MTCRBH)

Figure 33-6. Configuration of the Memory Timing Configuration Register (MTCRAH, MTCRBH)

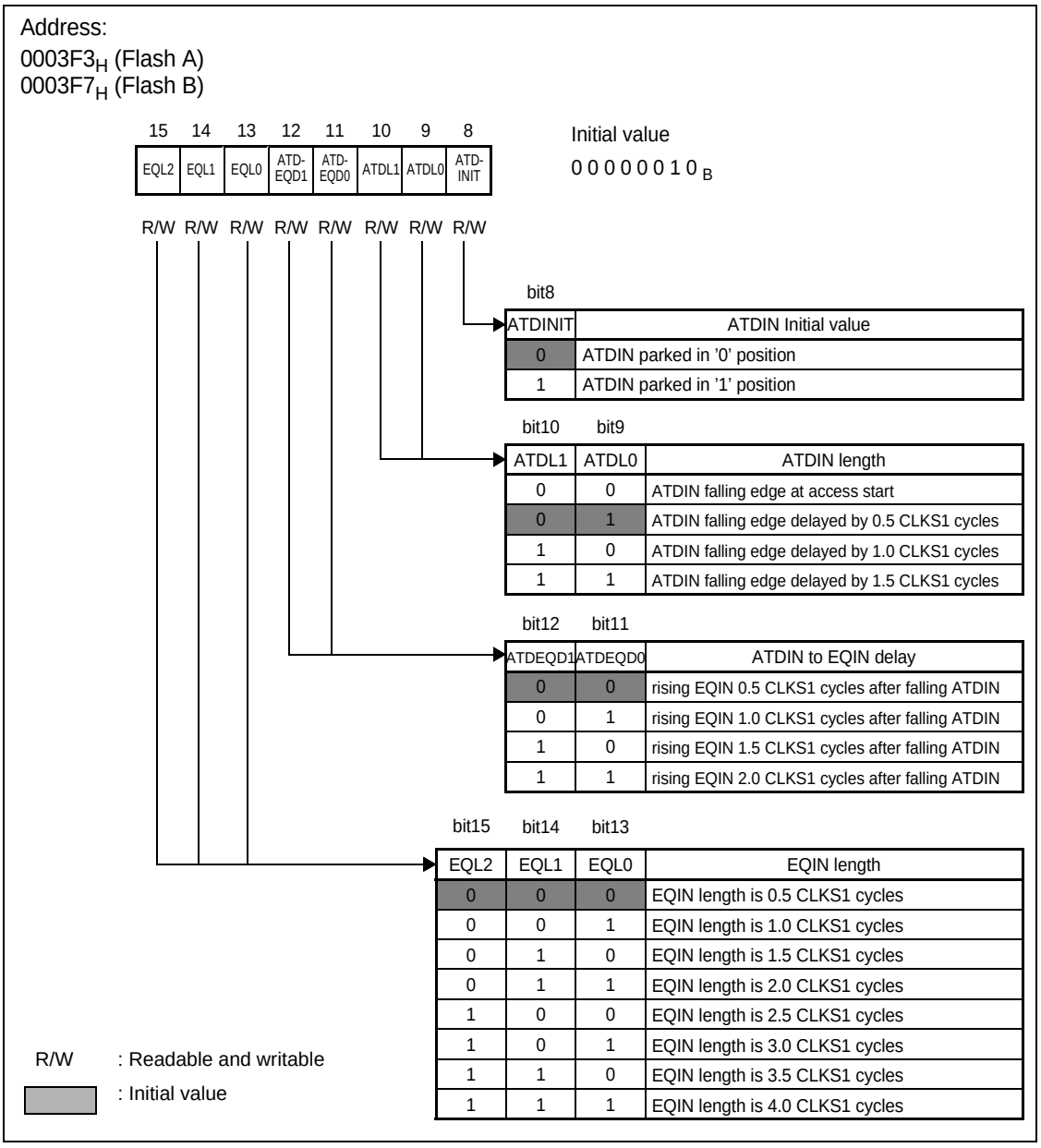


Table 33-3. Function of each bit of the Memory Timing Configuration Register (MTCRAH/MTCRBH)

| Bit name | | Function |
|---|---|---|
| bit 8 | ATDINIT:<br>ATDIN Initial value | • This bit defines the initial value of the ATDIN signal.<br>• When setting this signal to "0", then ATDIN is parked in the "0" position. It is asserted to "1" at the beginning of a Flash access (read or write). ATDL must be set to "01" or higher in this case.<br>• Setting this bit to "1" parks ATDIN in the "1" position. ATDIN becomes "1" already at the end of the pervious Flash access.<br>• This bit is initialized to "0" by any reset. |
| bit 9 - bit 10 | ATDL0 to ATDL1:<br>ATDIN length | • These bits define the length of the ATDIN pulse according to the table in Figure 33-6.<br>• The setting must be done depending on the CLKS1 frequency and the level of ATDINIT.<br>• These bits also affect the assertion time of WEX in case of a write access (in synchronous and asynchronous mode).<br>• These bits are initialized to "01" by any reset. |
| bit 11 - bit 12 | ATDEQD0 to ATDEQD1:<br>ATDIN to EQIN delay | • These bits define the length of the pause between the ATDIN falling edge and the EQIN rising edge according to the table in Figure 33-6.<br>• The setting must be done depending on the CLKS1 frequency.<br>• These bits also affect the assertion time of WEX in case of a write access (in synchronous and asynchronous mode).<br>• These bits are initialized to "00" by any reset. |
| bit 13 - bit 15 | EQL0 to EQL2:<br>EQIN length | • These bits define the length of the EQIN pulse according to the table in Figure 33-6.<br>• The setting must be done depending on the CLKS1 frequency.<br>• These bits are initialized to "000" by any reset. |

## Recommended Flash Timing Settings

Table 33-4 shows recommended Flash Memory Timing configurations. The maximum operating frequency of different timing configurations depends on the core supply voltage. Please refer to Standby Mode and Voltage Regulator Control Circuit on page 187 for a description how to program the core supply voltage.

Table 33-4. Recommended settings of MTCRA and MTCRB

| MTCRA, MTCRB | Permitted frequency range for CLKS1 *1) | | Permitted CLKB divider setting | Resulting CLKB frequency range *1) | | Purpose |
|---|---|---|---|---|---|---|
| | 1.8V operation | 1.9V operation | | 1.8V operation | 1.9V operation | |
| 6E3Dh | $f_{CLKS1} \leq 100$ MHz (can be used at any setting, especially for mode transitions) | | 1 ... 16 | $f_{CLKB} \leq 100$ MHz (can be used at any setting, especially for mode transitions) | | Synchronous read with 5 wait states |
| 6E3Fh | | | | | | Synchronous read/write with 7 wait states |
| 0239h (init val.) | $f_{CLKS1} \leq 25$ MHz | | 1 ... 16 | $f_{CLKB} \leq 25$ MHz | | Synchronous read with 1 wait state |
| 2129h | $f_{CLKS1} \leq 30$ MHz | $f_{CLKS1} \leq 32$ MHz | 1 ... 16 | $f_{CLKB} \leq 30$ MHz | $f_{CLKB} \leq 32$ MHz | Synchronous read with 1 wait state |
| 233Ah | $f_{CLKS1} \leq 50$ MHz | $f_{CLKS1} \leq 56$ MHz | 1 ... 16 | $f_{CLKB} \leq 50$ MHz | $f_{CLKB} \leq 56$ MHz | Synchronous read with 2 wait states |
| | $f_{CLKS1} \leq 20$ MHz | | | $f_{CLKB} \leq 20$ MHz | | Synchronous read/write with 2 wait states |
| 4B3Bh | $f_{CLKS1} \leq 60$ MHz | $f_{CLKS1} \leq 64$ MHz | 1 ... 16 | $f_{CLKB} \leq 60$ MHz | $f_{CLKB} \leq 64$ MHz | Synchronous read with 3 wait states |
| 4B3Dh | | | | | | Synchronous read/write with 5 wait states |
| 2128h | $f_{CLKS1} \leq 30$ MHz | $f_{CLKS1} \leq 32$ MHz | 2 ... 16 | $f_{CLKB} \leq 15$ MHz | $f_{CLKB} \leq 16$ MHz | Synchronous read with 0 wait states |

Table 33-4. Recommended settings of MTCRA and MTCRB

| MTCRA, MTCRB | Permitted frequency range for CLKS1 *1) | | Permitted CLKB divider setting | Resulting CLKB frequency range *1) | | Purpose |
|---|---|---|---|---|---|---|
| | 1.8V operation | 1.9V operation | | 1.8V operation | 1.9V operation | |
| 2208h | 20 MHz ≤ $f_{CLKS1}$ ≤ 50 MHz | 20 MHz ≤ $f_{CLKS1}$ ≤ 56 MHz | 2 | 10 MHz ≤ $f_{CLKB}$ ≤ 25 MHz | 10 MHz ≤ $f_{CLKB}$ ≤ 28 MHz | Synchronous read with 0 wait states |
| 4C09h | $f_{CLKS1}$ ≤ 76 MHz | $f_{CLKS1}$ ≤ 84 MHz | 2 | $f_{CLKB}$ ≤ 38 MHz | $f_{CLKB}$ ≤ 42 MHz | Synchronous read with 1 wait state |
| 6B09h | 20 MHz ≤ $f_{CLKS1}$ ≤ 92 MHz | 20 MHz ≤ $f_{CLKS1}$ ≤ 100 MHz | 2 | 10 MHz ≤ $f_{CLKB}$ ≤ 46 MHz | 10 MHz ≤ $f_{CLKB}$ ≤ 50 MHz | Synchronous read with 1 wait state |
| 2279h | $f_{CLKS1}$ ≤ 40 MHz | | 2 ... 16 | $f_{CLKB}$ ≤ 20 MHz | | Synchronous read/write with 1 wait state |
| 0231h | $f_{CLKS1}$ ≤ 7 MHz | | 1 ... 16 | $f_{CLKB}$ ≤ 7 MHz | | Asynchronous read/write with 1 wait state |

1*) Maximum values for $f_{CLKS1}$ and $f_{CLKB}$ must not exceed the limits defined in the Datasheet.

**Note:**

When changing any clock settings (change of CLKS1 clock source or CLKB divider setting), use a Flash timing which is valid for both, the old and the new clock setting. The settings 6E3Dh/6E3Fh can be used for any clock transition.

When using the modulated clock CLKMOD as clock source for CLKS1, make sure that the maximum and minimum values of the CLKMOD frequency are within the permitted range for the CLKS1 frequency.

This table is not valid for MB96F348T/HxA

## 33.4.3 Flash Memory Write Control registers 0-5 (FMWC0-FMWC5)

Flash Memory Write Control registers (FMWC0-FMWC5) control the erase/write permission of each sector of the flash memory.

### Flash Memory Write Control registers (FMWC0-FMWC5)

Figure 33-7. Configuration of the Flash Memory Write Control registers (FMWC0-FMWC5)

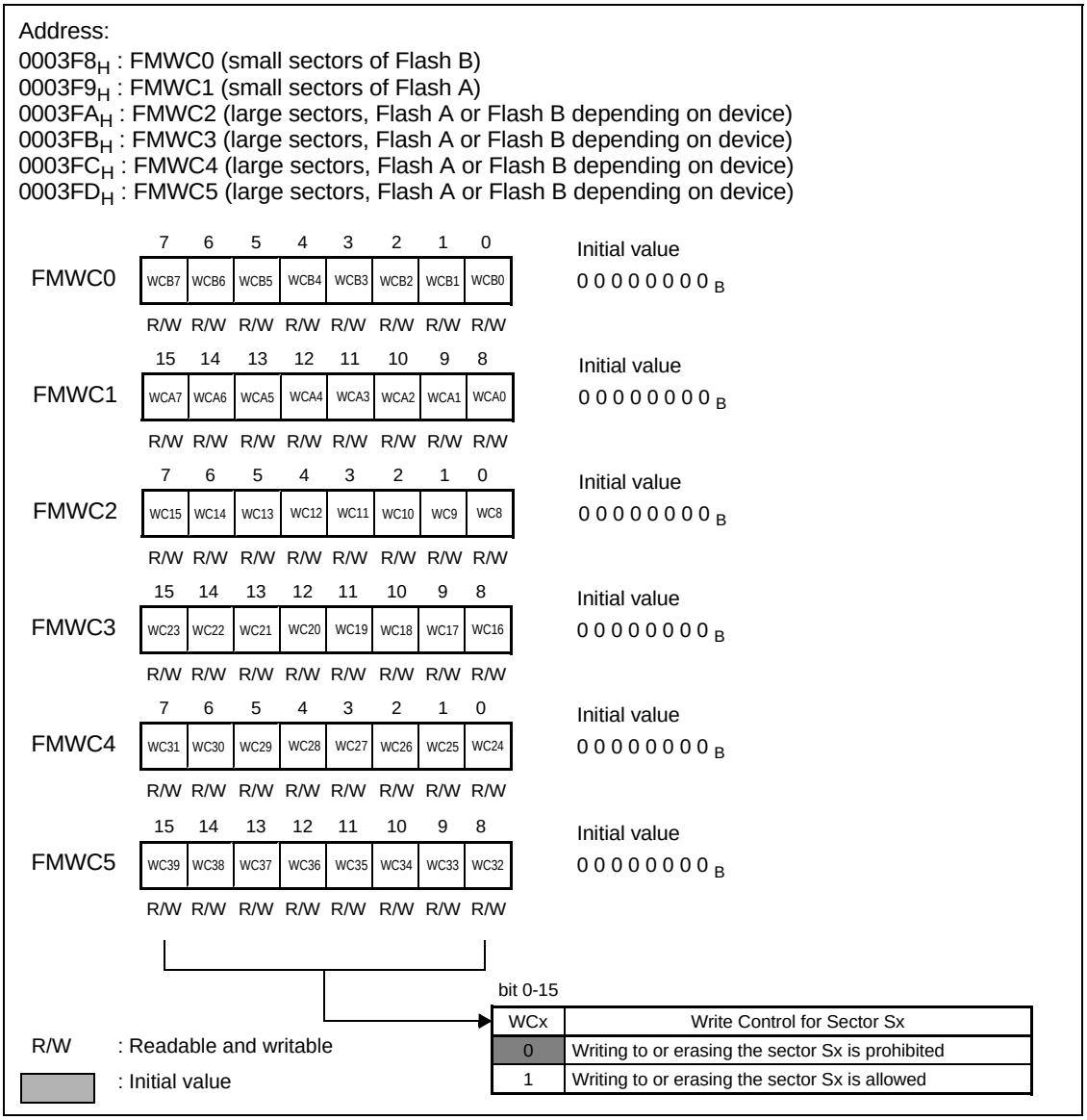



Table 33-5. Function of each Bit of the Flash Memory Write Control registers (FMWC0-7)

| Bit name | | Function |
|---|---|---|
| bit 0 - bit 15 | WCB0 to WCB7, WCA0 to WCA7 and WC8 to WC39:<br>Write Control bits of sector SB0-SB7, SA0-SA7 and S8-S39 | • These bits control if the corresponding sector can be written and erased or not.<br>• After reset, these bits are reset to '0' (no programming/erasing possible).<br>• Writing a bit to '1' enables programming/erasing of the corresponding sector. Setting bits to '1' however is possible only with the first write access to the register after any reset.<br>• Writing a bit back to '0' disables programming/erasing again. A bit can be written to '0' at any time.<br>• All bits of not available sectors must be written to '1'. The read value of these bits is undefined. |

## 33.5 Data Flash Memory Control Registers

This chapter describes the control/status registers of the Data Flash which are needed to control the flash memory erase/write process and to configure the read access timing.

### Memory Control Registers

The Data Flash A memory is controlled by the following registers:

- Data Flash Control Status register A (DFCSA)
- Data Flash Write command sequencer Control register A (DFWCA)
- Data Flash Write command sequencer Status register A (DFWSA)

The Data Flash B memory is controlled by the following registers:

- Data Flash Control Status register B (DFCSB)
- Data Flash Write command sequencer Control register B (DFWCB)
- Data Flash Write command sequencer Status register B (DFWSB)

### 33.5.1 Data Flash Control Status register (DFCSA, DFCSB)

The Data Flash Control Status register (DFCSA for Data Flash A memory, DFCSB for Data Flash B memory) controls all functions of the Data Flash memory interface circuit for reading and for program/erase (except the write command sequencer).

#### Data Flash Control Status register (DFCSA, DFCSB)

Figure 33-8. Configuration of the Data Flash Control Status register (DFCSA, DFCSB)

Address:
$0003E0_H$ (DFCSA for Data Flash A memory)
$0003E4_H$ (DFCSB for Data Flash B memory)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| - | TMG2 | TMG1 | TMG0 | - | INTE | RDYINT | RDY |
| - | R/W | R/W | R/W | - | R/W | R/W | R |

Initial value
$X\,1\,1\,1\,X\,0\,0\,X_B$

bit0

| RDY | Flash RDY status |
|---|---|
| 0 | Write/erase ongoing |
| 1 | Flash is ready (no write/erase ongoing) |

bit1

| RDYINT | RDY Interrupt flag |
|---|---|
| 0 | No write/erase command termination detected |
| 1 | Write/erase command has terminated |

bit2

| INTE | Interrupt Enable bit |
|---|---|
| 0 | Ready Interrupt function disabled |
| 1 | Ready Interrupt function active |

bit3

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

| bit6 | bit5 | bit4 | | | |
|---|---|---|---|---|---|
| TMG2 | TMG1 | TMG0 | Maximum CLKB frequency | CLKB wait cycles (Read) | CLKB wait cycles (Write) |
| 0 | 0 | 0 | 7.5 MHz | 2 | 2 |
| 0 | 0 | 1 | 16 MHz | 3 | 2 |
| 0 | 1 | 0 | 33 MHz | 4 | 2 |
| 0 | 1 | 1 | 50 MHz | 5 | 2 |
| 1 | 0 | 0 | 66 MHz | 7 | 3 |
| 1 | 0 | 1 | 83 MHz | 8 | 4 |
| 1 | 1 | 0 | 100 MHz | 9 | 5 |
| 1 | 1 | 1 | 100 MHz | 10 | 5 |

bit7

| - | Reserved |
|---|---|
| 0 | Always write "0" to this bit |

X : undefined value
R/W : Readable and writable
R : Read only
: Initial value

Table 33-6. Function of each bit of the Data Flash Control Status register (DFCSA, DFCSB)

<table>
<tr><th colspan="2">Bit names</th><th>Function</th></tr>
<tr><td>bit 0</td><td>RDY:<br>Flash RDY status</td><td>• This bit shows the status of the (sampled) Flash RDY output.<br>• Writing to this bit has no effect.<br>• A read value of '0' indicates that a program/erase command is currently executed. Only the reset and suspend commands are accepted in this state.<br>• A read value of '1' indicates that no program/erase command is currently executed or that the Flash is in the erase-suspend state. Write/erase commands can be written to the Flash.</td></tr>
<tr><td>bit 1</td><td>RDYINT:<br>RDY Interrupt flag</td><td>• This bit is the interrupt flag of the Flash ready interrupt function.<br>• This bit is initialized to '0' by reset.<br>• It is set to '1' by a rising edge of the RDY signal.<br>• The RDYINT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>• The read cycle of a read-modify-write access always returns '1'.</td></tr>
<tr><td>bit 2</td><td>INTE:<br>Interrupt Enable bit</td><td>• This bit enables the interrupt function of the RDYINT flag (write/erase command termination interrupt).<br>• An interrupt is generated when this bit is set to '1' and the RDYINT flag is '1' or gets set by a rising edge of the RDY signal.<br>• When this bit is set to '0', no RDYINT interrupt can be generated.<br>• The initial value of the INTE bit is '0'.<br>• This bit is read- and writable.</td></tr>
<tr><td>bit 3</td><td>Reserved</td><td>• Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected.</td></tr>
<tr><td>bit 4 - bit 6</td><td>TMG0 - TMG2: Timing configuration bits</td><td>• These bits control the read and write timing for the Data Flash (wait cycles).<br>• The initial value of these bit is '111' (maximum number of wait cycles).<br>• The required setting depends on the used CLKB frequency. Select a setting according to the following table:<br>bit6 bit5 bit4<br>
<table>
<tr><th>TMG2</th><th>TMG1</th><th>TMG0</th><th>Maximum CLKB frequency</th><th>CLKB wait cycles (Read)</th><th>CLKB wait cycles (Write)</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>7.5 MHz</td><td>2</td><td>2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>16 MHz</td><td>3</td><td>2</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>33 MHz</td><td>4</td><td>2</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>50 MHz</td><td>5</td><td>2</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>66 MHz</td><td>7</td><td>3</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>83 MHz</td><td>8</td><td>4</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>100 MHz</td><td>9</td><td>5</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>100 MHz</td><td>10</td><td>5</td></tr>
</table>
• These bits are read- and writable</td></tr>
<tr><td>bit 7</td><td>Reserved</td><td>• Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected.</td></tr>
</table>

## 33.5.2 Data Flash Write command sequencer Control register (DFWCA, DFWCB)

These registers (DFWCA for Data Flash A memory, DFWCB for Data Flash B memory) control all functions of the write command sequencer.

### Data Flash Write command sequencer Control register (DFWCA, DFWCB)

Figure 33-9. Configuration of the DFWCA and DFWCB registers)

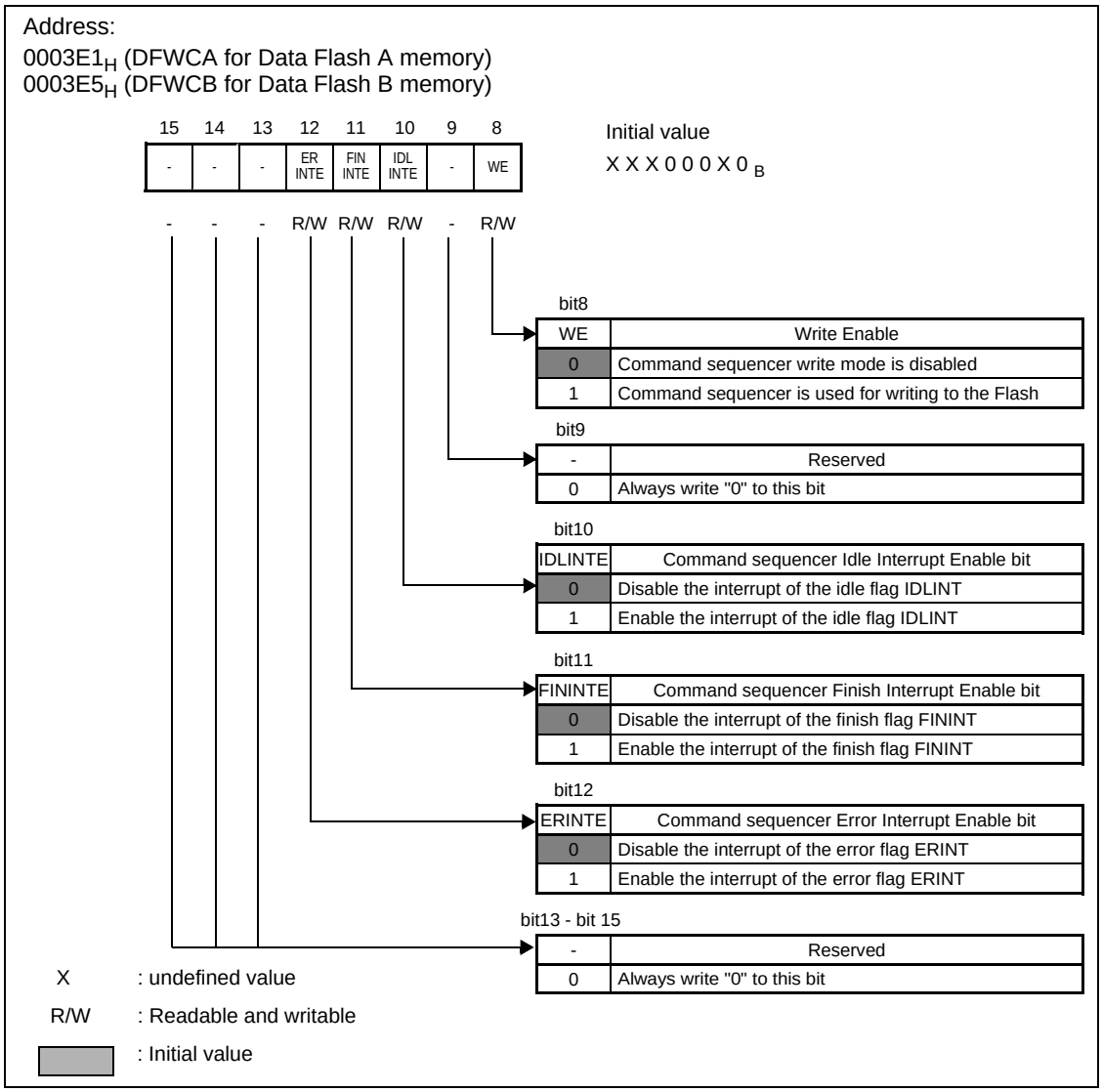


Table 33-7. Function of each bit of the Data Flash Write command sequencer Control register (DFWCA, DFWCB)

| Bit names | | Function |
|---|---|---|
| bit 8 | WE:<br>Write Enable bit | • This bit controls the write access mode.<br>• Setting this bit to ’1’ activates the Write command sequencer. Each CPU write access to the Data Flash memory area is considered as a write access to the target address. The "Write program" command (see Table 33-9) is automatically generated by the Flash interface.<br>• Setting this bit to ’0’ disables the Write command sequencer. The program sequence must be written to the Flash manually. This mode must also be used for sending any other command to the Flash (like chip/sector erase, sector erase suspend/resume). Disabling the command sequencer is only permitted in "Idle" state (DFWSA/DFWSB:ST[1:0]="00").<br>• Flash read operations are not affected by the setting of this bit.<br>• The initial value of this bit is ’0’ (direct write mode). |
| bit 9 | Reserved | • Always write "0" to this bit.<br>• The read value of this bit is undefined.<br>• Read modify write operations to this register are not affected. |
| bit 10 | IDLINTE:<br>Idle Interrupt Enable bit | • This bit enables the idle interrupt (IDLINT) of the Data Flash write command sequencer.<br>• An interrupt is generated when this bit is set to ’1’ and the IDLINT flag gets set (transition of the write command sequencer to idle state).<br>• When this bit is set to ’0’, no idle interrupt can be generated.<br>• The idle interrupt can only be generated when the write command sequencer is enabled (WE=’1’).<br>• The initial value of this bit is ’0’ (idle interrupt disabled). |
| bit 11 | FININTE:<br>Finish Interrupt Enable bit | • This bit enables the finish interrupt (FININT) of the Data Flash write command sequencer.<br>• An interrupt is generated when this bit is set to ’1’ and the FININT flag gets set (successful completion of a write operation).<br>• When this bit is set to ’0’, no finish interrupt can be generated.<br>• The finish interrupt can only be generated when the write command sequencer is enabled (WE=’1’).<br>• The initial value of this bit is ’0’ (finish interrupt disabled). |
| bit 12 | ERINTE:<br>Error Interrupt Enable bit | • This bit enables the error interrupts (TOERINT, WERINT and WIERINT) of the Data Flash write command sequencer.<br>• An interrupt is generated when this bit is set to ’1’ and one of the TOERINT, WERINT and WIERINT error flag gets set.<br>• When this bit is set to ’0’, no error interrupt can be generated.<br>• The error interrupt can only be generated when the write command sequencer is enabled (WE=’1’).<br>• The initial value of this bit is ’0’ (error interrupt disabled). |
| bit 13 - bit 15 | Reserved | • Always write "0" to these bits.<br>• The read value of these bits is undefined.<br>• Read modify write operations to this register are not affected. |

## 33.5.3 Data Flash Write command sequencer Status register (DFWSA, DFWSB)

These registers (DFWSA for Data Flash A memory, DFWSB for Data Flash B memory) contain all status bits of the write command sequencer.

### Data Flash Write command sequencer Status register (DFWSA, DFWSB)

Figure 33-10. Configuration of the DFWSA and DFWSB registers

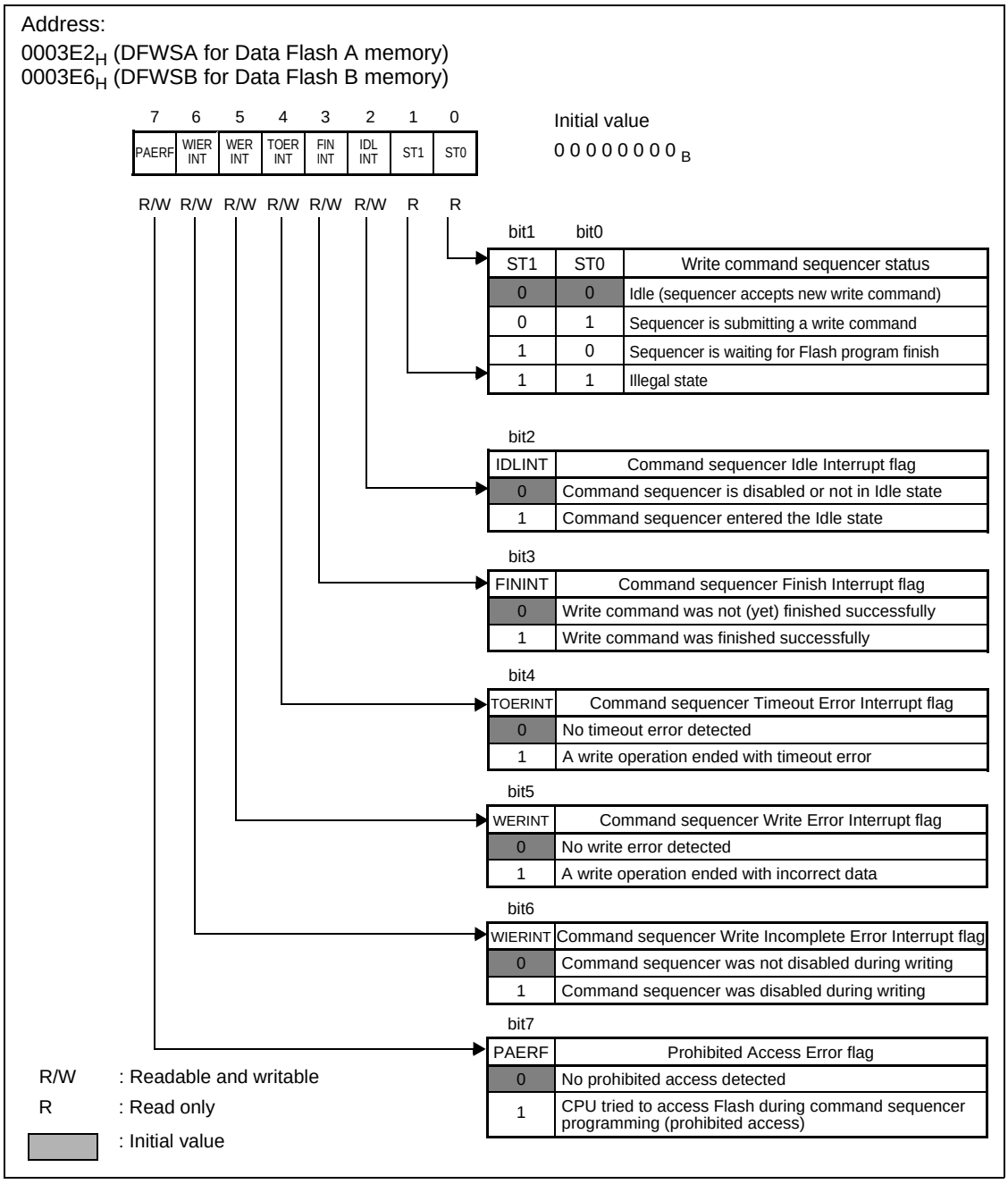


Table 33-8. Function of each bit of the Data Flash Write command sequencer Status register (DFWSA, DFWSB)

<table>
<tr><th colspan="2">Bit names</th><th>Function</th></tr>
<tr><td>bit 0 - bit 1</td><td>ST0 to ST1:<br>Status bits</td><td>
• These bits show the status of the Write command sequencer:
<table>
<tr><th>bit1</th><th>bit0</th><th></th></tr>
<tr><th>ST1</th><th>ST0</th><th>Write command sequencer status</th></tr>
<tr><td>0</td><td>0</td><td>Idle (sequencer accepts new write command)</td></tr>
<tr><td>0</td><td>1</td><td>Sequencer is submitting a write command</td></tr>
<tr><td>1</td><td>0</td><td>Sequencer is waiting for Flash program finish</td></tr>
<tr><td>1</td><td>1</td><td>Illegal state</td></tr>
</table>
• ST[1:0] = "00" indicates that the sequencer is either disabled (WE='0') or idle (DFWCx.WE='1'). In the Idle state, the sequencer is ready to receive a new write command.<br>
• ST[1:0] = "01" indicates that the sequencer is submitting the write command to the Flash module.<br>
• ST[1:0] = "10" indicates that the Write command sequencer is checking the progress of the write command by checking the Flash RDY status and the Flash hardware sequence flags.<br>
• ST[1:0] = "11" indicates an Illegal state: The Write command sequencer was disabled (DFWCx.WE='0') during Flash writing (in state "01" or "10").<br>A reset must be asserted in this case (for example a Software reset).<br>
• These bits are only readable.<br>
• The initial value of these bit is "00" (Write command sequencer disabled).
</td></tr>
<tr><td>bit 2</td><td>IDLINT:<br>Idle Interrupt flag</td><td>
• This bit is the idle interrupt flag of the Write command sequencer and can be used to initiate a data transfer to the Data Flash by DMA.<br>
• This bit is initialized to '0' by a reset.<br>
• It is set to '1' when the command sequencer is enabled or going to the idle state (after a write operation was finished).<br>
• The IDLINT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>
• This bit can also be cleared by DMA.<br>
• The read cycle of a read-modify-write access always returns '1'.<br>
• An interrupt is generated when this bit is set to '1' and the corresponding interrupt enable bit DFWCx.IDLINTE is '1'.
</td></tr>
<tr><td>bit 3</td><td>FININT:<br>Finish Interrupt flag</td><td>
• This bit is the finish interrupt flag of the Write command sequencer and can be used for flash writing by CPU.<br>
• This bit is initialized to '0' by a reset.<br>
• It is set to '1' when the command sequencer successfully finishes a write command.<br>
• The FININT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>
• This bit can also be cleared by DMA.<br>
• The read cycle of a read-modify-write access always returns '1'.<br>
• An interrupt is generated when this bit is set to '1' and the corresponding interrupt enable bit DFWCx.FININTE is '1'.
</td></tr>
<tr><td>bit 4</td><td>TOERINT:<br>Timeout Interrupt flag</td><td>
• This bit is the timeout error interrupt flag of the Write command sequencer.<br>
• This bit is initialized to '0' by a reset.<br>
• It is set to '1' when the command sequencer ends in the timeout state (DQ5 flag='1').<br>
• The TOERINT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>
• The read cycle of a read-modify-write access always returns '1'.<br>
• An interrupt is generated when this bit is set to '1' and the error interrupt enable bit DFWCx.ERINTE is '1'. A DMA Stop request is also submitted in this case.
</td></tr>
<tr><td>bit 5</td><td>WERINT:<br>Write Error Interrupt flag</td><td>
• This bit is the write error interrupt flag of the Write command sequencer.<br>
• This bit is initialized to '0' by a reset.<br>
• It is set to '1' when the write command terminates without having the correct data written (for example after trying to write to an erase suspended sector).<br>
• The WERINT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>
• The read cycle of a read-modify-write access always returns '1'.<br>
• An interrupt is generated when this bit is set to '1' and the error interrupt enable bit DFWCx.ERINTE is '1'. A DMA Stop request is also submitted in this case.
</td></tr>
</table>

Table 33-8. Function of each bit of the Data Flash Write command sequencer Status register (DFWSA, DFWSB)

| Bit names | | Function |
|---|---|---|
| bit 6 | WIERINT:<br>Write Incomplete Interrupt flag | • This bit is the write incomplete error interrupt flag of the Write command sequencer.<br>• This bit is initialized to '0' by a reset.<br>• It is set to '1' when the command sequencer is disabled (by writing WE='0') while the command sequencer was not in the Idle state. The status of the interrupted write sequence must be checked manually in this case. If the command sequencer stays in the "Illegal state", then a reset must be asserted.<br>• The WIERINT bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>• The read cycle of a read-modify-write access always returns '1'.<br>• An interrupt is generated when this bit is set to '1' and the error interrupt enable bit DFWCx.ERINTE is '1'. A DMA Stop request is also submitted in this case. |
| bit 7 | PAERF:<br>Prohibited Access flag | • This bit is an error flag showing a prohibited access.<br>• This bit is initialized to '0' by a reset.<br>• It is set to '1' when the CPU tries to read or write the Flash memory while the Write command sequencer is not in the idle state. Such a prohibited write access is ignored while a read access returns 00h.<br>• The PAERF bit can be cleared by writing to '0'. Writing to '1' has no effect.<br>• The read cycle of a read-modify-write access always returns '1'.<br>• This bit cannot generate an interrupt. |

## 33.6 Starting the Flash Memory Automatic Algorithm

Write and erase to flash memory is performed by launching the flash memory's own Automatic Algorithms. The following commands are available: Read/Reset, Write, Chip Erase and Sector Erase. The erase suspend and restart function is available during Sector erase.

### Command sequence table

Automatic Algorithms for flash memory write/erase are launched by writing one to six bytes or words to the Flash memory in succession according to Table 33-9. This table is valid for the Program/Data Flash (Main Flash) and for the Data Flash.

The data of the commands must be written to the low-order byte (except the program data of the write command). Hence the commands must be written to an even address.

For the Program/Data Flash (Main Flash), this is possible in byte or word mode. The data written to the high-order byte (in case of a word access) is ignored. The data width used for the 4th bus write cycle of the write command (program address and program data) determines the write mode of the Flash (byte or word).

For the Data Flash, the commands must be written in byte mode. Only programming of bytes is possible when using the write command.

The Data Flash interface includes a Write command sequencer which can automatically send the write sequence to the Flash. This Write command sequencer also supports word programming.

Disable the Write command sequencer (DFWCx:WE='0') for submitting any of the commands below.

Table 33-9. Command sequence table

| Command sequence | Bus write access | 1st bus write cycle | | 2nd bus write cycle | | 3rd bus write cycle | | 4th bus write cycle | | 5th bus write cycle | | 6th bus write cycle | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Address | Data | Address | Data | Address | Data | Address | Data | Address | Data | Address | Data |
| Read/Reset (*1) | 1 | mmmXXX (even) | F0 | - | - | - | - | - | - | - | - | - | - |
| Read/Reset (*1) | 3 | mmmAAA | AA | mmm554 | 55 | mmmAAA | F0 | - | - | - | - | - | - |
| Write program | 4 | mmmAAA | AA | mmm554 | 55 | mmmAAA | A0 | PA | PD | - | - | - | - |
| Chip Erase | 6 | mmmAAA | AA | mmm554 | 55 | mmmAAA | 80 | mmmAAA | AA | mmm554 | 55 | mmmAAA | 10 |
| Sector Erase | 6 | mmmAAA | AA | mmm554 | 55 | mmmAAA | 80 | mmmAAA | AA | mmm554 | 55 | sssXXX (even) | 30 |
| Sector Erase Suspend | 1 | sssXXX (even) | B0 | | | | | | | | | | |

Table 33-9. Command sequence table<Italic> (continued)

| Command sequence | Bus write access | 1st bus write cycle | | 2nd bus write cycle | | 3rd bus write cycle | | 4th bus write cycle | | 5th bus write cycle | | 6th bus write cycle | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Address | Data | Address | Data | Address | Data | Address | Data | Address | Data | Address | Data |
| Sector Erase Restart | 1 | sssXXX (even) | 30 | | | | | | | | | | |
| Set to fast mode | 3 | mmmAAA | AA | mmm554 | 55 | mmmAAA | 20h | | | | | | |
| Fast write (*2) | 2 | mmmXXX (even) | A0 | PA | PD | | | | | | | | |
| Reset from fast mode | 2 | mmmXXX (even) | 90 | mmmXXX (even) | F0/00 | | | | | | | | |

**Note:**

- The addresses in the table are the values in the CPU memory map. All addresses and data are represented using hexa-decimal notation. The letter X indicates an arbitrary value.
- CPU/Flash address bit 1 is ignored for command writing (except for the program address PA). Hence a command can be written for example to address AAA or AA8.
- The addresses mmm in the table must point to the flash memory **module** (Flash A/B or Data Flash A/B) on which the command is meant to operate.
- The addresses sss in the table must point to the flash memory **sector** on which the sector command is meant to operate. See section 33.2 Block Diagram of the Flash Memory and Sector Configuration of the Flash Memory. Care must be taken when sending the sector erase command to the sector 0 of the Data Flash (SDA0 or SDB0). The first 5 commands must be written to another sector of the same flash module to make sure "mmmAAA" and "mmm554" points to the correct flash module.
- PA: Write address. Only even addresses can be specified when writing in word mode. For byte mode, also odd addresses are permitted.
- PD: Write data. Only byte data can be specified for the Data Flash.
- Writing an illegal address or data, or writing them in the incorrect order, will reset the Flash memory to the read mode.
- It is possible to read data from the Flash between the write cycles of the commands. The command execution starts after the last write cycle.
- Writing an improper sequence (incorrect address or incorrect data or writing in an improper sequence) will reset the command detector in the Flash.
- Writing a command to the Program/Data Flash (Main Flash) is possible only when the corresponding write enable signal (MCSRA:WE, MCSRB:WE) is set to 1.
- Commands must not be written to sectors which are write protected. Any write access to a write protected sector activates the hardware reset of the corresponding flash module and forces the corresponding RDY flag to 0 for 64 CLKB clock cycles. Such a hardware reset initializes the Flash to the Read/Reset state, independent of the previous state (program, erase, suspend).

*1: Both of the two types of Read/Reset commands can reset the flash memory to read mode from the timeout state. The two commands are offered only for software compatibility. This is independent of the fast mode activation.

*2: This command is only valid in Fast Mode.

## 33.7 Confirming the Automatic Algorithm Execution State

The Flash memory performs the write/erase sequence via automatic algorithms. It thus has hardware for informing the outside world when it has finished internal operations.

### Hardware sequence flags

A read access to the target flash sector returns the status of the hardware sequence flags instead of the flash memory data as long as the automatic algorithm is ongoing. They can be used to check the current status of the write/erase operation.

The hardware sequence flags consist of the bits DQ7, DQ6, DQ5, (DQ4,) DQ3 and DQ2. The functions of these bits are:

- Data polling flag (DQ7)
- Toggle bit flag (DQ6)
- Timing limit exceeded flag (DQ5)
- Sector erase state flag (DQ4): Only available for Data Flash
- Sector erase timer flag (DQ3)
- Toggle bit-2 flag (DQ2)

Table 33-10. Bit assignments of hardware sequence flags

| Bit No. | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Hardware sequence flag | DQ7 | DQ6 | DQ5 | (DQ4) | DQ3 | DQ2 | - | - |

### Note:

- The values of the other bits (DQ[15:8], (DQ4), DQ[1:0]) are not defined. Software must be written in a way to tolerate any value of these bits.
- A read access at the beginning or the end of the automatic algorithm can return invalid data. Hence the hardware sequence flags must be read at least twice (or 3 times for checking the toggle bits) before making a branch decision.
- Reading the hardware sequence flags of the Program/Data Flash can be done independently of the setting of the corresponding write enable signal (MCSRA:WE, MCSRB:WE).

### Note:

For reading the hardware sequence flags of the Program/Data Flash (Main Flash), the following rules must be adhered:

- The hardware sequence flags must be read by accessing the low-order byte (even address), even after starting a byte write access to an upper-order byte (odd address).
- The Data read buffer must be disabled (set MCSRx:DRBE=0).
- Make sure at least one other instruction is executed between two read accesses to the hardware sequence flags (at least one NOP). Otherwise the DQ6 and DQ2 toggle bits cannot be read correctly.
- The MTCRA/MTCRB:ADS bit must be set to ’1’ when reading the hardware sequence flags (ADS is set to ’1’ in all recommended Flash timing settings for writing). Otherwise the DQ2 toggle bit cannot be read correctly and reading flash data in Sector erase suspend state will not work correctly.
- Do not access flash memory control registers (address $0003F0_H$ to $003FF_H$) between reading the hardware sequence flags. Otherwise the DQ2 toggle bit cannot be read correctly.

These rules are not applicable for the Data Flash.

For checking whether automatic writing or erasing is being executed, the hardware sequence flags or the RDY bit of the memory control status register (MCSRA/MCSRB or DFCSA/DFCSB) can be read. The hardware sequence flags however show more detailed information.

After writing/erasing has terminated successfully, the Flash returns to the read/reset state. This must be confirmed by one of these flags (RDY or hardware sequence flags) before performing the next operation such as data read.

In addition, the hardware sequence flags can be used to confirm whether the second or subsequent sector erase code write is valid.

The following table shows the function of all flags in each Flash status.

Table 33-11. Hardware sequence flag functions

| Status | | Accessed address | DQ7 | DQ6 | DQ5 | (DQ4) | DQ3 | DQ2 |
|---|---|---|---|---|---|---|---|---|
| Normal operation | Read/Reset | Any address | DATA:7 | DATA:6 | DATA:5 | DATA:4 | DATA:3 | DATA:2 |
| | Write (embedded program algorithm running) | Address being programmed | $\overline{\text{DATA:7}}$ | Toggle | 0 | 0 | 0 | 1 |
| | Sector erase wait (Flash waits for inputting further sector erase requests) | Sectors which are already specified for erase | 1 | Toggle | 0 | 0 | 0 | Toggle |
| | | Other sectors | | | | | | 1 |
| | Chip/sector erase (embedded erase algorithm running) | Sectors which are specified for erase (all sectors in case of Chip erase) | 0 | Toggle | 0 | 1: busy to suspend<br>0: ready to suspend | 1 | Toggle |
| | | Other sectors | | | | | | 1 |
| | Sector erase suspended | Sectors which are specified for erase | 1 | 1 | 0 | 0 | 0 | Toggle |
| | | Other sectors | DATA:7 | DATA:6 | DATA:5 | DATA:4 | DATA:3 | DATA:2 |
| | Writing of another sector while Flash is in erase suspended state | Address being programmed | $\overline{\text{DATA:7}}$ | Toggle | 0 | 0 | 0 | 1 |
| | Chip/sector erase finished (Read/Reset state) | Sector which was erased | DATA:7 = 1 | DATA:6 = 1 | DATA:5 = 1 | DATA:4 = 1 | DATA:3 = 1 | DATA:2 = 1 |
| Timing limit exceeded | Write | Address being programmed | $\overline{\text{DATA:7}}$ | Toggle | 1 | 0 | 0 | 1 |
| | Chip/sector erase | Sector which caused the Timeout during erase | 0 | Toggle | 1 | undefined | 1 | Toggle |
| | | Other sectors | | | | | | 1 |

The function of each hardware sequence flag is described in the following sections.

The DQ4 flag is available only for the Data Flash. The usage of this bit is described in chapter 33.8.6 Suspending Sector Erase.

## 33.7.1 Data Polling Flag (DQ7)

The data polling flag (DQ7) indicates if the automatic algorithm (write or erase) is being executed or has terminated.

### *33.7.1.1 Data polling flag (DQ7)*

The function of the DQ7 flag in each Flash status is described in Table 33-11.

**Write**

A read access during execution of the automatic write algorithm causes the flash memory to output the inverted value of bit 7 of the program data ($\overline{\text{DATA:7}}$) regardless which Flash address is being read.

A read access after termination of the automatic write algorithm is handled as a regular Flash read access and returns bit 7 (DATA:7) of the currently addressed Flash cell. Hence for correctly identifying the termination of a write command, polling should always be performed from the memory location which is being programmed.

#### Word and Byte writing

A word write command writes data to the high-order byte (DATA[15:8]) and low-order byte (DATA[7:0]). DQ7 shows the inverted value of DATA[7].

A byte write command to an even address writes data to the low-order byte at DATA[7:0] only. DQ7 shows the inverted value of DATA[7].

A byte write command to an odd address writes data to the high-order byte at DATA[15:8] only. In this case DQ7 shows the inverted value of DATA[15].

#### Chip/sector erase

A read access during execution of the automatic erase algorithm causes the flash memory to output a 0 regardless which Flash address is being read. During the sector erase wait time however, 1 is output.

A read access after termination of the automatic erase algorithm is handled as a regular Flash read access and returns the data of the currently addressed Flash cell. Accessing an erased cell returns 1.

#### Sector erase suspend

A read access in sector erase suspend state causes the flash memory to output DQ7 = 1 if the address belongs to the sector being erased.

The flash memory outputs bit 7 (DATA: 7) of the addressed memory cell if the address does not belong to the sector being erased.

Referencing this flag together with the toggle bit flag (DQ6) enables a decision to be made on whether the flash memory is in the erase suspended state and which sector is being erased.

#### Note:

When the automatic algorithm starts or stops, the Flash changes asynchronously between the automatic algorithm execution and the read/reset or sector erase suspend state. A possible read access to the Flash at this time can return invalid data. Hence the status of the DQ7 bit can only be identified safely if the corresponding data has been read at least twice.

#### Note:

For the Program/Data Flash (Main Flash), the hardware sequence flags must always be read from an even address, even when writing a byte to an odd address. This means the hardware sequence flags cannot be read from the same address which was written. During programming, DQ7 shows DATA:15. But after termination of the automatic algorithm, the currently addressed Flash cell is read. Depending on the content of this Flash cell, there may be no transition visible on bit 7. Hence the DQ7 bit cannot be used to identify the termination of a byte write to an odd address.

### 33.7.2 Toggle Bit Flag (DQ6)

The toggle bit flag (DQ6) together with the data polling flag (DQ7) indicates if the automatic algorithm (write or erase) is being executed or has terminated.

#### *33.7.2.1 Toggle bit flag (DQ6)*

The function of the DQ6 flag in each Flash status is described in Table 33-11.

#### Write and chip/sector erase

Successive read accesses during execution of the automatic write or erase algorithm causes the flash memory to toggle the DQ6 flag for every read cycle, regardless which Flash address is being read.

A read access after termination of the automatic algorithm is handled as a regular Flash read access and returns bit 6 (DATA:6) of the currently addressed Flash cell. Accessing an erased cell returns 1.

### Sector erase suspend

A read access in sector erase suspend state causes the flash memory to output DQ6 = 1 if the address belongs to the sector being erased.

The flash memory outputs bit 6 (DATA: 6) of the addressed memory cell if the address does not belong to the sector being erased.

### Note:

When the automatic algorithm starts or stops, the Flash changes asynchronously between the automatic algorithm execution and the read/reset or sector erase suspend state. A possible read access to the Flash at this time can return invalid data. Hence the status of the toggle bit can only be identified safely if the corresponding data has been read at least three times.

### Note:

For correctly reading the DQ6 flag of the Program/Data Flash (Main Flash), the following rule must be adhered:

- Make sure at least one other instruction is executed between two read accesses to the hardware sequence flags (at least one NOP).

This rule is not applicable for the Data Flash.

## 33.7.3 Timing Limit Exceeded Flag (DQ5)

The timing limit exceeded flag (DQ5) indicates that the execution of the automatic algorithm has exceeded the timing (internal pulse count) prescribed in the flash memory.

### *33.7.3.1 Timing limit exceeded flag (DQ5)*

The function of the DQ5 flag in each Flash status is described in Table 33-11.

### Write and chip/sector erase

A read access during execution of the automatic write or erase algorithm causes the flash memory to output the status of the DQ5 flag, regardless which Flash address is being read.

A read access after successful termination of the automatic write or erase algorithm is handled as a regular Flash read access and returns bit 5 (DATA:5) of the currently addressed Flash cell.

The DQ5 flag is read as 0 as long as the program or erase time is within the prescribed time (maximum time required for write/erase). After this time has been exceeded, 1 is returned as read value.

An unsuccessful write or erase operation can be determined if DQ5 is 1 while DQ6 and DQ7 shows that the automatic algorithm is still being executed.

For example, writing 1 to a flash memory address where 0 has been written will cause the fail state to occur. In this case, the flash memory will lock and execution of the automatic algorithm will not terminate. As a result, valid data will not be output from the data polling flag (DQ7). In addition, the toggle bit flag (DQ6) will exceed the time limit without stopping the toggle operation and the timing limit exceeded flag (DQ5) will output 1. Note that this state does not indicate that the flash memory is faulty, but has been used incorrectly. When this state occurs, execute the Read/Reset command.

### Sector erase suspend

A read access in sector erase suspend state causes the flash memory to output DQ5 = 0 if the address belongs to the sector being erased.

The flash memory outputs bit 5 (DATA: 5) of the addressed memory cell if the address does not belong to the sector being erased.

**Note:**

When the automatic algorithm starts or stops, the Flash changes asynchronously between the automatic algorithm execution and the read/reset or sector erase suspend state. A possible read access to the Flash at this time can return invalid data. Hence the status of the DQ5 bit can only be identified safely if the corresponding data has been read at least twice.

## 33.7.4 Sector Erase Timer Flag (DQ3)

The sector erase timer flag (DQ3) indicates if the execution of the sector erase command has been started of if the sector erase wait period is being applied which allows to submit further sector erase commands.

### *33.7.4.1 Sector erase timer Flag (DQ3)*

After submitting a sector erase command sequence, the sector erase wait period is applied. Within this period it is possible to submit further sector erase commands. The DQ3 flag can be used to identify this wait period.

The function of the DQ3 flag in each Flash status is described in Table 33-11.

**Sector erase**

A read access during execution of the automatic erase algorithm causes the flash memory to output the status of the DQ3 flag, regardless which Flash address is being read. A read access during the sector erase wait period returns 0. After exceeding this wait period, the actual sector erase starts and a read access to the DQ3 flag returns 1. Further sector erase commands will be ignored when DQ3 is 1.

A read access after successful termination of the automatic erase algorithm is handled as a regular Flash read access and returns bit 3 (DATA:3) of the currently addressed Flash cell. Accessing an erased cell returns 1.

See 33.8.5 Erasing Optional Data (Sector erase) for more details about submitting multiple sector erase commands.

The DQ3 flag should be checked before and after submitting further sector erase commands. If DQ3 is read as 1 after submitting the erase command, then this additional sector erase request may not be accepted.

**Sector erase suspend**

A read access in sector erase suspend state causes the flash memory to output DQ3 = 1 if the address belongs to the sector being erased.

The flash memory outputs bit 3 (DATA: 3) of the addressed memory cell if the address does not belong to the sector being erased.

**Note:**

When the automatic algorithm starts or stops, the Flash changes asynchronously between the automatic algorithm execution and the read/reset or sector erase suspend state. A possible read access to the Flash at this time can return invalid data. Hence the status of the DQ3 bit can only be identified safely if the corresponding data has been read at least twice.

## 33.7.5 Toggle Bit-2 Flag (DQ2)

The toggle bit-2 flag (DQ2) indicates if a sector is in the erase-suspended state. It can also be used to check which sectors are specified for erase.

### *33.7.5.1 Toggle bit-2 flag (DQ2)*

The DQ2 toggle bit can be used together with the DQ6 toggle bit to determine whether the Flash is in the sector erase suspend state or if the erase algorithm is currently being executed.

The function of the DQ2 flag in each Flash status is described in Table 33-11.

**Sector erase**

Successive read accesses during execution of the automatic sector erase algorithm causes the flash memory to toggle the DQ2 flag for every read cycle if the read address points to a sector which is specified for erase. A read access to another sec-

tor returns 1. This can be used to determine if the currently accessed sector belongs to the sectors which are specified for erase. For a chip erase, DQ2 toggles for all sectors.

A read access after termination of the automatic algorithm is handled as a regular Flash read access and returns bit 2 (DATA:2) of the currently addressed Flash cell. Accessing an erased cell returns 1.

### Sector erase suspend

Successive read accesses during sector erase suspend also causes the flash memory to toggle the DQ2 flag for every read cycle if the read address points to a sector which is specified for erase. A read access to another sector returns bit 2 (DATA:2) of the currently addressed Flash cell.

Both DQ2 and DQ6 are used for detecting an erase-suspended sector (DQ2 toggles, but DQ6 does not).

Sectors which are not specified for erase can be programmed in erase suspend state. Reading from such a sector during erase-suspend-program mode returns DQ2=1.

### Timing limit exceeded

In case of an unsuccessful chip or sector erase operation (Timing limit exceeded, DQ5=1), DQ2 can be used to identify which sector caused the timeout state. DQ2 will only toggle when accessing the sector which caused the timeout. For other sectors, 1 is read.

**Note:**

When the automatic algorithm starts or stops, the Flash changes asynchronously between the automatic algorithm execution and the read/reset or sector erase suspend state. A possible read access to the Flash at this time can return invalid data. Hence the status of the toggle bit can only be identified safely if the corresponding data has been read at least three times from the same location.

**Note:**

For correctly reading the DQ2 flag of the Program/Data Flash (Main Flash), the following rules must be adhered:

- Make sure at least one other instruction is executed between two read accesses to the hardware sequence flags (at least one NOP).
- The MTCRA/MTCRB:ADS bit must be set to ’1’ when reading the hardware sequence flags (ADS is set to ’1’ in all recommended Flash timing settings for writing).
- Do not access flash memory control registers (address $0003F0_H$ to $003FF_H$) between reading the hardware sequence flags.

These rules are not applicable for the Data Flash.

## 33.8 Detailed Explanation of Writing to and Erasing Flash Memory

This section describes each operation procedure of the flash memory: Read/Reset, Write, Chip Erase, Sector Erase, Sector Erase Suspend and Sector Erase Restart.

### Detailed explanation of flash memory write/erase

By issuing a command sequence (see Table 33-9 in Section 33.6 Starting the Flash Memory Automatic Algorithm) the flash memory executes the automatic algorithm to perform Read/Reset, Write, Chip Erase, Sector Erase, Sector Erase Suspend or Sector Erase Restart operations.

In between write accesses of the command sequence, other bus read/write cycles can be performed as long as no write access to the flash memory other than defined by the command sequence is performed. Even reading the Flash between the write accesses is possible.

Termination of the automatic algorithm can be determined by polling the hardware sequence flags, polling the RDY flag (MCSRA:RDY/MCSRB:RDY for Program/Data Flash, DFCSA:RDY/DFCSB:RDY for Data Flash) or by interrupt. At normal termination, the flash memory returns to the read/reset state.

Each operation of the flash memory is described in the following chapters:

- Setting the read/reset state
- Writing data by submitting the write command sequence
- Write to the Data Flash by using the Write Command Sequencer
- Erasing all data (chip erase)
- Erasing optional data (sector erase)
- Suspending sector erase
- Restarting sector erase

## 33.8.1 Setting The Read/Reset State

This section describes the procedure for issuing the Read/Reset command to set the flash memory to the read/reset state.

### Setting the flash memory to the read/reset state

The read/reset state is the initial state of the flash memory. When the power is turned on and when a command terminates normally, the flash memory is set to the read/reset state. In the read/reset state, any command can be input.

In the Timeout state (DQ5=1), the flash memory can be set to the read/reset state by sending the Read/Reset command in the command sequence table (see Table 33-9 in Section 33.6 Starting the Flash Memory Automatic Algorithm) continuously to the target sector in the flash memory.

The Read/Reset command has two types of command sequences that execute after the first and after the third bus operation. However, there is no essential difference between those two command sequences. The two types of the command are offered for software compatibility.

The Read/Reset command is not required to read data by a regular read. The Read/Reset command is mainly used to initialize the automatic algorithm when a command does not terminate normally.

The Read/Reset command has no effect in normal operation of the write or erase automatic algorithm. It cannot be used to interrupt an ongoing write or erase command execution. Also resetting the Flash from the sector erase suspended state is not possible. The Read/Reset command has no impact on the activation of the fast program mode.

## 33.8.2 Writing Data by submitting the write command sequence

This section describes how to write data to the flash memory by sending the write command sequence.

### *33.8.2.1 Starting the write automatic algorithm*

The data write automatic algorithm of the flash memory can be started by sending the Write command in the command sequence table (see Table 33-9 in Section 33.6 Starting the Flash Memory Automatic Algorithm) continuously to the target sector in the flash memory. When data write to the target address is completed in the fourth command cycle, the automatic algorithm for writing is started.

### Specifying addresses

Writing to the Program/Data Flash (Main Flash) is possible in byte or word mode. When writing in word mode, an even address must be specified in the write data cycle.

Writing to the Data Flash in direct write mode (DFWCA:WE='0' / DFWCB:WE='0') is possible in byte mode only.

Writing can be done in any order of addresses or even if the sector boundary is exceeded. However, the Write command writes only data of one byte or word for each execution.

### Notes on writing data

Writing cannot return a data bit in the Flash from 0 to 1. When trying to program a bit to 1 which is already set to 0, the data polling algorithm (DQ7) or toggle operation (DQ6) does not terminate and the timing limit exceeded flag (DQ5) be set after the prescribed maximum time for writing. A bit programmed to 0 can only be set to 1 by an erase operation.

All commands are ignored during execution of the automatic write algorithm. If a CPU reset (Power reset, external reset, software reset, watchdog reset or clock stop reset) is asserted during writing, the data of the written addresses will be unpredict-

able. A write access to a write protected sector also asserts the flash hardware resets and cancels an ongoing write operation.

#### 33.8.2.2 Fast write mode

The Fast write mode can be used for programming multiple data to the Flash. After writing the "Set to fast mode" command to the Flash, it is possible to program data by sending the two-cycle Fast write command instead of the standard four-cycle Write command.

Only submitting the "Reset from fast mode" command changes the flash back to the normal mode. Other commands have no influence on the fast mode setting, including Read/Reset and erase commands.

Asserting a flash hardware reset by asserting a CPU reset (Power reset, external reset, software reset, watchdog reset or clock stop reset) or by writing to a protected sector also resets the flash from the fast mode.

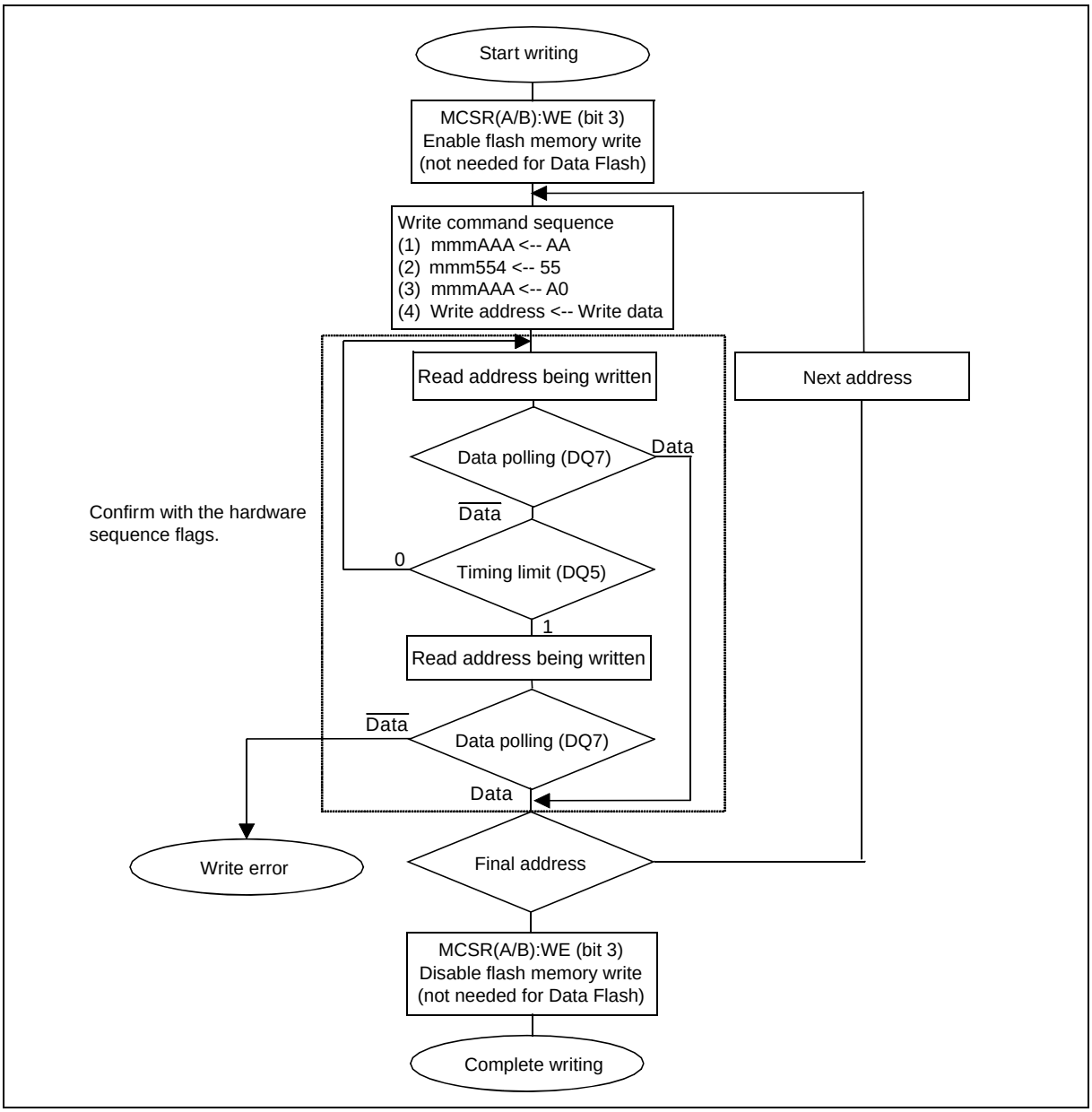
#### 33.8.2.3 Example for writing to the flash memory

Figure 33-11 shows an example of the procedure for writing to the flash memory. The hardware sequence flags (see Section 33.7 Confirming the Automatic Algorithm Execution State) can be used to determine the state of the automatic algorithm in the flash memory. Here, the data polling flag (DQ7) is used to confirm that writing has terminated.

Data polling must be performed to the memory location which is being programmed. Otherwise the transition of the DQ7 flag at the termination of the write command cannot be detected.

When the write automatic algorithm starts or stops, the Flash changes asynchronously between the automatic algorithm execution and the read/reset or sector erase suspend state. A possible read access to the Flash at this time can return invalid data. Hence the status of the hardware sequence flags must be rechecked to confirm the correct state.

Figure 33-11. Example of the flash memory write procedure




**Note:**

The address "mmm" must point to the flash memory module on which the command is meant to operate.

## 33.8.3 Writing to the Data Flash by using the Write command sequencer

This section describes how to write to the Data Flash by using the Write command sequencer instead of manually submitting the write command sequence.

### *33.8.3.1 Using the Write command sequencer*

The Data Flash can be written by two methods. Either in direct mode by manually submitting the write sequence as described in the previous section (DFWCA:WE='0' / DFWCB:WE='0') or by using the Write command sequencer (DFWCA:WE='1' / DFWCB:WE='1').

After activating the Write command sequencer, each CPU write access to the Data Flash is handled as a write request to the flash memory. The Flash interface prefixes each write data with the Write command sequence. It is not possible to submit any other command sequence to the Flash as long as DFWCA/B:WE is set to 1.

#### Activating and deactivating the Write command sequencer

After any reset, the Write command sequencer is always disabled. It can be activated by setting DFWCA/B:WE to 1. This should be done only when the Data Flash is in the Read/Reset or Sector erase suspend state and no command sequence has been written to the flash (not even the first write cycle of a new command sequence).

The Write command sequencer must not be disabled when it is not in Idle state (DFWSA/B:ST[1:0]="00").

#### Specifying addresses

Writing to the Data Flash in Write command sequencer mode is possible in byte or word mode. When writing in word mode, an even address must be specified. The Flash interface programs the word data to the Flash by automatically submitting two byte write commands.

#### Writing and reading the Data Flash with activated Write command sequencer

Writing and reading the Data Flash is possible only when the Write command sequencer is in the Idle state (DFWSA/B:ST[1:0]="00"). In this case, reading is possible in byte or word mode as with disabled Write command sequencer. When the command sequencer is not in the Idle state, no Flash read access will be executed. Instead the flash interface outputs the default value $00_H$ and sets the prohibited access error flag DFWSA/B:PAERF.

A flash write access when the Write command sequencer is not idle is also ignored and will set the error flag DFWSA/B:PAERF. Always check the Write command sequencer status bits DFWSA/B:ST[1:0] before reading or writing to the Data Flash with activated Write command sequencer.

### *33.8.3.2 Interrupt functions of the Write command sequencer and DMA access*

The Write command sequencer can only handle one write command at a time. A second write request is ignored when a write operation is ongoing. The progress of a write operation can be checked by reading the Write command sequencer state or by using the interrupt functions. Memory blocks can be transferred by using the DMA function. Write errors are indicated by 3 error interrupt flags.

#### Idle Interrupt IDLINT (mainly for programming by DMA)

The DFWSx:IDLINT flag is set when the Write command sequencer is ready to receive new write data. This is the case after enabling the Write command sequencer (setting DFWCx:WE to 1) and when the command sequencer finishes a write operation. The interrupt and DMA function of this flag is enabled by the corresponding DFWCx:IDLINTE bit.

This flag can be used to initiate a DMA transfer to the Data Flash. The DMA and interrupt module must be configured before activating the Write command sequencer. Setting DFWCx:WE or DFWCx:IDLINTE to 1 will then assert the first interrupt which starts the DMA operation.

The IDLINT flag can be cleared by CPU writing and by the DMA clear function.

### Finish Interrupt FININT (mainly for programming by CPU)

The DFWSx:FININT flag is set after a successful termination of a write command. It can be used to inform the CPU about the result of the started write command and that the Write command sequencer is ready to program the next data. The interrupt function of this flag is enabled by the corresponding DFWCx:FININTE bit.

The FININT flag can be cleared by CPU writing and by the DMA clear function.

### Error Interrupts (Timeout, Write error and Write incomplete)

An unsuccessful termination of a write operation when using the Write command sequencer is denoted with the setting of one of the following error flags:

- Timeout Interrupt flag DFWSx:TOERINT: This error flag is set when the Timing Limit Exceeded flag DQ5 is set during programming.
- Write Error Interrupt flag DFWSx:WERINT: This error flag is set when the write command terminates without having the correct data written (for example after trying to write to an erase suspended sector).
- Write Incomplete Error Interrupt flag DFWSx:WIERINT: This error flag is set when the write command sequencer is disabled while a write sequence is ongoing (DFWSx:ST[1:0]=01 or 10).

The interrupt function of these 3 flags is enabled by the corresponding DFWCx:ERINTE bit. A DMA Stop request is also asserted in case of an error interrupt.

## 33.8.4 Erasing All Data (Chip Erase)

This section describes the procedure for issuing the Chip Erase command to erase all data in a flash memory module.

### Erasing all data in the flash memory module (chip erase)

All data can be erased from a flash memory module by sending the Chip Erase command in the command sequence table (see Table 33-9 in Section 33.6 Starting the Flash Memory Automatic Algorithm) continuously to any sector in the target flash memory module.

The Chip Erase command is executed in six bus operations. When writing of the sixth cycle is completed, the chip erase operation is started. For chip erase, the user does not need to write 0 to the flash memory before erasing. During execution of the automatic erase algorithm, the flash memory automatically writes 0 for verification before all of the cells are erased (pre-programming).

## 33.8.5 Erasing Optional Data (Sector erase)

This section describes the procedure for issuing the Sector Erase command to erase optional data (sector erase) in the flash memory. Individual sectors can be erased. Multiple sectors can also be specified at one time.

### *33.8.5.1 Starting the Sector erase automatic algorithm*

Optional sectors in the flash memory can be erased by sending the Sector Erase command in the command sequence table (see Table 33-9 in Section 33.6 Starting the Flash Memory Automatic Algorithm) continuously to the target sector in the flash memory.

### Specifying sectors

A Sector erase is initiated by submitting the sector erase command (six write operations) to the target sector. After submitting the last command (30h) to an even address in the target sector, the sector erase wait time is applied for approximately 50µs. To erase multiple sectors, write the erase code 30h (sixth cycle of the command sequence) to the next target sector within this wait time. The first 5 cycles of the sector erase command do not have to be written in this case.

### Notes on specifying multiple sectors

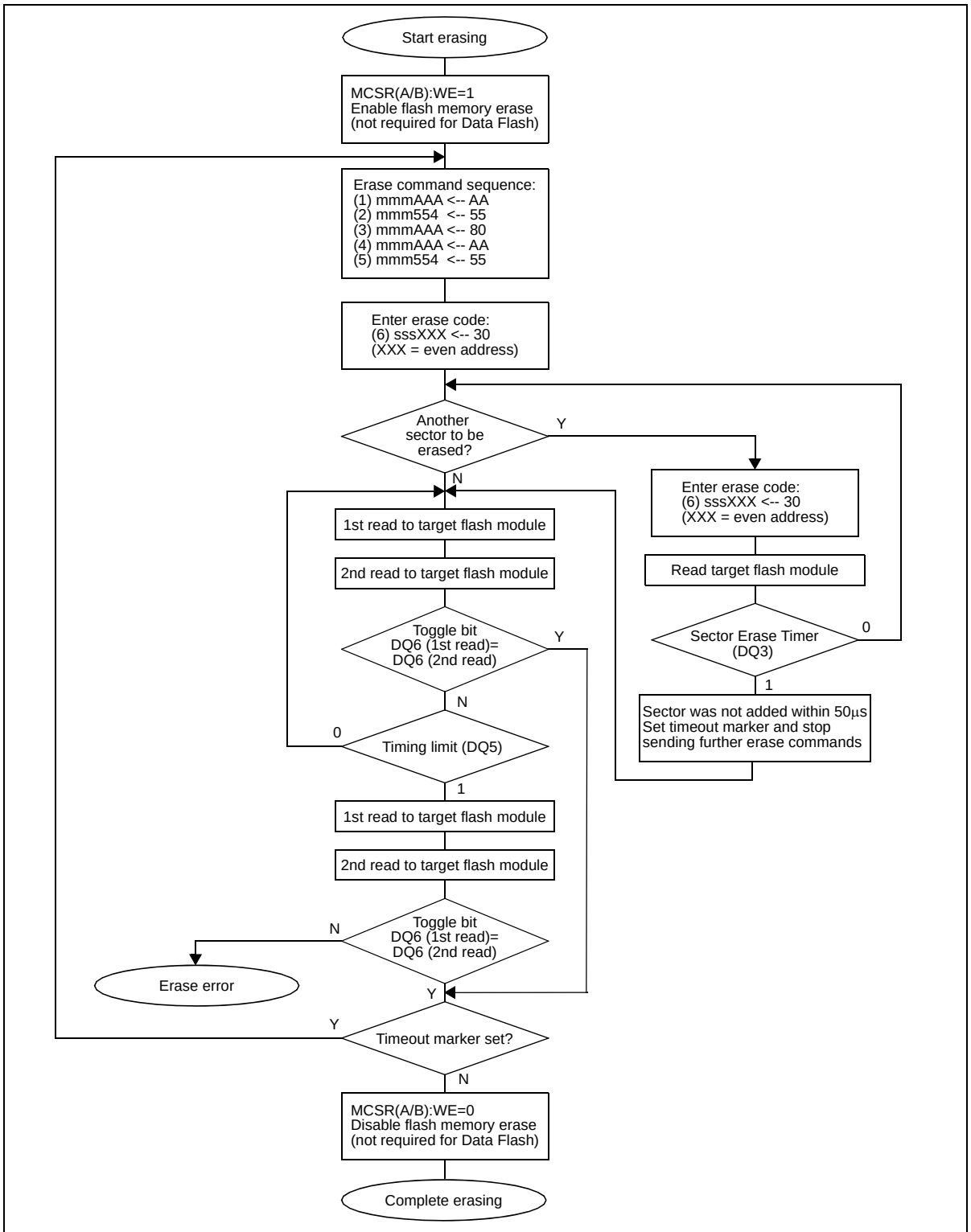
Erase is started when the sector erase wait period of 50µs terminates after the last sector erase code (30h) has been written. Each writing of a sector erase code restarts the sector erase wait period. The sector erase timer flag DQ3 must be checked after submitting each erase code to make sure it has been accepted.

### 33.8.5.2 Example for erasing sectors in the flash memory

The hardware sequence flags (see Section 33.7 Confirming the Automatic Algorithm Execution State) can be used to determine the state of the automatic algorithm in the flash memory. Figure 33-12 shows an example where the toggle bit flag (DQ6) is used to confirm that erasing has terminated.

When the automatic erase algorithm stops, the Flash changes asynchronously from the automatic algorithm execution and the read/reset state. A possible read access to the Flash at this time can return invalid data. For example to safely identify the Erase error state, the DQ6 toggle bits must be checked again after reading DQ5=1.

Figure 33-12. Example of the flash memory sector erase procedure




**Note:**

The address "mmm" must point to the flash memory module on which the command is meant to operate.
The address "sss" must point to the flash memory sector on which the command is meant to operate.

## 33.8.6 Suspending Sector Erase

This section describes the procedure for issuing the Sector Erase Suspend command to suspend erasing of flash memory sectors. In this state, data can be read from or written to sectors that are not specified to be erased.

### *33.8.6.1 Suspending erasing of flash memory sectors*

Erasing of flash memory sectors can be suspended by sending the Sector Erase Suspend command ($B0_H$) to an even address in the target flash memory module. Submitting this command suspends the sector erase operation being executed. In this suspend state, it is possible to read data from or to write data to sectors that are not specified to be erased. Further erase operations are forbidden.

The Sector Erase Suspend command is valid only while the sector erase operation is in progress or while the sector erase wait time is being applied. The command will be ignored during chip erase or write operations. If a Sector Erase Suspend command is issued during sector erase suspend, the command will be ignored.

Entering the Sector Erase Suspend command during the sector erase wait period will immediately terminate sector erase wait and cancel the requested erase operation. While the sector erase is in progress, it takes a maximum period of 20µs from submitting the Sector Erase Suspend command until the flash changes to the suspend state.

**Notes on using the sector erase suspend function of the Program/Data Flash (Main Flash):**

- After starting or restarting a sector erase command, a minimum wait time of 1.2ms must be applied before submitting a sector erase suspend command.
- The erase operation of one sector must not be suspended for more than 10 times.

**Notes on using the sector erase suspend function of the Data Flash:**

The same rules as stated above for the Program/Data Flash (Main Flash) are also valid for the Data Flash. However it is allowed to suspend a sector erase operation more frequently, when the following rules are obeyed:

- During sector erase, the DQ4 flag must be polled with a maximum read period of 20µs.
- If DQ4 is 1 (busy to suspend), polling must continue.
- If DQ4 becomes 0 (ready to suspend), the erase suspend command must be written immediately.
- The maximum length of the "busy to suspend" period is 2.4ms. Within this time, no suspend command should be written to the flash.

## 33.8.7 Restarting Sector Erase

This section describes the procedure for issuing the Sector Erase Restart command to restart suspended erasing of flash memory sectors.

**Restarting erasing of flash memory sectors**

Suspended erasing of flash memory sectors can be restarted by sending the Sector Erase Restart command ($30_H$) to an even address in the target flash memory module. If a Sector Erase Restart command is issued during sector erase, the command will be ignored.

The Sector Erase Restart command is used to restart erasing of sectors from the sector erase suspend state set by using the Sector Erase Suspend command.

## 33.8.8 Protecting sectors from being erased/written to

This section describes the procedure to protect sectors of the Program/Data Flash (Main Flash) from being unintentionally erased or written to.

### 33.8.8.1 Function of the sector write protection

Individual sectors of the Program/Data Flash (Main Flash) can be protected from being erased (by sector erase command) or from being written (by write command). This can be configured by programming the Flash Memory Write Control registers. For a description of these registers, please refer to 33.4.3 Flash Memory Write Control registers 0-5 (FMWC0-FMWC5). This feature can be used to prevent the application from an unintended erase or rewrite of specified Flash sectors. It is still possible to write/erase the flash memory in Parallel Flash programming mode or in Serial Communication mode.

An attempt to send a command sequence to a write protected sector is cancelled by asserting a flash hardware reset for the corresponding Flash module. This includes all write accesses of any command sequence. Hence for submitting a command sequence to the Flash, only sectors which are not write protected must be addressed.

The flash hardware reset is asserted for 64 CLKB clock cycles. During this time, the corresponding RDY flag (MCSRA:RDY or MCSRB:RDY) is forced to 0.

The sector write protection has no effect in Parallel Flash programming mode.

#### Notes on writing to the flash during execution of the automatic algorithm

Any write access to a write protected sector activates the flash hardware reset. Such a hardware reset stops any ongoing automatic algorithm execution (write or erase). This causes data being written to be undefined and sectors being erased to be corrupted. In such a case, the interrupted write or erase command must be submitted again.

#### Chip erase

It can be selected if a Chip Erase command is executed or not when sector protection for some existing sector is enabled.

The Chip Erase command is not executed when:

- Any write access of the Chip Erase command sequence is issued on a protected sector (Flash hardware reset is asserted). When all sector protection bits are set to 0, then the Chip Erase command sequence is always cancelled.
- The command data in the $5^{th}$ and $6^{th}$ write cycle is written in word mode as 0x0055 and 0x0010 (Chip Erase command is ignored).

The Chip Erase is performed even though any sector protection is enabled when:

- The command is issued to an unprotected and available sector, the command is issued in word mode, and the upper byte is different than 0x00.

### 33.8.8.2 Activating the write protection by user program

The user program can configure the sector write protection by direct programming of the Flash Memory Write Control registers (FMWC0-FMWC5).

### 33.8.8.3 Activating the write protection without user program interaction

Individual sectors of the flash memory can be protected from being erased or written to by filling in data into dedicated locations of the Flash memory (ROM configuration blocks RCBA/RCBB). No user program interaction is needed to activate the write protection in this case.

The write protection for sectors of Flash A is defined by the Flash Write Protection Activation Marker A (FWPAMA1/0) and the Flash Write Protection Sector Markers A (FWPSMA0 - FWPSMA4) stored in RCBA.

The write protection for sectors of Flash B is defined by the Flash Write Protection Activation Marker B (FWPAMB1/0) and the Flash Write Protection Sector Markers B (FWPSMB0 - FWPSMB4) stored in RCBB.

After any reset and before user program execution the following is executed (in Internal and External Vector modes):

#### Flash A:

When the content of {FWPAMA1, FWPAMA0} = 292D3A7BH, then the content of FWPSMA0-4 is transferred to FMWC1-5 for all existing sectors of Flash A.

For any other value of {FWPAMA1, FWPAMA0}, the content of FWPSMA0-4 is ignored and no transfer to the FMWC registers is performed.

### Flash B (only devices with Flash B):

When the content of {FWPAMB1, FWPAMB0} = 292D3A7BH, then the content of FWPSMB0-4 is transferred to FMWC0,2-5 for all existing sectors of Flash B.

For any other value of {FWPAMB1, FWPAMB0}, the content of FWPSMB0-4 is ignored and no transfer to the FMWC registers is performed.

As long as the sector containing the flash write protection markers is not protected against write/erase (for Flash A memory bit FWPSMA0:WCA0 = 1, for Flash B memory bit FWPSMB0:WCB0 = 1), it is possible to modify the content of the markers by writing to this marker or by erasing the Flash memory sector containing the marker. The new content is activated by asserting any reset.

After write protection for a set of sectors is activated by Flash Write Protection Markers, the user program can still add individual sectors to the set of write protected sectors by setting the corresponding bits in the Flash Memory Write Control registers 0-5 (FMWC0-FMWC5) to "0".

In Serial communication mode, the configuration data is not automatically transferred to the FMWC registers.

If the temporary UART scan is enabled in Internal Vector mode, then the configuration data is transferred to the FMWC registers after leaving the serial communication.

Figure 33-13. Configuration of the Flash Write Protection Activation Marker (FWPAMA0/1, FWPAMB0/1)

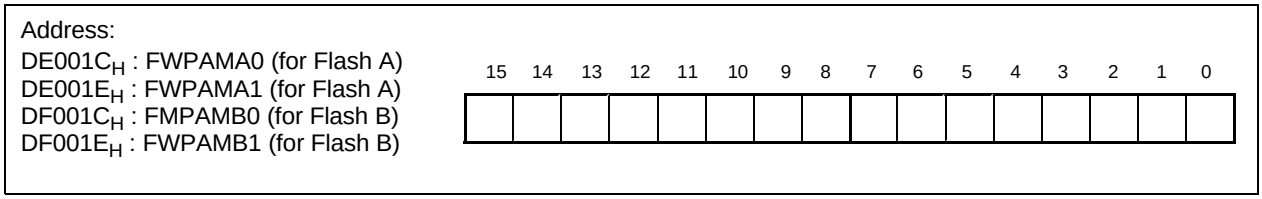
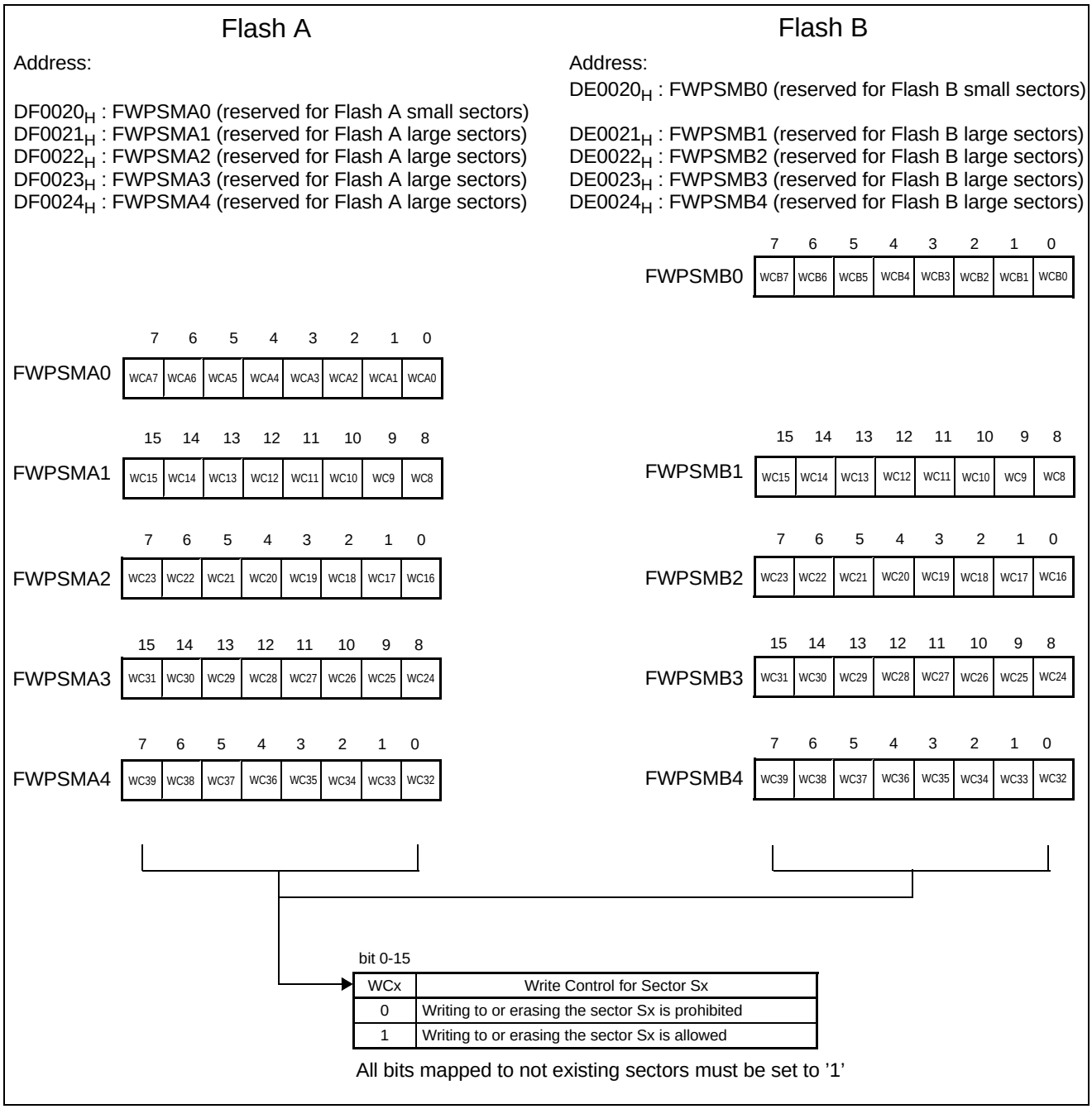


Figure 33-14. Configuration of the Flash Write Protection Sector Markers (FWPSMA0 -FWPSMA5, FWPSMB0 -FWPSMB5)



## 33.9 Notes on using Flash Memory

This section contains notes on using flash memory.

### 33.9.1 Notes on using flash memory

#### Compatibility to MBM29LV200TC

For the Program Data Flash (Main Flash), please review the MBM29LV200TC data sheet in conjunction with this document.

#### Input of a CPU reset (RST)

A CPU reset (Power reset, external reset, software reset, watchdog reset or clock stop reset) asserts the flash hardware reset. Such a reset assertion during flash writing causes the data being written to be undefined.

Resetting the device once execution of sector erase has begun will corrupt the data in the sector. In that case, restart the erase on this sector and allow it to complete.

#### Program access to flash memory

When the automatic algorithm is operating, read access to the flash memory at which the automatic algorithm is active, is disabled.

Hence, make sure that the CPU is not executing code from a flash memory that is erased/written. If the Flash A memory is to be erased/written, make sure that code is executed only from the Flash B memory (if available) or from RAM. If the Flash B memory is to be erased/written, make sure that code is executed only from the Flash A memory or from RAM.

For the same reason, make sure that the table base register (TBR) is not pointing to an interrupt vector table located in flash memory to be erased or written to. Program TBR to point to an interrupt vector table in the other flash memory (if available) or in RAM, or disable interrupts completely before starting the automatic algorithm.

#### DMA

DMA can only be used to serve write and erase interrupts of the Data Flash memory, not for the Program/Data Flash (Main Flash).

## 33.10 Flash memory programming example

This section presents a Flash memory programming example.

### Programming example

```
Flash memory sample program:
;=====================================================================
; THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.
;  CYPRESS ACCEPTS NO RESPONSIBILITY OR LIABILITY
;      FOR ANY ERRORS OR ELIGIBILITY FOR ANY PURPOSES.
;
;          (C) CYPRESS EUROPE 1998-2007
;=====================================================================
;
; This program erases Flash A sector SA2 and programs 0xAA55
; to SA2. Afterwards the results are output on port P00 as follows:
;
; Bit# 0 1 2 3 4 5 6 7
;      ---------------
;      | | |   | | |
;      | | |   | | |
;      | | |   | | +---- EERR: Erase time out error
;      | | |   | +------ EOK1: Successfully erased (with "time out warning")
;      | | |   +-------- EOK0: Successfully erased
;      | | +------------ PERR: Program time out error
;      | +-------------- POK1: Successfully programmed (with "time out warning")
;      +---------------- POK0: Successfully programmed

                .PROGRAM  FLASH_W_E
                .TITLE    FLASH_WRITE_ERASE

; Defines
#set     POK0     0x0001
#set     POK1     0x0002
#set     PERR     0x0004
#set     EOK0     0x0010
#set     EOK1     0x0020
#set     EERR     0x0040

#set     DQ3      0x0008                    ; sector erase start
#set     DQ5      0x0020                    ; time out bit
#set     DQ7      0x0080                    ; data polling bit

; I/O
PDR00    .EQU     0x0000                    ; Port data register 00
DDR00    .EQU     0x0430                    ; Port direction register 00

MCSRA    .EQU     0x03F1                    ; Flash A Timing 1
MTCRA    .EQU     0x03F2                    ; Flash A Timing 2
FMWC1    .EQU     0x03F9                    ; Sector SA0 - SA7 unlock
CKSR     .EQU     0x0401                    ; Clock select
CKSSR    .EQU     0x0402                    ; Clock stabilization control
CKMR     .EQU     0x0403                    ; Clock stabilization status
CKFCR    .EQU     0x0404                    ; Clock frequency control
PLLCR    .EQU     0x0406                    ; PLL settings

; RAM
RAMSTART .EQU     0x4000                    ; Start address of RAM code

; FLASH
SA1      .EQU     0x00DF2000                ; SA1 data sector
SA1_554  .EQU     0x00DF2554                ; Flash sequence address 1
```

```
SA1_AAA  .EQU     0x00DF2AAA                ; Flash sequence address 2

                  .SECTION  RESVECT, CONST, LOCATE=0xFFFFDC
                  .DATA.E START
                  .SECTION  BOOT_SELECT, CONST, LOCATE=0xDF0030
                  .DATA.L 0xFFFFFFFF

;=====================================================================
;
; Main program in Flash memory

                  .SECTION  CODE_FLASH, CODE, LOCATE=0xFF8000

START:   AND      CCR,     #0x80            ; disable interrupts
                  MOV      RP, #0           ; register bank #0

                  MOV      CKSSR, #0xFC     ; set clock stabilization time 2^16/CLKMC
                  MOVW     PLLCR, #0x000D   ; CPU: 56 MHz, CLKP2: 14 MHz for 4 MHz ext.
                  MOVW     CKFCR, #0x3001   ; CLKB/1, CLKP1/1, CLKP2/4
                  MOVW     MTCRA, #0x4B3B   ; 3 wait states
                  MOV      CKSR,  #0xFA

                  MOVL     A, #0x00002540   ; top of system stack pointer
                  MOVW     SP, A
                  SWAPW
                  MOV      SSB, A

                  ; copy RAM code to RAM
                  MOVW     A, #RAMSTART     ; start of RAM code
                  MOVW     A, #(0xFFFF & FLASH) ; start of ROM code
                  MOVW     RW0, #SIZEOF(CODE_FLASH_RAM) ; number of code bytes
                  MOVS     ADB, PCB

                  MOV      A, #0xFF         ; init port
                  MOV      DDR00, A
                  MOVN     A, #0            ; 0x00 to port 00
                  MOV      I:PDR00, A

                  MOV      A, #0xFF         ; unlock SA0 - SA7
                  MOV      FMWC1, A

PLLWAIT: BBCCKMR:6, PLLWAIT; wait for clock stabilization

                  CALLP    (RAMSTART)       ; erase sector
                  MOV      PDR00, A         ; status flag to port 00
                  MOVW     RW6, A           ; save status flag

                  CALLP    (RAMSTART + 4)   ; write Data
                  MOVW     A, RW6           ; restore status flag
                  ORW      A
                  MOV      PDR00, A         ; show result

LOOP:    BRA LOOP                          ; endless loop

;====== FLASH ERASE/WRITE RAM CODE ================================
;
; RAM code functions

                  .SECTION  CODE_FLASH_RAM, CODE, LOCATE=0xFF9000

; jump table
FLASH:   JMPP     (ERASE - 0xFF9000 + RAMSTART)
         JMPP     (WRITE - 0xFF9000 + RAMSTART)
```

```
;====== SECTOR ERASE ============================================
;
; erases sector SA1
; input: none
; output (in A): EOK0 = successfully erased
;                EOK1 = successfully erased with "pre-time-out"
;                EERR = time out error

ERASE:            MOV      A, MCSRA          ; save Flash Timings
                  PUSHW    A
                  MOV      A, MTCRA
                  PUSHW    A

                  CLRB     MCSRA:4           ; disable Code Read Buffer
                  CLRB     MCSRA:5           ; disable Data Read Buffer
                  MOVW     MTCRA, #0x4B3D    ; slow down Flash access to 4 wait states
                  SETB     MCSRA:3           ; write enable

                  ; Sector erase sequence
                  MOVL     A,#SA1_AAA
                  MOVL     RL0, A
                  MOV      A, #0xAA          ; *0xDF2AAA = 0xAA
                  MOVW     @RL0, A

                  MOVL     A,#SA1_554
                  MOVL     RL0, A
                  MOV      A, #0x55          ; *0xDF2554 = 0x55
                  MOVW     @RL0, A

                  MOVL     A,#SA1_AAA
                  MOVL     RL0, A
                  MOV      A, #0x80          ; *0xDF2AAA = 0x80
                  MOVW     @RL0, A

                  MOVL     A,#SA1_AAA
                  MOVL     RL0, A
                  MOV      A, #0xAA          ; *0xDF2AAA = 0xAA
                  MOVW     @RL0, A

                  MOVL     A,#SA1_554
                  MOVL     RL0, A
                  MOV      A, #0x55          ; *0xDF2554 = 0x55
                  MOVW     @RL0, A

                  MOVL     A,#SA1
                  MOVL     RL0, A
                  MOV      A, #0x30          ; *0xDF2000 = 0x30
                  MOVW     @RL0, A

                  ; Wait for sector erase start
E_DQ3:            MOVW     A, @RL0+0         ; read sector state
                  MOVN     A, #DQ3
                  ANDW     A
                  MOVN     A, #DQ3
                  CMPW     A
                  BNZ      E_DQ3

                  ; Data polling algorithm
                  MOVN     A, #0             ; status flag in RW2
                  MOVW     RW2, A

E_LOOP:           MOVW     A, @RL0+0         ; read sector state
                  ANDW     A, #DQ7           ; data polling
                  CWBNE    A, #DQ7, E_1
```

```
                MOV     A, #EOK0          ; successful erased
                MOVW    RW2, A

E_1:            MOVW    A, @RL0+0         ; read sector state
                ANDW    A, #DQ5           ; time out?
                CWBNE   A, #DQ5, E_2

                MOVW    A, @RL0+0         ; read sector state
                ANDW    A, #DQ7           ; data polling
                CWBNE   A, #DQ7, E_ERR

                MOV     A, #EOK1          ; successful erased
                MOVW    RW2, A
                BRA     E_2

E_ERR:          MOV     A, #EERR          ; time out error
                MOVW    RW2, A

E_2:            MOVW    A, RW2
                BZ      E_LOOP

                CLRB    MCSRA:3           ; reset write enable

                POPW    A                 ; restore Flash Timings
                MOV     MTCRA, A
                POPW    A
                MOV     MCSRA, A

                MOVW    A, RW2            ; status flag in A

RAMEND:  RETP

;====== FLASH WRITE ===========================================
;
; writes 0xAA55 to start of sector SA1 (0xDF2000)
; input: none
; output (in A): POK0 = successfully written
;                POK1 = successfully written with "pre-time-out"
;                PERR = time out error

WRITE:           MOV     A, MCSRA          ; Save Flash Timings
                 PUSHW   A
                 MOV     A, MTCRA
                 PUSHW   A

                 CLRB    MCSRA:4           ; disable Code Read Buffer
                 CLRB    MCSRA:5           ; disable Data Read Buffer
                 MOVW    MTCRA, #0x4B3D    ; slow down Flash access to 4 wait states
                 SETB    MCSRA:3           ; write enable

                 ; Flash write sequence
                 MOVL    A,#SA1_AAA
                 MOVL    RL0, A
                 MOV     A, #0xAA          ; *0xDF2AAA = 0xAA
                 MOVW    @RL0, A

                 MOVL    A,#SA1_554
                 MOVL    RL0, A
                 MOV     A, #0x55          ; *0xDF2554 = 0x55
                 MOVW    @RL0, A

                 MOVL    A,#SA1_AAA
                 MOVL    RL0, A
```

```
                MOV     A, #0xA0          ; *0xDF2AAA = 0xA0
                MOVW    @RL0, A

                MOVL    A, #SA1
                MOVL    RL0, A
                MOVW    A, #0xAA55        ; *0xDF2000 = dummy data
                MOVW    @RL0, A

                ; Data polling algorithm
                MOVN    A, #0             ; status flag in RW2
                MOVW    RW2, A

W_LOOP:         MOVW    A, @RL0+0         ; read Flash state
                MOVW    RW4, #0xAA55      ; dummy data
                ANDW    A, #DQ7           ; data polling
                MOVW    RW5, A
                MOVW    A, RW4
                ANDW    A, #DQ7
                CMPW    A, RW5
                BNZ     W_1

                MOVW    A, #POK0          ; successful written
                MOVW    RW2, A

W_1:            MOVW    A, @RL0+0         ; read Flash state
                ANDW    A, #DQ5           ; time out?
                CWBNE   A, #DQ5, W_2

                MOVW    A, @RL0+0         ; read Flash state
                MOVW    RW4, #0xAA55      ; dummy data
                ANDW    A, #DQ7           ; data polling
                MOVW    RW5, A
                MOVW    A, RW4
                ANDW    A, #DQ7
                CMPW    A, RW5
                BNZ     W_ERR

                MOVW    A, #POK1          ; successful written
                MOVW    RW2, A
                BRA     W_2

W_ERR:          MOVW    A, #PERR          ; time out error
                MOVW    RW2, A

W_2:            MOVW    A, RW2
                BZ      W_LOOP

                CLRB    MCSRA:3           ; reset write enable

                POPW    A                 ; restore Flash Timings
                MOV     MTCRA, A
                POPW    A
                MOV     MCSRA, A

                MOVW    A, RW2            ; status flag in A

                RETP

; ===== SA1 DUMMY DATA ===================================
;
; SA1 filled with 4 0x00 bytes to confirm sector erase

                .SECTION SA1_DUMMY_DATA, CODE, LOCATE=0xDF2000
```

```
.DATA.L  0x00000000

.END START                ; for debugging
```

# 34. Mask-ROM Memory

This chapter explains the functions and operation of the Mask-ROM memory.

34.1 Overview of the Mask-ROM memory

34.2 Mask-ROM Memory Control Registers

## 34.1 Overview

This chapter describes the main features of the Mask-ROM memory. The interface for the  usage of Mask-ROM is compatible with the one for Flash memory.

### Main features

The Mask-ROM memory supports the following features:

- Configuration of the Mask-ROM memory interface is compatible with the Flash interface.
- For timing compatibility with flash devices the wait cycles can be configured by Memory Timing Configuration Register (MTCRA).
- Two separate read buffers for code and data fetch for compatibility with Flash memory. The buffers can be either enabled (default) or disabled.
- Code security (read protection) as for Flash memory. Please refer to ROM/Flash Security on page 713 for details.

## 34.2 Mask-ROM Memory Control Registers

This chapter describes the Mask-ROM Memory Control Registers.

### Memory Control Status Register (MCSRA)

In the description below, the bit described in grey do exist in the Mask-ROM memory interface and indicate thus the compatibility with the Flash interface. In case of a Mask-ROM memory, those bits have no functionallity.

Figure 34-1. Memory Control Status Register A (MCSRA)

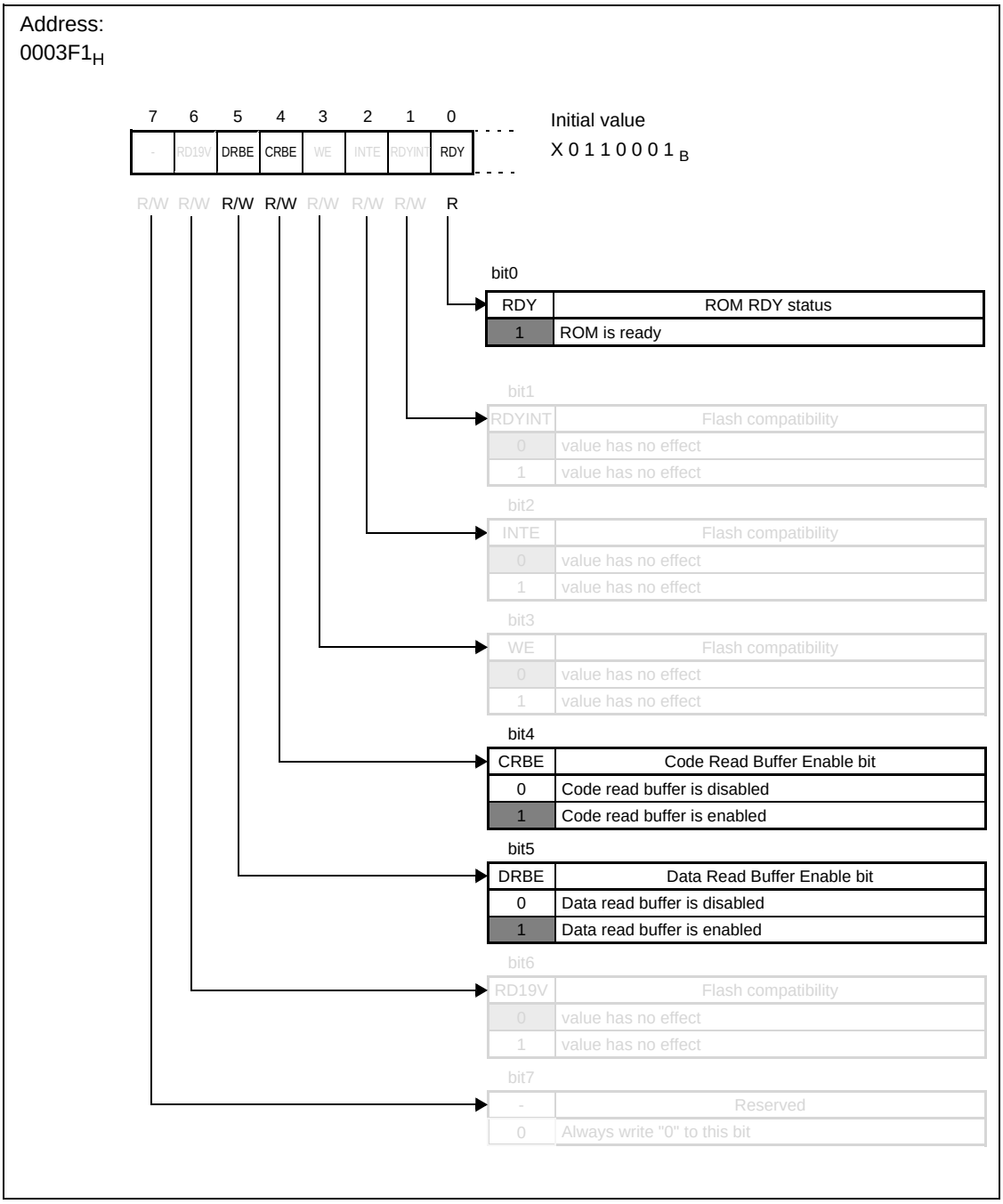


Table 34-1. Memory Control Status Register (MCSRA)

| Bit name | | Function |
|---|---|---|
| bit 0 | RDY | This bit is kept for compatibility to Flash interface.<br>Only a read value of '1' is possible which indicates that the ROM is ready. |
| bit 1 - bit 3 | - | Only available for compatibility with Flash interface but without functionallity. Any values can be writen here. |
| bit 4 | CRBE | This bit enables/disables the code read buffer.<br>Setting this bit to '1' enables the code read buffer. A code read access from any address reads two 16-bit words, the one that is addressed and the neighboring one, such that the resulting two 16-bit words are aligned to an even address. Both words are stored into a read buffer. A later CPU read access to the address of the buffered data will be executed without wait cycles.<br>Setting this bit to '0' disables the code read buffer. Code is always read directly from the Mask-ROM.<br>The initial value of this bit is '1' (buffer enabled). |
| bit 5 | DRBE | This bit enables/disables the data read buffer.<br>Setting this bit to '1' enables the data read buffer. A data read access from any address reads two 16-bit words, the one that is addressed and the neighboring one, such that the resulting two 16-bit words are aligned to an even address. Both words are stored into a read buffer. A later CPU read access to the address of the buffered data will be executed without wait cycles.<br>Setting this bit to '0' disables the data read buffer. Data is always read directly from the Mask-ROM.<br>The initial value of this bit is '1' (buffer enabled). |
| bit 6 | - | Only available for compatibility with Flash interface but without functionallity. Any values can be writen here. |
| bit 7 | Reserved | Always write "0" to this bit.<br>The read value of this bit is undefined.<br>Read modify write operations to this register are not affected. |

### Memory Timing Configuration Register A (MTCRA)

In the description below, the bit described in grey do exist in the Mask-ROM memory interface and indicate thus the compatibility with the Flash interface. In case of a Mask-ROM memory, those bits have no functionallity.

Figure 34-2. Memory Timing Configuration Register A low byte (MTCRAL)

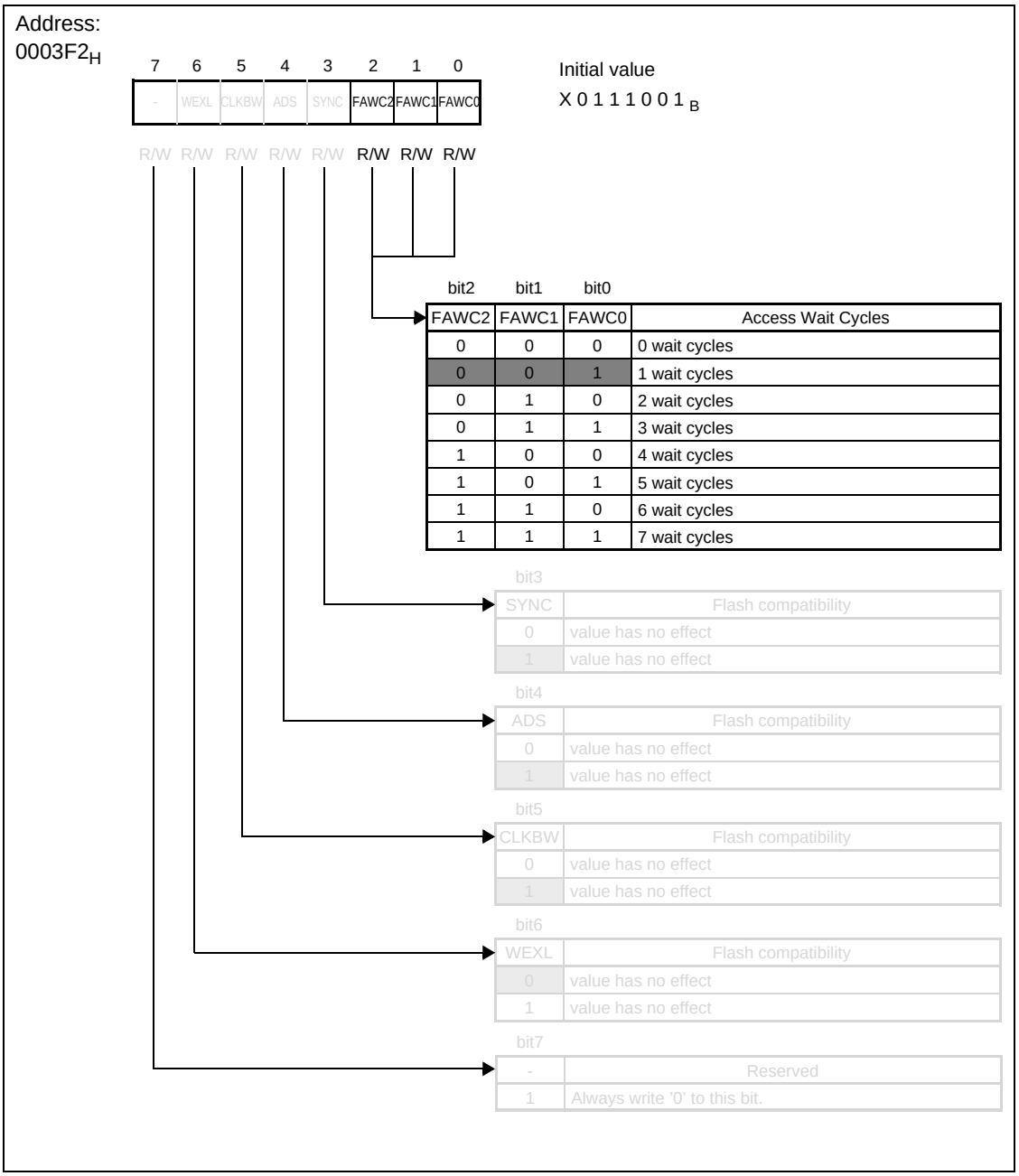



Table 34-2. Memory Timing Configuration Register low byte (MTCRAL)

| Bit name | | Function |
|---|---|---|
| bit 0 - bit 2 | FAWC0 to FAWC2 | • These bits define how many CPU wait cycles (CLKB cycles) are added at any read and write access to the Mask-ROM.<br>• These bits must be set depending on the machine clock frequency and the other configuration bits.<br>• The initial value of these bits is "001" (1 wait cycle) |
| bit 3 - bit 6 | - | Only available for compatibility with Flash interface but without functionallity. Any values can be writen here. |
| bit 7 | Reserved | • Always write "0" to these bits.<br>• The read value of these bits is undefined.<br>• Read modify write operations to this register are not affected. |

Figure 34-3. Memory Timing Configuration Register A high byte (MTCRAH)

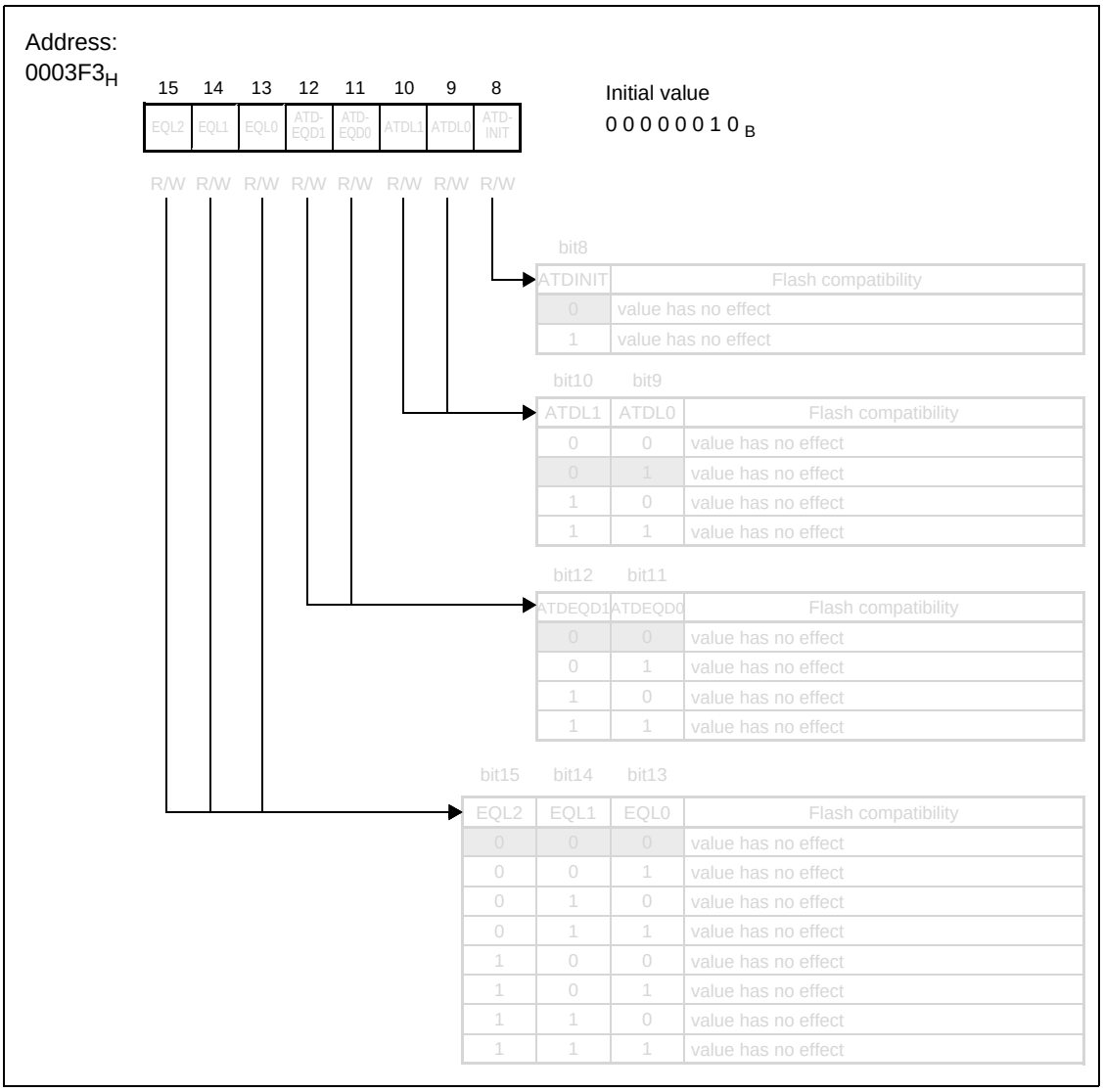



Table 34-3. Memory Timing Configuration Register A high byte (MTCRAH)

| Bit name | | Function |
|---|---|---|
| bit 8 - bit 15 | - | Only available for compatibility with Flash interface but without functionallity. Any values can be writen here. |

# 35. ROM/Flash Security

This chapter explains the functions and operation of the ROM/Flash security.

35.1 Overview of the ROM/Flash Security

35.2 Usage of the ROM/Flash Security

## 35.1 Overview of the ROM/Flash Security

The ROM/Flash security is a feature to prevent that the content of the ROM/Flash memory is read-out in a way not intended in the application.

### ROM/Flash security features

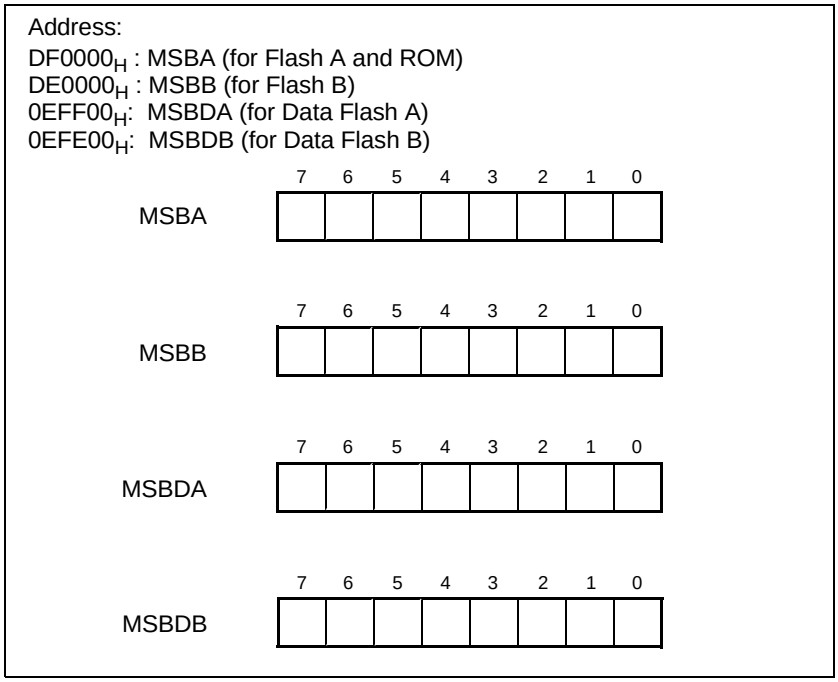
- Read protection feature to prevent reading out the content of the ROM/Flash memory when not intended in the application. Only internal user program that has been started in internal vector mode has access to the content of the ROM/Flash memory. The ROM/Flash memory cannot be read by:
  - program activated by external boot vector fetch (modes 0/1/6)
  - external parallel flash programmer (mode 7)
  - serial communication mode (mode 2)
- In all modes the internal Flash can still be erased.
- Separate protection of Flash A, Flash B, Data Flash A and Data Flash B.
- Unlock of read protection by 128-bit security key in serial communication mode, if configured. Security key can also be disabled permanently.

## 35.2 Usage of the ROM/Flash Security

This section describes how the ROM/Flash Security is enabled and disabled and how the content of the ROM/Flash memory is protected against unintended read-out.

### Enabling the ROM/Flash security

Figure 35-1. Configuration of the ROM/Flash Memory Security Byte




Flash memory security can be enabled independently for Flash A, Flash B, Data Flash A and Data Flash B by programming the Flash Memory Security Bytes MSBA, MSBB, MSBDA and MSBDB. In ROM devices, the security is controlled by the MSBA register only.

When the content of MSBA is $99_H$, then the Flash A or ROM memory is protected. If the content of MSBA is $66_H$, then the Flash A or ROM memory is not protected. No other value than $99_H$ or $66_H$ should be written to the MSBA register.

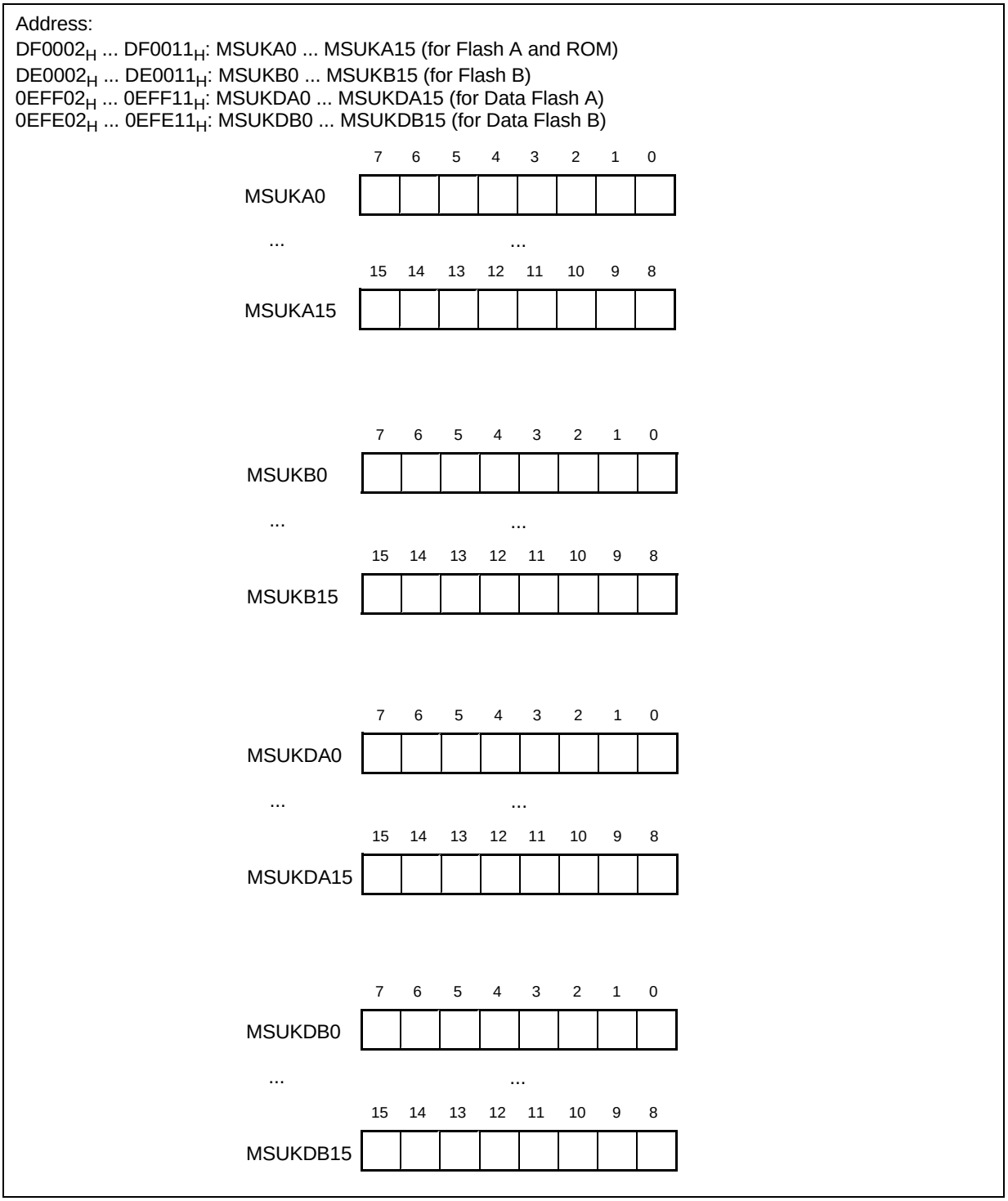
When the content of MSBB, MSBDA or MSBDB is $99_H$, then the according Flash memory is protected. If the content of MSBB, MSBDA or MSBDB is $66_H$, then the according Flash memory is not protected. No other value than $99_H$ or $66_H$ should be written to the MSBB, MSBDA or MSBDB register.

**Note:**

After erasing the flash, the value of all bits in flash memory is '1'. Hence, when the Flash memory location of MSBA, MSBB, MSBDA or MSBDB is not written, the content of it is $FF_H$. In this case, the content of the Flash memory is not protected.

### Unlocking the ROM/Flash security

Figure 35-2. Configuration of the ROM/Flash Memory Security Unlock Key




A Flash or ROM memory which is protected (Memory Security Byte is $99_H$) can still be read-out in serial programming mode, after an unlock key has been received by the UART which is used for programming. For details how to send the unlock key via the UART, please refer to application note MCU-AN-300213-E.

The unlock key is compared against the key stored in the unlock key bytes MSUKA0 ... MSUKA15 (for Flash A and ROM memory), MSUKB0 ... MSUKB15 (for Flash B memory), MSUKDA0 ... MSUKDA15 (for Data Flash A) and MSUKDB0 ... MSUKDB15 (for Data Flash B). If the received key matches the key stored in the Flash/ROM memory, the memory can be read-out by subsequent serial programming commands.

When the unlock key value stored in MSUKA0 ... MSUKA15/MSUKB0 ... MSUKB15/MSUKDA0 ... MSUKDA15/MSUKDB0 ... MSUKDB15 is all zero, the protection of the corresponding Flash memory can not be unlocked. For ROM devices this key value is not permitted.

**Note:**

Deactivation of unlocking key also prevents failure analysis of Flash contents. Feature to be used only if necessary.

### Disabling ROM/Flash security

On mask ROM devices, the ROM/Flash security can not be disabled.

On flash memory devices, the ROM/Flash security can be disabled as follows:

- When starting in internal vector mode (MD[2:0] = $011_B$), the application program can disable the Flash security by
  - changing the value of the Memory security byte to a value different to $99_H$ by writing to this Flash memory location.
  - clearing the value of the Memory security byte by erasing the sector containing this byte.
  - clearing the value of the Memory security byte by erasing the complete flash memory with the "Chip erase" command.
- When starting in external vector mode (MD[2:0] = $000_B$ or MD[2:0] = $001_B$ or MD[2:0] = $110_B$), the application program can disable the Flash security only by
  - clearing the value of the Memory security byte by erasing the complete flash memory with the "Chip erase" command.
- When starting in serial communication mode (MD[2:0] = $010_B$), the Flash security can be disabled by
  - changing the value of the Memory security byte to a value different to $99_H$ by writing to this Flash memory location after the Flash security has been unlocked.
  - clearing the value of the Memory security byte by erasing the sector containing this byte after the Flash security has been unlocked.
  - clearing the value of the Memory security byte by erasing the complete flash memory with the "Chip erase" command.
- When starting in parallel flash programming mode (MD[2:0] = $111_B$) the Flash security can be disabled only by
  - clearing the value of the Memory security byte by erasing the complete flash memory with the "Chip erase" command.

# 36. Examples of Serial Programming Connection

This chapter describes how to connect the MCU for in-circuit serial programming of the Flash memory.

36.1 Basic Configuration of Serial Programming Connection

36.2 Example of connecting a PC for programming the Flash Microcontroller

36.3 Example of connecting a programming tool for programming the Flash Microcontroller

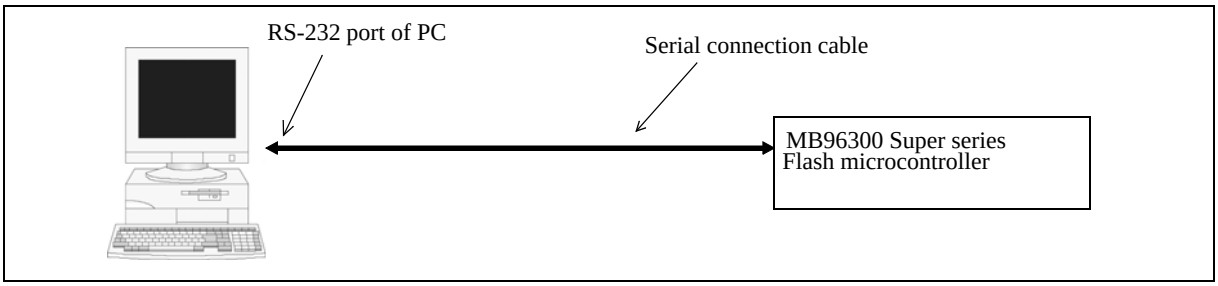
## 36.1 Basic Configuration of Serial Programming Connection

The MB96300 Super series Flash Microcontrollers support Flash ROM serial onboard programming. This section describes how to connect the Microcontroller to the programming equipment.

### Basic configuration of serial programming

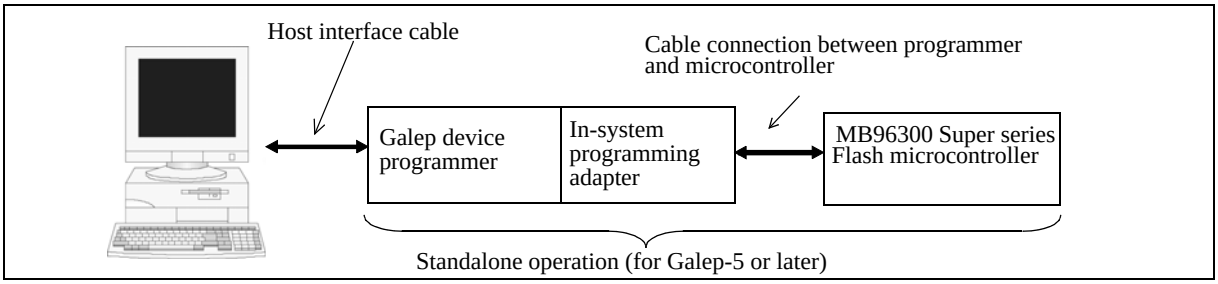
In a minimal system, the MB96300 Super series Flash micrcocontroller is connected via the USART in asynchronous mode to the RS-232 port of the PC.

Figure 36-1. Connection between PC serial port to Cypress MB96300 Super series flash microcontrollers




For high speed serial on board programming, programming tools of different vendors can be used. For example, the Galep device programmer from CONITEC DATASYSTEMS supports Cypress MB96300 Super series Flash memory microcontrollers.

Figure 36-2. Connection between CONITEC DATASYSTEMS Galep device programmer to Cypress MB96300 Super series flash microcontrollers



**Note:**

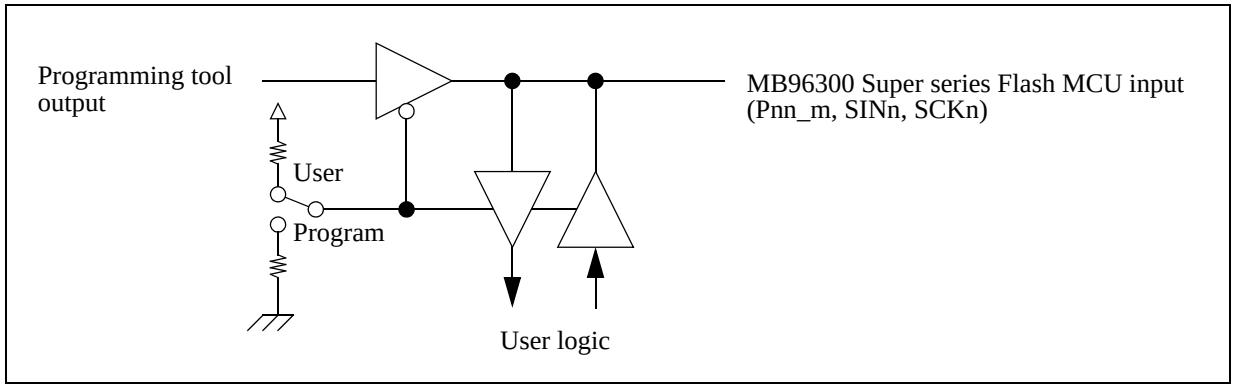
Please refer to CONITEC DATASYSTEMS for details about the Galep device programmer.

Table 36-1. Pins used for Cypress serial onboard programming

| Pin | Function | Additional information |
|---|---|---|
| MD2, MD1<br>MD0 | Mode pins | Controls the operation mode of the MCU. For details, please refer to chapter Boot ROM program execution and Operation mode and ROM Configuration Block on page 165. |
| X0, X1 | Oscillation pins | Some programming tools require quartz precise timing of the MCU. For this reason, the on-chip quartz oscillator circuit must be used. |
| Pnn_m | Communication handshaking | Some programming tools require a handshake signal to speed up communicating with the MCU. Please refer to the datasheet about the port recommended for handshaking. |
| $\overline{RST}$ | Reset pin | Some programming tools require to control the reset line of the MCU. |
| SINn | Serial data input pin | The USART is used for communication between the programming tool and the MCU. The MCU detects automatically, which USART from a predefined set of USARTs is connected to the programming tool. Please refer to the datasheet for the USART channel numbers that can be used for programing the MCU. |
| SOTn | Serial data output pin | |
| SCKn | Serial clock signal input pin | |
| C | C pin | This external capacitor pin is used to stabilize the power supply. Please refer to the data-sheet for the recommended capacitance values and types. |
| $V_{CC}$ | Power voltage supply pin | If the power is supplied from the user system, the device programmer power supply does not need to be connected. In case the device programmer supply is connected, please connect so that the power supply of the user side is not short-circuited. |
| $V_{SS}$ | GND pin | Common to the ground of the flash microcomputer programmer. |

When the signals driven by the device programmer are also used for the user system, the control circuit shown in the figure below is required. The circuit enables or disables the programming connection depending on the state of a jumper.

Figure 36-3. Connecting user circuitry to ports used for serial programming




## Serial data baud rate

The MCU determines automatically the serial data baud rate as adjusted in the programming tool or PC running programming software.

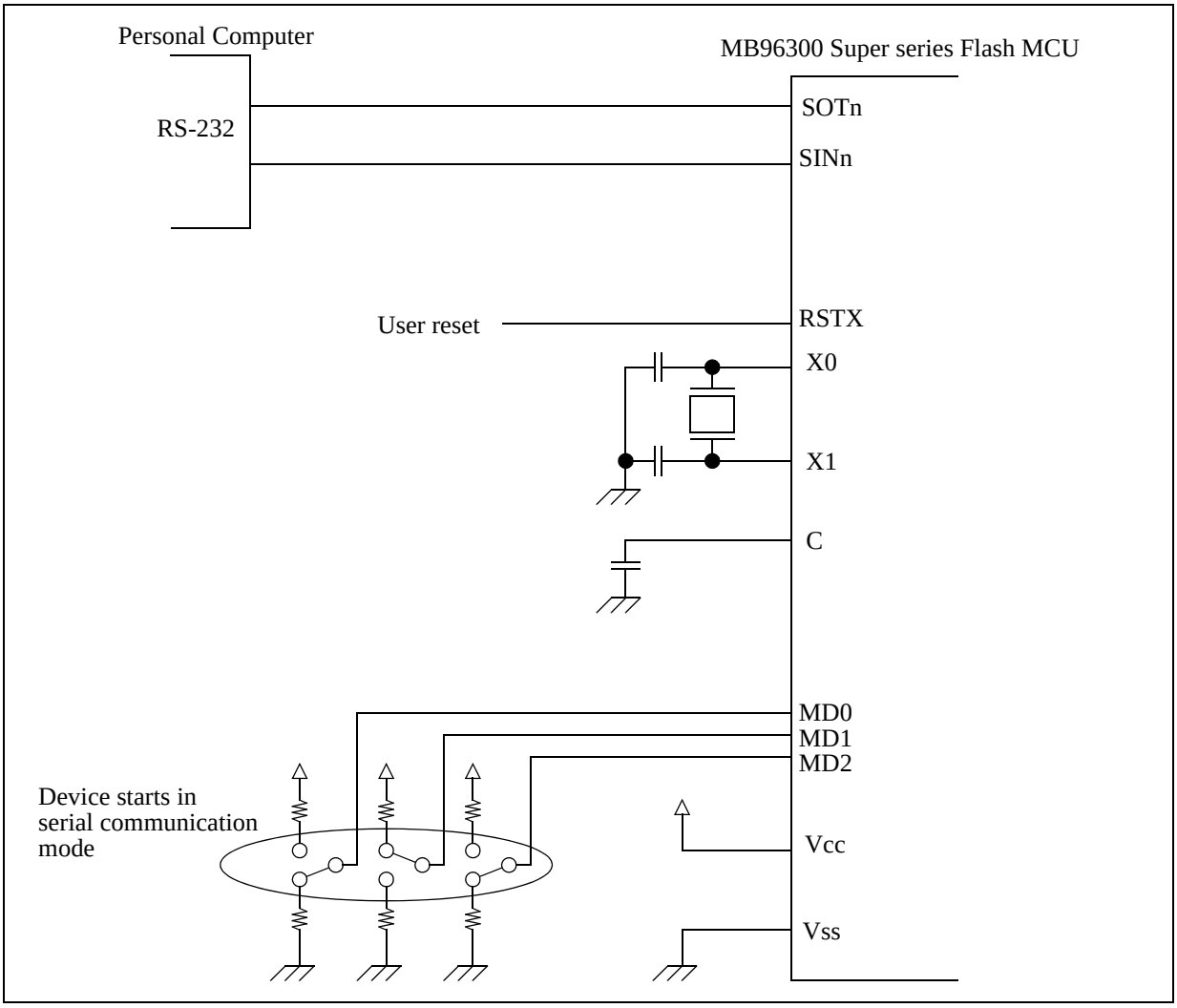
## 36.2 Example of connecting a PC for programming the Flash Microcontroller

Figure 36-4 shows an example of the minimum connection to a PC to program the flash microcontroller.

**Example of minimum connection to a PC to program the flash microcontroller.**

For a minimum connection between a PC and the Flash MCU, only a USART in asynchronous mode needs to be connected. Please refer to the datasheet for an overview of USART channels available for flash serial programming.

Figure 36-4. Example of minimum connection to a PC to program the flash microcontroller.



## 36.3 Example of connecting a programming tool for programming the Flash Microcontroller

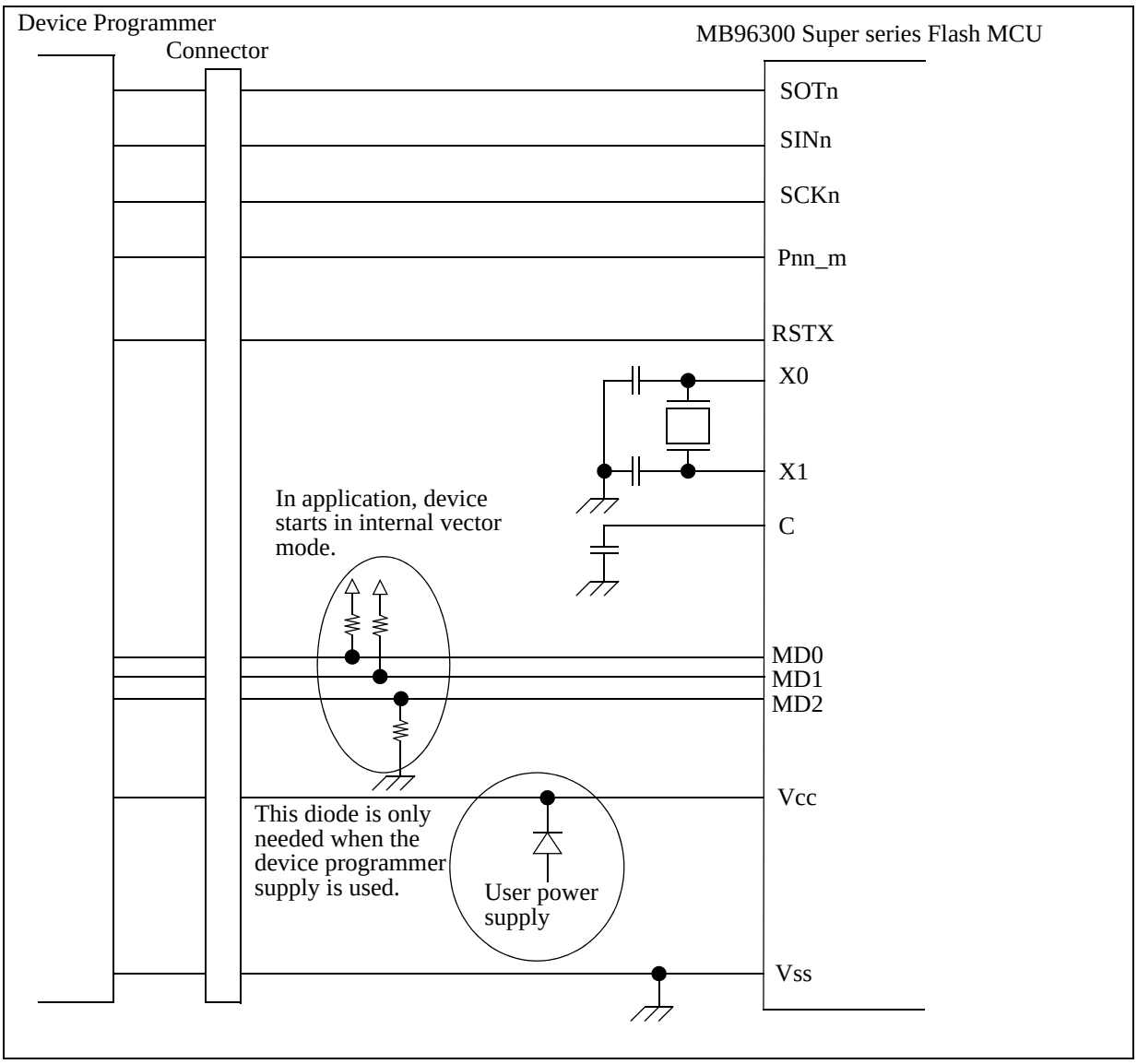
Figure 36-5 is an example of connecting a programming tool to program the flash microcontroller.

### Example of connecting a programming tool to program the flash microcontroller

For high speed programming a device programmer must be connected as shown in the example below. Please refer to the datasheet for an overview of USART channels available for flash serial programming.

Please refer to the datasheet to determine the port used for handshaking between the device programmer and the MCU.

Figure 36-5. Example of a connection to a device programmer to program the flash microcontroller which is started in internal vector mode in the application.



# 37. Appendix

The appendixes provide I/O maps, instructions, and other information.

37.1 I/O Map of MB96V300

37.2 Instructions

37.3 Timing Diagrams in Flash Memory Mode

## 37.1 I/O Map of MB96V300

This appendix lists the addresses to be assigned to the registers in the peripheral blocks.

### 37.1.1 I/O maps of MB96V300

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000000H | I/O Port P00 - Port Data Register | PDR00 | | RW |
| 000001H | I/O Port P01 - Port Data Register | PDR01 | | RW |
| 000002H | I/O Port P02 - Port Data Register | PDR02 | | RW |
| 000003H | I/O Port P03 - Port Data Register | PDR03 | | RW |
| 000004H | I/O Port P04 - Port Data Register | PDR04 | | RW |
| 000005H | I/O Port P05 - Port Data Register | PDR05 | | RW |
| 000006H | I/O Port P06 - Port Data Register | PDR06 | | RW |
| 000007H | I/O Port P07 - Port Data Register | PDR07 | | RW |
| 000008H | I/O Port P08 - Port Data Register | PDR08 | | RW |
| 000009H | I/O Port P09 - Port Data Register | PDR09 | | RW |
| 00000AH | I/O Port P10 - Port Data Register | PDR10 | | RW |
| 00000BH | I/O Port P11 - Port Data Register | PDR11 | | RW |
| 00000CH | I/O Port P12 - Port Data Register | PDR12 | | RW |
| 00000DH | I/O Port P13 - Port Data Register | PDR13 | | RW |
| 00000EH | I/O Port P14 - Port Data Register | PDR14 | | RW |
| 00000FH | I/O Port P15 - Port Data Register | PDR15 | | RW |
| 000010H | I/O Port P16 - Port Data Register | PDR16 | | RW |
| 000011H | I/O Port P17 - Port Data Register | PDR17 | | RW |
| 000012H-000017H | Reserved | | | - |
| 000018H | ADC0 - Control Status register Low | ADCSL | ADCS | RW |
| 000019H | ADC0 - Control Status register High | ADCSH | | RW |
| 00001AH | ADC0 - Data Register Low | ADCRL | ADCR | R |
| 00001BH | ADC0 - Data Register High | ADCRH | | R |
| 00001CH | ADC0 - Setting Register | | ADSR | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00001DH | ADC0 - Setting Register | | | RW |
| 00001EH | ADC0 - Extended Configuration Register | ADECR | | RW |
| 00001FH | Reserved | | | - |
| 000020H | FRT0 - Data register of free-running timer | | TCDT0 | RW |
| 000021H | FRT0 - Data register of free-running timer | | | RW |
| 000022H | FRT0 - Control status register of free-running timer Low | TCCSL0 | TCCS0 | RW |
| 000023H | FRT0 - Control status register of free-running timer High | TCCSH0 | | RW |
| 000024H | FRT1 - Data register of free-running timer | | TCDT1 | RW |
| 000025H | FRT1 - Data register of free-running timer | | | RW |
| 000026H | FRT1 - Control status register of free-running timer Low | TCCSL1 | TCCS1 | RW |
| 000027H | FRT1 - Control status register of free-running timer High | TCCSH1 | | RW |
| 000028H | OCU0 - Output Compare Control Status | OCS0 | | RW |
| 000029H | OCU1 - Output Compare Control Status | OCS1 | | RW |
| 00002AH | OCU0 - Compare Register | | OCCP0 | RW |
| 00002BH | OCU0 - Compare Register | | | RW |
| 00002CH | OCU1 - Compare Register | | OCCP1 | RW |
| 00002DH | OCU1 - Compare Register | | | RW |
| 00002EH | OCU2 - Output Compare Control Status | OCS2 | | RW |
| 00002FH | OCU3 - Output Compare Control Status | OCS3 | | RW |
| 000030H | OCU2 - Compare Register | | OCCP2 | RW |
| 000031H | OCU2 - Compare Register | | | RW |
| 000032H | OCU3 - Compare Register | | OCCP3 | RW |
| 000033H | OCU3 - Compare Register | | | RW |
| 000034H | OCU4 - Output Compare Control Status | OCS4 | | RW |
| 000035H | OCU5 - Output Compare Control Status | OCS5 | | RW |
| 000036H | OCU4 - Compare Register | | OCCP4 | RW |
| 000037H | OCU4 - Compare Register | | | RW |
| 000038H | OCU5 - Compare Register | | OCCP5 | RW |
| 000039H | OCU5 - Compare Register | | | RW |
| 00003AH | OCU6 - Output Compare Control Status | OCS6 | | RW |
| 00003BH | OCU7 - Output Compare Control Status | OCS7 | | RW |
| 00003CH | OCU6 - Compare Register | | OCCP6 | RW |
| 00003DH | OCU6 - Compare Register | | | RW |
| 00003EH | OCU7 - Compare Register | | OCCP7 | RW |
| 00003FH | OCU7 - Compare Register | | | RW |
| 000040H | ICU0/ICU1 - Control Status Register | ICS01 | | RW |
| 000041H | ICU0/ICU1 - Edge register | ICE01 | | RW |
| 000042H | ICU0 - Capture Register Low | IPCPL0 | IPCP0 | R |
| 000043H | ICU0 - Capture Register High | IPCPH0 | | R |
| 000044H | ICU1 - Capture Register Low | IPCPL1 | IPCP1 | R |
| 000045H | ICU1 - Capture Register High | IPCPH1 | | R |
| 000046H | ICU2/ICU3 - Control Status Register | ICS23 | | RW |
| 000047H | ICU2/ICU3 - Edge register | ICE23 | | RW |
| 000048H | ICU2 - Capture Register Low | IPCPL2 | IPCP2 | R |
| 000049H | ICU2 - Capture Register High | IPCPH2 | | R |
| 00004AH | ICU3 - Capture Register Low | IPCPL3 | IPCP3 | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00004BH | ICU3 - Capture Register High | IPCPH3 | | R |
| 00004CH | ICU4/ICU5 - Control Status Register | ICS45 | | RW |
| 00004DH | ICU4/ICU5 - Edge register | ICE45 | | RW |
| 00004EH | ICU4 - Capture Register Low | IPCPL4 | IPCP4 | R |
| 00004FH | ICU4 - Capture Register High | IPCPH4 | | R |
| 000050H | ICU5 - Capture Register Low | IPCPL5 | IPCP5 | R |
| 000051H | ICU5 - Capture Register High | IPCPH5 | | R |
| 000052H | ICU6/ICU7 - Control Status Register | ICS67 | | RW |
| 000053H | ICU6/ICU7 - Edge register | ICE67 | | RW |
| 000054H | ICU6 - Capture Register Low | IPCPL6 | IPCP6 | R |
| 000055H | ICU6 - Capture Register High | IPCPH6 | | R |
| 000056H | ICU7 - Capture Register Low | IPCPL7 | IPCP7 | R |
| 000057H | ICU7 - Capture Register High | IPCPH7 | | R |
| 000058H | EXTINT0 - External Interrupt Enable Register | ENIR0 | | RW |
| 000059H | EXTINT0 - External Interrupt Interrupt request Register | EIRR0 | | RW |
| 00005AH | EXTINT0 - External Interrupt Level Select Low | ELVRL0 | ELVR0 | RW |
| 00005BH | EXTINT0 - External Interrupt Level Select High | ELVRH0 | | RW |
| 00005CH | EXTINT1 - External Interrupt Enable Register | ENIR1 | | RW |
| 00005DH | EXTINT1 - External Interrupt Interrupt request Register | EIRR1 | | RW |
| 00005EH | EXTINT1 - External Interrupt Level Select Low | ELVRL1 | ELVR1 | RW |
| 00005FH | EXTINT1 - External Interrupt Level Select High | ELVRH1 | | RW |
| 000060H | RLT0 - Timer Control Status Register Low | TMCSRL0 | TMCSR0 | RW |
| 000061H | RLT0 - Timer Control Status Register High | TMCSRH0 | | RW |
| 000062H | RLT0 - Reload Register - for writing | | TMRLR0 | W |
| 000062H | RLT0 - Reload Register - for reading | | TMR0 | R |
| 000063H | RLT0 - Reload Register - for writing | | | W |
| 000063H | RLT0 - Reload Register - for reading | | | R |
| 000064H | RLT1 - Timer Control Status Register Low | TMCSRL1 | TMCSR1 | RW |
| 000065H | RLT1 - Timer Control Status Register High | TMCSRH1 | | RW |
| 000066H | RLT1 - Reload Register - for writing | | TMRLR1 | W |
| 000066H | RLT1 - Reload Register - for reading | | TMR1 | R |
| 000067H | RLT1 - Reload Register - for writing | | | W |
| 000067H | RLT1 - Reload Register - for reading | | | R |
| 000068H | RLT2 - Timer Control Status Register Low | TMCSRL2 | TMCSR2 | RW |
| 000069H | RLT2 - Timer Control Status Register High | TMCSRH2 | | RW |
| 00006AH | RLT2 - Reload Register - for writing | | TMRLR2 | W |
| 00006AH | RLT2 - Reload Register - for reading | | TMR2 | R |
| 00006BH | RLT2 - Reload Register - for writing | | | W |
| 00006BH | RLT2 - Reload Register - for reading | | | R |
| 00006CH | RLT3 - Timer Control Status Register Low | TMCSRL3 | TMCSR3 | RW |
| 00006DH | RLT3 - Timer Control Status Register High | TMCSRH3 | | RW |
| 00006EH | RLT3 - Reload Register - for writing | | TMRLR3 | W |
| 00006EH | RLT3 - Reload Register - for reading | | TMR3 | R |
| 00006FH | RLT3 - Reload Register - for writing | | | W |
| 00006FH | RLT3 - Reload Register - for reading | | | R |
| 000070H | RLT6 - Timer Control Status Register Low (dedic. RLT for PPG) | TMCSRL6 | TMCSR6 | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000071H | RLT6 - Timer Control Status Register High (dedic. RLT for PPG) | TMCSRH6 | | RW |
| 000072H | RLT6 - Reload Register (dedic. RLT for PPG) - for writing | | TMRLR6 | W |
| 000072H | RLT6 - Reload Register (dedic. RLT for PPG) - for reading | | TMR6 | R |
| 000073H | RLT6 - Reload Register (dedic. RLT for PPG) - for writing | | | W |
| 000073H | RLT6 - Reload Register (dedic. RLT for PPG) - for reading | | | R |
| 000074H | PPG3-PPG0 - General Control register 1 Low | GCN1L0 | GCN10 | RW |
| 000075H | PPG3-PPG0 - General Control register 1 High | GCN1H0 | | RW |
| 000076H | PPG3-PPG0 - General Control register 2 Low | GCN2L0 | GCN20 | RW |
| 000077H | PPG3-PPG0 - General Control register 2 High | GCN2H0 | | RW |
| 000078H | PPG0 - Timer register | | PTMR0 | R |
| 000079H | PPG0 - Timer register | | | R |
| 00007AH | PPG0 - Period setting register | | PCSR0 | W |
| 00007BH | PPG0 - Period setting register | | | W |
| 00007CH | PPG0 - Duty cycle register | | PDUT0 | W |
| 00007DH | PPG0 - Duty cycle register | | | W |
| 00007EH | PPG0 - Control status register Low | PCNL0 | PCN0 | RW |
| 00007FH | PPG0 - Control status register High | PCNH0 | | RW |
| 000080H | PPG1 - Timer register | | PTMR1 | R |
| 000081H | PPG1 - Timer register | | | R |
| 000082H | PPG1 - Period setting register | | PCSR1 | W |
| 000083H | PPG1 - Period setting register | | | W |
| 000084H | PPG1 - Duty cycle register | | PDUT1 | W |
| 000085H | PPG1 - Duty cycle register | | | W |
| 000086H | PPG1 - Control status register Low | PCNL1 | PCN1 | RW |
| 000087H | PPG1 - Control status register High | PCNH1 | | RW |
| 000088H | PPG2 - Timer register | | PTMR2 | R |
| 000089H | PPG2 - Timer register | | | R |
| 00008AH | PPG2 - Period setting register | | PCSR2 | W |
| 00008BH | PPG2 - Period setting register | | | W |
| 00008CH | PPG2 - Duty cycle register | | PDUT2 | W |
| 00008DH | PPG2 - Duty cycle register | | | W |
| 00008EH | PPG2 - Control status register Low | PCNL2 | PCN2 | RW |
| 00008FH | PPG2 - Control status register High | PCNH2 | | RW |
| 000090H | PPG3 - Timer register | | PTMR3 | R |
| 000091H | PPG3 - Timer register | | | R |
| 000092H | PPG3 - Period setting register | | PCSR3 | W |
| 000093H | PPG3 - Period setting register | | | W |
| 000094H | PPG3 - Duty cycle register | | PDUT3 | W |
| 000095H | PPG3 - Duty cycle register | | | W |
| 000096H | PPG3 - Control status register Low | PCNL3 | PCN3 | RW |
| 000097H | PPG3 - Control status register High | PCNH3 | | RW |
| 000098H | PPG7-PPG4 - General Control register 1 Low | GCN1L1 | GCN11 | RW |
| 000099H | PPG7-PPG4 - General Control register 1 High | GCN1H1 | | RW |
| 00009AH | PPG7-PPG4 - General Control register 2 Low | GCN2L1 | GCN21 | RW |
| 00009BH | PPG7-PPG4 - General Control register 2 High | GCN2H1 | | RW |
| 00009CH | PPG4 - Timer register | | PTMR4 | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00009DH | PPG4 - Timer register | | | R |
| 00009EH | PPG4 - Period setting register | | PCSR4 | W |
| 00009FH | PPG4 - Period setting register | | | W |
| 0000A0H | PPG4 - Duty cycle register | | PDUT4 | W |
| 0000A1H | PPG4 - Duty cycle register | | | W |
| 0000A2H | PPG4 - Control status register Low | PCNL4 | PCN4 | RW |
| 0000A3H | PPG4 - Control status register High | PCNH4 | | RW |
| 0000A4H | PPG5 - Timer register | | PTMR5 | R |
| 0000A5H | PPG5 - Timer register | | | R |
| 0000A6H | PPG5 - Period setting register | | PCSR5 | W |
| 0000A7H | PPG5 - Period setting register | | | W |
| 0000A8H | PPG5 - Duty cycle register | | PDUT5 | W |
| 0000A9H | PPG5 - Duty cycle register | | | W |
| 0000AAH | PPG5 - Control status register Low | PCNL5 | PCN5 | RW |
| 0000ABH | PPG5 - Control status register High | PCNH5 | | RW |
| 0000ACH | I2C0 - Bus Status Register | IBSR0 | | R |
| 0000ADH | I2C0 - Bus Control Register | IBCR0 | | RW |
| 0000AEH | I2C0 - Ten bit Slave address Register Low | ITBAL0 | ITBA0 | RW |
| 0000AFH | I2C0 - Ten bit Slave address Register High | ITBAH0 | | RW |
| 0000B0H | I2C0 - Ten bit Address mask Register Low | ITMKL0 | ITMK0 | RW |
| 0000B1H | I2C0 - Ten bit Address mask Register High | ITMKH0 | | RW |
| 0000B2H | I2C0 - Seven bit Slave address Register | ISBA0 | | RW |
| 0000B3H | I2C0 - Seven bit Address mask Register | ISMK0 | | RW |
| 0000B4H | I2C0 - Data Register | IDAR0 | | RW |
| 0000B5H | I2C0 - Clock Control Register | ICCR0 | | RW |
| 0000B6H | I2C1 - Bus Status Register | IBSR1 | | R |
| 0000B7H | I2C1 - Bus Control Register | IBCR1 | | RW |
| 0000B8H | I2C1 - Ten bit Slave address Register Low | ITBAL1 | ITBA1 | RW |
| 0000B9H | I2C1 - Ten bit Slave address Register High | ITBAH1 | | RW |
| 0000BAH | I2C1 - Ten bit Address mask Register Low | ITMKL1 | ITMK1 | RW |
| 0000BBH | I2C1 - Ten bit Address mask Register High | ITMKH1 | | RW |
| 0000BCH | I2C1 - Seven bit Slave address Register | ISBA1 | | RW |
| 0000BDH | I2C1 - Seven bit Address mask Register | ISMK1 | | RW |
| 0000BEH | I2C1 - Data Register | IDAR1 | | RW |
| 0000BFH | I2C1 - Clock Control Register | ICCR1 | | RW |
| 0000C0H | USART0 - Serial Mode Register | SMR0 | | RW |
| 0000C1H | USART0 - Serial Control Register | SCR0 | | RW |
| 0000C2H | USART0 - TX Register | TDR0 | | W |
| 0000C2H | USART0 - RX Register | RDR0 | | R |
| 0000C3H | USART0 - Serial Status | SSR0 | | RW |
| 0000C4H | USART0 - Control/Com. Register | ECCR0 | | RW |
| 0000C5H | USART0 - Ext. Status Register | ESCR0 | | RW |
| 0000C6H | USART0 - Baud Rate Generator Register Low | BGRL0 | BGR0 | RW |
| 0000C7H | USART0 - Baud Rate Generator Register High | BGRH0 | | RW |
| 0000C8H | USART0 - Extended Serial Interrupt Register | ESIR0 | | RW |
| 0000C9H | Reserved | | | - |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0000CAH | USART1 - Serial Mode Register | SMR1 | | RW |
| 0000CBH | USART1 - Serial Control Register | SCR1 | | RW |
| 0000CCH | USART1 - TX Register | TDR1 | | W |
| 0000CCH | USART1 - RX Register | RDR1 | | R |
| 0000CDH | USART1 - Serial Status | SSR1 | | RW |
| 0000CEH | USART1 - Control/Com. Register | ECCR1 | | RW |
| 0000CFH | USART1 - Ext. Status Register | ESCR1 | | RW |
| 0000D0H | USART1 - Baud Rate Generator Register Low | BGRL1 | BGR1 | RW |
| 0000D1H | USART1 - Baud Rate Generator Register High | BGRH1 | | RW |
| 0000D2H | USART1 - Extended Serial Interrupt Register | ESIR1 | | RW |
| 0000D3H | Reserved | | | - |
| 0000D4H | USART2 - Serial Mode Register | SMR2 | | RW |
| 0000D5H | USART2 - Serial Control Register | SCR2 | | RW |
| 0000D6H | USART2 - TX Register | TDR2 | | W |
| 0000D6H | USART2 - RX Register | RDR2 | | R |
| 0000D7H | USART2 - Serial Status | SSR2 | | RW |
| 0000D8H | USART2 - Control/Com. Register | ECCR2 | | RW |
| 0000D9H | USART2 - Ext. Status Register | ESCR2 | | RW |
| 0000DAH | USART2 - Baud Rate Generator Register Low | BGRL2 | BGR2 | RW |
| 0000DBH | USART2 - Baud Rate Generator Register High | BGRH2 | | RW |
| 0000DCH | USART2 - Extended Serial Interrupt Register | ESIR2 | | RW |
| 0000DDH | Reserved | | | - |
| 0000DEH | USART3 - Serial Mode Register | SMR3 | | RW |
| 0000DFH | USART3 - Serial Control Register | SCR3 | | RW |
| 0000E0H | USART3 - TX Register | TDR3 | | W |
| 0000E0H | USART3 - RX Register | RDR3 | | R |
| 0000E1H | USART3 - Serial Status | SSR3 | | RW |
| 0000E2H | USART3 - Control/Com. Register | ECCR3 | | RW |
| 0000E3H | USART3 - Ext. Status Register | ESCR3 | | RW |
| 0000E4H | USART3 - Baud Rate Generator Register Low | BGRL3 | BGR3 | RW |
| 0000E5H | USART3 - Baud Rate Generator Register High | BGRH3 | | RW |
| 0000E6H | USART3 - Extended Serial Interrupt Register | ESIR3 | | RW |
| 0000E7H-0000EFH | Reserved | | | - |
| 0000F0H-0000FFH | External Bus area | EXTBUS0 | | RW |
| 000100H | DMA0 - Buffer address pointer low byte | BAPL0 | | RW |
| 000101H | DMA0 - Buffer address pointer middle byte | BAPM0 | | RW |
| 000102H | DMA0 - Buffer address pointer high byte | BAPH0 | | RW |
| 000103H | DMA0 - DMA control register | DMACS0 | | RW |
| 000104H | DMA0 - I/O register address pointer low byte | IOAL0 | IOA0 | RW |
| 000105H | DMA0 - I/O register address pointer high byte | IOAH0 | | RW |
| 000106H | DMA0 - Data counter low byte | DCTL0 | DCT0 | RW |
| 000107H | DMA0 - Data counter high byte | DCTH0 | | RW |
| 000108H | DMA1 - Buffer address pointer low byte | BAPL1 | | RW |
| 000109H | DMA1 - Buffer address pointer middle byte | BAPM1 | | RW |
| 00010AH | DMA1 - Buffer address pointer high byte | BAPH1 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00010BH | DMA1 - DMA control register | DMACS1 | | RW |
| 00010CH | DMA1 - I/O register address pointer low byte | IOAL1 | IOA1 | RW |
| 00010DH | DMA1 - I/O register address pointer high byte | IOAH1 | | RW |
| 00010EH | DMA1 - Data counter low byte | DCTL1 | DCT1 | RW |
| 00010FH | DMA1 - Data counter high byte | DCTH1 | | RW |
| 000110H | DMA2 - Buffer address pointer low byte | BAPL2 | | RW |
| 000111H | DMA2 - Buffer address pointer middle byte | BAPM2 | | RW |
| 000112H | DMA2 - Buffer address pointer high byte | BAPH2 | | RW |
| 000113H | DMA2 - DMA control register | DMACS2 | | RW |
| 000114H | DMA2 - I/O register address pointer low byte | IOAL2 | IOA2 | RW |
| 000115H | DMA2 - I/O register address pointer high byte | IOAH2 | | RW |
| 000116H | DMA2 - Data counter low byte | DCTL2 | DCT2 | RW |
| 000117H | DMA2 - Data counter high byte | DCTH2 | | RW |
| 000118H | DMA3 - Buffer address pointer low byte | BAPL3 | | RW |
| 000119H | DMA3 - Buffer address pointer middle byte | BAPM3 | | RW |
| 00011AH | DMA3 - Buffer address pointer high byte | BAPH3 | | RW |
| 00011BH | DMA3 - DMA control register | DMACS3 | | RW |
| 00011CH | DMA3 - I/O register address pointer low byte | IOAL3 | IOA3 | RW |
| 00011DH | DMA3 - I/O register address pointer high byte | IOAH3 | | RW |
| 00011EH | DMA3 - Data counter low byte | DCTL3 | DCT3 | RW |
| 00011FH | DMA3 - Data counter high byte | DCTH3 | | RW |
| 000120H | DMA4 - Buffer address pointer low byte | BAPL4 | | RW |
| 000121H | DMA4 - Buffer address pointer middle byte | BAPM4 | | RW |
| 000122H | DMA4 - Buffer address pointer high byte | BAPH4 | | RW |
| 000123H | DMA4 - DMA control register | DMACS4 | | RW |
| 000124H | DMA4 - I/O register address pointer low byte | IOAL4 | IOA4 | RW |
| 000125H | DMA4 - I/O register address pointer high byte | IOAH4 | | RW |
| 000126H | DMA4 - Data counter low byte | DCTL4 | DCT4 | RW |
| 000127H | DMA4 - Data counter high byte | DCTH4 | | RW |
| 000128H | DMA5 - Buffer address pointer low byte | BAPL5 | | RW |
| 000129H | DMA5 - Buffer address pointer middle byte | BAPM5 | | RW |
| 00012AH | DMA5 - Buffer address pointer high byte | BAPH5 | | RW |
| 00012BH | DMA5 - DMA control register | DMACS5 | | RW |
| 00012CH | DMA5 - I/O register address pointer low byte | IOAL5 | IOA5 | RW |
| 00012DH | DMA5 - I/O register address pointer high byte | IOAH5 | | RW |
| 00012EH | DMA5 - Data counter low byte | DCTL5 | DCT5 | RW |
| 00012FH | DMA5 - Data counter high byte | DCTH5 | | RW |
| 000130H | DMA6 - Buffer address pointer low byte | BAPL6 | | RW |
| 000131H | DMA6 - Buffer address pointer middle byte | BAPM6 | | RW |
| 000132H | DMA6 - Buffer address pointer high byte | BAPH6 | | RW |
| 000133H | DMA6 - DMA control register | DMACS6 | | RW |
| 000134H | DMA6 - I/O register address pointer low byte | IOAL6 | IOA6 | RW |
| 000135H | DMA6 - I/O register address pointer high byte | IOAH6 | | RW |
| 000136H | DMA6 - Data counter low byte | DCTL6 | DCT6 | RW |
| 000137H | DMA6 - Data counter high byte | DCTH6 | | RW |
| 000138H | DMA7 - Buffer address pointer low byte | BAPL7 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000139H | DMA7 - Buffer address pointer middle byte | BAPM7 | | RW |
| 00013AH | DMA7 - Buffer address pointer high byte | BAPH7 | | RW |
| 00013BH | DMA7 - DMA control register | DMACS7 | | RW |
| 00013CH | DMA7 - I/O register address pointer low byte | IOAL7 | IOA7 | RW |
| 00013DH | DMA7 - I/O register address pointer high byte | IOAH7 | | RW |
| 00013EH | DMA7 - Data counter low byte | DCTL7 | DCT7 | RW |
| 00013FH | DMA7 - Data counter high byte | DCTH7 | | RW |
| 000140H | DMA8 - Buffer address pointer low byte | BAPL8 | | RW |
| 000141H | DMA8 - Buffer address pointer middle byte | BAPM8 | | RW |
| 000142H | DMA8 - Buffer address pointer high byte | BAPH8 | | RW |
| 000143H | DMA8 - DMA control register | DMACS8 | | RW |
| 000144H | DMA8 - I/O register address pointer low byte | IOAL8 | IOA8 | RW |
| 000145H | DMA8 - I/O register address pointer high byte | IOAH8 | | RW |
| 000146H | DMA8 - Data counter low byte | DCTL8 | DCT8 | RW |
| 000147H | DMA8 - Data counter high byte | DCTH8 | | RW |
| 000148H | DMA9 - Buffer address pointer low byte | BAPL9 | | RW |
| 000149H | DMA9 - Buffer address pointer middle byte | BAPM9 | | RW |
| 00014AH | DMA9 - Buffer address pointer high byte | BAPH9 | | RW |
| 00014BH | DMA9 - DMA control register | DMACS9 | | RW |
| 00014CH | DMA9 - I/O register address pointer low byte | IOAL9 | IOA9 | RW |
| 00014DH | DMA9 - I/O register address pointer high byte | IOAH9 | | RW |
| 00014EH | DMA9 - Data counter low byte | DCTL9 | DCT9 | RW |
| 00014FH | DMA9 - Data counter high byte | DCTH9 | | RW |
| 000150H | DMA10 - Buffer address pointer low byte | BAPL10 | | RW |
| 000151H | DMA10 - Buffer address pointer middle byte | BAPM10 | | RW |
| 000152H | DMA10 - Buffer address pointer high byte | BAPH10 | | RW |
| 000153H | DMA10 - DMA control register | DMACS10 | | RW |
| 000154H | DMA10 - I/O register address pointer low byte | IOAL10 | IOA10 | RW |
| 000155H | DMA10 - I/O register address pointer high byte | IOAH10 | | RW |
| 000156H | DMA10 - Data counter low byte | DCTL10 | DCT10 | RW |
| 000157H | DMA10 - Data counter high byte | DCTH10 | | RW |
| 000158H | DMA11 - Buffer address pointer low byte | BAPL11 | | RW |
| 000159H | DMA11 - Buffer address pointer middle byte | BAPM11 | | RW |
| 00015AH | DMA11 - Buffer address pointer high byte | BAPH11 | | RW |
| 00015BH | DMA11 - DMA control register | DMACS11 | | RW |
| 00015CH | DMA11 - I/O register address pointer low byte | IOAL11 | IOA11 | RW |
| 00015DH | DMA11 - I/O register address pointer high byte | IOAH11 | | RW |
| 00015EH | DMA11 - Data counter low byte | DCTL11 | DCT11 | RW |
| 00015FH | DMA11 - Data counter high byte | DCTH11 | | RW |
| 000160H | DMA12 - Buffer address pointer low byte | BAPL12 | | RW |
| 000161H | DMA12 - Buffer address pointer middle byte | BAPM12 | | RW |
| 000162H | DMA12 - Buffer address pointer high byte | BAPH12 | | RW |
| 000163H | DMA12 - DMA control register | DMACS12 | | RW |
| 000164H | DMA12 - I/O register address pointer low byte | IOAL12 | IOA12 | RW |
| 000165H | DMA12 - I/O register address pointer high byte | IOAH12 | | RW |
| 000166H | DMA12 - Data counter low byte | DCTL12 | DCT12 | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000167H | DMA12 - Data counter high byte | DCTH12 | | RW |
| 000168H | DMA13 - Buffer address pointer low byte | BAPL13 | | RW |
| 000169H | DMA13 - Buffer address pointer middle byte | BAPM13 | | RW |
| 00016AH | DMA13 - Buffer address pointer high byte | BAPH13 | | RW |
| 00016BH | DMA13 - DMA control register | DMACS13 | | RW |
| 00016CH | DMA13 - I/O register address pointer low byte | IOAL13 | IOA13 | RW |
| 00016DH | DMA13 - I/O register address pointer high byte | IOAH13 | | RW |
| 00016EH | DMA13 - Data counter low byte | DCTL13 | DCT13 | RW |
| 00016FH | DMA13 - Data counter high byte | DCTH13 | | RW |
| 000170H | DMA14 - Buffer address pointer low byte | BAPL14 | | RW |
| 000171H | DMA14 - Buffer address pointer middle byte | BAPM14 | | RW |
| 000172H | DMA14 - Buffer address pointer high byte | BAPH14 | | RW |
| 000173H | DMA14 - DMA control register | DMACS14 | | RW |
| 000174H | DMA14 - I/O register address pointer low byte | IOAL14 | IOA14 | RW |
| 000175H | DMA14 - I/O register address pointer high byte | IOAH14 | | RW |
| 000176H | DMA14 - Data counter low byte | DCTL14 | DCT14 | RW |
| 000177H | DMA14 - Data counter high byte | DCTH14 | | RW |
| 000178H | DMA15 - Buffer address pointer low byte | BAPL15 | | RW |
| 000179H | DMA15 - Buffer address pointer middle byte | BAPM15 | | RW |
| 00017AH | DMA15 - Buffer address pointer high byte | BAPH15 | | RW |
| 00017BH | DMA15 - DMA control register | DMACS15 | | RW |
| 00017CH | DMA15 - I/O register address pointer low byte | IOAL15 | IOA15 | RW |
| 00017DH | DMA15 - I/O register address pointer high byte | IOAH15 | | RW |
| 00017EH | DMA15 - Data counter low byte | DCTL15 | DCT15 | RW |
| 00017FH | DMA15 - Data counter high byte | DCTH15 | | RW |
| 000180H-00037FH | CPU - General Purpose registers (RAM access) | GPR_RAM | | RW |
| 000380H | DMA0 - Interrupt select | DISEL0 | | RW |
| 000381H | DMA1 - Interrupt select | DISEL1 | | RW |
| 000382H | DMA2 - Interrupt select | DISEL2 | | RW |
| 000383H | DMA3 - Interrupt select | DISEL3 | | RW |
| 000384H | DMA4 - Interrupt select | DISEL4 | | RW |
| 000385H | DMA5 - Interrupt select | DISEL5 | | RW |
| 000386H | DMA6 - Interrupt select | DISEL6 | | RW |
| 000387H | DMA7 - Interrupt select | DISEL7 | | RW |
| 000388H | DMA8 - Interrupt select | DISEL8 | | RW |
| 000389H | DMA9 - Interrupt select | DISEL9 | | RW |
| 00038AH | DMA10 - Interrupt select | DISEL10 | | RW |
| 00038BH | DMA11 - Interrupt select | DISEL11 | | RW |
| 00038CH | DMA12 - Interrupt select | DISEL12 | | RW |
| 00038DH | DMA13 - Interrupt select | DISEL13 | | RW |
| 00038EH | DMA14 - Interrupt select | DISEL14 | | RW |
| 00038FH | DMA15 - Interrupt select | DISEL15 | | RW |
| 000390H | DMA - Status register low byte | DSRL | DSR | RW |
| 000391H | DMA - Status register high byte | DSRH | | RW |
| 000392H | DMA - Stop status register low byte | DSSRL | DSSR | RW |
| 000393H | DMA - Stop status register high byte | DSSRH | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000394H | DMA - Enable register low byte | DERL | DER | RW |
| 000395H | DMA - Enable register high byte | DERH | | RW |
| 000396H-00039FH | Reserved | | | - |
| 0003A0H | Interrupt level register | ILR | ICR | RW |
| 0003A1H | Interrupt index register | IDX | | RW |
| 0003A2H | Interrupt vector table base register Low | TBRL | TBR | RW |
| 0003A3H | Interrupt vector table base register High | TBRH | | RW |
| 0003A4H | Delayed Interrupt register | DIRR | | RW |
| 0003A5H | Non Maskable Interrupt register | NMI | | RW |
| 0003A6H-0003ABH | Reserved | | | - |
| 0003ACH | EDSU communication interrupt selection Low | EDSU2L | EDSU2 | RW |
| 0003ADH | EDSU communication interrupt selection High | EDSU2H | | RW |
| 0003AEH | ROM mirror control register | ROMM | | RW |
| 0003AFH | EDSU configuration register | EDSU | | RW |
| 0003B0H | Memory patch control/status register ch 0/1 | | PFCS0 | RW |
| 0003B1H | Memory patch control/status register ch 0/1 | | | RW |
| 0003B2H | Memory patch control/status register ch 2/3 | | PFCS1 | RW |
| 0003B3H | Memory patch control/status register ch 2/3 | | | RW |
| 0003B4H | Memory patch control/status register ch 4/5 | | PFCS2 | RW |
| 0003B5H | Memory patch control/status register ch 4/5 | | | RW |
| 0003B6H | Memory patch control/status register ch 6/7 | | PFCS3 | RW |
| 0003B7H | Memory patch control/status register ch 6/7 | | | RW |
| 0003B8H | Memory Patch function - Patch address 0 low | PFAL0 | | RW |
| 0003B9H | Memory Patch function - Patch address 0 middle | PFAM0 | | RW |
| 0003BAH | Memory Patch function - Patch address 0 high | PFAH0 | | RW |
| 0003BBH | Memory Patch function - Patch address 1 low | PFAL1 | | RW |
| 0003BCH | Memory Patch function - Patch address 1 middle | PFAM1 | | RW |
| 0003BDH | Memory Patch function - Patch address 1 high | PFAH1 | | RW |
| 0003BEH | Memory Patch function - Patch address 2 low | PFAL2 | | RW |
| 0003BFH | Memory Patch function - Patch address 2 middle | PFAM2 | | RW |
| 0003C0H | Memory Patch function - Patch address 2 high | PFAH2 | | RW |
| 0003C1H | Memory Patch function - Patch address 3 low | PFAL3 | | RW |
| 0003C2H | Memory Patch function - Patch address 3 middle | PFAM3 | | RW |
| 0003C3H | Memory Patch function - Patch address 3 high | PFAH3 | | RW |
| 0003C4H | Memory Patch function - Patch address 4 low | PFAL4 | | RW |
| 0003C5H | Memory Patch function - Patch address 4 middle | PFAM4 | | RW |
| 0003C6H | Memory Patch function - Patch address 4 high | PFAH4 | | RW |
| 0003C7H | Memory Patch function - Patch address 5 low | PFAL5 | | RW |
| 0003C8H | Memory Patch function - Patch address 5 middle | PFAM5 | | RW |
| 0003C9H | Memory Patch function - Patch address 5 high | PFAH5 | | RW |
| 0003CAH | Memory Patch function - Patch address 6 low | PFAL6 | | RW |
| 0003CBH | Memory Patch function - Patch address 6 middle | PFAM6 | | RW |
| 0003CCH | Memory Patch function - Patch address 6 high | PFAH6 | | RW |
| 0003CDH | Memory Patch function - Patch address 7 low | PFAL7 | | RW |
| 0003CEH | Memory Patch function - Patch address 7 middle | PFAM7 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0003CFH | Memory Patch function - Patch address 7 high | PFAH7 | | RW |
| 0003D0H | Memory Patch function - Patch data 0 Low | PFDL0 | PFD0 | RW |
| 0003D1H | Memory Patch function - Patch data 0 High | PFDH0 | | RW |
| 0003D2H | Memory Patch function - Patch data 1 Low | PFDL1 | PFD1 | RW |
| 0003D3H | Memory Patch function - Patch data 1 High | PFDH1 | | RW |
| 0003D4H | Memory Patch function - Patch data 2 Low | PFDL2 | PFD2 | RW |
| 0003D5H | Memory Patch function - Patch data 2 High | PFDH2 | | RW |
| 0003D6H | Memory Patch function - Patch data 3 Low | PFDL3 | PFD3 | RW |
| 0003D7H | Memory Patch function - Patch data 3 High | PFDH3 | | RW |
| 0003D8H | Memory Patch function - Patch data 4 Low | PFDL4 | PFD4 | RW |
| 0003D9H | Memory Patch function - Patch data 4 High | PFDH4 | | RW |
| 0003DAH | Memory Patch function - Patch data 5 Low | PFDL5 | PFD5 | RW |
| 0003DBH | Memory Patch function - Patch data 5 High | PFDH5 | | RW |
| 0003DCH | Memory Patch function - Patch data 6 Low | PFDL6 | PFD6 | RW |
| 0003DDH | Memory Patch function - Patch data 6 High | PFDH6 | | RW |
| 0003DEH | Memory Patch function - Patch data 7 Low | PFDL7 | PFD7 | RW |
| 0003DFH | Memory Patch function - Patch data 7 High | PFDH7 | | RW |
| 0003E0H-0003F0H | Reserved | | | - |
| 0003F1H | Memory Control Status Register A | MCSRA | | RW |
| 0003F2H | Memory Timing Configuration Register A Low | MTCRAL | MTCRA | RW |
| 0003F3H-0003F5H | Reserved | | | - |
| 0003F6H | Memory Timing Configuration Register B Low | MTCRBL | MTCRB | RW |
| 0003F7H-0003FFH | Reserved | | | - |
| 000400H | Standby Mode control register | SMCR | | RW |
| 000401H | Clock select register | CKSR | | RW |
| 000402H | Clock Stabilisation select register | CKSSR | | RW |
| 000403H | Clock monitor register | CKMR | | R |
| 000404H | Clock Frequency control register Low | CKFCRL | CKFCR | RW |
| 000405H | Clock Frequency control register High | CKFCRH | | RW |
| 000406H | PLL Control register Low | PLLCRL | PLLCR | RW |
| 000407H | PLL Control register High | PLLCRH | | RW |
| 000408H | RC clock timer control register | RCTCR | | RW |
| 000409H | Main clock timer control register | MCTCR | | RW |
| 00040AH | Sub clock timer control register | SCTCR | | RW |
| 00040BH | Reset cause and clock status register with clear function | RCCSRC | | R |
| 00040CH | Reset configuration register | RCR | | RW |
| 00040DH | Reset cause and clock status register | RCCSR | | R |
| 00040EH | Watch dog timer configuration register | WDTC | | RW |
| 00040FH | Watch dog timer clear pattern register | WDTCP | | W |
| 000410H-000414H | Reserved | | | - |
| 000415H | Clock output activation register | COAR | | RW |
| 000416H | Clock output configuration register 0 | COCR0 | | RW |
| 000417H | Clock output configuration register 1 | COCR1 | | RW |
| 000418H | Clock Modulator control register | CMCR | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000419H | Reserved | | | - |
| 00041AH | Clock Modulator Parameter register Low | CMPRL | CMPR | RW |
| 00041BH | Clock Modulator Parameter register High | CMPRH | | RW |
| 00041CH-00042BH | Reserved | | | - |
| 00042CH | Voltage Regulator Control register | VRCR | | RW |
| 00042DH | Clock Input and LVD Control Register | CILCR | | RW |
| 00042EH-00042FH | Reserved | | | - |
| 000430H | I/O Port P00 - Data Direction Register | DDR00 | | RW |
| 000431H | I/O Port P01 - Data Direction Register | DDR01 | | RW |
| 000432H | I/O Port P02 - Data Direction Register | DDR02 | | RW |
| 000433H | I/O Port P03 - Data Direction Register | DDR03 | | RW |
| 000434H | I/O Port P04 - Data Direction Register | DDR04 | | RW |
| 000435H | I/O Port P05 - Data Direction Register | DDR05 | | RW |
| 000436H | I/O Port P06 - Data Direction Register | DDR06 | | RW |
| 000437H | I/O Port P07 - Data Direction Register | DDR07 | | RW |
| 000438H | I/O Port P08 - Data Direction Register | DDR08 | | RW |
| 000439H | I/O Port P09 - Data Direction Register | DDR09 | | RW |
| 00043AH | I/O Port P10 - Data Direction Register | DDR10 | | RW |
| 00043BH | I/O Port P11 - Data Direction Register | DDR11 | | RW |
| 00043CH | I/O Port P12 - Data Direction Register | DDR12 | | RW |
| 00043DH | I/O Port P13 - Data Direction Register | DDR13 | | RW |
| 00043EH | I/O Port P14 - Data Direction Register | DDR14 | | RW |
| 00043FH | I/O Port P15 - Data Direction Register | DDR15 | | RW |
| 000440H | I/O Port P16 - Data Direction Register | DDR16 | | RW |
| 000441H | I/O Port P17 - Data Direction Register | DDR17 | | RW |
| 000442H-000443H | Reserved | | | - |
| 000444H | I/O Port P00 - Port Input Enable Register | PIER00 | | RW |
| 000445H | I/O Port P01 - Port Input Enable Register | PIER01 | | RW |
| 000446H | I/O Port P02 - Port Input Enable Register | PIER02 | | RW |
| 000447H | I/O Port P03 - Port Input Enable Register | PIER03 | | RW |
| 000448H | I/O Port P04 - Port Input Enable Register | PIER04 | | RW |
| 000449H | I/O Port P05 - Port Input Enable Register | PIER05 | | RW |
| 00044AH | I/O Port P06 - Port Input Enable Register | PIER06 | | RW |
| 00044BH | I/O Port P07 - Port Input Enable Register | PIER07 | | RW |
| 00044CH | I/O Port P08 - Port Input Enable Register | PIER08 | | RW |
| 00044DH | I/O Port P09 - Port Input Enable Register | PIER09 | | RW |
| 00044EH | I/O Port P10 - Port Input Enable Register | PIER10 | | RW |
| 00044FH | I/O Port P11 - Port Input Enable Register | PIER11 | | RW |
| 000450H | I/O Port P12 - Port Input Enable Register | PIER12 | | RW |
| 000451H | I/O Port P13 - Port Input Enable Register | PIER13 | | RW |
| 000452H | I/O Port P14 - Port Input Enable Register | PIER14 | | RW |
| 000453H | I/O Port P15 - Port Input Enable Register | PIER15 | | RW |
| 000454H | I/O Port P16 - Port Input Enable Register | PIER16 | | RW |
| 000455H | I/O Port P17 - Port Input Enable Register | PIER17 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000456H-000457H | Reserved | | | - |
| 000458H | I/O Port P00 - Port Input Level Register | PILR00 | | RW |
| 000459H | I/O Port P01 - Port Input Level Register | PILR01 | | RW |
| 00045AH | I/O Port P02 - Port Input Level Register | PILR02 | | RW |
| 00045BH | I/O Port P03 - Port Input Level Register | PILR03 | | RW |
| 00045CH | I/O Port P04 - Port Input Level Register | PILR04 | | RW |
| 00045DH | I/O Port P05 - Port Input Level Register | PILR05 | | RW |
| 00045EH | I/O Port P06 - Port Input Level Register | PILR06 | | RW |
| 00045FH | I/O Port P07 - Port Input Level Register | PILR07 | | RW |
| 000460H | I/O Port P08 - Port Input Level Register | PILR08 | | RW |
| 000461H | I/O Port P09 - Port Input Level Register | PILR09 | | RW |
| 000462H | I/O Port P10 - Port Input Level Register | PILR10 | | RW |
| 000463H | I/O Port P11 - Port Input Level Register | PILR11 | | RW |
| 000464H | I/O Port P12 - Port Input Level Register | PILR12 | | RW |
| 000465H | I/O Port P13 - Port Input Level Register | PILR13 | | RW |
| 000466H | I/O Port P14 - Port Input Level Register | PILR14 | | RW |
| 000467H | I/O Port P15 - Port Input Level Register | PILR15 | | RW |
| 000468H | I/O Port P16 - Port Input Level Register | PILR16 | | RW |
| 000469H | I/O Port P17 - Port Input Level Register | PILR17 | | RW |
| 00046AH-00046BH | Reserved | | | - |
| 00046CH | I/O Port P00 - Extended Port Input Level Register | EPILR00 | | RW |
| 00046DH | I/O Port P01 - Extended Port Input Level Register | EPILR01 | | RW |
| 00046EH | I/O Port P02 - Extended Port Input Level Register | EPILR02 | | RW |
| 00046FH | I/O Port P03 - Extended Port Input Level Register | EPILR03 | | RW |
| 000470H | I/O Port P04 - Extended Port Input Level Register | EPILR04 | | RW |
| 000471H | I/O Port P05 - Extended Port Input Level Register | EPILR05 | | RW |
| 000472H | I/O Port P06 - Extended Port Input Level Register | EPILR06 | | RW |
| 000473H | I/O Port P07 - Extended Port Input Level Register | EPILR07 | | RW |
| 000474H | I/O Port P08 - Extended Port Input Level Register | EPILR08 | | RW |
| 000475H | I/O Port P09 - Extended Port Input Level Register | EPILR09 | | RW |
| 000476H | I/O Port P10 - Extended Port Input Level Register | EPILR10 | | RW |
| 000477H | I/O Port P11 - Extended Port Input Level Register | EPILR11 | | RW |
| 000478H | I/O Port P12 - Extended Port Input Level Register | EPILR12 | | RW |
| 000479H | I/O Port P13 - Extended Port Input Level Register | EPILR13 | | RW |
| 00047AH | I/O Port P14 - Extended Port Input Level Register | EPILR14 | | RW |
| 00047BH | I/O Port P15 - Extended Port Input Level Register | EPILR15 | | RW |
| 00047CH | I/O Port P16 - Extended Port Input Level Register | EPILR16 | | RW |
| 00047DH | I/O Port P17 - Extended Port Input Level Register | EPILR17 | | RW |
| 00047EH-00047FH | Reserved | | | - |
| 000480H | I/O Port P00 - Port Output Drive Register | PODR00 | | RW |
| 000481H | I/O Port P01 - Port Output Drive Register | PODR01 | | RW |
| 000482H | I/O Port P02 - Port Output Drive Register | PODR02 | | RW |
| 000483H | I/O Port P03 - Port Output Drive Register | PODR03 | | RW |
| 000484H | I/O Port P04 - Port Output Drive Register | PODR04 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000485H | I/O Port P05 - Port Output Drive Register | PODR05 | | RW |
| 000486H | I/O Port P06 - Port Output Drive Register | PODR06 | | RW |
| 000487H | I/O Port P07 - Port Output Drive Register | PODR07 | | RW |
| 000488H | I/O Port P08 - Port Output Drive Register | PODR08 | | RW |
| 000489H | I/O Port P09 - Port Output Drive Register | PODR09 | | RW |
| 00048AH | I/O Port P10 - Port Output Drive Register | PODR10 | | RW |
| 00048BH | I/O Port P11 - Port Output Drive Register | PODR11 | | RW |
| 00048CH | I/O Port P12 - Port Output Drive Register | PODR12 | | RW |
| 00048DH | I/O Port P13 - Port Output Drive Register | PODR13 | | RW |
| 00048EH | I/O Port P14 - Port Output Drive Register | PODR14 | | RW |
| 00048FH | I/O Port P15 - Port Output Drive Register | PODR15 | | RW |
| 000490H | I/O Port P16 - Port Output Drive Register | PODR16 | | RW |
| 000491H | I/O Port P17 - Port Output Drive Register | PODR17 | | RW |
| 000492H-00049BH | Reserved | | | - |
| 00049CH | I/O Port P08 - Port High Drive Register | PHDR08 | | RW |
| 00049DH | I/O Port P09 - Port High Drive Register | PHDR09 | | RW |
| 00049EH | I/O Port P10 - Port High Drive Register | PHDR10 | | RW |
| 00049FH-0004A7H | Reserved | | | - |
| 0004A8H | I/O Port P00 - Pull-Up resistor Control Register | PUCR00 | | RW |
| 0004A9H | I/O Port P01 - Pull-Up resistor Control Register | PUCR01 | | RW |
| 0004AAH | I/O Port P02 - Pull-Up resistor Control Register | PUCR02 | | RW |
| 0004ABH | I/O Port P03 - Pull-Up resistor Control Register | PUCR03 | | RW |
| 0004ACH | I/O Port P04 - Pull-Up resistor Control Register | PUCR04 | | RW |
| 0004ADH | I/O Port P05 - Pull-Up resistor Control Register | PUCR05 | | RW |
| 0004AEH | I/O Port P06 - Pull-Up resistor Control Register | PUCR06 | | RW |
| 0004AFH | I/O Port P07 - Pull-Up resistor Control Register | PUCR07 | | RW |
| 0004B0H | I/O Port P08 - Pull-Up resistor Control Register | PUCR08 | | RW |
| 0004B1H | I/O Port P09 - Pull-Up resistor Control Register | PUCR09 | | RW |
| 0004B2H | I/O Port P10 - Pull-Up resistor Control Register | PUCR10 | | RW |
| 0004B3H | I/O Port P11 - Pull-Up resistor Control Register | PUCR11 | | RW |
| 0004B4H | I/O Port P12 - Pull-Up resistor Control Register | PUCR12 | | RW |
| 0004B5H | I/O Port P13 - Pull-Up resistor Control Register | PUCR13 | | RW |
| 0004B6H | I/O Port P14 - Pull-Up resistor Control Register | PUCR14 | | RW |
| 0004B7H | I/O Port P15 - Pull-Up resistor Control Register | PUCR15 | | RW |
| 0004B8H | I/O Port P16 - Pull-Up resistor Control Register | PUCR16 | | RW |
| 0004B9H | I/O Port P17 - Pull-Up resistor Control Register | PUCR17 | | RW |
| 0004BAH-0004BBH | Reserved | | | - |
| 0004BCH | I/O Port P00 - External Pin State Register | EPSR00 | | R |
| 0004BDH | I/O Port P01 - External Pin State Register | EPSR01 | | R |
| 0004BEH | I/O Port P02 - External Pin State Register | EPSR02 | | R |
| 0004BFH | I/O Port P03 - External Pin State Register | EPSR03 | | R |
| 0004C0H | I/O Port P04 - External Pin State Register | EPSR04 | | R |
| 0004C1H | I/O Port P05 - External Pin State Register | EPSR05 | | R |
| 0004C2H | I/O Port P06 - External Pin State Register | EPSR06 | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0004C3H | I/O Port P07 - External Pin State Register | EPSR07 | | R |
| 0004C4H | I/O Port P08 - External Pin State Register | EPSR08 | | R |
| 0004C5H | I/O Port P09 - External Pin State Register | EPSR09 | | R |
| 0004C6H | I/O Port P10 - External Pin State Register | EPSR10 | | R |
| 0004C7H | I/O Port P11 - External Pin State Register | EPSR11 | | R |
| 0004C8H | I/O Port P12 - External Pin State Register | EPSR12 | | R |
| 0004C9H | I/O Port P13 - External Pin State Register | EPSR13 | | R |
| 0004CAH | I/O Port P14 - External Pin State Register | EPSR14 | | R |
| 0004CBH | I/O Port P15 - External Pin State Register | EPSR15 | | R |
| 0004CCH | I/O Port P16 - External Pin State Register | EPSR16 | | R |
| 0004CDH | I/O Port P17 - External Pin State Register | EPSR17 | | R |
| 0004CEH-0004CFH | Reserved | | | - |
| 0004D0H | ADC analog input enable register 0 | ADER0 | | RW |
| 0004D1H | ADC analog input enable register 1 | ADER1 | | RW |
| 0004D2H | ADC analog input enable register 2 | ADER2 | | RW |
| 0004D3H | ADC analog input enable register 3 | ADER3 | | RW |
| 0004D4H | ADC analog input enable register 4 | ADER4 | | RW |
| 0004D5H | Reserved | | | - |
| 0004D6H | Peripheral Resource Relocation Register 0 | PRRR0 | | RW |
| 0004D7H | Peripheral Resource Relocation Register 1 | PRRR1 | | RW |
| 0004D8H | Peripheral Resource Relocation Register 2 | PRRR2 | | RW |
| 0004D9H | Peripheral Resource Relocation Register 3 | PRRR3 | | RW |
| 0004DAH | Peripheral Resource Relocation Register 4 | PRRR4 | | RW |
| 0004DBH | Peripheral Resource Relocation Register 5 | PRRR5 | | RW |
| 0004DCH | Peripheral Resource Relocation Register 6 | PRRR6 | | RW |
| 0004DDH | Peripheral Resource Relocation Register 7 | PRRR7 | | RW |
| 0004DEH | Peripheral Resource Relocation Register 8 | PRRR8 | | RW |
| 0004DFH | Peripheral Resource Relocation Register 9 | PRRR9 | | RW |
| 0004E0H | RTC - Sub Second Register L | WTBRL0 | WTBR0 | RW |
| 0004E1H | RTC - Sub Second Register M | WTBRH0 | | RW |
| 0004E2H | RTC - Sub-Second Register H | WTBR1 | | RW |
| 0004E3H | RTC - Second Register | WTSR | | RW |
| 0004E4H | RTC - Minutes | WTMR | | RW |
| 0004E5H | RTC - Hour | WTHR | | RW |
| 0004E6H | RTC - Timer Control Extended Register | WTCER | | RW |
| 0004E7H | RTC - Clock select register | WTCKSR | | RW |
| 0004E8H | RTC - Timer Control Register Low | WTCRL | WTCR | RW |
| 0004E9H | RTC - Timer Control Register High | WTCRH | | RW |
| 0004EAH | CAL - Calibration unit Control register | CUCR | | RW |
| 0004EBH | Reserved | | | - |
| 0004ECH | CAL - Duration Timer Data Register Low | CUTDL | CUTD | RW |
| 0004EDH | CAL - Duration Timer Data Register High | CUTDH | | RW |
| 0004EEH | CAL - Calibration Timer Register 2 Low | CUTR2L | CUTR2 | R |
| 0004EFH | CAL - Calibration Timer Register 2 High | CUTR2H | | R |
| 0004F0H | CAL - Calibration Timer Register 1 Low | CUTR1L | CUTR1 | R |
| 0004F1H | CAL - Calibration Timer Register 1 High | CUTR1H | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0004F2H | RLT4 - Timer Control Status Register Low | TMCSRL4 | TMCSR4 | RW |
| 0004F3H | RLT4 - Timer Control Status Register High | TMCSRH4 | | RW |
| 0004F4H | RLT4 - Reload Register - for writing | | TMRLR4 | W |
| 0004F4H | RLT4 - Reload Register - for reading | | TMR4 | R |
| 0004F5H | RLT4 - Reload Register - for writing | | | W |
| 0004F5H | RLT4 - Reload Register - for reading | | | R |
| 0004F6H | RLT5 - Timer Control Status Register Low | TMCSRL5 | TMCSR5 | RW |
| 0004F7H | RLT5 - Timer Control Status Register High | TMCSRH5 | | RW |
| 0004F8H | RLT5 - Reload Register - for writing | | TMRLR5 | W |
| 0004F8H | RLT5 - Reload Register - for reading | | TMR5 | R |
| 0004F9H | RLT5 - Reload Register - for writing | | | W |
| 0004F9H | RLT5 - Reload Register - for reading | | | R |
| 0004FAH | RLT - Timer input select (for Cascading) | TMISR | | RW |
| 0004FBH-0004FFH | Reserved | | | - |
| 000500H | FRT2 - Data register of free-running timer | | TCDT2 | RW |
| 000501H | FRT2 - Data register of free-running timer | | | RW |
| 000502H | FRT2 - Control status register of free-running timer Low | TCCSL2 | TCCS2 | RW |
| 000503H | FRT2 - Control status register of free-running timer High | TCCSH2 | | RW |
| 000504H | FRT3 - Data register of free-running timer | | TCDT3 | RW |
| 000505H | FRT3 - Data register of free-running timer | | | RW |
| 000506H | FRT3 - Control status register of free-running timer Low | TCCSL3 | TCCS3 | RW |
| 000507H | FRT3 - Control status register of free-running timer High | TCCSH3 | | RW |
| 000508H | OCU8 - Output Compare Control Status | OCS8 | | RW |
| 000509H | OCU9 - Output Compare Control Status | OCS9 | | RW |
| 00050AH | OCU8 - Compare Register | | OCCP8 | RW |
| 00050BH | OCU8 - Compare Register | | | RW |
| 00050CH | OCU9 - Compare Register | | OCCP9 | RW |
| 00050DH | OCU9 - Compare Register | | | RW |
| 00050EH | OCU10 - Output Compare Control Status | OCS10 | | RW |
| 00050FH | OCU11 - Output Compare Control Status | OCS11 | | RW |
| 000510H | OCU10 - Compare Register | | OCCP10 | RW |
| 000511H | OCU10 - Compare Register | | | RW |
| 000512H | OCU11 - Compare Register | | OCCP11 | RW |
| 000513H | OCU11 - Compare Register | | | RW |
| 000514H | ICU8/ICU9 - Control Status Register | ICS89 | | RW |
| 000515H | ICU8/ICU9 - Edge Register | ICE89 | | RW |
| 000516H | ICU8 - Capture Register Low | IPCPL8 | IPCP8 | R |
| 000517H | ICU8 - Capture Register High | IPCPH8 | | R |
| 000518H | ICU9 - Capture Register Low | IPCPL9 | IPCP9 | R |
| 000519H | ICU9 - Capture Register High | IPCPH9 | | R |
| 00051AH | ICU10/ICU11 - Control Status Register | ICS1011 | | RW |
| 00051BH | ICU10/ICU11 - Edge Register | ICE1011 | | RW |
| 00051CH | ICU10 - Capture Register Low | IPCPL10 | IPCP10 | R |
| 00051DH | ICU10 - Capture Register High | IPCPH10 | | R |
| 00051EH | ICU11 - Capture Register Low | IPCPL11 | IPCP11 | R |
| 00051FH | ICU11 - Capture Register High | IPCPH11 | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000520H | USART4 - Serial Mode Register | SMR4 | | RW |
| 000521H | USART4 - Serial Control Register | SCR4 | | RW |
| 000522H | USART4 - TX Register | TDR4 | | W |
| 000522H | USART4 - RX Register | RDR4 | | R |
| 000523H | USART4 - Serial Status | SSR4 | | RW |
| 000524H | USART4 - Control/Com. Register (internal) | ECCR4 | | RW |
| 000525H | USART4 - Ext. Status Register | ESCR4 | | RW |
| 000526H | USART4 - Baud Rate Generator Register Low | BGRL4 | BGR4 | RW |
| 000527H | USART4 - Baud Rate Generator Register High | BGRH4 | | RW |
| 000528H | USART4 - Extended Serial Interrupt Register | ESIR4 | | RW |
| 000529H | Reserved | | | - |
| 00052AH | USART5 - Serial Mode Register | SMR5 | | RW |
| 00052BH | USART5 - Serial Control Register | SCR5 | | RW |
| 00052CH | USART5 - RX Register | TDR5 | | W |
| 00052CH | USART5 - TX Register | RDR5 | | R |
| 00052DH | USART5 - Serial Status | SSR5 | | RW |
| 00052EH | USART5 - Control/Com. Register | ECCR5 | | RW |
| 00052FH | USART5 - Ext. Status Register | ESCR5 | | RW |
| 000530H | USART5 - Baud Rate Generator Register Low | BGRL5 | BGR5 | RW |
| 000531H | USART5 - Baud Rate Generator Register High | BGRH5 | | RW |
| 000532H | USART5 - Extended Serial Interrupt Register | ESIR5 | | RW |
| 000533H | Reserved | | | - |
| 000534H | USART6 - Serial Mode Register | SMR6 | | RW |
| 000535H | USART6 - Serial Control Register | SCR6 | | RW |
| 000536H | USART6 - Serial TX Register | TDR6 | | W |
| 000536H | USART6 - Serial RX Register | RDR6 | | R |
| 000537H | USART6 - Serial Status Register | SSR6 | | RW |
| 000538H | USART6 - Ext. Control/Com. Register | ECCR6 | | RW |
| 000539H | USART6 - Ext. Status Com. Register | ESCR6 | | RW |
| 00053AH | USART6 - Baud Rate Generator Register Low | BGRL6 | BGR6 | RW |
| 00053BH | USART6 - Baud Rate Generator Register High | BGRH6 | | RW |
| 00053CH | USART6 - Extended Serial Interrupt Register | ESIR6 | | RW |
| 00053DH | Reserved | | | - |
| 00053EH | USART7 - Serial Mode Register | SMR7 | | RW |
| 00053FH | USART7 - Serial Control Register | SCR7 | | RW |
| 000540H | USART7 - Serial TX Register | TDR7 | | W |
| 000540H | USART7 - Serial RX Register | RDR7 | | R |
| 000541H | USART7 - Serial Status Register | SSR7 | | RW |
| 000542H | USART7 - Ext. Control/Com. Register | ECCR7 | | RW |
| 000543H | USART7 - Ext. Status Com. Register | ESCR7 | | RW |
| 000544H | USART7 - Baud Rate Generator Register Low | BGRL7 | BGR7 | RW |
| 000545H | USART7 - Baud Rate Generator Register High | BGRH7 | | RW |
| 000546H | USART7 - Extended Serial Interrupt Register | ESIR7 | | RW |
| 000547H | Reserved | | | - |
| 000548H | USART8 - Serial Mode Register | SMR8 | | RW |
| 000549H | USART8 - Serial Control Register | SCR8 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00054AH | USART8 - Serial TX Register | TDR8 | | W |
| 00054AH | USART8 - Serial RX Register | RDR8 | | R |
| 00054BH | USART8 - Serial Status Register | SSR8 | | RW |
| 00054CH | USART8 - Ext. Control/Com. Register | ECCR8 | | RW |
| 00054DH | USART8 - Ext. Status Com. Register | ESCR8 | | RW |
| 00054EH | USART8 - Baud Rate Generator Register Low | BGRL8 | BGR8 | RW |
| 00054FH | USART8 - Baud Rate Generator Register High | BGRH8 | | RW |
| 000550H | USART8 - Extended Serial Interrupt Register | ESIR8 | | RW |
| 000551H | Reserved | | | - |
| 000552H | USART9 - Serial Mode Register | SMR9 | | RW |
| 000553H | USART9 - Serial Control Register | SCR9 | | RW |
| 000554H | USART9 - Serial TX Register | TDR9 | | W |
| 000554H | USART9 - Serial RX Register | RDR9 | | R |
| 000555H | USART9 - Serial Status Register | SSR9 | | RW |
| 000556H | USART9 - Ext. Control/Com. Register | ECCR9 | | RW |
| 000557H | USART9 - Ext. Status Com. Register | ESCR9 | | RW |
| 000558H | USART9 - Baud Rate Generator Register Low | BGRL9 | BGR9 | RW |
| 000559H | USART9 - Baud Rate Generator Register High | BGRH9 | | RW |
| 00055AH | USART9 - Extended Serial Interrupt Register | ESIR9 | | RW |
| 00055BH-00055FH | Reserved | | | - |
| 000560H | ALARM0 - Control Status Register | ACSR0 | | RW |
| 000561H | ALARM0 - Extended Control Status Register | AECSR0 | | RW |
| 000562H | ALARM1 - Control Status Register | ACSR1 | | RW |
| 000563H | ALARM1 - Extended Control Status Register | AECSR1 | | RW |
| 000564H | PPG6 - Timer register | | PTMR6 | R |
| 000565H | PPG6 - Timer register | | | R |
| 000566H | PPG6 - Period setting register | | PCSR6 | W |
| 000567H | PPG6 - Period setting register | | | W |
| 000568H | PPG6 - Duty cycle register | | PDUT6 | W |
| 000569H | PPG6 - Duty cycle register | | | W |
| 00056AH | PPG6 - Control status register Low | PCNL6 | PCN6 | RW |
| 00056BH | PPG6 - Control status register High | PCNH6 | | RW |
| 00056CH | PPG7 - Timer register | | PTMR7 | R |
| 00056DH | PPG7 - Timer register | | | R |
| 00056EH | PPG7 - Period setting register | | PCSR7 | W |
| 00056FH | PPG7 - Period setting register | | | W |
| 000570H | PPG7 - Duty cycle register | | PDUT7 | W |
| 000571H | PPG7 - Duty cycle register | | | W |
| 000572H | PPG7 - Control status register Low | PCNL7 | PCN7 | RW |
| 000573H | PPG7 - Control status register High | PCNH7 | | RW |
| 000574H | PPG11-PPG8 - General Control register 1 Low | GCN1L2 | GCN12 | RW |
| 000575H | PPG11-PPG8 - General Control register 1 High | GCN1H2 | | RW |
| 000576H | PPG11-PPG8 - General Control register 2 Low | GCN2L2 | GCN22 | RW |
| 000577H | PPG11-PPG8 - General Control register 2 High | GCN2H2 | | RW |
| 000578H | PPG8 - Timer register | | PTMR8 | R |
| 000579H | PPG8 - Timer register | | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00057AH | PPG8 - Period setting register | | PCSR8 | W |
| 00057BH | PPG8 - Period setting register | | | W |
| 00057CH | PPG8 - Duty cycle register | | PDUT8 | W |
| 00057DH | PPG8 - Duty cycle register | | | W |
| 00057EH | PPG8 - Control status register Low | PCNL8 | PCN8 | RW |
| 00057FH | PPG8 - Control status register High | PCNH8 | | RW |
| 000580H | PPG9 - Timer register | | PTMR9 | R |
| 000581H | PPG9 - Timer register | | | R |
| 000582H | PPG9 - Period setting register | | PCSR9 | W |
| 000583H | PPG9 - Period setting register | | | W |
| 000584H | PPG9 - Duty cycle register | | PDUT9 | W |
| 000585H | PPG9 - Duty cycle register | | | W |
| 000586H | PPG9 - Control status register Low | PCNL9 | PCN9 | RW |
| 000587H | PPG9 - Control status register High | PCNH9 | | RW |
| 000588H | PPG10 - Timer register | | PTMR10 | R |
| 000589H | PPG10 - Timer register | | | R |
| 00058AH | PPG10 - Period setting register | | PCSR10 | W |
| 00058BH | PPG10 - Period setting register | | | W |
| 00058CH | PPG10 - Duty cycle register | | PDUT10 | W |
| 00058DH | PPG10 - Duty cycle register | | | W |
| 00058EH | PPG10 - Control status register Low | PCNL10 | PCN10 | RW |
| 00058FH | PPG10 - Control status register High | PCNH10 | | RW |
| 000590H | PPG11 - Timer register | | PTMR11 | R |
| 000591H | PPG11 - Timer register | | | R |
| 000592H | PPG11 - Period setting register | | PCSR11 | W |
| 000593H | PPG11 - Period setting register | | | W |
| 000594H | PPG11 - Duty cycle register | | PDUT11 | W |
| 000595H | PPG11 - Duty cycle register | | | W |
| 000596H | PPG11 - Control status register Low | PCNL11 | PCN11 | RW |
| 000597H | PPG11 - Control status register High | PCNH11 | | RW |
| 000598H | PPG15-PPG12 - General Control register 1 Low | GCN1L3 | GCN13 | RW |
| 000599H | PPG15-PPG12 - General Control register 1 High | GCN1H3 | | RW |
| 00059AH | PPG15-PPG12 - General Control register 2 Low | GCN2L3 | GCN23 | RW |
| 00059BH | PPG15-PPG12 - General Control register 2 High | GCN2H3 | | RW |
| 00059CH | PPG12 - Timer register | | PTMR12 | R |
| 00059DH | PPG12 - Timer register | | | R |
| 00059EH | PPG12 - Period setting register | | PCSR12 | W |
| 00059FH | PPG12 - Period setting register | | | W |
| 0005A0H | PPG12 - Duty cycle register | | PDUT12 | W |
| 0005A1H | PPG12 - Duty cycle register | | | W |
| 0005A2H | PPG12 - Control status register Low | PCNL12 | PCN12 | RW |
| 0005A3H | PPG12 - Control status register High | PCNH12 | | RW |
| 0005A4H | PPG13 - Timer register | | PTMR13 | R |
| 0005A5H | PPG13 - Timer register | | | R |
| 0005A6H | PPG13 - Period setting register | | PCSR13 | W |
| 0005A7H | PPG13 - Period setting register | | | W |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0005A8H | PPG13 - Duty cycle register | | PDUT13 | W |
| 0005A9H | PPG13 - Duty cycle register | | | W |
| 0005AAH | PPG13 - Control status register Low | PCNL13 | PCN13 | RW |
| 0005ABH | PPG13 - Control status register High | PCNH13 | | RW |
| 0005ACH | PPG14 - Timer register | | PTMR14 | R |
| 0005ADH | PPG14 - Timer register | | | R |
| 0005AEH | PPG14 - Period setting register | | PCSR14 | W |
| 0005AFH | PPG14 - Period setting register | | | W |
| 0005B0H | PPG14 - Duty cycle register | | PDUT14 | W |
| 0005B1H | PPG14 - Duty cycle register | | | W |
| 0005B2H | PPG14 - Control status register Low | PCNL14 | PCN14 | RW |
| 0005B3H | PPG14 - Control status register High | PCNH14 | | RW |
| 0005B4H | PPG15 - Timer register | | PTMR15 | R |
| 0005B5H | PPG15 - Timer register | | | R |
| 0005B6H | PPG15 - Period setting register | | PCSR15 | W |
| 0005B7H | PPG15 - Period setting register | | | W |
| 0005B8H | PPG15 - Duty cycle register | | PDUT15 | W |
| 0005B9H | PPG15 - Duty cycle register | | | W |
| 0005BAH | PPG15 - Control status register Low | PCNL15 | PCN15 | RW |
| 0005BBH | PPG15 - Control status register High | PCNH15 | | RW |
| 0005BCH | PPG19-PPG16 - General Control register 1 Low | GCN1L4 | GCN14 | RW |
| 0005BDH | PPG19-PPG16 - General Control register 1 High | GCN1H4 | | RW |
| 0005BEH | PPG19-PPG16 - General Control register 2 Low | GCN2L4 | GCN24 | RW |
| 0005BFH | PPG19-PPG16 - General Control register 2 High | GCN2H4 | | RW |
| 0005C0H | PPG16 - Timer register | | PTMR16 | R |
| 0005C1H | PPG16 - Timer register | | | R |
| 0005C2H | PPG16 - Period setting register | | PCSR16 | W |
| 0005C3H | PPG16 - Period setting register | | | W |
| 0005C4H | PPG16 - Duty cycle register | | PDUT16 | W |
| 0005C5H | PPG16 - Duty cycle register | | | W |
| 0005C6H | PPG16 - Control status register Low | PCNL16 | PCN16 | RW |
| 0005C7H | PPG16 - Control status register High | PCNH16 | | RW |
| 0005C8H | PPG17 - Timer register | | PTMR17 | R |
| 0005C9H | PPG17 - Timer register | | | R |
| 0005CAH | PPG17 - Period setting register | | PCSR17 | W |
| 0005CBH | PPG17 - Period setting register | | | W |
| 0005CCH | PPG17 - Duty cycle register | | PDUT17 | W |
| 0005CDH | PPG17 - Duty cycle register | | | W |
| 0005CEH | PPG17 - Control status register Low | PCNL17 | PCN17 | RW |
| 0005CFH | PPG17 - Control status register High | PCNH17 | | RW |
| 0005D0H | PPG18 - Timer register | | PTMR18 | R |
| 0005D1H | PPG18 - Timer register | | | R |
| 0005D2H | PPG18 - Period setting register | | PCSR18 | W |
| 0005D3H | PPG18 - Period setting register | | | W |
| 0005D4H | PPG18 - Duty cycle register | | PDUT18 | W |
| 0005D5H | PPG18 - Duty cycle register | | | W |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0005D6H | PPG18 - Control status register Low | PCNL18 | PCN18 | RW |
| 0005D7H | PPG18 - Control status register High | PCNH18 | | RW |
| 0005D8H | PPG19 - Timer register | | PTMR19 | R |
| 0005D9H | PPG19 - Timer register | | | R |
| 0005DAH | PPG19 - Period setting register | | PCSR19 | W |
| 0005DBH | PPG19 - Period setting register | | | W |
| 0005DCH | PPG19 - Duty cycle register | | PDUT19 | W |
| 0005DDH | PPG19 - Duty cycle register | | | W |
| 0005DEH | PPG19 - Control status register Low | PCNL19 | PCN19 | RW |
| 0005DFH | PPG19 - Control status register High | PCNH19 | | RW |
| 0005E0H | SMC0 - PWM control register | PWC0 | | RW |
| 0005E1H | SMC0 - Extended control register (Output enable) | PWEC0 | | RW |
| 0005E2H | SMC0 - PWM compare register PWM 1 | | PWC10 | RW |
| 0005E3H | SMC0 - PWM compare register PWM 1 | | | RW |
| 0005E4H | SMC0 - PWM compare register PWM 2 | | PWC20 | RW |
| 0005E5H | SMC0 - PWM compare register PWM 2 | | | RW |
| 0005E6H | SMC0 - PWM Select register | PWS10 | | RW |
| 0005E7H | SMC0 - PWM Select register | PWS20 | | RW |
| 0005E8H-0005E9H | Reserved | | | - |
| 0005EAH | SMC1 - PWM control register | PWC1 | | RW |
| 0005EBH | SMC1 - Extended control register (Output enable) | PWEC1 | | RW |
| 0005ECH | SMC1 - PWM compare register PWM 1 | | PWC11 | RW |
| 0005EDH | SMC1 - PWM compare register PWM 1 | | | RW |
| 0005EEH | SMC1 - PWM compare register PWM 2 | | PWC21 | RW |
| 0005EFH | SMC1 - PWM compare register PWM 2 | | | RW |
| 0005F0H | SMC1 - PWM Select register | PWS11 | | RW |
| 0005F1H | SMC1 - PWM Select register | PWS21 | | RW |
| 0005F2H-0005F3H | Reserved | | | - |
| 0005F4H | SMC2 - PWM control register | PWC2 | | RW |
| 0005F5H | SMC2 - Extended control register (Output enable) | PWEC2 | | RW |
| 0005F6H | SMC2 - PWM compare register PWM 1 | | PWC12 | RW |
| 0005F7H | SMC2 - PWM compare register PWM 1 | | | RW |
| 0005F8H | SMC2 - PWM compare register PWM 2 | | PWC22 | RW |
| 0005F9H | SMC2 - PWM compare register PWM 2 | | | RW |
| 0005FAH | SMC2 - PWM Select register | PWS12 | | RW |
| 0005FBH | SMC2 - PWM Select register | PWS22 | | RW |
| 0005FCH-0005FDH | Reserved | | | - |
| 0005FEH | SMC3 - PWM control register | PWC3 | | RW |
| 0005FFH | SMC3 - Extended control register (Output enable) | PWEC3 | | RW |
| 000600H | SMC3 - PWM compare register PWM 1 | | PWC13 | RW |
| 000601H | SMC3 - PWM compare register PWM 1 | | | RW |
| 000602H | SMC3 - PWM compare register PWM 2 | | PWC23 | RW |
| 000603H | SMC3 - PWM compare register PWM 2 | | | RW |
| 000604H | SMC3 - PWM Select register | PWS13 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000605H | SMC3 - PWM Select register | PWS23 | | RW |
| 000606H-000607H | Reserved | | | - |
| 000608H | SMC4 - PWM control register | PWC4 | | RW |
| 000609H | SMC4 - Extended control register (Output enable) | PWEC4 | | RW |
| 00060AH | SMC4 - PWM compare register PWM 1 | | PWC14 | RW |
| 00060BH | SMC4 - PWM compare register PWM 1 | | | RW |
| 00060CH | SMC4 - PWM compare register PWM 2 | | PWC24 | RW |
| 00060DH | SMC4 - PWM compare register PWM 2 | | | RW |
| 00060EH | SMC4 - PWM Select register | PWS14 | | RW |
| 00060FH | SMC4 - PWM Select register | PWS24 | | RW |
| 000610H-000611H | Reserved | | | - |
| 000612H | SMC5 - PWM control register | PWC5 | | RW |
| 000613H | SMC5 - Extended control register (Output enable) | PWEC5 | | RW |
| 000614H | SMC5 - PWM compare register PWM 1 | | PWC15 | RW |
| 000615H | SMC5 - PWM compare register PWM 1 | | | RW |
| 000616H | SMC5 - PWM compare register PWM 2 | | PWC25 | RW |
| 000617H | SMC5 - PWM compare register PWM 2 | | | RW |
| 000618H | SMC5 - PWM Select register | PWS15 | | RW |
| 000619H | SMC5 - PWM Select register | PWS25 | | RW |
| 00061AH-00061BH | Reserved | | | - |
| 00061CH | LCD - Output Enable Register 0 (Seg 7-0) | LCDER0 | | RW |
| 00061DH | LCD - Output Enable Register 1 (Seg 15-8) | LCDER1 | | RW |
| 00061EH | LCD - Output Enable Register 2 (Seg 23-16) | LCDER2 | | RW |
| 00061FH | LCD - Output Enable Register 3 (Seg 31-24) | LCDER3 | | RW |
| 000620H | LCD - Output Enable Register 4 (Seg 39-32) | LCDER4 | | RW |
| 000621H | LCD - Output Enable Register 5 (Seg 47-40) | LCDER5 | | RW |
| 000622H | LCD - Output Enable Register 6 (Seg 55-48) | LCDER6 | | RW |
| 000623H | LCD - Output Enable Register 7 (Seg 63-56) | LCDER7 | | RW |
| 000624H | LCD - Output Enable Register 8 (Seg 71-64) | LCDER8 | | RW |
| 000625H | Reserved | | | - |
| 000626H | LCD - Output Enable Register V (Vx) | LCDVER | | RW |
| 000627H | LCD - Extended Control Register | LECR | | RW |
| 000628H | LCD - Common pin switching register | LCDCMR | | RW |
| 000629H | LCD - Control Register | LCR | | RW |
| 00062AH | LCD - Data register for Segment 1-0 | VRAM0 | | RW |
| 00062BH | LCD - Data register for Segment 3-2 | VRAM1 | | RW |
| 00062CH | LCD - Data register for Segment 5-4 | VRAM2 | | RW |
| 00062DH | LCD - Data register for Segment 7-6 | VRAM3 | | RW |
| 00062EH | LCD - Data register for Segment 9-8 | VRAM4 | | RW |
| 00062FH | LCD - Data register for Segment 11-10 | VRAM5 | | RW |
| 000630H | LCD - Data register for Segment 13-12 | VRAM6 | | RW |
| 000631H | LCD - Data register for Segment 15-14 | VRAM7 | | RW |
| 000632H | LCD - Data register for Segment 17-16 | VRAM8 | | RW |
| 000633H | LCD - Data register for Segment 19-18 | VRAM9 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000634H | LCD - Data register for Segment 21-20 | VRAM10 | | RW |
| 000635H | LCD - Data register for Segment 23-22 | VRAM11 | | RW |
| 000636H | LCD - Data register for Segment 25-24 | VRAM12 | | RW |
| 000637H | LCD - Data register for Segment 27-26 | VRAM13 | | RW |
| 000638H | LCD - Data register for Segment 29-28 | VRAM14 | | RW |
| 000639H | LCD - Data register for Segment 31-30 | VRAM15 | | RW |
| 00063AH | LCD - Data register for Segment 33-32 | VRAM16 | | RW |
| 00063BH | LCD - Data register for Segment 35-34 | VRAM17 | | RW |
| 00063CH | LCD - Data register for Segment 37-36 | VRAM18 | | RW |
| 00063DH | LCD - Data register for Segment 39-38 | VRAM19 | | RW |
| 00063EH | LCD - Data register for Segment 41-40 | VRAM20 | | RW |
| 00063FH | LCD - Data register for Segment 43-42 | VRAM21 | | RW |
| 000640H | LCD - Data register for Segment 45-44 | VRAM22 | | RW |
| 000641H | LCD - Data register for Segment 47-46 | VRAM23 | | RW |
| 000642H | LCD - Data register for Segment 49-48 | VRAM24 | | RW |
| 000643H | LCD - Data register for Segment 51-50 | VRAM25 | | RW |
| 000644H | LCD - Data register for Segment 53-52 | VRAM26 | | RW |
| 000645H | LCD - Data register for Segment 55-54 | VRAM27 | | RW |
| 000646H | LCD - Data register for Segment 57-56 | VRAM28 | | RW |
| 000647H | LCD - Data register for Segment 59-58 | VRAM29 | | RW |
| 000648H | LCD - Data register for Segment 61-60 | VRAM30 | | RW |
| 000649H | LCD - Data register for Segment 63-62 | VRAM31 | | RW |
| 00064AH | LCD - Data register for Segment 65-64 | VRAM32 | | RW |
| 00064BH | LCD - Data register for Segment 67-66 | VRAM33 | | RW |
| 00064CH | LCD - Data register for Segment 69-68 | VRAM34 | | RW |
| 00064DH | LCD - Data register for Segment 71-70 | VRAM35 | | RW |
| 00064EH-00065FH | Reserved | | | - |
| 000660H | Peripheral Resource Relocation Register 10 | PRRR10 | | RW |
| 000661H | Peripheral Resource Relocation Register 11 | PRRR11 | | RW |
| 000662H | Peripheral Resource Relocation Register 12 | PRRR12 | | RW |
| 000663H | Peripheral Resource Relocation Register 13 | PRRR13 | | W |
| 000664H-0006DFH | Reserved | | | - |
| 0006E0H | External Bus - Area configuration register 0 Low | EACL0 | EAC0 | RW |
| 0006E1H | External Bus - Area configuration register 0 High | EACH0 | | RW |
| 0006E2H | External Bus - Area configuration register 1 Low | EACL1 | EAC1 | RW |
| 0006E3H | External Bus - Area configuration register 1 High | EACH1 | | RW |
| 0006E4H | External Bus - Area configuration register 2 Low | EACL2 | EAC2 | RW |
| 0006E5H | External Bus - Area configuration register 2 High | EACH2 | | RW |
| 0006E6H | External Bus - Area configuration register 3 Low | EACL3 | EAC3 | RW |
| 0006E7H | External Bus - Area configuration register 3 High | EACH3 | | RW |
| 0006E8H | External Bus - Area configuration register 4 Low | EACL4 | EAC4 | RW |
| 0006E9H | External Bus - Area configuration register 4 High | EACH4 | | RW |
| 0006EAH | External Bus - Area configuration register 5 Low | EACL5 | EAC5 | RW |
| 0006EBH | External Bus - Area configuration register 5 High | EACH5 | | RW |
| 0006ECH | External Bus - Area select register 2 | EAS2 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0006EDH | External Bus - Area select register 3 | EAS3 | | RW |
| 0006EEH | External Bus - Area select register 4 | EAS4 | | RW |
| 0006EFH | External Bus - Area select register 5 | EAS5 | | RW |
| 0006F0H | External Bus - Mode register | EBM | | RW |
| 0006F1H | External Bus - Clock and Function register | EBCF | | RW |
| 0006F2H | External Bus - Address output enable register 0 | EBAE0 | | RW |
| 0006F3H | External Bus - Address output enable register 1 | EBAE1 | | RW |
| 0006F4H | External Bus - Address output enable register 2 | EBAE2 | | RW |
| 0006F5H | External Bus - Control signal register | EBCS | | RW |
| 0006F6H-0006FFH | Reserved | | | - |
| 000700H | CAN0 - Control register Low | CTRLRL0 | CTRLR0 | RW |
| 000701H | CAN0 - Control register High (reserved) | CTRLRH0 | | R |
| 000702H | CAN0 - Status register Low | STATRL0 | STATR0 | RW |
| 000703H | CAN0 - Status register High (reserved) | STATRH0 | | R |
| 000704H | CAN0 - Error Counter Low (Transmit) | ERRCNTL0 | ERRCNT0 | R |
| 000705H | CAN0 - Error Counter High (Receive) | ERRCNTH0 | | R |
| 000706H | CAN0 - Bit Timing Register Low | BTRL0 | BTR0 | RW |
| 000707H | CAN0 - Bit Timing Register High | BTRH0 | | RW |
| 000708H | CAN0 - Interrupt Register Low | INTRL0 | INTR0 | R |
| 000709H | CAN0 - Interrupt Register High | INTRH0 | | R |
| 00070AH | CAN0 - Test Register Low | TESTRL0 | TESTR0 | RW |
| 00070BH | CAN0 - Test Register High (reserved) | TESTRH0 | | R |
| 00070CH | CAN0 - BRP Extension register Low | BRPERL0 | BRPER0 | RW |
| 00070DH | CAN0 - BRP Extension register High (reserved) | BRPERH0 | | R |
| 00070EH-00070FH | Reserved | | | - |
| 000710H | CAN0 - IF1 Command request register Low | IF1CREQL0 | IF1CREQ0 | RW |
| 000711H | CAN0 - IF1 Command request register High | IF1CREQH0 | | RW |
| 000712H | CAN0 - IF1 Command Mask register Low | IF1CMSKL0 | IF1CMSK0 | RW |
| 000713H | CAN0 - IF1 Command Mask register High (reserved) | IF1CMSKH0 | | R |
| 000714H | CAN0 - IF1 Mask 1 Register Low | IF1MSK1L0 | IF1MSK10 | RW |
| 000715H | CAN0 - IF1 Mask 1 Register High | IF1MSK1H0 | | RW |
| 000716H | CAN0 - IF1 Mask 2 Register Low | IF1MSK2L0 | IF1MSK20 | RW |
| 000717H | CAN0 - IF1 Mask 2 Register High | IF1MSK2H0 | | RW |
| 000718H | CAN0 - IF1 Arbitration 1 Register Low | IF1ARB1L0 | IF1ARB10 | RW |
| 000719H | CAN0 - IF1 Arbitration 1 Register High | IF1ARB1H0 | | RW |
| 00071AH | CAN0 - IF1 Arbitration 2 Register Low | IF1ARB2L0 | IF1ARB20 | RW |
| 00071BH | CAN0 - IF1 Arbitration 2 Register High | IF1ARB2H0 | | RW |
| 00071CH | CAN0 - IF1 Message Control Register Low | IF1MCTRL0 | IF1MCTR0 | RW |
| 00071DH | CAN0 - IF1 Message Control Register High | IF1MCTRH0 | | RW |
| 00071EH | CAN0 - IF1 Data A1 Low | IF1DTA1L0 | IF1DTA10 | RW |
| 00071FH | CAN0 - IF1 Data A1 High | IF1DTA1H0 | | RW |
| 000720H | CAN0 - IF1 Data A2 Low | IF1DTA2L0 | IF1DTA20 | RW |
| 000721H | CAN0 - IF1 Data A2 High | IF1DTA2H0 | | RW |
| 000722H | CAN0 - IF1 Data B1 Low | IF1DTB1L0 | IF1DTB10 | RW |
| 000723H | CAN0 - IF1 Data B1 High | IF1DTB1H0 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000724H | CAN0 - IF1 Data B2 Low | IF1DTB2L0 | IF1DTB20 | RW |
| 000725H | CAN0 - IF1 Data B2 High | IF1DTB2H0 | | RW |
| 000726H-00073FH | Reserved | | | - |
| 000740H | CAN0 - IF2 Command request register Low | IF2CREQL0 | IF2CREQ0 | RW |
| 000741H | CAN0 - IF2 Command request register High | IF2CREQH0 | | RW |
| 000742H | CAN0 - IF2 Command Mask register Low | IF2CMSKL0 | IF2CMSK0 | RW |
| 000743H | CAN0 - IF2 Command Mask register High (reserved) | IF2CMSKH0 | | R |
| 000744H | CAN0 - IF2 Mask 1 Register Low | IF2MSK1L0 | IF2MSK10 | RW |
| 000745H | CAN0 - IF2 Mask 1 Register High | IF2MSK1H0 | | RW |
| 000746H | CAN0 - IF2 Mask 2 Register Low | IF2MSK2L0 | IF2MSK20 | RW |
| 000747H | CAN0 - IF2 Mask 2 Register High | IF2MSK2H0 | | RW |
| 000748H | CAN0 - IF2 Arbitration 1 Register Low | IF2ARB1L0 | IF2ARB10 | RW |
| 000749H | CAN0 - IF2 Arbitration 1 Register High | IF2ARB1H0 | | RW |
| 00074AH | CAN0 - IF2 Arbitration 2 Register Low | IF2ARB2L0 | IF2ARB20 | RW |
| 00074BH | CAN0 - IF2 Arbitration 2 Register High | IF2ARB2H0 | | RW |
| 00074CH | CAN0 - IF2 Message Control Register Low | IF2MCTRL0 | IF2MCTR0 | RW |
| 00074DH | CAN0 - IF2 Message Control Register High | IF2MCTRH0 | | RW |
| 00074EH | CAN0 - IF2 Data A1 Low | IF2DTA1L0 | IF2DTA10 | RW |
| 00074FH | CAN0 - IF2 Data A1 High | IF2DTA1H0 | | RW |
| 000750H | CAN0 - IF2 Data A2 Low | IF2DTA2L0 | IF2DTA20 | RW |
| 000751H | CAN0 - IF2 Data A2 High | IF2DTA2H0 | | RW |
| 000752H | CAN0 - IF2 Data B1 Low | IF2DTB1L0 | IF2DTB10 | RW |
| 000753H | CAN0 - IF2 Data B1 High | IF2DTB1H0 | | RW |
| 000754H | CAN0 - IF2 Data B2 Low | IF2DTB2L0 | IF2DTB20 | RW |
| 000755H | CAN0 - IF2 Data B2 High | IF2DTB2H0 | | RW |
| 000756H-00077FH | Reserved | | | - |
| 000780H | CAN0 - Transmission Request 1 Register Low | TREQR1L0 | TREQR10 | R |
| 000781H | CAN0 - Transmission Request 1 Register High | TREQR1H0 | | R |
| 000782H | CAN0 - Transmission Request 2 Register Low | TREQR2L0 | TREQR20 | R |
| 000783H | CAN0 - Transmission Request 2 Register High | TREQR2H0 | | R |
| 000784H-00078FH | Reserved | | | - |
| 000790H | CAN0 - New Data 1 Register Low | NEWDT1L0 | NEWDT10 | R |
| 000791H | CAN0 - New Data 1 Register High | NEWDT1H0 | | R |
| 000792H | CAN0 - New Data 2 Register Low | NEWDT2L0 | NEWDT20 | R |
| 000793H | CAN0 - New Data 2 Register High | NEWDT2H0 | | R |
| 000794H-00079FH | Reserved | | | - |
| 0007A0H | CAN0 - Interrupt Pending 1 Register Low | INTPND1L0 | INTPND10 | R |
| 0007A1H | CAN0 - Interrupt Pending 1 Register High | INTPND1H0 | | R |
| 0007A2H | CAN0 - Interrupt Pending 2 Register Low | INTPND2L0 | INTPND20 | R |
| 0007A3H | CAN0 - Interrupt Pending 2 Register High | INTPND2H0 | | R |
| 0007A4H-0007AFH | Reserved | | | - |
| 0007B0H | CAN0 - Message Valid 1 Register Low | MSGVAL1L0 | MSGVAL10 | R |
| 0007B1H | CAN0 - Message Valid 1 Register High | MSGVAL1H0 | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0007B2H | CAN0 - Message Valid 2 Register Low | MSGVAL2L0 | MSGVAL20 | R |
| 0007B3H | CAN0 - Message Valid 2 Register High | MSGVAL2H0 | | R |
| 0007B4H-0007CDH | Reserved | | | - |
| 0007CEH | CAN0 - Output enable register | COER0 | | RW |
| 0007CFH | Reserved | | | - |
| 0007D0H | SG0 - Sound Generator Control Register Low | SGCRL0 | SGCR0 | RW |
| 0007D1H | SG0 - Sound Generator Control Register High | SGCRH0 | | RW |
| 0007D2H | SG0 - Sound Generator Frequency Register | SGFR0 | | RW |
| 0007D3H | SG0 - Sound Generator Amplitude Register | SGAR0 | | RW |
| 0007D4H | SG0 - Sound Generator Decrement Register | SGDR0 | | RW |
| 0007D5H | SG0 - Sound Generator Tone Register | SGTR0 | | RW |
| 0007D6H | SG1 - Sound Generator Control Register Low | SGCRL1 | SGCR1 | RW |
| 0007D7H | SG1 - Sound Generator Control Register High | SGCRH1 | | RW |
| 0007D8H | SG1 - Sound Generator Frequency Register | SGFR1 | | RW |
| 0007D9H | SG1 - Sound Generator Amplitude Register | SGAR1 | | RW |
| 0007DAH | SG1 - Sound Generator Decrement Register | SGDR1 | | RW |
| 0007DBH | SG1 - Sound Generator Tone Register | SGTR1 | | RW |
| 0007DCH-0007FFH | Reserved | | | - |
| 000800H | CAN1 - Control register Low | CTRLRL1 | CTRLR1 | RW |
| 000801H | CAN1 - Control register High (reserved) | CTRLRH1 | | R |
| 000802H | CAN1 - Status register Low | STATRL1 | STATR1 | RW |
| 000803H | CAN1 - Status register High (reserved) | STATRH1 | | R |
| 000804H | CAN1 - Error Counter Low (Transmit) | ERRCNTL1 | ERRCNT1 | R |
| 000805H | CAN1 - Error Counter High (Receive) | ERRCNTH1 | | R |
| 000806H | CAN1 - Bit Timing Register Low | BTRL1 | BTR1 | RW |
| 000807H | CAN1 - Bit Timing Register High | BTRH1 | | RW |
| 000808H | CAN1 - Interrupt Register Low | INTRL1 | INTR1 | R |
| 000809H | CAN1 - Interrupt Register High | INTRH1 | | R |
| 00080AH | CAN1 - Test Register Low | TESTRL1 | TESTR1 | RW |
| 00080BH | CAN1 - Test Register High (reserved) | TESTRH1 | | R |
| 00080CH | CAN1 - BRP Extension register Low | BRPERL1 | BRPER1 | RW |
| 00080DH | CAN1 - BRP Extension register High (reserved) | BRPERH1 | | R |
| 00080EH-00080FH | Reserved | | | - |
| 000810H | CAN1 - IF1 Command request register Low | IF1CREQL1 | IF1CREQ1 | RW |
| 000811H | CAN1 - IF1 Command request register High | IF1CREQH1 | | RW |
| 000812H | CAN1 - IF1 Command Mask register Low | IF1CMSKL1 | IF1CMSK1 | RW |
| 000813H | CAN1 - IF1 Command Mask register High (reserved) | IF1CMSKH1 | | R |
| 000814H | CAN1 - IF1 Mask 1 Register Low | IF1MSK1L1 | IF1MSK11 | RW |
| 000815H | CAN1 - IF1 Mask 1 Register High | IF1MSK1H1 | | RW |
| 000816H | CAN1 - IF1 Mask 2 Register Low | IF1MSK2L1 | IF1MSK21 | RW |
| 000817H | CAN1 - IF1 Mask 2 Register High | IF1MSK2H1 | | RW |
| 000818H | CAN1 - IF1 Arbitration 1 Register Low | IF1ARB1L1 | IF1ARB11 | RW |
| 000819H | CAN1 - IF1 Arbitration 1 Register High | IF1ARB1H1 | | RW |
| 00081AH | CAN1 - IF1 Arbitration 2 Register Low | IF1ARB2L1 | IF1ARB21 | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00081BH | CAN1 - IF1 Arbitration 2 Register High | IF1ARB2H1 | | RW |
| 00081CH | CAN1 - IF1 Message Control Register Low | IF1MCTRL1 | IF1MCTR1 | RW |
| 00081DH | CAN1 - IF1 Message Control Register High | IF1MCTRH1 | | RW |
| 00081EH | CAN1 - IF1 Data A1 Low | IF1DTA1L1 | IF1DTA11 | RW |
| 00081FH | CAN1 - IF1 Data A1 High | IF1DTA1H1 | | RW |
| 000820H | CAN1 - IF1 Data A2 Low | IF1DTA2L1 | IF1DTA21 | RW |
| 000821H | CAN1 - IF1 Data A2 High | IF1DTA2H1 | | RW |
| 000822H | CAN1 - IF1 Data B1 Low | IF1DTB1L1 | IF1DTB11 | RW |
| 000823H | CAN1 - IF1 Data B1 High | IF1DTB1H1 | | RW |
| 000824H | CAN1 - IF1 Data B2 Low | IF1DTB2L1 | IF1DTB21 | RW |
| 000825H | CAN1 - IF1 Data B2 High | IF1DTB2H1 | | RW |
| 000826H-00083FH | Reserved | | | - |
| 000840H | CAN1 - IF2 Command request register Low | IF2CREQL1 | IF2CREQ1 | RW |
| 000841H | CAN1 - IF2 Command request register High | IF2CREQH1 | | RW |
| 000842H | CAN1 - IF2 Command Mask register Low | IF2CMSKL1 | IF2CMSK1 | RW |
| 000843H | CAN1 - IF2 Command Mask register High (reserved) | IF2CMSKH1 | | R |
| 000844H | CAN1 - IF2 Mask 1 Register Low | IF2MSK1L1 | IF2MSK11 | RW |
| 000845H | CAN1 - IF2 Mask 1 Register High | IF2MSK1H1 | | RW |
| 000846H | CAN1 - IF2 Mask 2 Register Low | IF2MSK2L1 | IF2MSK21 | RW |
| 000847H | CAN1 - IF2 Mask 2 Register High | IF2MSK2H1 | | RW |
| 000848H | CAN1 - IF2 Arbitration 1 Register Low | IF2ARB1L1 | IF2ARB11 | RW |
| 000849H | CAN1 - IF2 Arbitration 1 Register High | IF2ARB1H1 | | RW |
| 00084AH | CAN1 - IF2 Arbitration 2 Register Low | IF2ARB2L1 | IF2ARB21 | RW |
| 00084BH | CAN1 - IF2 Arbitration 2 Register High | IF2ARB2H1 | | RW |
| 00084CH | CAN1 - IF2 Message Control Register Low | IF2MCTRL1 | IF2MCTR1 | RW |
| 00084DH | CAN1 - IF2 Message Control Register High | IF2MCTRH1 | | RW |
| 00084EH | CAN1 - IF2 Data A1 Low | IF2DTA1L1 | IF2DTA11 | RW |
| 00084FH | CAN1 - IF2 Data A1 High | IF2DTA1H1 | | RW |
| 000850H | CAN1 - IF2 Data A2 Low | IF2DTA2L1 | IF2DTA21 | RW |
| 000851H | CAN1 - IF2 Data A2 High | IF2DTA2H1 | | RW |
| 000852H | CAN1 - IF2 Data B1 Low | IF2DTB1L1 | IF2DTB11 | RW |
| 000853H | CAN1 - IF2 Data B1 High | IF2DTB1H1 | | RW |
| 000854H | CAN1 - IF2 Data B2 Low | IF2DTB2L1 | IF2DTB21 | RW |
| 000855H | CAN1 - IF2 Data B2 High | IF2DTB2H1 | | RW |
| 000856H-00087FH | Reserved | | | - |
| 000880H | CAN1 - Transmission Request 1 Register Low | TREQR1L1 | TREQR11 | R |
| 000881H | CAN1 - Transmission Request 1 Register High | TREQR1H1 | | R |
| 000882H | CAN1 - Transmission Request 2 Register Low | TREQR2L1 | TREQR21 | R |
| 000883H | CAN1 - Transmission Request 2 Register High | TREQR2H1 | | R |
| 000884H-00088FH | Reserved | | | - |
| 000890H | CAN1 - New Data 1 Register Low | NEWDT1L1 | NEWDT11 | R |
| 000891H | CAN1 - New Data 1 Register High | NEWDT1H1 | | R |
| 000892H | CAN1 - New Data 2 Register Low | NEWDT2L1 | NEWDT21 | R |
| 000893H | CAN1 - New Data 2 Register High | NEWDT2H1 | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000894H-00089FH | Reserved | | | - |
| 0008A0H | CAN1 - Interrupt Pending 1 Register Low | INTPND1L1 | INTPND11 | R |
| 0008A1H | CAN1 - Interrupt Pending 1 Register High | INTPND1H1 | | R |
| 0008A2H | CAN1 - Interrupt Pending 2 Register Low | INTPND2L1 | INTPND21 | R |
| 0008A3H | CAN1 - Interrupt Pending 2 Register High | INTPND2H1 | | R |
| 0008A4H-0008AFH | Reserved | | | - |
| 0008B0H | CAN1 - Message Valid 1 Register Low | MSGVAL1L1 | MSGVAL11 | R |
| 0008B1H | CAN1 - Message Valid 1 Register High | MSGVAL1H1 | | R |
| 0008B2H | CAN1 - Message Valid 2 Register Low | MSGVAL2L1 | MSGVAL21 | R |
| 0008B3H | CAN1 - Message Valid 2 Register High | MSGVAL2H1 | | R |
| 0008B4H-0008CDH | Reserved | | | - |
| 0008CEH | CAN1 - Output enable register | COER1 | | RW |
| 0008CFH-0008FFH | Reserved | | | - |
| 000900H | CAN2 - Control register Low | CTRLRL2 | CTRLR2 | RW |
| 000901H | CAN2 - Control register High (reserved) | CTRLRH2 | | R |
| 000902H | CAN2 - Status register Low | STATRL2 | STATR2 | RW |
| 000903H | CAN2 - Status register High (reserved) | STATRH2 | | R |
| 000904H | CAN2 - Error Counter Low (Transmit) | ERRCNTL2 | ERRCNT2 | R |
| 000905H | CAN2 - Error Counter High (Receive) | ERRCNTH2 | | R |
| 000906H | CAN2 - Bit Timing Register Low | BTRL2 | BTR2 | RW |
| 000907H | CAN2 - Bit Timing Register High | BTRH2 | | RW |
| 000908H | CAN2 - Interrupt Register Low | INTRL2 | INTR2 | R |
| 000909H | CAN2 - Interrupt Register High | INTRH2 | | R |
| 00090AH | CAN2 - Test Register Low | TESTRL2 | TESTR2 | RW |
| 00090BH | CAN2 - Test Register High (reserved) | TESTRH2 | | R |
| 00090CH | CAN2 - BRP Extension register Low | BRPERL2 | BRPER2 | RW |
| 00090DH | CAN2 - BRP Extension register High (reserved) | BRPERH2 | | R |
| 00090EH-00090FH | Reserved | | | - |
| 000910H | CAN2 - IF1 Command request register Low | IF1CREQL2 | IF1CREQ2 | RW |
| 000911H | CAN2 - IF1 Command request register High | IF1CREQH2 | | RW |
| 000912H | CAN2 - IF1 Command Mask register Low | IF1CMSKL2 | IF1CMSK2 | RW |
| 000913H | CAN2 - IF1 Command Mask register High (reserved) | IF1CMSKH2 | | R |
| 000914H | CAN2 - IF1 Mask 1 Register Low | IF1MSK1L2 | IF1MSK12 | RW |
| 000915H | CAN2 - IF1 Mask 1 Register High | IF1MSK1H2 | | RW |
| 000916H | CAN2 - IF1 Mask 2 Register Low | IF1MSK2L2 | IF1MSK22 | RW |
| 000917H | CAN2 - IF1 Mask 2 Register High | IF1MSK2H2 | | RW |
| 000918H | CAN2 - IF1 Arbitration 1 Register Low | IF1ARB1L2 | IF1ARB12 | RW |
| 000919H | CAN2 - IF1 Arbitration 1 Register High | IF1ARB1H2 | | RW |
| 00091AH | CAN2 - IF1 Arbitration 2 Register Low | IF1ARB2L2 | IF1ARB22 | RW |
| 00091BH | CAN2 - IF1 Arbitration 2 Register High | IF1ARB2H2 | | RW |
| 00091CH | CAN2 - IF1 Message Control Register Low | IF1MCTRL2 | IF1MCTR2 | RW |
| 00091DH | CAN2 - IF1 Message Control Register High | IF1MCTRH2 | | RW |
| 00091EH | CAN2 - IF1 Data A1 Low | IF1DTA1L2 | IF1DTA12 | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 00091FH | CAN2 - IF1 Data A1 High | IF1DTA1H2 | | RW |
| 000920H | CAN2 - IF1 Data A2 Low | IF1DTA2L2 | IF1DTA22 | RW |
| 000921H | CAN2 - IF1 Data A2 High | IF1DTA2H2 | | RW |
| 000922H | CAN2 - IF1 Data B1 Low | IF1DTB1L2 | IF1DTB12 | RW |
| 000923H | CAN2 - IF1 Data B1 High | IF1DTB1H2 | | RW |
| 000924H | CAN2 - IF1 Data B2 Low | IF1DTB2L2 | IF1DTB22 | RW |
| 000925H | CAN2 - IF1 Data B2 High | IF1DTB2H2 | | RW |
| 000926H-00093FH | Reserved | | | - |
| 000940H | CAN2 - IF2 Command request register Low | IF2CREQL2 | IF2CREQ2 | RW |
| 000941H | CAN2 - IF2 Command request register High | IF2CREQH2 | | RW |
| 000942H | CAN2 - IF2 Command Mask register Low | IF2CMSKL2 | IF2CMSK2 | RW |
| 000943H | CAN2 - IF2 Command Mask register High (reserved) | IF2CMSKH2 | | R |
| 000944H | CAN2 - IF2 Mask 1 Register Low | IF2MSK1L2 | IF2MSK12 | RW |
| 000945H | CAN2 - IF2 Mask 1 Register High | IF2MSK1H2 | | RW |
| 000946H | CAN2 - IF2 Mask 2 Register Low | IF2MSK2L2 | IF2MSK22 | RW |
| 000947H | CAN2 - IF2 Mask 2 Register High | IF2MSK2H2 | | RW |
| 000948H | CAN2 - IF2 Arbitration 1 Register Low | IF2ARB1L2 | IF2ARB12 | RW |
| 000949H | CAN2 - IF2 Arbitration 1 Register High | IF2ARB1H2 | | RW |
| 00094AH | CAN2 - IF2 Arbitration 2 Register Low | IF2ARB2L2 | IF2ARB22 | RW |
| 00094BH | CAN2 - IF2 Arbitration 2 Register High | IF2ARB2H2 | | RW |
| 00094CH | CAN2 - IF2 Message Control Register Low | IF2MCTRL2 | IF2MCTR2 | RW |
| 00094DH | CAN2 - IF2 Message Control Register High | IF2MCTRH2 | | RW |
| 00094EH | CAN2 - IF2 Data A1 Low | IF2DTA1L2 | IF2DTA12 | RW |
| 00094FH | CAN2 - IF2 Data A1 High | IF2DTA1H2 | | RW |
| 000950H | CAN2 - IF2 Data A2 Low | IF2DTA2L2 | IF2DTA22 | RW |
| 000951H | CAN2 - IF2 Data A2 High | IF2DTA2H2 | | RW |
| 000952H | CAN2 - IF2 Data B1 Low | IF2DTB1L2 | IF2DTB12 | RW |
| 000953H | CAN2 - IF2 Data B1 High | IF2DTB1H2 | | RW |
| 000954H | CAN2 - IF2 Data B2 Low | IF2DTB2L2 | IF2DTB22 | RW |
| 000955H | CAN2 - IF2 Data B2 High | IF2DTB2H2 | | RW |
| 000956H-00097FH | Reserved | | | - |
| 000980H | CAN2 - Transmission Request 1 Register Low | TREQR1L2 | TREQR12 | R |
| 000981H | CAN2 - Transmission Request 1 Register High | TREQR1H2 | | R |
| 000982H | CAN2 - Transmission Request 2 Register Low | TREQR2L2 | TREQR22 | R |
| 000983H | CAN2 - Transmission Request 2 Register High | TREQR2H2 | | R |
| 000984H-00098FH | Reserved | | | - |
| 000990H | CAN2 - New Data 1 Register Low | NEWDT1L2 | NEWDT12 | R |
| 000991H | CAN2 - New Data 1 Register High | NEWDT1H2 | | R |
| 000992H | CAN2 - New Data 2 Register Low | NEWDT2L2 | NEWDT22 | R |
| 000993H | CAN2 - New Data 2 Register High | NEWDT2H2 | | R |
| 000994H-00099FH | Reserved | | | - |
| 0009A0H | CAN2 - Interrupt Pending 1 Register Low | INTPND1L2 | INTPND12 | R |
| 0009A1H | CAN2 - Interrupt Pending 1 Register High | INTPND1H2 | | R |
| 0009A2H | CAN2 - Interrupt Pending 2 Register Low | INTPND2L2 | INTPND22 | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 0009A3H | CAN2 - Interrupt Pending 2 Register High | INTPND2H2 | | R |
| 0009A4H-0009AFH | Reserved | | | - |
| 0009B0H | CAN2 - Message Valid 1 Register Low | MSGVAL1L2 | MSGVAL12 | R |
| 0009B1H | CAN2 - Message Valid 1 Register High | MSGVAL1H2 | | R |
| 0009B2H | CAN2 - Message Valid 2 Register Low | MSGVAL2L2 | MSGVAL22 | R |
| 0009B3H | CAN2 - Message Valid 2 Register High | MSGVAL2H2 | | R |
| 0009B4H-0009CDH | Reserved | | | - |
| 0009CEH | CAN2 - Output enable register | COER2 | | RW |
| 0009CFH-0009FFH | Reserved | | | - |
| 000A00H | CAN3 - Control register Low | CTRLRL3 | CTRLR3 | RW |
| 000A01H | CAN3 - Control register High (reserved) | CTRLRH3 | | R |
| 000A02H | CAN3 - Status register Low | STATRL3 | STATR3 | RW |
| 000A03H | CAN3 - Status register High (reserved) | STATRH3 | | R |
| 000A04H | CAN3 - Error Counter Low (Transmit) | ERRCNTL3 | ERRCNT3 | R |
| 000A05H | CAN3 - Error Counter High (Receive) | ERRCNTH3 | | R |
| 000A06H | CAN3 - Bit Timing Register Low | BTRL3 | BTR3 | RW |
| 000A07H | CAN3 - Bit Timing Register High | BTRH3 | | RW |
| 000A08H | CAN3 - Interrupt Register Low | INTRL3 | INTR3 | R |
| 000A09H | CAN3 - Interrupt Register High | INTRH3 | | R |
| 000A0AH | CAN3 - Test Register Low | TESTRL3 | TESTR3 | RW |
| 000A0BH | CAN3 - Test Register High (reserved) | TESTRH3 | | R |
| 000A0CH | CAN3 - BRP Extension register Low | BRPERL3 | BRPER3 | RW |
| 000A0DH | CAN3 - BRP Extension register High (reserved) | BRPERH3 | | R |
| 000A0EH-000A0FH | Reserved | | | - |
| 000A10H | CAN3 - IF1 Command request register Low | IF1CREQL3 | IF1CREQ3 | RW |
| 000A11H | CAN3 - IF1 Command request register High | IF1CREQH3 | | RW |
| 000A12H | CAN3 - IF1 Command Mask register Low | IF1CMSKL3 | IF1CMSK3 | RW |
| 000A13H | CAN3 - IF1 Command Mask register High (reserved) | IF1CMSKH3 | | R |
| 000A14H | CAN3 - IF1 Mask 1 Register Low | IF1MSK1L3 | IF1MSK13 | RW |
| 000A15H | CAN3 - IF1 Mask 1 Register High | IF1MSK1H3 | | RW |
| 000A16H | CAN3 - IF1 Mask 2 Register Low | IF1MSK2L3 | IF1MSK23 | RW |
| 000A17H | CAN3 - IF1 Mask 2 Register High | IF1MSK2H3 | | RW |
| 000A18H | CAN3 - IF1 Arbitration 1 Register Low | IF1ARB1L3 | IF1ARB13 | RW |
| 000A19H | CAN3 - IF1 Arbitration 1 Register High | IF1ARB1H3 | | RW |
| 000A1AH | CAN3 - IF1 Arbitration 2 Register Low | IF1ARB2L3 | IF1ARB23 | RW |
| 000A1BH | CAN3 - IF1 Arbitration 2 Register High | IF1ARB2H3 | | RW |
| 000A1CH | CAN3 - IF1 Message Control Register Low | IF1MCTRL3 | IF1MCTR3 | RW |
| 000A1DH | CAN3 - IF1 Message Control Register High | IF1MCTRH3 | | RW |
| 000A1EH | CAN3 - IF1 Data A1 Low | IF1DTA1L3 | IF1DTA13 | RW |
| 000A1FH | CAN3 - IF1 Data A1 High | IF1DTA1H3 | | RW |
| 000A20H | CAN3 - IF1 Data A2 Low | IF1DTA2L3 | IF1DTA23 | RW |
| 000A21H | CAN3 - IF1 Data A2 High | IF1DTA2H3 | | RW |
| 000A22H | CAN3 - IF1 Data B1 Low | IF1DTB1L3 | IF1DTB13 | RW |
| 000A23H | CAN3 - IF1 Data B1 High | IF1DTB1H3 | | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000A24H | CAN3 - IF1 Data B2 Low | IF1DTB2L3 | IF1DTB23 | RW |
| 000A25H | CAN3 - IF1 Data B2 High | IF1DTB2H3 | | RW |
| 000A26H-000A3FH | Reserved | | | - |
| 000A40H | CAN3 - IF2 Command request register Low | IF2CREQL3 | IF2CREQ3 | RW |
| 000A41H | CAN3 - IF2 Command request register High | IF2CREQH3 | | RW |
| 000A42H | CAN3 - IF2 Command Mask register Low | IF2CMSKL3 | IF2CMSK3 | RW |
| 000A43H | CAN3 - IF2 Command Mask register High (reserved) | IF2CMSKH3 | | R |
| 000A44H | CAN3 - IF2 Mask 1 Register Low | IF2MSK1L3 | IF2MSK13 | RW |
| 000A45H | CAN3 - IF2 Mask 1 Register High | IF2MSK1H3 | | RW |
| 000A46H | CAN3 - IF2 Mask 2 Register Low | IF2MSK2L3 | IF2MSK23 | RW |
| 000A47H | CAN3 - IF2 Mask 2 Register High | IF2MSK2H3 | | RW |
| 000A48H | CAN3 - IF2 Arbitration 1 Register Low | IF2ARB1L3 | IF2ARB13 | RW |
| 000A49H | CAN3 - IF2 Arbitration 1 Register High | IF2ARB1H3 | | RW |
| 000A4AH | CAN3 - IF2 Arbitration 2 Register Low | IF2ARB2L3 | IF2ARB23 | RW |
| 000A4BH | CAN3 - IF2 Arbitration 2 Register High | IF2ARB2H3 | | RW |
| 000A4CH | CAN3 - IF2 Message Control Register Low | IF2MCTRL3 | IF2MCTR3 | RW |
| 000A4DH | CAN3 - IF2 Message Control Register High | IF2MCTRH3 | | RW |
| 000A4EH | CAN3 - IF2 Data A1 Low | IF2DTA1L3 | IF2DTA13 | RW |
| 000A4FH | CAN3 - IF2 Data A1 High | IF2DTA1H3 | | RW |
| 000A50H | CAN3 - IF2 Data A2 Low | IF2DTA2L3 | IF2DTA23 | RW |
| 000A51H | CAN3 - IF2 Data A2 High | IF2DTA2H3 | | RW |
| 000A52H | CAN3 - IF2 Data B1 Low | IF2DTB1L3 | IF2DTB13 | RW |
| 000A53H | CAN3 - IF2 Data B1 High | IF2DTB1H3 | | RW |
| 000A54H | CAN3 - IF2 Data B2 Low | IF2DTB2L3 | IF2DTB23 | RW |
| 000A55H | CAN3 - IF2 Data B2 High | IF2DTB2H3 | | RW |
| 000A56H-000A7FH | Reserved | | | - |
| 000A80H | CAN3 - Transmission Request 1 Register Low | TREQR1L3 | TREQR13 | R |
| 000A81H | CAN3 - Transmission Request 1 Register High | TREQR1H3 | | R |
| 000A82H | CAN3 - Transmission Request 2 Register Low | TREQR2L3 | TREQR23 | R |
| 000A83H | CAN3 - Transmission Request 2 Register High | TREQR2H3 | | R |
| 000A84H-000A8FH | Reserved | | | - |
| 000A90H | CAN3 - New Data 1 Register Low | NEWDT1L3 | NEWDT13 | R |
| 000A91H | CAN3 - New Data 1 Register High | NEWDT1H3 | | R |
| 000A92H | CAN3 - New Data 2 Register Low | NEWDT2L3 | NEWDT23 | R |
| 000A93H | CAN3 - New Data 2 Register High | NEWDT2H3 | | R |
| 000A94H-000A9FH | Reserved | | | - |
| 000AA0H | CAN3 - Interrupt Pending 1 Register Low | INTPND1L3 | INTPND13 | R |
| 000AA1H | CAN3 - Interrupt Pending 1 Register High | INTPND1H3 | | R |
| 000AA2H | CAN3 - Interrupt Pending 2 Register Low | INTPND2L3 | INTPND23 | R |
| 000AA3H | CAN3 - Interrupt Pending 2 Register High | INTPND2H3 | | R |
| 000AA4H-000AAFH | Reserved | | | - |
| 000AB0H | CAN3 - Message Valid 1 Register Low | MSGVAL1L3 | MSGVAL13 | R |
| 000AB1H | CAN3 - Message Valid 1 Register High | MSGVAL1H3 | | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000AB2H | CAN3 - Message Valid 2 Register Low | MSGVAL2L3 | MSGVAL23 | R |
| 000AB3H | CAN3 - Message Valid 2 Register High | MSGVAL2H3 | | R |
| 000AB4H-000ACDH | Reserved | | | - |
| 000ACEH | CAN3 - Output enable register | COER3 | | RW |
| 000ACFH-000AFFH | Reserved | | | - |
| 000B00H | CAN4 - Control register Low | CTRLRL4 | CTRLR4 | RW |
| 000B01H | CAN4 - Control register High (reserved) | CTRLRH4 | | R |
| 000B02H | CAN4 - Status register Low | STATRL4 | STATR4 | RW |
| 000B03H | CAN4 - Status register High (reserved) | STATRH4 | | R |
| 000B04H | CAN4 - Error Counter Low (Transmit) | ERRCNTL4 | ERRCNT4 | R |
| 000B05H | CAN4 - Error Counter High (Receive) | ERRCNTH4 | | R |
| 000B06H | CAN4 - Bit Timing Register Low | BTRL4 | BTR4 | RW |
| 000B07H | CAN4 - Bit Timing Register High | BTRH4 | | RW |
| 000B08H | CAN4 - Interrupt Register Low | INTRL4 | INTR4 | R |
| 000B09H | CAN4 - Interrupt Register High | INTRH4 | | R |
| 000B0AH | CAN4 - Test Register Low | TESTRL4 | TESTR4 | RW |
| 000B0BH | CAN4 - Test Register High (reserved) | TESTRH4 | | R |
| 000B0CH | CAN4 - BRP Extension register Low | BRPERL4 | BRPER4 | RW |
| 000B0DH | CAN4 - BRP Extension register High (reserved) | BRPERH4 | | R |
| 000B0EH-000B0FH | Reserved | | | - |
| 000B10H | CAN4 - IF1 Command request register Low | IF1CREQL4 | IF1CREQ4 | RW |
| 000B11H | CAN4 - IF1 Command request register High | IF1CREQH4 | | RW |
| 000B12H | CAN4 - IF1 Command Mask register Low | IF1CMSKL4 | IF1CMSK4 | RW |
| 000B13H | CAN4 - IF1 Command Mask register High (reserved) | IF1CMSKH4 | | R |
| 000B14H | CAN4 - IF1 Mask 1 Register Low | IF1MSK1L4 | IF1MSK14 | RW |
| 000B15H | CAN4 - IF1 Mask 1 Register High | IF1MSK1H4 | | RW |
| 000B16H | CAN4 - IF1 Mask 2 Register Low | IF1MSK2L4 | IF1MSK24 | RW |
| 000B17H | CAN4 - IF1 Mask 2 Register High | IF1MSK2H4 | | RW |
| 000B18H | CAN4 - IF1 Arbitration 1 Register Low | IF1ARB1L4 | IF1ARB14 | RW |
| 000B19H | CAN4 - IF1 Arbitration 1 Register High | IF1ARB1H4 | | RW |
| 000B1AH | CAN4 - IF1 Arbitration 2 Register Low | IF1ARB2L4 | IF1ARB24 | RW |
| 000B1BH | CAN4 - IF1 Arbitration 2 Register High | IF1ARB2H4 | | RW |
| 000B1CH | CAN4 - IF1 Message Control Register Low | IF1MCTRL4 | IF1MCTR4 | RW |
| 000B1DH | CAN4 - IF1 Message Control Register High | IF1MCTRH4 | | RW |
| 000B1EH | CAN4 - IF1 Data A1 Low | IF1DTA1L4 | IF1DTA14 | RW |
| 000B1FH | CAN4 - IF1 Data A1 High | IF1DTA1H4 | | RW |
| 000B20H | CAN4 - IF1 Data A2 Low | IF1DTA2L4 | IF1DTA24 | RW |
| 000B21H | CAN4 - IF1 Data A2 High | IF1DTA2H4 | | RW |
| 000B22H | CAN4 - IF1 Data B1 Low | IF1DTB1L4 | IF1DTB14 | RW |
| 000B23H | CAN4 - IF1 Data B1 High | IF1DTB1H4 | | RW |
| 000B24H | CAN4 - IF1 Data B2 Low | IF1DTB2L4 | IF1DTB24 | RW |
| 000B25H | CAN4 - IF1 Data B2 High | IF1DTB2H4 | | RW |
| 000B26H-000B3FH | Reserved | | | - |
| 000B40H | CAN4 - IF2 Command request register Low | IF2CREQL4 | IF2CREQ4 | RW |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000B41H | CAN4 - IF2 Command request register High | IF2CREQH4 | | RW |
| 000B42H | CAN4 - IF2 Command Mask register Low | IF2CMSKL4 | IF2CMSK4 | RW |
| 000B43H | CAN4 - IF2 Command Mask register High (reserved) | IF2CMSKH4 | | R |
| 000B44H | CAN4 - IF2 Mask 1 Register Low | IF2MSK1L4 | IF2MSK14 | RW |
| 000B45H | CAN4 - IF2 Mask 1 Register High | IF2MSK1H4 | | RW |
| 000B46H | CAN4 - IF2 Mask 2 Register Low | IF2MSK2L4 | IF2MSK24 | RW |
| 000B47H | CAN4 - IF2 Mask 2 Register High | IF2MSK2H4 | | RW |
| 000B48H | CAN4 - IF2 Arbitration 1 Register Low | IF2ARB1L4 | IF2ARB14 | RW |
| 000B49H | CAN4 - IF2 Arbitration 1 Register High | IF2ARB1H4 | | RW |
| 000B4AH | CAN4 - IF2 Arbitration 2 Register Low | IF2ARB2L4 | IF2ARB24 | RW |
| 000B4BH | CAN4 - IF2 Arbitration 2 Register High | IF2ARB2H4 | | RW |
| 000B4CH | CAN4 - IF2 Message Control Register Low | IF2MCTRL4 | IF2MCTR4 | RW |
| 000B4DH | CAN4 - IF2 Message Control Register High | IF2MCTRH4 | | RW |
| 000B4EH | CAN4 - IF2 Data A1 Low | IF2DTA1L4 | IF2DTA14 | RW |
| 000B4FH | CAN4 - IF2 Data A1 High | IF2DTA1H4 | | RW |
| 000B50H | CAN4 - IF2 Data A2 Low | IF2DTA2L4 | IF2DTA24 | RW |
| 000B51H | CAN4 - IF2 Data A2 High | IF2DTA2H4 | | RW |
| 000B52H | CAN4 - IF2 Data B1 Low | IF2DTB1L4 | IF2DTB14 | RW |
| 000B53H | CAN4 - IF2 Data B1 High | IF2DTB1H4 | | RW |
| 000B54H | CAN4 - IF2 Data B2 Low | IF2DTB2L4 | IF2DTB24 | RW |
| 000B55H | CAN4 - IF2 Data B2 High | IF2DTB2H4 | | RW |
| 000B56H-000B7FH | Reserved | | | - |
| 000B80H | CAN4 - Transmission Request 1 Register Low | TREQR1L4 | TREQR14 | R |
| 000B81H | CAN4 - Transmission Request 1 Register High | TREQR1H4 | | R |
| 000B82H | CAN4 - Transmission Request 2 Register Low | TREQR2L4 | TREQR24 | R |
| 000B83H | CAN4 - Transmission Request 2 Register High | TREQR2H4 | | R |
| 000B84H-000B8FH | Reserved | | | - |
| 000B90H | CAN4 - New Data 1 Register Low | NEWDT1L4 | NEWDT14 | R |
| 000B91H | CAN4 - New Data 1 Register High | NEWDT1H4 | | R |
| 000B92H | CAN4 - New Data 2 Register Low | NEWDT2L4 | NEWDT24 | R |
| 000B93H | CAN4 - New Data 2 Register High | NEWDT2H4 | | R |
| 000B94H-000B9FH | Reserved | | | - |
| 000BA0H | CAN4 - Interrupt Pending 1 Register Low | INTPND1L4 | INTPND14 | R |
| 000BA1H | CAN4 - Interrupt Pending 1 Register High | INTPND1H4 | | R |
| 000BA2H | CAN4 - Interrupt Pending 2 Register Low | INTPND2L4 | INTPND24 | R |
| 000BA3H | CAN4 - Interrupt Pending 2 Register High | INTPND2H4 | | R |
| 000BA4H-000BAFH | Reserved | | | - |
| 000BB0H | CAN4 - Message Valid 1 Register Low | MSGVAL1L4 | MSGVAL14 | R |
| 000BB1H | CAN4 - Message Valid 1 Register High | MSGVAL1H4 | | R |
| 000BB2H | CAN4 - Message Valid 2 Register Low | MSGVAL2L4 | MSGVAL24 | R |

Table 37-1. I/O map of MB96V300B

| Address | Register | Abbreviation 8-bit access | Abbreviation 16-bit access | Access |
|---|---|---|---|---|
| 000BB3H | CAN4 - Message Valid 2 Register High | MSGVAL2H4 | | R |
| 000BB4H-000BCDH | Reserved | | | - |
| 000BCEH | CAN4 - Output enable register | COER4 | | RW |
| 000BCFH-000BFFH | Reserved | | | - |

### Explanation of write and read

R/W: Both read and write enabled

R: Only read enabled

W: Only write enabled

## 37.2 Instructions

Appendix B describes the instructions used by the F$^2$MC-16FX.

37.2.1 Instruction Types

37.2.2 Addressing

37.2.3 Direct Addressing

37.2.4 Indirect Addressing

37.2.5 Execution Cycle Count

37.2.6 Effective address field

37.2.7 How to Read the Instruction List

37.2.8 F$^2$MC-16FX Instruction List

37.2.9 Instruction Map

### 37.2.1 Instruction Types

The F$^2$MC-16FX supports 351 types of instructions. Addressing is enabled by using an effective address field of each instruction or using the instruction code itself.

#### Instruction types

The F$^2$MC-16FX supports the following 351 types of instructions:

- 41 transfer instructions (byte)
- 38 transfer instructions (word or long word)
- 42 addition/subtraction instructions (byte, word, or long word)
- 12 increment/decrement instructions (byte, word, or long word)
- 11 comparison instructions (byte, word, or long word)
- 11 unsigned multiplication/division instructions (word or long word)
- 11 signed multiplication/division instructions (word or long word)
- 39 logic instructions (byte or word)
- 6 logic instructions (long word)
- 6 sign inversion instructions (byte or word)
- 1 normalization instruction (long word)

- 18 shift instructions (byte, word, or long word)
- 50 branch instructions
- 6 accumulator operation instructions (byte or word)
- 28 other control instructions (byte, word, or long word)
- 21 bit operation instructions
- 10 string instructions

### 37.2.2 Addressing

With the F$^2$MC-16FX, the address format is determined by the instruction effective address field or the instruction code itself (implied). When the address format is determined by the instruction code itself, specify an address in accordance with the instruction code used. Some instructions permit the user to select several types of addressing.

#### Addressing

The F$^2$MC-16FX supports the following 23 types of addressing:

- Immediate (#imm)
- Register direct
- Direct branch address (addr16)
- Physical direct branch address (addr24)
- I/O direct (io)
- Abbreviated direct address (dir)
- Direct address (addr16)
- I/O direct bit address (io:bp)
- Abbreviated direct bit address (dir:bp)
- Direct bit address (addr16:bp)
- Vector address (#vct)
- Register indirect (@RWj  j = 0 to 3)
- Register indirect with post increment (@RWj+  j = 0 to 3)
- Register indirect with displacement (@RWi + disp8  i = 0 to 7, @RWj + disp16  j = 0 to 3)
- Long register indirect with displacement (@RLi + disp8  i = 0 to 3)
- Program counter indirect with displacement (@PC + disp16)
- Register indirect with base index (@RW0 + RW7, @RW1 + RW7)
- Program counter relative branch address (rel)
- Register list (rlst)
- Accumulator indirect (@A)
- Accumulator indirect branch address (@A)
- Indirectly-specified branch address (@ear)
- Indirectly-specified branch address (@eam)

### Effective address field

Table 37-2 lists the address formats specified by the effective address field.

Table 37-2. Effective address field

| Code | Representation | | | Address format | Default bank |
|---|---|---|---|---|---|
| 00 | R0 | RW0 | RL0 | Register direct: Individual parts correspond to the byte, word, and long word types in order from the left. | None |
| 01 | R1 | RW1 | (RL0) | | |
| 02 | R2 | RW2 | RL1 | | |
| 03 | R3 | RW3 | (RL1) | | |
| 04 | R4 | RW4 | RL2 | | |
| 05 | R5 | RW5 | (RL2) | | |
| 06 | R6 | RW6 | RL3 | | |
| 07 | R7 | RW7 | (RL3) | | |
| 08 | @RW0 | | | Register indirect | DTB |
| 09 | @RW1 | | | | DTB |
| 0A | @RW2 | | | | ADB |
| 0B | @RW3 | | | | SPB |
| 0C | @RW0+ | | | Register indirect with post increment | DTB |
| 0D | @RW1+ | | | | DTB |
| 0E | @RW2+ | | | | ADB |
| 0F | @RW3+ | | | | SPB |
| 10 | @RW0+disp8 | | | Register indirect with 8-bit displacement | DTB |
| 11 | @RW1+disp8 | | | | DTB |
| 12 | @RW2+disp8 | | | | ADB |
| 13 | @RW3+disp8 | | | | SPB |
| 14 | @RW4+disp8 | | | Register indirect with 8-bit displacement | DTB |
| 15 | @RW5+disp8 | | | | DTB |
| 16 | @RW6+disp8 | | | | ADB |
| 17 | @RW7+disp8 | | | | SPB |
| 18 | @RW0+disp16 | | | Register indirect with 16-bit displacement | DTB |
| 19 | @RW1+disp16 | | | | DTB |
| 1A | @RW2+disp16 | | | | ADB |
| 1B | @RW3+disp16 | | | | SPB |
| 1C | @RW0+RW7 | | | Register indirect with index | DTB |
| 1D | @RW1+RW7 | | | Register indirect with index | DTB |
| 1E | @PC+disp16 | | | PC indirect with 16-bit displacement | PCB |
| 1F | addr16 | | | Direct address | DTB |

## 37.2.3 Direct Addressing

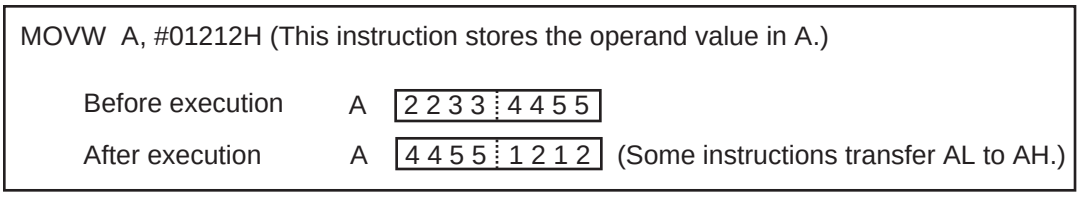
An operand value, register, or address is specified explicitly in direct addressing mode.

### *37.2.3.1 Direct addressing*

#### Immediate addressing (#imm)

Specify an operand value explicitly (#imm4/ #imm8/ #imm16/ #imm32).

Figure 37-1. Example of immediate addressing (#imm)




#### Register direct addressing

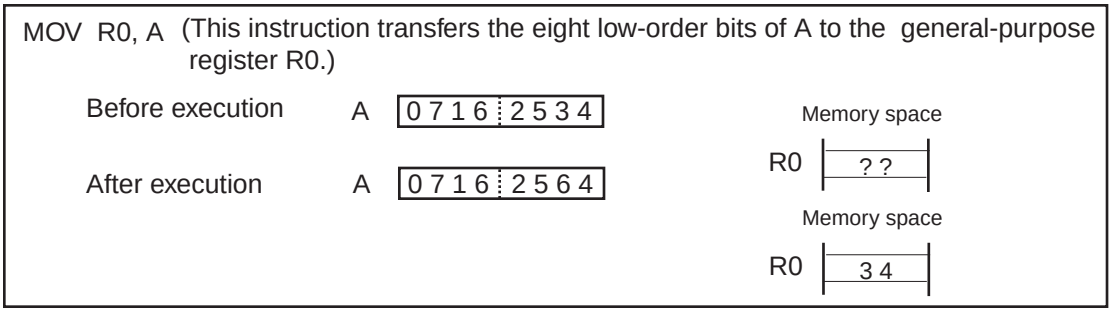
Specify a register explicitly as an operand. Table 37-3 lists the registers that can be specified. Figure 37-2 shows an example of register direct addressing.

Table 37-3. Direct addressing registers

| | | |
|---|---|---|
| General-purpose register | Byte | R0, R1, R2, R3, R4, R5, R6, R7 |
| | Word | RW0, RW1, RW2, RW3, RW4, R5W, RW6, RW7 |
| | Long word | RL0, RL1, RL2, RL3 |
| Special-purpose register | Accumulator | A, AL |
| | Pointer | SP *1 |
| | Bank | PCB, DTB, USB, SSB, ADB |
| | Page | DPR |
| | Control | PS, CCR, RP, ILM |

*1: One of the user stack pointer (USP) and system stack pointer (SSP) is selected and used depending on the value of the S flag bit in the condition code register (CCR). For branch instructions, the program counter (PC) is not specified in an instruction operand but is specified implicitly.

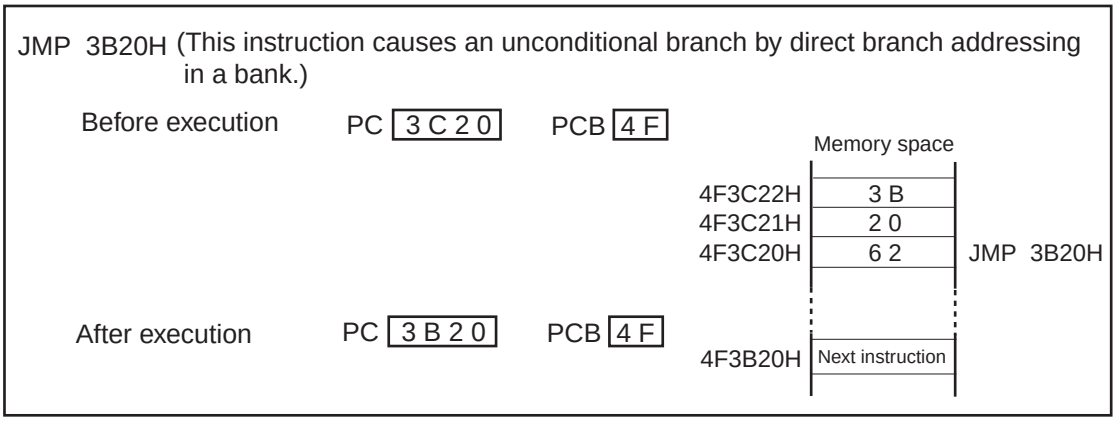
Figure 37-2. Example of register direct addressing




#### Direct branch addressing (addr16)

Specify an offset explicitly for the branch destination address. The size of the offset is 16 bits, which indicates the branch destination in the logical address space. Direct branch addressing is used for an unconditional branch, subroutine call, or software interrupt instruction. Bits 23 to 16 of the address are specified by the program bank register (PCB).

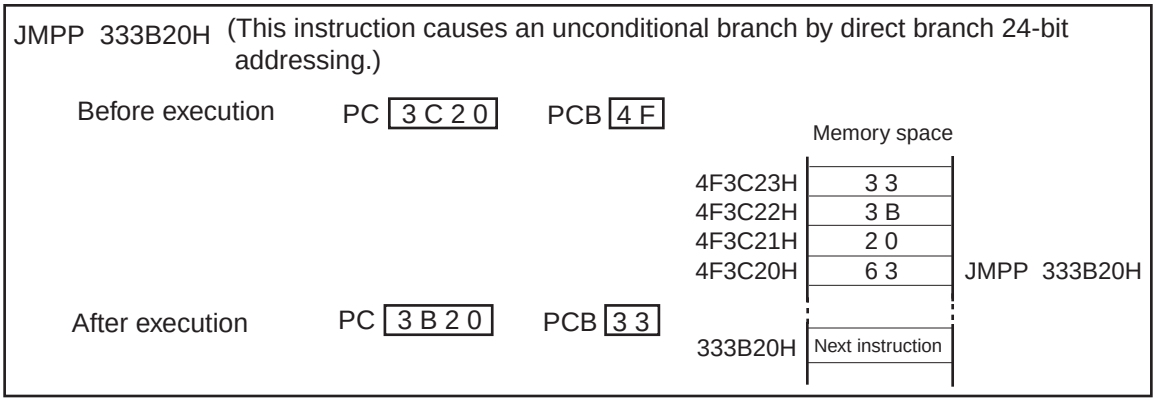
Figure 37-3. Example of direct branch addressing (addr16)




### Physical direct branch addressing (addr24)

Specify an offset explicitly for the branch destination address. The size of the offset is 24 bits. Physical direct branch addressing is used for unconditional branch, subroutine call, or software interrupt instruction.

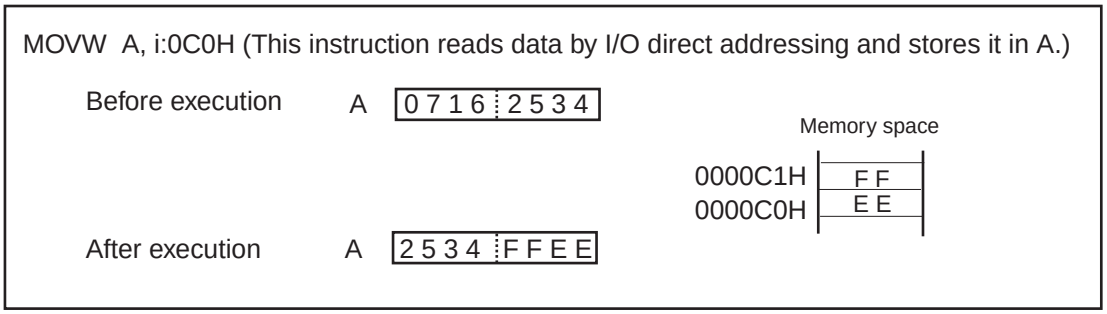
Figure 37-4. Example of direct branch addressing (addr24)




### I/O direct addressing (io)

Specify an 8-bit offset explicitly for the memory address in an operand. The I/O address space in the physical address space from 000000H to 0000FFH is accessed regardless of the data bank register (DTB) and direct page register (DPR). A bank select prefix for bank addressing is invalid if specified before an instruction using I/O direct addressing.

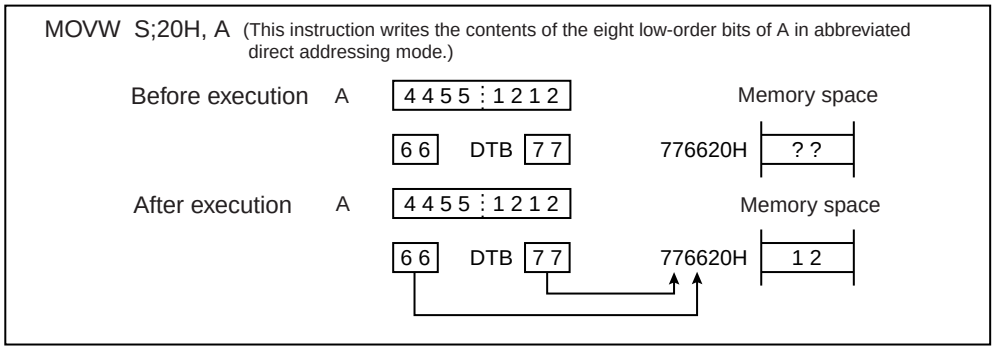
Figure 37-5. Example of I/O direct addressing (io)



### Abbreviated direct addressing (dir)

Specify the eight low-order bits of a memory address explicitly in an operand. Address bits 8 to 15 are specified by the direct page register (DPR). Address bits 16 to 23 are specified by the data bank register (DTB).

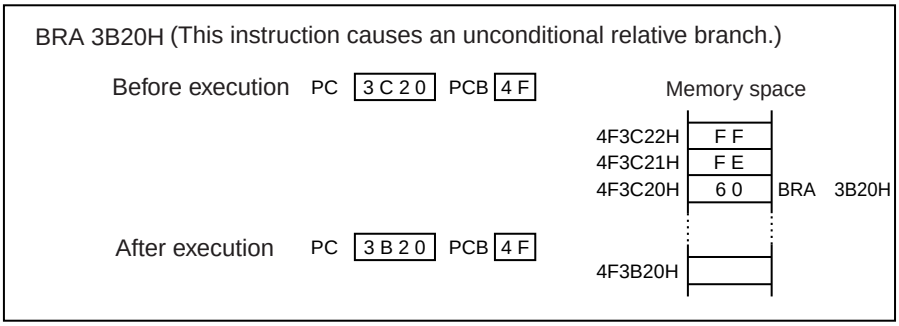
Figure 37-6. Example of abbreviated direct addressing (dir)




### Direct addressing (addr16)

Specify the 16 low-order bits of a memory address explicitly in an operand. Address bits 16 to 23 are specified by the data bank register (DTB). A prefix instruction for access space addressing is invalid for this mode of addressing.

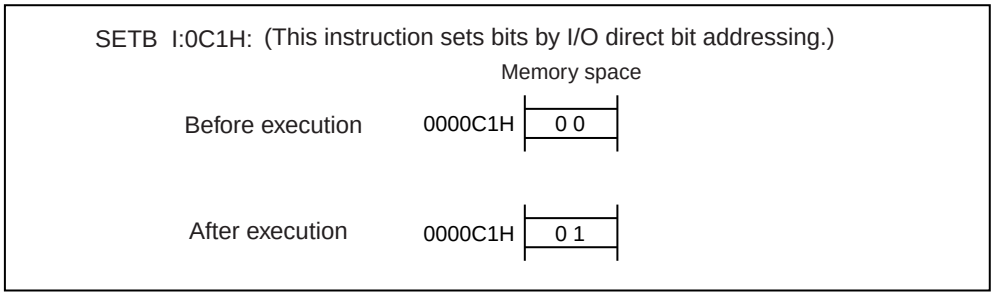
Figure 37-7. Example of direct addressing (addr16)




### I/O direct bit addressing (io:bp)

Specify bits in physical addresses 000000H to 0000FFH explicitly. Bit positions are indicated by ":bp", where the larger number indicates the most significant bit (MSB) and the lower number indicates the least significant bit (LSB).

Figure 37-8. Example of I/O direct bit addressing (io:bp)

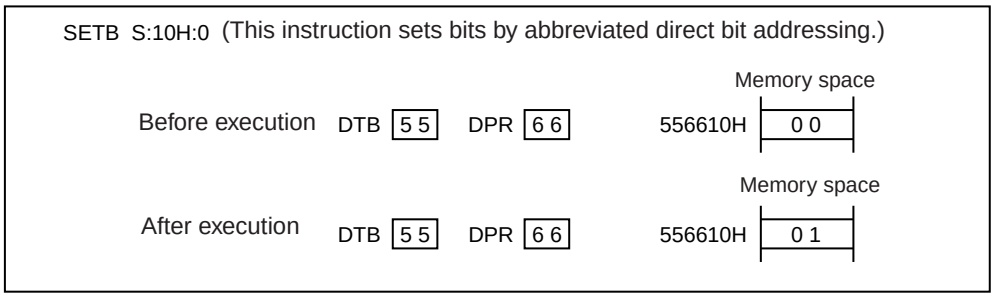



### Abbreviated direct bit addressing (dir:bp)

Specify the eight low-order bits of a memory address explicitly in an operand. Address bits 8 to 15 are specified by the direct page register (DPR). Address bits 16 to 23 are specified by the data bank register (DTB). Bit positions are indicated by ":bp",

where the larger number indicates the most significant bit (MSB) and the lower number indicates the least significant bit (LSB).

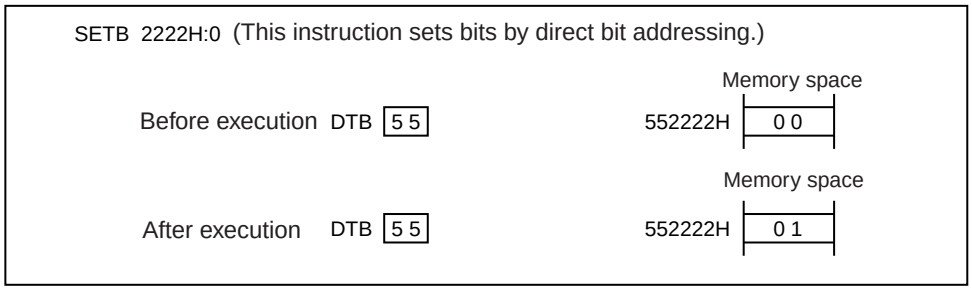
Figure 37-9. Example of abbreviated direct bit addressing (dir:bp)




### Direct bit addressing (addr16:bp)

Specify arbitrary bits in 64 kilobytes explicitly. Address bits 16 to 23 are specified by the data bank register (DTB). Bit positions are indicated by ":bp", where the larger number indicates the most significant bit (MSB) and the lower number indicates the least significant bit (LSB).

Figure 37-10. Example of direct bit addressing (addr16:bp)




### Vector Addressing (#vct)

Specify vector data in an operand to indicate the branch destination address. There are two sizes for vector numbers: 4 bits and 8 bits. Vector addressing is used for a subroutine call or software interrupt instruction.

Figure 37-11. Example of vector addressing (#vct)

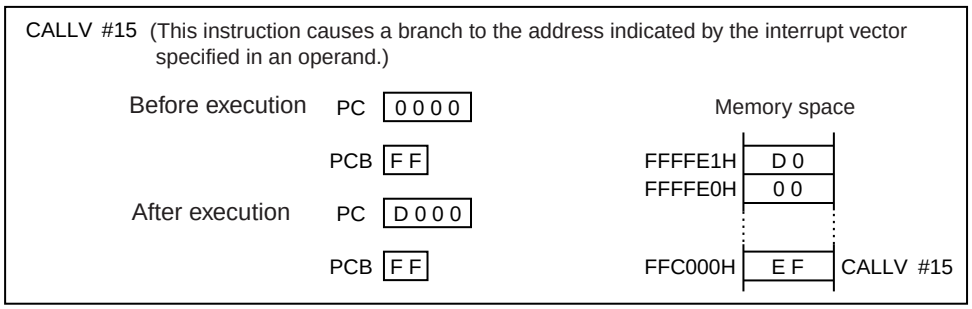


Table 37-4. CALLV vector list

| Instruction | Vector address L | Vector address H |
|---|---|---|
| CALLV #0 | $XXFFFE_H$ | $XXFFFF_H$ |
| CALLV #1 | $XXFFFC_H$ | $XXFFFD_H$ |
| CALLV #2 | $XXFFFA_H$ | $XXFFFB_H$ |
| CALLV #3 | $XXFFF8_H$ | $XXFFF9_H$ |
| CALLV #4 | $XXFFF6_H$ | $XXFFF7_H$ |
| CALLV #5 | $XXFFF4_H$ | $XXFFF5_H$ |
| CALLV #6 | $XXFFF2_H$ | $XXFFF3_H$ |
| CALLV #7 | $XXFFF0_H$ | $XXFFF1_H$ |
| CALLV #8 | $XXFFEE_H$ | $XXFFEF_H$ |
| CALLV #9 | $XXFFEC_H$ | $XXFFED_H$ |
| CALLV #10 | $XXFFEA_H$ | $XXFFEB_H$ |
| CALLV #11 | $XXFFE8_H$ | $XXFFE9_H$ |
| CALLV #12 | $XXFFE6_H$ | $XXFFE7_H$ |
| CALLV #13 | $XXFFE4_H$ | $XXFFE5_H$ |
| CALLV #14 | $XXFFE2_H$ | $XXFFE3_H$ |
| CALLV #15 | $XXFFE0_H$ | $XXFFE1_H$ |

Note: A PCB register value is set in XX.

**Note:**

When the program bank register (PCB) is $FF_H$, the vector area overlaps the vector area of INT #vct8 (#0 to #7). Use vector addressing carefully (see Table 37-4).

## 37.2.4 Indirect Addressing

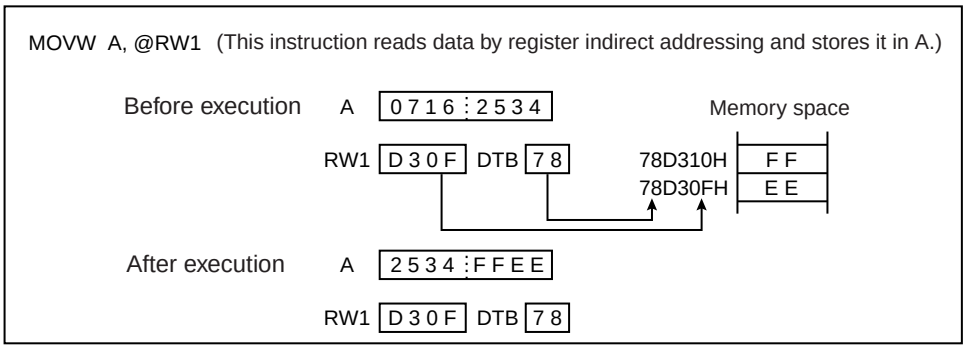
In indirect addressing mode, an address is specified indirectly by the address data of an operand.

### *37.2.4.1 Indirect addressing*

**Register indirect addressing (@RWj  j = 0 to 3)**

Memory is accessed using the contents of general-purpose register RWj as an address. Address bits 16 to 23 are indicated by the data bank register (DTB) when RW0 or RW1 is used, system stack bank register (SSB) or user stack bank register (USB) when RW3 is used, or additional data bank register (ADB) when RW2 is used.

Figure 37-12. Example of register indirect addressing (@RWj j = 0 to 3)

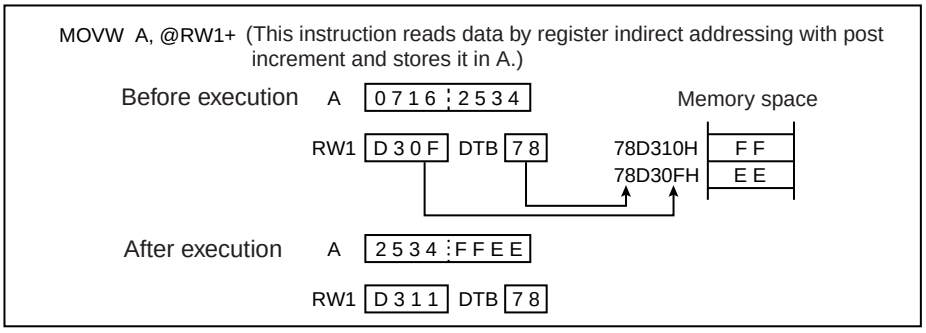



### Register indirect addressing with post increment (@RWj+ j = 0 to 3)

Memory is accessed using the contents of general-purpose register RWj as an address. After operand operation, RWj is incremented by the operand size (1 for a byte, 2 for a word, or 4 for a long word). Address bits 16 to 23 are indicated by the data bank register (DTB) when RW0 or RW1 is used, system stack bank register (SSB) or user stack bank register (USB) when RW3 is used, or additional data bank register (ADB) when RW2 is used.

If the post increment results in the address of the register that specifies the increment, the incremented value is referenced after that. In this case, if the next instruction is a write instruction, priority is given to writing by an instruction and, therefore, the register that would be incremented becomes write data.

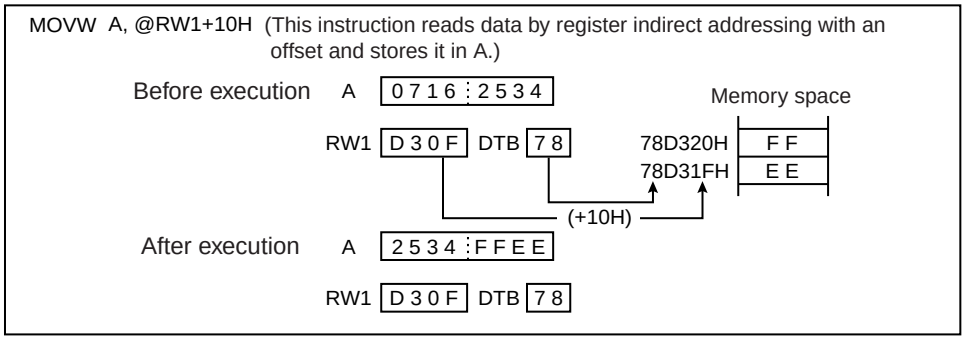
Figure 37-13. Example of register indirect addressing with post increment (@RWj+ j = 0 to 3)




### Register indirect addressing with offset (@RWi + disp8 i = 0 to 7, @RWj + disp16 j = 0 to 3)

Memory is accessed using the address obtained by adding an offset to the contents of general-purpose register RWj. Two types of offset, byte and word offsets, are used. They are added as signed numeric values. Address bits 16 to 23 are indicated by the data bank register (DTB) when RW0, RW1, RW4, or RW5 is used, system stack bank register (SSB) or user stack bank register (USB) when RW3 or RW7 is used, or additional data bank register (ADB) when RW2 or RW6 is used.

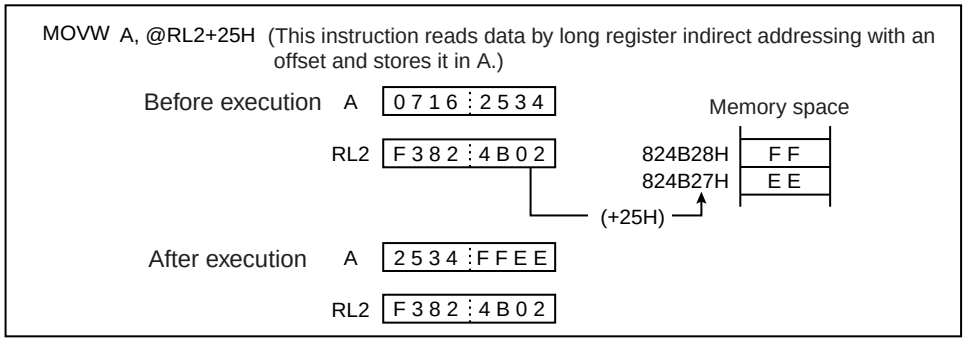
Figure 37-14. Example of register indirect addressing with offset (@RWi + disp8 i = 0 to 7, @RWj + disp16 j = 0 to 3)




### Long register indirect addressing with offset (@RLi + disp8 i = 0 to 3)

Memory is accessed using the address that is the 24 low-order bits obtained by adding an offset to the contents of general-purpose register RLi. The offset is 8-bits long and is added as a signed numeric value.

Figure 37-15. Example of long register indirect addressing with offset (@RLi + disp8 i = 0 to 3)

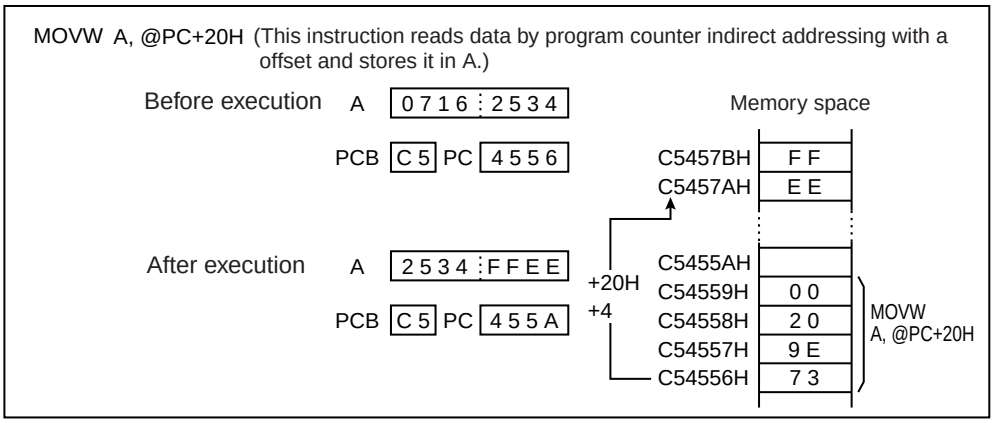



### Program counter indirect addressing with offset (@PC + disp16)

Memory is accessed using the address indicated by (instruction address + 4 + disp16). The offset is one word long. Address bits 16 to 23 are specified by the program bank register (PCB). Note that the operand address of each of the following instructions is not deemed to be (next instruction address + disp16):

- DBNZ eam, rel
- DWBNZ eam, rel
- CBNE eam, #imm8, rel
- CWBNE eam, #imm16, rel
- MOV eam, #imm8
- MOVW eam, #imm16

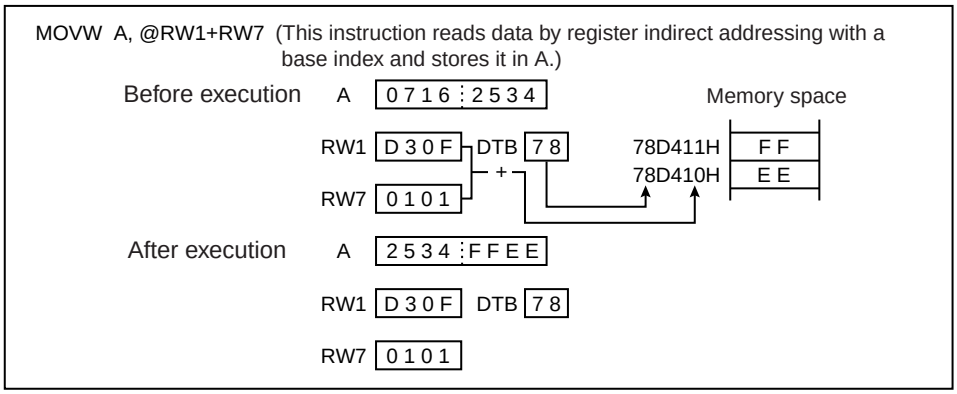
Figure 37-16. Example of program counter indirect addressing with offset (@PC + disp16)




## Register indirect addressing with base index (@RW0 + RW7, @RW1 + RW7)

Memory is accessed using the address determined by adding RW0 or RW1 to the contents of general-purpose register RW7. Address bits 16 to 23 are indicated by the data bank register (DTB).

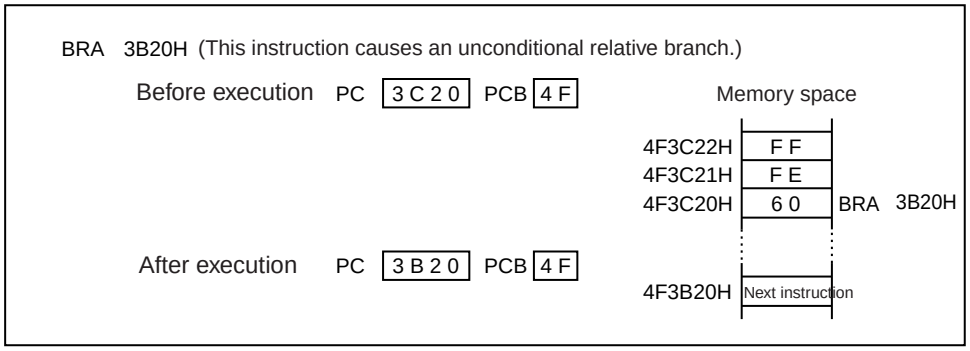
Figure 37-17. Example of register indirect addressing with base index (@RW0 + RW7, @RW1 + RW7)




## Program counter relative branch addressing (rel)

The address of the branch destination is a value determined by adding an 8-bit offset to the program counter (PC) value. If the result of addition exceeds 16 bits, bank register incrementing or decrementing is not performed and the excess part is ignored, and therefore the address is contained within a 64-kilobyte bank. This addressing is used for both conditional and unconditional branch instructions. Address bits 16 to 23 are indicated by the program bank register (PCB).

Figure 37-18. Example of program counter relative branch addressing (rel)



### Register list (rlst)

Specify a register to be pushed onto or popped from a stack.

Figure 37-19. Configuration of the register list

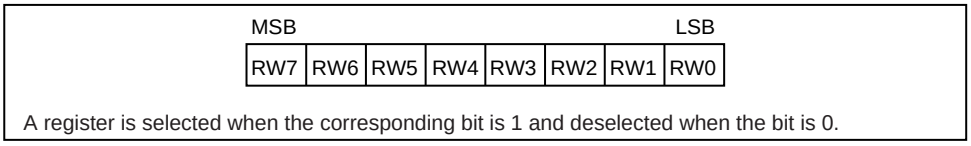
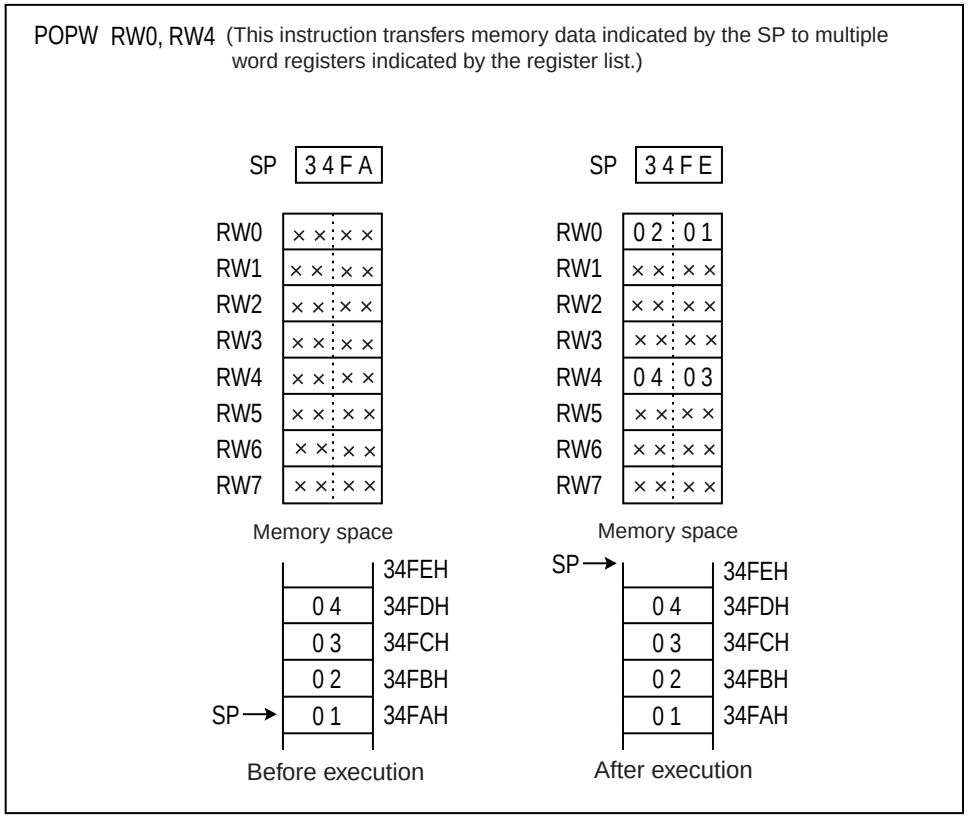


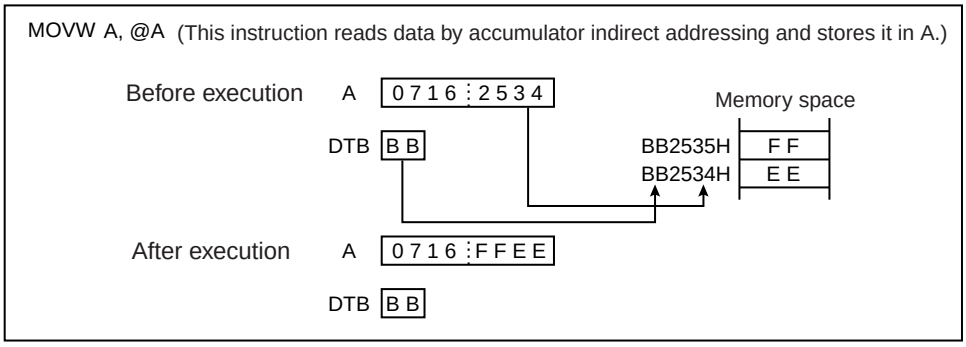

Figure 37-20. Example of register list (rlist)




### Accumulator indirect addressing (@A)

Memory is accessed using the address indicated by the contents of the low-order bytes (16 bits) of the accumulator (AL). Address bits 16 to 23 are specified by a mnemonic in the data bank register (DTB).

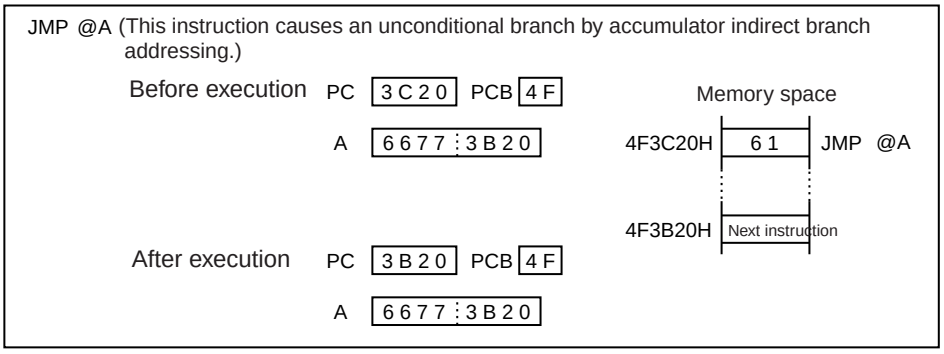
Figure 37-21. Example of accumulator indirect addressing (@A)




### Accumulator indirect branch addressing (@A)

The address of the branch destination is the content (16 bits) of the low-order bytes (AL) of the accumulator. It indicates the branch destination in the bank address space. Address bits 16 to 23 are specified by the program bank register (PCB). For the Jump Context (JCTX) instruction, however, address bits 16 to 23 are specified by the data bank register (DTB). This addressing is used for unconditional branch instructions.

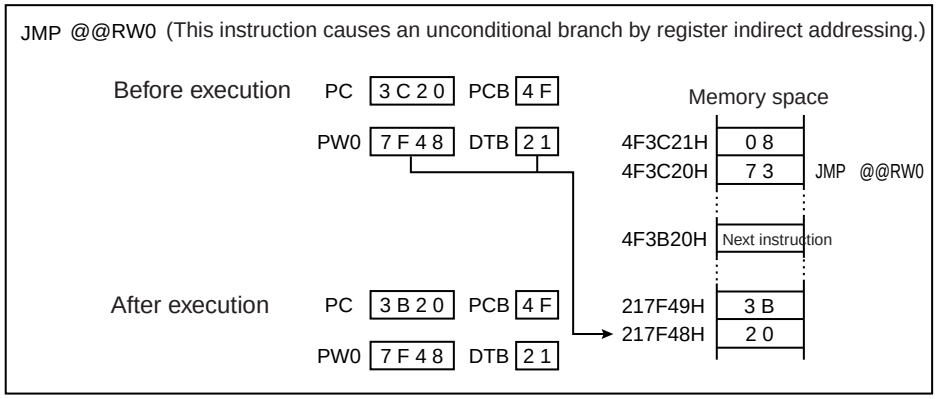
Figure 37-22. Example of accumulator indirect branch addressing (@A)




### Indirect specification branch addressing (@ear)

The address of the branch destination is the word data at the address indicated by ear.

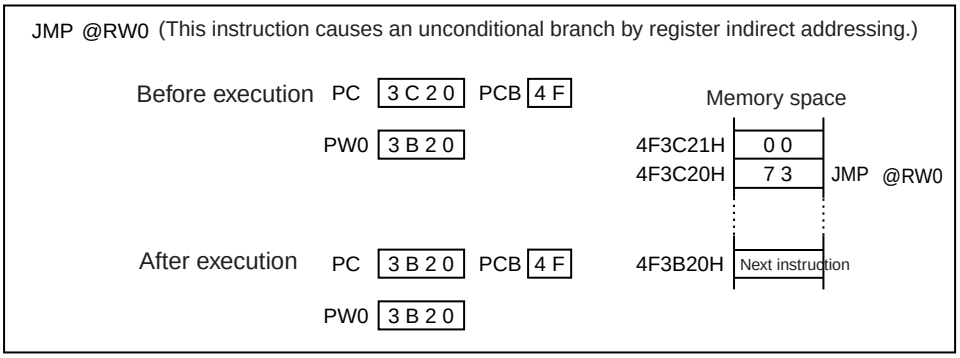
Figure 37-23. Example of indirect specification branch addressing (@ear)




### Indirect specification branch addressing (@eam)

The address of the branch destination is the word data at the address indicated by eam.

Figure 37-24. Example of indirect specification branch addressing (@eam)




## 37.2.5 Execution Cycle Count

The execution cycle count for 16FX is TBD and will be shown here in a future version of the hardware manual.

## 37.2.6 Effective address field

Table 37-5 shows the effective address field.

**Effective Address Field**

Table 37-5. Effective address field

| Code | Representation | | | Address format | Byte count of extended address part |
|---|---|---|---|---|---|
| 00 | R0 | RW0 | RL0 | Register direct: Individual parts correspond to the byte, word, and long word types in order from the left. | - |
| 01 | R1 | RW1 | (RL0) | | |
| 02 | R2 | RW2 | RL1 | | |
| 03 | R3 | RW3 | (RL1) | | |
| 04 | R4 | RW4 | RL2 | | |
| 05 | R5 | RW5 | (RL2) | | |
| 06 | R6 | RW6 | RL3 | | |
| 07 | R7 | RW7 | (RL3) | | |
| 08 | @RW0 | | | Register indirect | 0 |
| 09 | @RW1 | | | | |
| 0A | @RW2 | | | | |
| 0B | @RW3 | | | | |
| 0C | @RW0+ | | | Register indirect with post increment | 0 |
| 0D | @RW1+ | | | | |
| 0E | @RW2+ | | | | |
| 0F | @RW3+ | | | | |

Table 37-5. Effective address field<Italic> (continued)

| Code | Representation | Address format | Byte count of extended address part |
|---|---|---|---|
| 10 | @RW0+disp8 | Register indirect with 8-bit displacement | 1 |
| 11 | @RW1+disp8 | | |
| 12 | @RW2+disp8 | | |
| 13 | @RW3+disp8 | | |
| 14 | @RW4+disp8 | | |
| 15 | @RW5+disp8 | | |
| 16 | @RW6+disp8 | | |
| 17 | @RW7+disp8 | | |
| 18 | @RW0+disp16 | Register indirect with 16-bit displacement | 2 |
| 19 | @RW1+disp16 | | |
| 1A | @RW2+disp16 | | |
| 1B | @RW3+disp16 | | |
| 1C | @RW0+RW7 | Register indirect with index | 0 |
| 1D | @RW1+RW7 | Register indirect with index | 0 |
| 1E | @PC+disp16 | PC indirect with 16-bit displacement | 2 |
| 1F | addr16 | Direct address | 2 |

*1: Each byte count of the extended address part applies to + in the # (byte count) column in 37.2.8 F$^2$MC-16FX Instruction List.

## 37.2.7 How to Read the Instruction List

Table 37-6 describes the items used in the F$^2$MC-16FX Instruction List, and Table 37-7 describes the symbols used in the same list.

### 37.2.7.1 Description of instruction presentation items and symbols

Table 37-6. Description of items in the instruction list

| Item | Description |
|---|---|
| Mnemonic | Uppercase, symbol: Represented as is in the assembler.<br>Lowercase: Rewritten in the assembler.<br>Number of following lowercase: Indicates bit length in the instruction. |
| # | Indicates the number of bytes. |
| ~ | Indicates the number of cycles.<br>See Table 37-2 for the alphabetical letters in items. |
| B | Indicates the correction value used to calculate the actual number of cycles during instruction execution.<br>The actual number of cycles during instruction execution can be determined by adding the value in the ~ column to this value. |
| Operation | Indicates the instruction operation. |
| LH | Indicates the special operation for bits 15 to 08 of the accumulator.<br>Z: Transfers 0.<br>X: Transfers after sign extension.<br>-: No transfer |
| AH | Indicates the special operation for the 16 high-order bits of the accumulator.<br>*: Transfers from AL to AH.<br>-: No transfer<br>Z: Transfers 00 to AH.<br>X: Transfers $00_H$ or $FF_H$ to AH after AL sign extension. |
| I | Each indicates the state of each flag: I (interrupt enable), S (stack), T (sticky bit), N (negative), Z (zero), V (overflow), C (carry).<br>*: Changes upon instruction execution.<br>-: No change<br>Z: Set upon instruction execution.<br>X: Reset upon instruction execution. |
| S | |
| T | |
| N | |
| Z | |
| V | |
| C | |
| RMW | Indicates whether the instruction is a Read Modify Write instruction (reading data from memory by the I instruction and writing the result to memory).<br>*: Read Modify Write instruction<br>-: Not Read Modify Write instruction<br>**Note:**<br>Cannot be used for an address that has different meanings between read and write operations. |

Table 37-7. Explanation on symbols in the instruction List

| Symbol | Explanation |
|---|---|
| A | The bit length used varies depending on the 32-bit accumulator instruction.<br>Byte: Low-order 8 bits of byte AL<br>Word: 16 bits of word AL<br>Long word: 32 bits of AL and AH |
| AH | 16 high-order bits of A |
| AL | 16 low-order bits of A |
| SP | Stack pointer (USP or SSP) |

Table 37-7. Explanation on symbols in the instruction List<Italic> (continued)

| Symbol | Explanation |
|---|---|
| PC | Program counter |
| PCB | Program bank register |
| DTB | Data bank register |
| ADB | Additional data bank register |
| SSB | System stack bank register |
| USB | User stack bank register |
| SPB | Current stack bank register (SSB or USB) |
| DPR | Direct page register |
| brg1 | DTB, ADB, SSB, USB, DPR, PCB, SPB |
| brg2 | DTB, ADB, SSB, USB, DPR, SPB |
| brg3 | DTB, ADB, PCB, SPB |
| Ri | R0, R1, R2, R3, R4, R5, R6, R7 |
| RWi | RW0, RW1, RW2, RW3, RW4, RW5, RW6, RW7 |
| RWj | RW0, RW1, RW2, RW3 |
| RLi | RL0, RL1, RL2, RL3 |
| dir | Abbreviated direct addressing |
| addr16 | Direct addressing |
| addr24 | Physical direct addressing |
| ad24 0-15 | Bits 0 to 15 of addr24 |
| ad24 16-23 | Bits 16 to 23 of addr24 |
| io | I/O area (000000H to 0000FFH) |
| #imm4 | 4-bit immediate data |
| #imm8 | 8-bit immediate data |
| #imm16 | 16-bit immediate data |
| #imm32 | 32-bit immediate data |
| ext (imm8) | 16-bit data obtained by sign extension of 8-bit immediate data |
| disp8 | 8-bit displacement |
| disp16 | 16-bit displacement |
| bp | Bit offset |
| vct4 | Vector number (0 to 15) |
| vct8 | Vector number (0 to 255) |
| ( ) b | Bit address |
| rel | PC relative branch |
| ear | Effective addressing (code 00 to 07) |
| eam | Effective addressing (code 08 to 1F) |
| rlst | Register list |

## 37.2.8 F²MC-16FX Instruction List

Table 37-8 to Table 37-25 list the instructions used by the F²MC-16FX.

### F²MC-16FX instruction list

Table 37-8. 41 Transfer instructions (byte)

| Mnemonic | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOV A,dir | 2 | | | byte (A) <-- (dir) | Z | * | - | - | * | * | - | - | - | - |
| MOV A,addr16 | 3 | | | byte (A) <-- (addr16) | Z | * | - | - | * | * | - | - | - | - |
| MOV A,Ri | 1 | | | byte (A) <-- (Ri) | Z | * | - | - | * | * | - | - | - | - |
| MOV A,ear | 2 | | | byte (A) <-- (ear) | Z | * | - | - | * | * | - | - | - | - |
| MOV A,eam | 2+ | | | byte (A) <-- (eam) | Z | * | - | - | * | * | - | - | - | - |
| MOV A,io | 2 | | | byte (A) <-- (io) | Z | * | - | - | * | * | - | - | - | - |
| MOV A,#imm8 | 2 | | | byte (A) <-- imm8 | Z | * | - | - | * | * | - | - | - | - |
| MOV A,@A | 2 | | | byte (A) <-- ((A)) | Z | - | - | - | * | * | - | - | - | - |
| MOV A,@RLi+disp8 | 3 | | | byte (A) <-- ((RLi)+disp8) | Z | * | - | - | * | * | - | - | - | - |
| MOVN A,#imm4 | 1 | | | byte (A) <-- imm4 | Z | * | - | - | R | * | - | - | - | - |
| MOVX A,dir | 2 | | | byte (A) <-- (dir) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,addr16 | 3 | | | byte (A) <-- (addr16) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,Ri | 2 | | | byte (A) <-- (Ri) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,ear | 2 | | | byte (A) <-- (ear) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,eam | 2+ | | | byte (A) <-- (eam) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,io | 2 | | | byte (A) <-- (io) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,#imm8 | 2 | | | byte (A) <-- imm8 | X | * | - | - | - | * | * | - | - | - |
| MOVX A,@A | 2 | | | byte (A) <-- ((A)) | X | - | - | - | - | * | * | - | - | - |
| MOVX A,@RWi+disp8 | 2 | | | byte (A) <-- ((RWi)+disp8) | X | * | - | - | - | * | * | - | - | - |
| MOVX A,@RLi+disp8 | 3 | | | byte (A) <-- ((RLi)+disp8 | X | * | - | - | - | * | * | - | - | - |
| MOV dir,A | 2 | | | byte (dir) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV addr16,A | 3 | | | byte (addr16) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV Ri,A | 1 | | | byte (Ri) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV ear,A | 2 | | | byte (ear) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV eam,A | 2+ | | | byte (eam) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV io,A | 2 | | | byte (io) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV @RLi+disp8,A | 3 | | | byte ((RLi)+disp8) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOV Ri,ear | 2 | | | byte (Ri) <-- (ear) | - | - | - | - | - | * | * | - | - | - |
| MOV Ri,eam | 2+ | | | byte (Ri) <-- (eam) | - | - | - | - | - | * | * | - | - | - |
| MOV ear,Ri | 2 | | | byte (ear) <-- (Ri) | - | - | - | - | - | * | * | - | - | - |
| MOV eam,Ri | 2+ | | | byte (eam) <-- (Ri) | - | - | - | - | - | * | * | - | - | - |
| MOV Ri,#imm8 | 2 | | | byte (Ri) <-- imm8 | - | - | - | - | - | * | * | - | - | - |
| MOV io,#imm8 | 3 | | | byte (io) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV dir,#imm8 | 3 | | | byte (dir) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV ear,#imm8 | 3 | | | byte (ear) <-- imm8 | - | - | - | - | - | * | * | - | - | - |
| MOV eam,#imm8 | 3+ | | | byte (eam) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV @AL,AH / MOV @A,T | 2 | | | byte ((A)) <-- (AH) | - | - | - | - | - | * | * | - | - | - |
| XCH A,ear | 2 | | | byte (A) <--> (ear) | Z | - | - | - | - | - | - | - | - | - |
| XCH A,eam | 2+ | | | byte (A) <--> (eam) | Z | - | - | - | - | - | - | - | - | - |
| XCH Ri,ear | 2 | | | byte (Ri) <--> (ear) | - | - | - | - | - | - | - | - | - | - |
| XCH Ri,eam | 2+ | | | byte (Ri) <--> (eam) | - | - | - | - | - | - | - | - | - | - |

Table 37-9. 38 Transfer instructions (word / long word)

| Mnemonic | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOVW A,dir | 2 | | | word (A) <-- (dir) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,addr16 | 3 | | | word (A) <-- (addr16) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,SP | 3 | | | word (A) <-- (SP) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,RWi | 1 | | | word (A) <-- (RWi) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,ear | 2 | | | word (A) <-- (ear) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,eam | 2+ | | | word (A) <-- (eam) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,io | 2 | | | word (A) <-- (io) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,@A | 2 | | | word (A) <-- ((A)) | - | - | - | - | - | * | * | - | - | - |
| MOVW A,#imm16 | 3 | | | word (A) <-- imm16 | - | * | - | - | - | * | * | - | - | - |
| MOVW A,@RWi+disp8 | 2 | | | word (A) <-- ((RWi)+disp8) | - | * | - | - | - | * | * | - | - | - |
| MOVW A,@RLi+disp8 | 3 | | | word (A) <-- ((RLi)+disp8) | - | * | - | - | - | * | * | - | - | - |
| MOVW dir,A | 2 | | | word (dir) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW addr16,A | 3 | | | word (addr16) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW SP,A | 1 | | | word (SP) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW RWi,A | 1 | | | word (RWi) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW ear,A | 2 | | | word (ear) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW eam,A | 2+ | | | word (eam) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW io,A | 2 | | | word (io) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW @RWi+disp8,A | 2 | | | word ((RWi)+disp8) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW @RLi+disp8,A | 3 | | | word ((RLi)+disp8) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVW RWi,ear | 2 | | | word (RWi) <-- (ear) | - | - | - | - | - | * | * | - | - | - |
| MOVW | 2+ | | | word (RWi) <-- (eam) | - | - | - | - | - | * | * | - | - | - |
| MOVW ear,Rwi | 2 | | | word (ear) <-- (RWi) | - | - | - | - | - | * | * | - | - | - |
| MOVW eam,Rwi | 2+ | | | word (eam) <-- (RWi) | - | - | - | - | - | * | * | - | - | - |
| MOVW RWi,#imm16 | 3 | | | word (RWi) <-- imm16 | - | - | - | - | - | * | * | - | - | - |
| MOVW io,#imm16 | 4 | | | word (io) <-- imm16 | - | - | - | - | - | - | - | - | - | - |
| MOVW ear,#imm16 | 4 | | | word (ear) <-- imm16 | - | - | - | - | - | * | * | - | - | - |
| MOVW eam,#imm16 | 4+ | | | word (eam) <-- imm16 | - | - | - | - | - | - | - | - | - | - |
| MOVW @AL,AH / MOVW @A,T | 2 | | | word ((A)) <-- (AH) | - | - | - | - | - | * | * | - | - | - |
| XCHW A,ear | 2 | | | word (A) <--> (ear) | - | - | - | - | - | - | - | - | - | - |
| XCHW A,eam | 2+ | | | word (A) <-- >(eam) | - | - | - | - | - | - | - | - | - | - |
| XCHW RWi, ear | 2 | | | word (RWi) <--> (ear) | - | - | - | - | - | - | - | - | - | - |
| XCHW RWi, eam | 2+ | | | word (RWi) <--> (eam) | - | - | - | - | - | - | - | - | - | - |
| MOVL A,ear | 2 | | | long (A) <-- (ear) | - | - | - | - | - | * | * | - | - | - |
| MOVL A,eam | 2+ | | | long (A) <-- (eam) | - | - | - | - | - | * | * | - | - | - |
| MOVL A,#imm32 | 5 | | | long (A) <-- imm32 | - | - | - | - | - | * | * | - | - | - |
| MOVL ear,A | 2 | | | long (ear1) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| MOVL eam,A | 2+ | | | long(eam1) <-- (A) | - | - | - | - | - | * | * | - | - | - |

Table 37-10. 42 Addition/subtraction instructions (byte, word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD | A,#imm8 | 2 | | | byte (A) <-- (A) + imm8 | Z | - | - | - | - | * | * | * | * | - |
| ADD | A,dir | 2 | | | byte (A) <-- (A) + (dir) | Z | - | - | - | - | * | * | * | * | - |
| ADD | A,ear | 2 | | | byte (A) <-- (A) + (ear) | Z | - | - | - | - | * | * | * | * | - |
| ADD | A,eam | 2+ | | | byte (A) <-- (A) + (eam) | Z | - | - | - | - | * | * | * | * | - |
| ADD | ear,A | 2 | | | byte (ear) <-- (ear) + (A) | - | - | - | - | - | * | * | * | * | - |
| ADD | eam,A | 2+ | | | byte (eam) <-- (eam) + (A) | Z | - | - | - | - | * | * | * | * | * |
| ADDC | A | 1 | | | byte (A) <-- (AH) + (AL) + (C) | Z | - | - | - | - | * | * | * | * | - |
| ADDC | A,ear | 2 | | | byte (A) <-- (A) + (ear)+ (C) | Z | - | - | - | - | * | * | * | * | - |
| ADDC | A,eam | 2+ | | | byte (A) <-- (A) + (eam)+ (C) | Z | - | - | - | - | * | * | * | * | - |
| ADDDC | A | 1 | | | byte (A) <-- (AH) + (AL) + (C) (decimal) | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,#imm8 | 2 | | | byte (A) <-- (A) - imm8 | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,dir | 2 | | | byte (A) <-- (A) - (dir) | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,ear | 2 | | | byte (A) <-- (A) - (ear) | Z | - | - | - | - | * | * | * | * | - |
| SUB | A,eam | 2+ | | | byte (A) <-- (A) - (eam) | Z | - | - | - | - | * | * | * | * | - |
| SUB | ear,A | 2 | | | byte (ear) <-- (ear) - (A) | - | - | - | - | - | * | * | * | * | - |
| SUB | eam,A | 2+ | | | byte (eam) <-- (eam) - (A) | - | - | - | - | - | * | * | * | * | * |
| SUBC | A | 1 | | | byte (A) <-- (AH) - (AL) - (C) | Z | - | - | - | - | * | * | * | * | - |
| SUBC | A,ear | 2 | | | byte (A) <-- (A) - (ear) - (C) | Z | - | - | - | - | * | * | * | * | - |
| SUBC | A,eam | 2+ | | | byte (A) <-- (A) - (eam) - (C) | Z | - | - | - | - | * | * | * | * | - |
| SUBDC | A | 1 | | | byte (A) <-- (AH) - (AL) - (C) (decimal) | Z | - | - | - | - | * | * | * | * | - |
| ADDW | A | 1 | | | word (A) <-- (AH) + (AL) | - | - | - | - | - | * | * | * | * | - |
| ADDW | A,ear | 2 | | | word (A) <-- (A) + (ear) | - | - | - | - | - | * | * | * | * | - |
| ADDW | A,eam | 2+ | | | word (A) <-- (A) + (eam) | - | - | - | - | - | * | * | * | * | - |
| ADDW | A,#imm16 | 3 | | | word (A) <-- (A) + imm16 | - | - | - | - | - | * | * | * | * | - |
| ADDW | ear,A | 2 | | | word (ear) <-- (ear) + (A) | - | - | - | - | - | * | * | * | * | - |
| ADDW | eam,A | 2+ | | | word (eam) <-- (eam) + (A) | - | - | - | - | - | * | * | * | * | * |
| ADDCW | A,ear | 2 | | | word (A) <-- (A) + (ear) + (C) | - | - | - | - | - | * | * | * | * | - |
| ADDCW | A,eam | 2+ | | | word (A) <-- (A) + (eam) + (C) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A | 1 | | | word (A) <-- (AH) - (AL) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A,ear | 2 | | | word (A) <-- (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A,eam | 2+ | | | word (A) <-- (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| SUBW | A,#imm16 | 3 | | | word (A) <-- (A) - imm16 | - | - | - | - | - | * | * | * | * | - |
| SUBW | ear,A | 2 | | | word (ear) <-- (ear) - (A) | - | - | - | - | - | * | * | * | * | - |
| SUBW | eam,A | 2+ | | | word (eam) <-- (eam) - (A) | - | - | - | - | - | * | * | * | * | * |
| SUBCW | A,ear | 2 | | | word (A) <-- (A) - (ear) - (C) | - | - | - | - | - | * | * | * | * | - |
| SUBCW | A,eam | 2+ | | | word (A) <-- (A) - (eam) - (C) | - | - | - | - | - | * | * | * | * | - |
| ADDL | A,ear | 2 | | | long (A) <-- (A) + (ear) | - | - | - | - | - | * | * | * | * | - |
| ADDL | A,eam | 2+ | | | long (A) <-- (A) + (eam) | - | - | - | - | - | * | * | * | * | - |
| ADDL | A,#imm32 | 5 | | | long (A) <-- (A) + imm32 | - | - | - | - | - | * | * | * | * | - |
| SUBL | A,ear | 2 | | | long (A) <-- (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| SUBL | A,eam | 2+ | | | long (A) <-- (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| SUBL | A,#imm32 | 5 | | | long (A) <-- (A) - imm32 | - | - | - | - | - | * | * | * | * | - |

Table 37-11. 12 Increment/decrement instructions (byte, word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INC | ear | 2 | | | byte (ear) <-- (ear) + 1 | - | - | - | - | - | * | * | * | - | - |
| INC | eam | 2+ | | | byte (eam) <-- (eam) + 1 | - | - | - | - | - | * | * | * | - | * |
| DEC | ear | 2 | | | byte (ear) <-- (ear) - 1 | - | - | - | - | - | * | * | * | - | - |
| DEC | eam | 2+ | | | byte (eam) <-- (eam) - 1 | - | - | - | - | - | * | * | * | - | * |
| INCW | ear | 2 | | | word (ear) <-- (ear) + 1 | - | - | - | - | - | * | * | * | - | - |
| INCW | eam | 2+ | | | word (eam) <-- (eam) + 1 | - | - | - | - | - | * | * | * | - | * |
| DECW | ear | 2 | | | word (ear) <-- (ear) - 1 | - | - | - | - | - | * | * | * | - | - |
| DECW | eam | 2+ | | | word (eam) <-- (eam) - 1 | - | - | - | - | - | * | * | * | - | * |
| INCL | ear | 2 | | | long (ear) <-- (ear) + 1 | - | - | - | - | - | * | * | * | - | - |
| INCL | eam | 2+ | | | long (eam) <-- (eam) + 1 | - | - | - | - | - | * | * | * | - | * |
| DECL | ear | 2 | | | long (ear) <-- (ear) - 1 | - | - | - | - | - | * | * | * | - | - |
| DECL | eam | 2+ | | | long (eam) <-- (eam) - 1 | - | - | - | - | - | * | * | * | - | * |

Table 37-12. 11 Compare instructions (byte, word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CMP | A | 1 | | | byte (AH) - (AL) | - | - | - | - | - | * | * | * | * | - |
| CMP | A,ear | 2 | | | byte (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| CMP | A,eam | 2+ | | | byte (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| CMP | A,#imm8 | 2 | | | byte (A) - imm8 | - | - | - | - | - | * | * | * | * | - |
| CMPW | A | 1 | | | word (AH) - (AL) | - | - | - | - | - | * | * | * | * | - |
| CMPW | A,ear | 2 | | | word (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| CMPW | A,eam | 2+ | | | word (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| CMPW | A,#imm16 | 3 | | | word (A) - imm16 | - | - | - | - | - | * | * | * | * | - |
| CMPL | A,ear | 2 | | | long (A) - (ear) | - | - | - | - | - | * | * | * | * | - |
| CMPL | A,eam | 2+ | | | long (A) - (eam) | - | - | - | - | - | * | * | * | * | - |
| CMPL | A,#imm32 | 5 | | | long (A) - imm32 | - | - | - | - | - | * | * | * | * | - |

Table 37-13. 11 Unsigned multiplication/division instructions (word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIVU | A | 1 | | | word (AH) / byte (AL)<br>quotient --> byte (AL) remainder --> byte (AH) | - | - | - | - | - | - | - | * | * | - |
| DIVU | A,ear | 2 | | | word (A) / byte (ear)<br>quotient --> byte (A) remainder --> byte (ear) | - | - | - | - | - | - | - | * | * | - |
| DIVU | A,eam | 2+ | | | word (A) / byte (eam)<br>quotient --> byte (A) remainder --> byte (eam) | - | - | - | - | - | - | - | * | * | - |
| DIVUW | A,ear | 2 | | | long (A) / word (ear)<br>quotient --> word (A) remainder --> word (ear) | - | - | - | - | - | - | - | * | * | - |
| DIVUW | A,eam | 2+ | | | long (A) / word (eam)<br>quotient --> word (A) remainder --> word (eam) | - | - | - | - | - | - | - | * | * | - |
| MULU | A | 1 | | | byte (AH) * byte (AL) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULU | A,ear | 2 | | | byte (A) * byte (ear) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULU | A,eam | 2+ | | | byte (A) * byte (eam) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULUW | A | 1 | | | word (AH) * word (AL) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULUW | A,ear | 2 | | | word (A) * word (ear) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULUW | A,eam | 2+ | | | word (A) * word (eam) --> Long (A) | - | - | - | - | - | - | - | - | - | - |

Table 37-14. 11 Signed multiplication/division instructions (word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DIV | A | 2 | | | word (AH) / byte (AL)<br>quotient --> byte (AL) remainder --> byte (AH) | Z | - | - | - | - | - | - | * | * | - |
| DIV | A,ear | 2 | | | word (A) / byte (ear)<br>quotient --> byte (A) remainder --> byte (ear) | Z | - | - | - | - | - | - | * | * | - |
| DIV | A,eam | 2+ | | | word (A) / byte (eam)<br>quotient --> byte (A) remainder --> byte (eam) | Z | - | - | - | - | - | - | * | * | - |
| DIVW | A,ear | 2 | | | long (A) / word (ear)<br>quotient --> word (A) remainder --> word (ear) | - | - | - | - | - | - | - | * | * | - |
| DIVW | A,eam | 2+ | | | long (A) / word (eam)<br>quotient --> word (A) remainder --> word (eam) | - | - | - | - | - | - | - | * | * | - |
| MUL | A | 2 | | | byte (AH) * byte (AL) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MUL | A,ear | 2 | | | byte (A) * byte (ear) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MUL | A,eam | 2+ | | | byte (A) * byte (eam) --> word (A) | - | - | - | - | - | - | - | - | - | - |
| MULW | A | 2 | | | word (AH) * word (AL) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULW | A,ear | 2 | | | word (A) * word (ear) --> Long (A) | - | - | - | - | - | - | - | - | - | - |
| MULW | A,eam | 2+ | | | word (A) * word (eam) --> Long (A) | - | - | - | - | - | - | - | - | - | - |

Table 37-15. 39 Logic 1 instructions (byte, word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AND | A,#imm8 | 2 | | | byte (A) <-- (A) and imm8 | - | - | - | - | - | * | * | R | - | - |
| AND | A,ear | 2 | | | byte (A) <-- (A) and (ear) | - | - | - | - | - | * | * | R | - | - |
| AND | A,eam | 2+ | | | byte (A) <-- (A) and (eam) | - | - | - | - | - | * | * | R | - | - |
| AND | ear,A | 2 | | | byte (ear) <-- (ear) and (A) | - | - | - | - | - | * | * | R | - | - |
| AND | eam,A | 2+ | | | byte (eam) <-- (eam) and (A) | - | - | - | - | - | * | * | R | - | * |
| OR | A,#imm8 | 2 | | | byte (A) <-- (A) or imm8 | - | - | - | - | - | * | * | R | - | - |
| OR | A,ear | 2 | | | byte (A) <-- (A) or (ear) | - | - | - | - | - | * | * | R | - | - |
| OR | A,eam | 2+ | | | byte (A) <-- (A) or (eam) | - | - | - | - | - | * | * | R | - | - |
| OR | ear,A | 2 | | | byte (ear) <-- (ear) or (A) | - | - | - | - | - | * | * | R | - | - |
| OR | eam,A | 2+ | | | byte (eam) <-- (eam) or (A) | - | - | - | - | - | * | * | R | - | * |
| XOR | A,#imm8 | 2 | | | byte (A) <-- (A) xor imm8 | - | - | - | - | - | * | * | R | - | - |
| XOR | A,ear | 2 | | | byte (A) <-- (A) xor (ear) | - | - | - | - | - | * | * | R | - | - |
| XOR | A,eam | 2+ | | | byte (A) <-- (A) xor (eam) | - | - | - | - | - | * | * | R | - | - |
| XOR | ear,A | 2 | | | byte (ear) <-- (ear) xor (A) | - | - | - | - | - | * | * | R | - | - |
| XOR | eam,A | 2+ | | | byte (eam) <-- (eam) xor (A) | - | - | - | - | - | * | * | R | - | * |
| NOT | A | 1 | | | byte (A) <-- not (A) | - | - | - | - | - | * | * | R | - | - |
| NOT | ear | 2 | | | byte (ear) <-- not (ear) | - | - | - | - | - | * | * | R | - | - |
| NOT | eam | 2+ | | | byte (eam) <-- not (eam) | - | - | - | - | - | * | * | R | - | * |
| ANDW | A | 1 | | | word (A) <-- (AH) and (A) | - | - | - | - | - | * | * | R | - | - |
| ANDW | A,#imm16 | 3 | | | word (A) <-- (A) and imm16 | - | - | - | - | - | * | * | R | - | - |
| ANDW | A,ear | 2 | | | word (A) <-- (A) and (ear) | - | - | - | - | - | * | * | R | - | - |
| ANDW | A,eam | 2+ | | | word (A) <-- (A) and (eam) | - | - | - | - | - | * | * | R | - | - |
| ANDW | ear,A | 2 | | | word (ear) <-- (ear) and (A) | - | - | - | - | - | * | * | R | - | - |
| ANDW | eam,A | 2+ | | | word (eam) <-- (eam) and (A) | - | - | - | - | - | * | * | R | - | * |
| ORW | A | 1 | | | word (A) <-- (AH) or (A) | - | - | - | - | - | * | * | R | - | - |
| ORW | A,#imm16 | 3 | | | word (A) <-- (A) or imm16 | - | - | - | - | - | * | * | R | - | - |
| ORW | A,ear | 2 | | | word (A) <-- (A) or (ear) | - | - | - | - | - | * | * | R | - | - |
| ORW | A,eam | 2+ | | | word (A) <-- (A) or (eam) | - | - | - | - | - | * | * | R | - | - |
| ORW | ear,A | 2 | | | word (ear) <-- (ear) or (A) | - | - | - | - | - | * | * | R | - | - |
| ORW | eam,A | 2+ | | | word (eam) <-- (eam) or (A) | - | - | - | - | - | * | * | R | - | * |
| XORW | A | 1 | | | word (A) <-- (AH) xor (A) | - | - | - | - | - | * | * | R | - | - |
| XORW | A,#imm16 | 3 | | | word (A) <-- (A) xor imm16 | - | - | - | - | - | * | * | R | - | - |
| XORW | A,ear | 2 | | | word (A) <-- (A) xor (ear) | - | - | - | - | - | * | * | R | - | - |
| XORW | A,eam | 2+ | | | word (A) <-- (A) xor (eam) | - | - | - | - | - | * | * | R | - | - |
| XORW | ear,A | 2 | | | word (ear) <-- (ear) xor (A) | - | - | - | - | - | * | * | R | - | - |
| XORW | eam,A | 2+ | | | word (eam) <-- (eam) xor (A) | - | - | - | - | - | * | * | R | - | * |
| NOTW | A | 1 | | | word (A) <-- not (A) | - | - | - | - | - | * | * | R | - | - |
| NOTW | ear | 2 | | | word (ear) <-- not (ear) | - | - | - | - | - | * | * | R | - | - |
| NOTW | eam | 2+ | | | word (eam) <-- not (eam) | - | - | - | - | - | * | * | R | - | * |

Table 37-16. 6 Logic 2 instructions (long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ANDL | A,ear | 2 | | | long (A) <-- (A) and (ear) | - | - | - | - | - | * | * | R | - | - |
| ANDL | A,eam | 2+ | | | long (A) <-- (A) and (eam) | - | - | - | - | - | * | * | R | - | - |
| ORL | A,ear | 2 | | | long (A) <-- (A) or (ear) | - | - | - | - | - | * | * | R | - | - |
| ORL | A,eam | 2+ | | | long (A) <-- (A) or (eam) | - | - | - | - | - | * | * | R | - | - |
| XORL | A,ear | 2 | | | long (A) <-- (A) xor (ear) | - | - | - | - | - | * | * | R | - | - |
| XORL | A,eam | 2+ | | | long (A) <-- (A) xor (eam) | - | - | - | - | - | * | * | R | - | - |

Table 37-17. 6 Sign inversion instructions (byte, word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NEG | A | 1 | | | byte (A) <-- 0 - (A) | X | - | - | - | - | * | * | * | * | - |
| NEG | ear | 2 | | | byte (ear) <-- 0 - (ear) | - | - | - | - | - | * | * | * | * | - |
| NEG | eam | 2+ | | | byte (eam) <-- 0 - (eam) | - | - | - | - | - | * | * | * | * | * |
| NEGW | A | 1 | | | word (A) <-- 0 - (A) | - | - | - | - | - | * | * | * | * | - |
| NEGW | ear | 2 | | | word (ear) <-- 0 - (ear) | - | - | - | - | - | * | * | * | * | - |
| NEGW | eam | 2+ | | | word (eam) <-- 0 - (eam) | - | - | - | - | - | * | * | * | * | * |

Table 37-18. 1 Normalization instruction (long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NRML | A,R0 | 2 | | | long (A) <-- Shifts to the position where '1' is set for the first time.<br>byte (RD) <-- Shift count at that time | - | - | - | - | - | - | * | - | - | - |

Table 37-19. 18 Shift instructions (byte, word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RORC | A | 2 | | | byte (A) <-- With right rotation carry | - | - | - | - | - | * | * | - | * | - |
| ROLC | A | 2 | | | byte (A) <-- With left rotation carry | - | - | - | - | - | * | * | - | * | - |
| RORC | ear | 2 | | | byte (ear) <-- With right rotation carry | - | - | - | - | - | * | * | - | * | - |
| RORC | eam | 2+ | | | byte (eam) <-- With right rotation carry | - | - | - | - | - | * | * | - | * | * |
| ROLC | ear | 2 | | | byte (ear) <-- With left rotation carry | - | - | - | - | - | * | * | - | * | - |
| ROLC | eam | 2+ | | | byte (eam) <-- With left rotation carry | - | - | - | - | - | * | * | - | * | * |
| ASR | A,R0 | 2 | | | byte (A) <-- Arithmetic right shift (A, 1 bit) | - | - | - | - | - | * | * | - | * | - |
| LSR | A,R0 | 2 | | | byte (A) <-- Logical right barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |
| LSL | A,R0 | 2 | | | byte (A) <-- Logical left barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |
| ASRW | A | 1 | | | word (A) <-- Arithmetic right shift (A, 1 bit) | - | - | - | - | * | * | * | - | * | - |
| LSRW | A/SHRW A | 1 | | | word (A) <-- Logical right shift (A, 1 bit) | - | - | - | - | * | R | * | - | * | - |
| LSLW | A/SHLW A | 1 | | | word (A) <-- Logical left shift (A, 1 bit) | - | - | - | - | - | * | * | - | * | - |
| ASRW | A,R0 | 2 | | | word (A) <-- Arithmetic right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSRW | A,R0 | 2 | | | word (A) <-- Logical right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSLW | A,R0 | 2 | | | word (A) <-- Logical left barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |
| ASRL | A,R0 | 2 | | | long (A) <-- Arithmetic right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSRL | A,R0 | 2 | | | long (A) <-- Logical right barrel shift (A, R0) | - | - | - | - | * | * | * | - | * | - |
| LSLL | A,R0 | 2 | | | long (A) <-- Logical left barrel shift (A, R0) | - | - | - | - | - | * | * | - | * | - |

Table 37-20. 31 Branch 1 instructions

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BZ/BEQ | rel | 2 | | | Branch on (Z) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNZ/BNE | rel | 2 | | | Branch on (Z) = 0 | - | - | - | - | - | - | - | - | - | - |
| BC/BLO | rel | 2 | | | Branch on (C) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNC/BHS | rel | 2 | | | Branch on (C) = 0 | - | - | - | - | - | - | - | - | - | - |
| BN | rel | 2 | | | Branch on (N) = 1 | - | - | - | - | - | - | - | - | - | - |
| BP | rel | 2 | | | Branch on (N) = 0 | - | - | - | - | - | - | - | - | - | - |
| BV | rel | 2 | | | Branch on (V) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNV | rel | 2 | | | Branch on (V) = 0 | - | - | - | - | - | - | - | - | - | - |
| BT | rel | 2 | | | Branch on (T) = 1 | - | - | - | - | - | - | - | - | - | - |
| BNT | rel | 2 | | | Branch on (T) = 0 | - | - | - | - | - | - | - | - | - | - |
| BLT | rel | 2 | | | Branch on (V) nor (N) = 1 | - | - | - | - | - | - | - | - | - | - |
| BGE | rel | 2 | | | Branch on (V) nor (N) = 0 | - | - | - | - | - | - | - | - | - | - |
| BLE | rel | 2 | | | Branch on ((V) xor (N)) or (Z) = 1 | - | - | - | - | - | - | - | - | - | - |
| BGT | rel | 2 | | | Branch on ((V) xor (N)) or (Z) = 0 | - | - | - | - | - | - | - | - | - | - |
| BLS | rel | 2 | | | Branch on (C) or (Z) = 1 | - | - | - | - | - | - | - | - | - | - |
| BHI | rel | 2 | | | Branch on (C) or (Z) = 0 | - | - | - | - | - | - | - | - | - | - |
| BRA | rel | 2 | | | Unconditional branch | - | - | - | - | - | - | - | - | - | - |
| JMP | @A | 1 | | | word (PC) <-- (A) | - | - | - | - | - | - | - | - | - | - |
| JMP | addr16 | 3 | | | word (PC) <-- addr16 | - | - | - | - | - | - | - | - | - | - |
| JMP | @ear | 2 | | | word (PC) <-- (ear) | - | - | - | - | - | - | - | - | - | - |
| JMP | @eam | 2+ | | | word (PC) <-- (eam) | - | - | - | - | - | - | - | - | - | - |
| JMPP | @ear *3 | 2 | | | word (PC) <-- (ear), (PCB) <-- (ear+2) | - | - | - | - | - | - | - | - | - | - |
| JMPP | @eam *3 | 2+ | | | word (PC) <-- (eam), (PCB) <-- (eam+2) | - | - | - | - | - | - | - | - | - | - |
| JMPP | addr24 | 4 | | | word (PC) <-- ad24 0-15, (PCB) <-- ad24 16-23 | - | - | - | - | - | - | - | - | - | - |
| CALL | @ear *4 | 2 | | | word (PC) <-- (ear) | - | - | - | - | - | - | - | - | - | - |
| CALL | addr16 *5 | 2+ | | | word (PC) <-- (eam) | - | - | - | - | - | - | - | - | - | - |
| CALL | @eam *4 | 3 | | | word (PC) <-- addr16 | - | - | - | - | - | - | - | - | - | - |
| CALLV | #vct4 *5 | 1 | | | Vector call instruction | - | - | - | - | - | - | - | - | - | - |
| CALLP | @ear *6 | 2 | | | word (PC) <-- (ear)0-15, (PCB) <-- (ear)16-23 | - | - | - | - | - | - | - | - | - | - |
| CALLP | @eam *6 | 2+ | | | word (PC) <-- (eam)0-15, (PCB) <-- (eam)16-23 | - | - | - | - | - | - | - | - | - | - |
| CALLP | addr24 *7 | 4 | | | word (PC) <-- addr0-15, (PCB) <-- addr16-23 | - | - | - | - | - | - | - | - | - | - |

*3: Read (word) of branch destination address
*4: W: Save to stack (word) R: Read (word) of branch destination address
*5: Save to stack (word)
*6: W: Save to stack (long word), R: Read (long word) of branch destination address
*7: Save to stack (long word)

Table 37-21. 19 Branch 2 instructions

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CBNE | A,#imm8,rel | 3 | | | Branch on byte (A) not equal to imm8 | - | - | - | - | - | * | * | * | * | - |
| CWBNE | A,#imm16,rel | 4 | | | Branch on word (A) not equal to imm16 | - | - | - | - | - | * | * | * | * | - |
| CBNE | ear,#imm8,rel | 4 | | | Branch on byte (ear) not equal to imm8 | - | - | - | - | - | * | * | * | * | - |
| CBNE | eam,#imm8,rel *9 | 4+ | | | Branch on byte (eam) not equal to imm8 | - | - | - | - | - | * | * | * | * | - |
| CWBNE | ear,#imm16,rel | 5 | | | Branch on word (ear) not equal to imm16 | - | - | - | - | - | * | * | * | * | - |
| CWBNE | eam,#imm16,rel*9 | 5+ | | | Branch on word (eam) not equal to imm16 | - | - | - | - | - | * | * | * | * | - |
| DBNZ | ear,rel | 3 | | | Branch on byte (ear) = (ear) - 1, (ear) not equal to 0 | - | - | - | - | - | * | * | * | - | - |
| DBNZ | eam,rel | 3+ | | | Branch on byte (eam) = (eam) - 1, (eam) not equal to 0 | - | - | - | - | - | * | * | * | - | * |
| DWBNZ | ear,rel | 3 | | | Branch on word (ear) = (ear) - 1, (ear) not equal to 0 | - | - | - | - | - | * | * | * | - | - |
| DWBNZ | eam,rel | 3+ | | | Branch on word (eam) = (eam) - 1, (eam) not equal to 0 | - | - | - | - | - | * | * | * | - | * |
| INT | #vct8 | 2 | | | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| INT | addr16 | 3 | | | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| INTP | addr24 | 4 | | | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| INT9 | | 1 | | | Software interrupt | - | - | R | S | - | - | - | - | - | - |
| RETI | | 1 | | | Return from interrupt | - | - | * | * | * | * | * | * | * | - |
| LINK | #imm8 | 2 | | | Saves the old frame pointer in the stack upon entering the function, then sets the new frame pointer and reserves the local pointer area. | - | - | - | - | - | - | - | - | - | - |
| UNLINK | | 1 | | | Recovers the old frame pointer from the stack upon exiting the function. | - | - | - | - | - | - | - | - | - | - |
| RET | *10 | 1 | | | Return from subroutine | - | - | - | - | - | - | - | - | - | - |
| RETP | *11 | 1 | | | Return from subroutine | - | - | - | - | - | - | - | - | - | - |

*9: Do not use RWj+ addressing mode with a CBNE or CWBNE instruction.
*10: Return from stack (word)
*11: Return from stack (long word)

Table 37-22. 28 Other control instructions (byte, word, long word)

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PUSHW | A | 1 | | | word (SP) <-- (SP) - 2, ((SP)) <-- (A) | - | - | - | - | - | - | - | - | - | - |
| PUSHW | AH | 1 | | | word (SP) <-- (SP) - 2, ((SP)) <-- (AH) | - | - | - | - | - | - | - | - | - | - |
| PUSHW | PS | 1 | | | word (SP) <-- (SP) - 2, ((SP)) <-- (PS) | - | - | - | - | - | - | - | - | - | - |
| PUSHW | rlst | 2 | | | (SP) <-- (SP) - 2n, ((SP)) <-- (rlst) | - | - | - | - | - | - | - | - | - | - |
| POPW | A | 1 | | | word (A) <-- ((SP)), (SP) <-- (SP) + 2 | - | * | - | - | - | - | - | - | - | - |
| POPW | AH | 1 | | | word (AH) <-- ((SP)), (SP) <-- (SP) + 2 | - | - | - | - | - | - | - | - | - | - |
| POPW | PS | 1 | | | word (PS) <-- ((SP)), (SP) <-- (SP) + 2 | - | - | * | * | * | * | * | * | * | - |
| POPW | rlst | 2 | | | (rlst) <-- ((SP)), (SP) <-- (SP) | - | - | - | - | - | - | - | - | - | - |
| JCTX | @A | 1 | | | Context switch instruction | - | - | * | * | * | * | * | * | * | - |
| AND | CCR,#imm8 | 2 | | | byte (CCR) <-- (CCR) and imm8 | - | - | * | * | * | * | * | * | * | - |
| OR | CCR,#imm8 | 2 | | | byte (CCR) <-- (CCR) or imm8 | - | - | * | * | * | * | * | * | * | - |
| MOV | RP,#imm8 | 2 | | | byte (RP) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOV | ILM,#imm8 | 2 | | | byte (ILM) <-- imm8 | - | - | - | - | - | - | - | - | - | - |
| MOVEA | RWi,ear | 2 | | | word (RWi) <-- ear | - | - | - | - | - | - | - | - | - | - |
| MOVEA | RWi,eam | 2+ | | | word (RWi) <-- eam | - | - | - | - | - | - | - | - | - | - |
| MOVEA | A,ear | 2 | | | word (A) <-- ear | - | * | - | - | - | - | - | - | - | - |
| MOVEA | A,eam | 2+ | | | word (A) <-- eam | - | * | - | - | - | - | - | - | - | - |
| ADDSP | #imm8 | 2 | | | word (SP) <-- ext(imm8) | - | - | - | - | - | - | - | - | - | - |
| ADDSP | #imm16 | 3 | | | word (SP) <-- imm16 | - | - | - | - | - | - | - | - | - | - |
| MOV | A,brg1 | 2 | | | byte (A) <-- (brg1) | Z | * | - | - | - | * | * | - | - | - |
| MOV | brg2,A | 2 | | | byte (brg2) <-- (A) | - | - | - | - | - | * | * | - | - | - |
| NOP | | 1 | | | No operation | - | - | - | - | - | - | - | - | - | - |
| ADB | | 1 | | | Prefix code for AD space access | - | - | - | - | - | - | - | - | - | - |
| DTB | | 1 | | | Prefix code for DT space access | - | - | - | - | - | - | - | - | - | - |
| PCB | | 1 | | | Prefix code for PC space access | - | - | - | - | - | - | - | - | - | - |
| SPB | | 1 | | | Prefix code for SP space access | - | - | - | - | - | - | - | - | - | - |
| NCC | | 1 | | | Prefix code for flag no-change | - | - | - | - | - | - | - | - | - | - |
| CMR | | 1 | | | Prefix code for common register bank | - | - | - | - | - | - | - | - | - | - |

Table 37-23. 21 Bit operand instructions

| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOVB | A,dir:bp | 3 | | | byte (A) <-- (dir:bp)b | Z | * | - | - | - | * | * | - | - | - |
| MOVB | A,addr16:bp | 4 | | | byte (A) <-- (addr16:bp)b | Z | * | - | - | - | * | * | - | - | - |
| MOVB | A,io:bp | 3 | | | byte (A) <-- (io:bp)b | Z | * | - | - | - | * | * | - | - | - |
| MOVB | dir:bp,A | 3 | | | bit (dir:bp)b <-- (A) | - | - | - | - | - | * | * | - | - | * |
| MOVB | addr16:bp,A | 4 | | | bit (addr16:bp)b <-- (A) | - | - | - | - | - | * | * | - | - | * |
| MOVB | io:bp,A | 3 | | | bit (io:bp)b <-- (A) | - | - | - | - | - | * | * | - | - | * |
| SETB | dir:bp | 3 | | | bit (dir:bp)b <-- 1 | - | - | - | - | - | - | - | - | - | * |
| SETB | addr16:bp | 4 | | | bit (addr16:bp)b <-- 1 | - | - | - | - | - | - | - | - | - | * |
| SETB | io:bp | 3 | | | bit (io:bp)b <-- 1 | - | - | - | - | - | - | - | - | - | * |
| CLRB | dir:bp | 3 | | | bit (dir:bp)b <-- 0 | - | - | - | - | - | - | - | - | - | * |
| CLRB | addr16:bp | 4 | | | bit (addr16:bp)b <-- 0 | - | - | - | - | - | - | - | - | - | * |
| CLRB | io:bp | 3 | | | bit (io:bp)b <-- 0 | - | - | - | - | - | - | - | - | - | * |
| BBC | dir:bp,rel | 4 | | | Branch on (dir:bp) b = 0 | - | - | - | - | - | - | * | - | - | - |
| BBC | addr16:bp,rel | 5 | | | Branch on (addr16:bp) b = 0 | - | - | - | - | - | - | * | - | - | - |
| BBC | io:bp,rel | 4 | | | Branch on (io:bp) b = 0 | - | - | - | - | - | - | * | - | - | - |
| BBS | dir:bp,rel | 4 | | | Branch on (dir:bp) b = 1 | - | - | - | - | - | - | * | - | - | - |
| BBS | addr16:bp,rel | 5 | | | Branch on (addr16:bp) b = 1 | - | - | - | - | - | - | * | - | - | - |
| BBS | io:bp,rel | 4 | | | Branch on (io:bp) b = 1 | - | - | - | - | - | - | * | - | - | - |
| SBBS | addr16:bp,rel | 5 | | | Branch on (addr16:bp) b = 1, bit = 1 | - | - | - | - | - | - | * | - | - | * |
| WBTS | io:bp | 3 | | | Waits until (io:bp) b = 1 | - | - | - | - | - | - | - | - | - | - |
| WBTC | io:bp | 3 | | | Waits until (io:bp) b = 0 | - | - | - | - | - | - | - | - | - | - |

Table 37-24. 6 Accumulator operation instructions (byte, word)

| Mnemonic | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SWAP | 1 | | | byte (A)0-7 <--> (A)8-15 | - | - | - | - | - | - | - | - | - | - |
| SWAPW / XCHW A,T | 1 | | | word (AH) <--> (AL) | - | * | - | - | - | - | - | - | - | - |
| EXT | 1 | | | Byte sign extension | X | - | - | - | - | * | * | - | - | - |
| EXTW | 1 | | | Word sign extension | - | X | - | - | - | * | * | - | - | - |
| ZEXT | 1 | | | Byte zero extension | Z | - | - | - | - | R | * | - | - | - |
| ZEXTW | 1 | | | Word zero extension | - | z | - | - | - | R | * | - | - | - |

Table 37-25. 10 String instructions

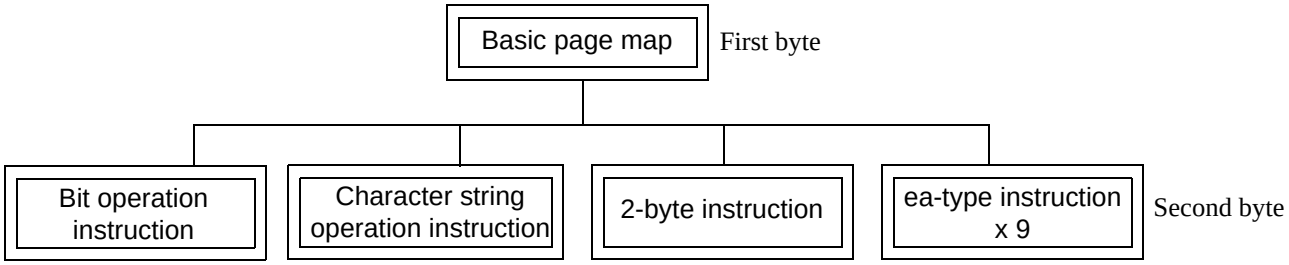
| Mnemonic | | # | ~ | B | Operation | LH | AH | I | S | T | N | Z | V | C | RMW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOVS / MOVSI | brg3, brg3 | 2 | | | byte transfer @AH+ <-- @AL+, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| MOVSD | brg3, brg3 | 2 | | | byte transfer @AH- <-- @AL-, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| SCEQ / SCEQI | brg3 | 2 | | | byte search @AH+ <-- AL, counter RW0 | - | - | - | - | - | * | * | * | * | - |
| SCEQD | brg3 | 2 | | | byte search @AH- <-- AL, counter RW0 | - | - | - | - | - | * | * | * | * | - |
| FILS / FILSI | brg3 | 2 | | | byte fill @AH+ <-- AL, counter RW0 | - | - | - | - | - | * | * | - | - | - |
| MOVSW / MOVSWI | brg3, brg3 | 2 | | | word transfer @AH+ <-- @AL+, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| MOVSWD | brg3, brg3 | 2 | | | word transfer @AH- <-- @AL-, counter = RW0 | - | - | - | - | - | - | - | - | - | - |
| SCWEQ / SCWEQI | brg3 | 2 | | | word search @AH+ - AL, counter = RW0 | - | - | - | - | - | * | * | * | * | - |
| SCWEQD | brg3 | 2 | | | word search @AH- - AL, counter = RW0 | - | - | - | - | - | * | * | * | * | - |
| FILSW / FILSWI | brg3 | 2 | | | word fill @AH+ <-- AL, counter = RW0 | - | - | - | - | - | * | * | - | - | - |

## 37.2.9 Instruction Map

Each F$^2$MC-16FX instruction code consists of 1 or 2 bytes. Therefore, the instruction map consists of multiple pages. Table 37-27 to Table 37-46 summarize the F$^2$MC-16FX instruction map.

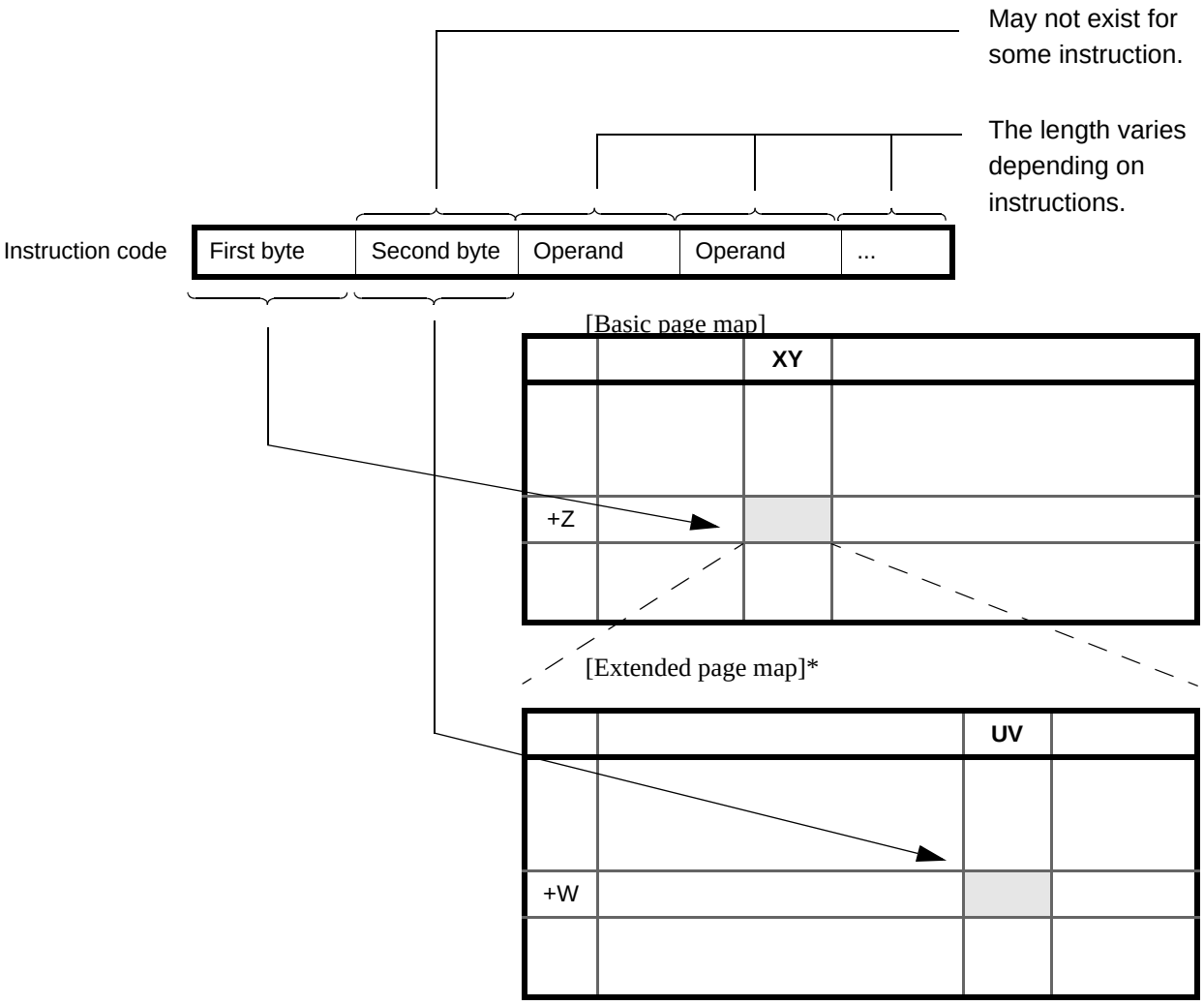
### Structure of instruction map

Figure 37-25. Structure of instruction map




An instruction such as the NOP instruction that ends in one byte is completed within the basic page. An instruction such as the MOVS instruction that requires two bytes recognizes the existence of byte 2 when it references byte 1, and can check the following one byte by referencing the map for byte 2. Figure 37-26 shows the correspondence between an actual instruction code and instruction map.

Figure 37-26. Correspondence between actual instruction code and instruction map

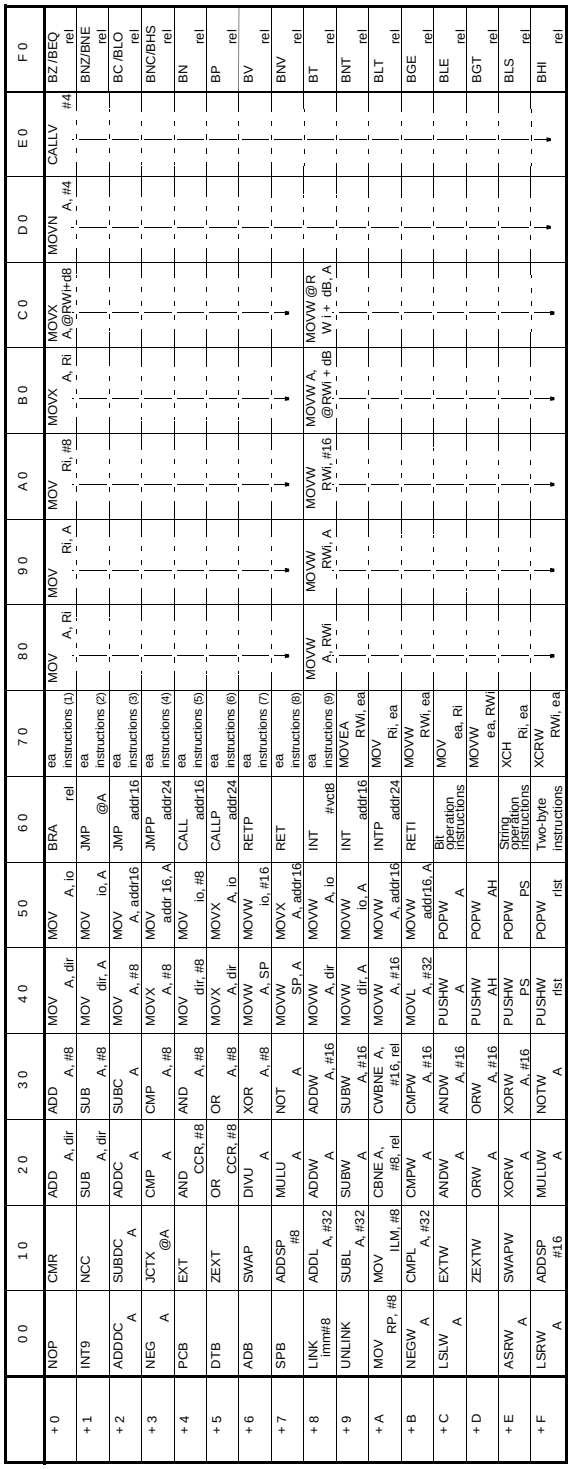



An example of an instruction code is shown in Table 37-26.

Table 37-26. Example of an instruction code

| Instruction | Byte 1 (from basic page map) | Byte 2 (from extended page map) |
|---|---|---|
| NOP | 00 +0=00 | - |
| AND A, #8 | 30 +4=34 | - |
| MOV A, ADB | 60 +F=6F | 00 +0=00 |
| @RW2+d8, #8rel | 70 +0=70 | F0 +2=F2 |

Table 37-27. Basic page map



| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | NOP | CMR | ADD A, dir | ADD A, #8 | MOV A, dir | MOV A, io | BRA rel | ea instructions (1) | MOV A, Ri | MOV Ri, A | MOV Ri, #8 | MOVX A, Ri | MOVX A,@RWi+d8 | MOVN A, #4 | CALLV #4 | BZ /BEQ rel |
| +1 | INT9 | NCC | SUB A, dir | SUB A, #8 | MOV dir, A | MOV io, A | JMP @A | ea instructions (2) | | | | | | | | BNZ/BNE rel |
| +2 | ADDDC A | SUBDC A | ADDC A | SUBC A | MOV A, #8 | MOV A, addr16 | JMP addr16 | ea instructions (3) | | | | | | | | BC /BLO rel |
| +3 | NEG A | JCTX @A | CMP A | CMP A, #8 | MOVX A, #8 | MOV addr 16, A | JMPP addr24 | ea instructions (4) | | | | | | | | BNC/BHS rel |
| +4 | PCB | EXT | AND CCR, #8 | AND A, #8 | MOV dir, #8 | MOV io, #8 | CALL addr16 | ea instructions (5) | | | | | | | | BN rel |
| +5 | DTB | ZEXT | OR CCR, #8 | OR A, #8 | MOVX A, dir | MOVX A, io | CALLP addr24 | ea instructions (6) | | | | | | | | BP rel |
| +6 | ADB | SWAP | DIVU A | XOR A, #8 | MOVW A, SP | MOVW io, #16 | RETP | ea instructions (7) | | | | | | | | BV rel |
| +7 | SPB | ADDSP #8 | MULU A | NOT A | MOVW SP, A | MOVX A, addr16 | RET | ea instructions (8) | | | | | | | | BNV rel |
| +8 | LINK imm#8 | ADDL A, #32 | ADDW A | ADDW A, #16 | MOVW A, dir | MOVW A, io | INT #vct8 | ea instructions (9) | MOVW A, RWi | MOVW RWi, A | MOVW RWi, #16 | MOVW A, @RWi + dB | MOVW @R W i + dB, A | | | BT rel |
| +9 | UNLINK | SUBL A, #32 | SUBW A | SUBW A, #16 | MOVW dir, A | MOVW io, A | INT addr16 | MOVEA RWi, ea | | | | | | | | BNT rel |
| +A | MOV RP, #8 | MOV ILM, #8 | CBNE A, #8, rel | CWBNE A, #16, rel | MOVW A, #16 | MOVW A, addr16 | INTP addr24 | MOV Ri, ea | | | | | | | | BLT rel |
| +B | NEGW A | CMPL A, #32 | CMPW A | CMPW A, #16 | MOVL A, #32 | MOVW addr16, A | RETI | MOVW RWi, ea | | | | | | | | BGE rel |
| +C | LSLW A | EXTW A | ANDW A | ANDW A, #16 | PUSHW A | POPW A | Bit operation instructions | MOV ea, Ri | | | | | | | | BLE rel |
| +D | | ZEXTW | ORW A | ORW A, #16 | PUSHW AH | POPW AH | | MOVW ea, RWi | | | | | | | | BGT rel |
| +E | ASRW A | SWAPW | XORW A | XORW A, #16 | PUSHW PS | POPW PS | String operation instructions | XCH Ri, ea | | | | | | | | BLS rel |
| +F | LSRW A | ADDSP #16 | MULUW A | NOTW A | PUSHW rlst | POPW rlst | Two-byte instructions | XCRW RWi, ea | | | | | | | | BHI rel |

Table 37-28. Bit operation instruction map (first byte = $6C_H$)

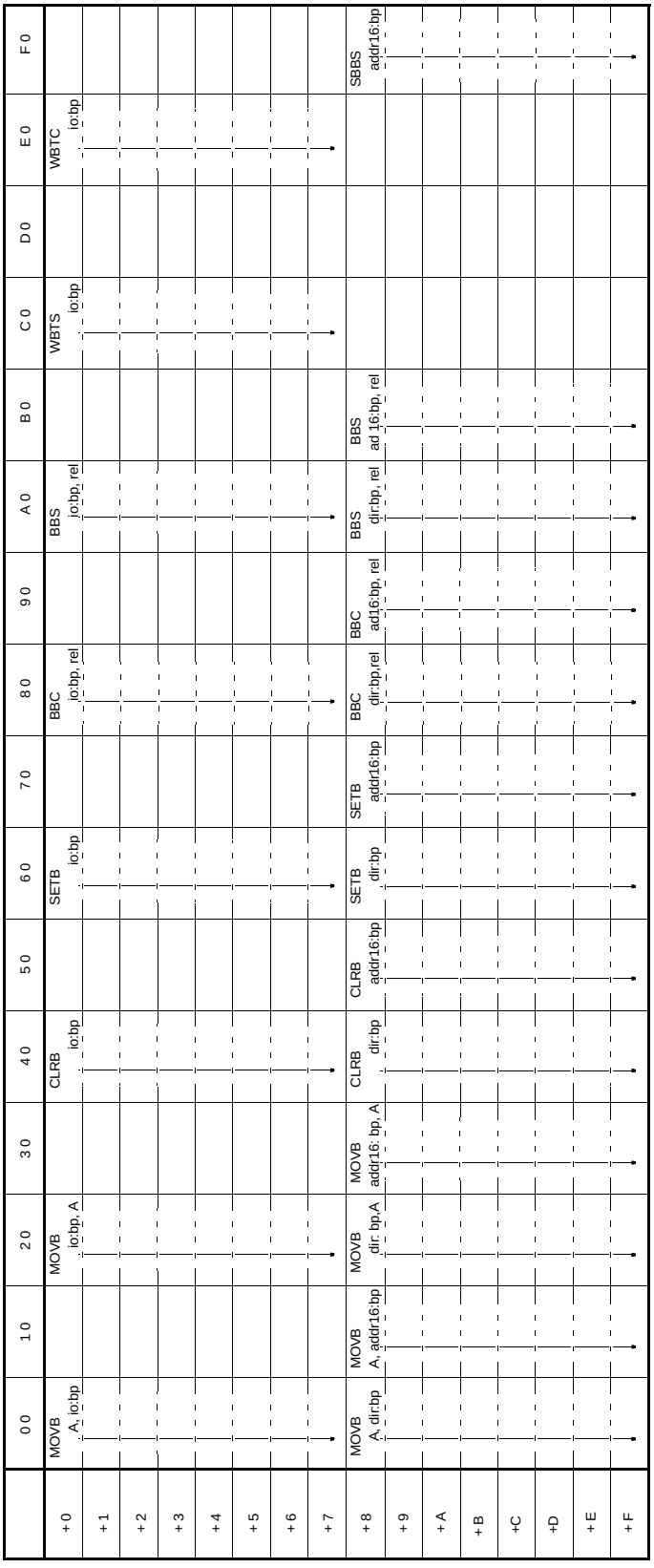


Table 37-29. Character string operation instruction map (first byte = $6E_H$)

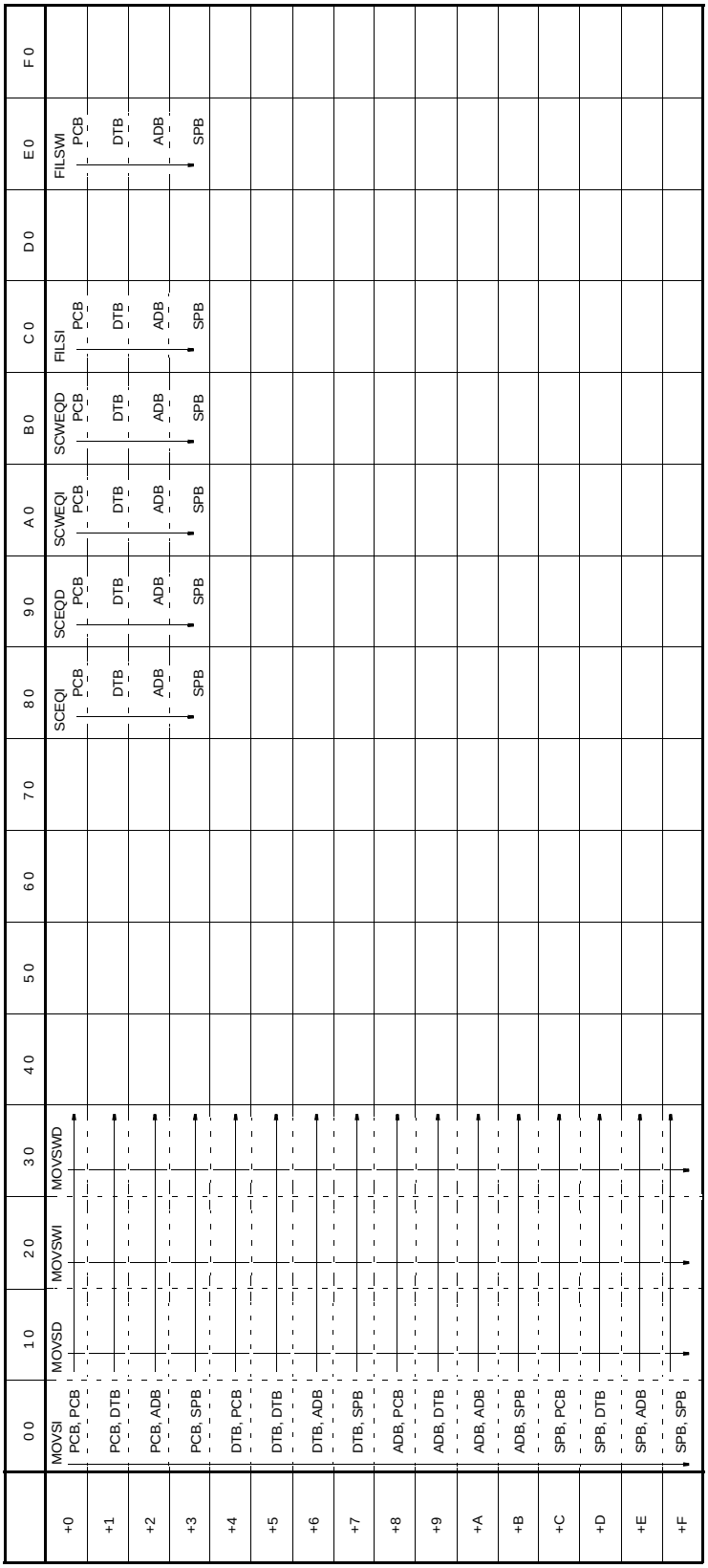


Table 37-30. 2-byte instruction map (first byte = $6F_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOV A, DTB | MOV DTB, A | MOVX A, @RL0 + d8 | MOV @RL0 + d8, A | MOV A, @RL0 + d8 | | | | | | | | | | | |
| +1 | MOV A, ADB | MOV ADB, A | - - - - | - - - - | - - - - | | | | | | | | | | | |
| +2 | MOV A, SSB | MOV SSB, A | MOVX A, @RL1+ d8 | MOV @RL1 + d8, A | MOV A, @RL1 + d8 | | | | | | | | | | | |
| +3 | MOV A, USB | MOV USB, A | - - - - | - - - - | - - - - | | | | | | | | | | | |
| +4 | MOV A, DPR | MOV DPR, A | MOVX A, @RL2 + d8 | MOV @RL2 + d8, A | MOV A, @RL2 + d8 | | | | | | | | | | | |
| +5 | MOV A, @A | MOV @AL, AH | - - - - | - - - - | - - - - | | | | | | | | | | | |
| +6 | MOV A, PCB | MOVX A, @A | MOVX A, @RL3 + d8 | MOV @RL3 + d8, A | MOV A, @RL3 + d8 | | | | | | | | | | | |
| +7 | ROLC A | RORC A | | | | | | | | | | | | | | |
| +8 | | | | MOVW @RL0 + d8, A | MOVW A, @RL0 + d8 | | | MUL A | | | | | | | | |
| +9 | | | | - - - - | - - - - | | | MULW A | | | | | | | | |
| +A | | | | MOVW @RL1 + d8, A | MOVW A, @RL1 + d8 | | | DIV A | | | | | | | | |
| +B | | | | - - - - | - - - - | | | | | | | | | | | |
| +C | LSLW A, R0 | LSLL A, R0 | LSL A, R0 | MOVW @RL2 + d8, A | MOVW A, @RL2 + d8 | | | | | | | | | | | |
| +D | MOVW A, @A | MOVW @AL, AH | NRML A, R0 | - - - - | - - - - | | | | | | | | | | | |
| +E | ASRW A, R0 | ASRL A, R0 | ASR A, R0 | MOVW @RL3 + d8, A | MOVW A, @RL3 + d8 | | | | | | | | | | | |
| +F | LSRW A, R0 | LSRL A, R0 | LSR A, R0 | | | | | | | | | | | | | |

Table 37-31. ea instruction 1 (first byte = $70_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | CWBNE ↓ | CWBNE ↓ | | | | | | | | | CBNE ↓ | CBNE ↓ |
| +0 | ADDL A, RL0 | ADDL A, @RW0 + d8 | SUBL A, RL0 | SUBL A, @RW0 + d8 | RW0, #16, rel | @RW0 + d8 #16, rel | CMPL A, RL0 | CMPL A, @RW0 + d8 | ANDL A, RL0 | ANDL A, @RW0 + d8 | ORL A, RL0 | ORL A, @RW0 + d8 | XORL A, RL0 | XORL A, @RW0 + d8 | R0, #8, rel | @RW0 + d8 #8, rel |
| +1 | ADDL A, RL0 | ADDL A, @RW1 + d8 | SUBL A, RL0 | SUBL A, @RW1 + d8 | RW1, #16, rel | @RW1 + d8 #16, rel | CMPL A, RL0 | CMPL A, @RW1 + d8 | ANDL A, RL0 | ANDL A, @RW1 + d8 | ORL A, RL0 | ORL A, @RW1 + d8 | XORL A, RL0 | XORL A, @RW1 + d8 | R1, #8, rel | @RW1 + d8 #8, rel |
| +2 | ADDL A, RL1 | ADDL A, @RW2 + d8 | SUBL A, RL1 | SUBL A, @RW2 + d8 | RW2, #16, rel | @RW2 + d8 #16, rel | CMPL A, RL1 | CMPL A, @RW2 + d8 | ANDL A, RL1 | ANDL A, @RW2 + d8 | ORL A, RL1 | ORL A, @RW2 + d8 | XORL A, RL1 | XORL A, @RW2 + d8 | R2, #8, rel | @RW2 + d8 #8, rel |
| +3 | ADDL A, RL1 | ADDL A, @RW3 + d8 | SUBL A, RL1 | SUBL A, @RW3 + d8 | RW3, #16, rel | @RW3 + d8 #16, rel | CMPL A, RL1 | CMPL A, @RW3 + d8 | ANDL A, RL1 | ANDL A, @RW3 + d8 | ORL A, RL1 | ORL A, @RW3 + d8 | XORL A, RL1 | XORL A, @RW3 + d8 | R3, #8, rel | @RW3 + d8 #8, rel |
| +4 | ADDL A, RL2 | ADDL A, @RW4 + d8 | SUBL A, RL2 | SUBL A, @RW4 + d8 | RW4, #16, rel | @RW4 + d8 #16, rel | CMPL A, RL2 | CMPL A, @RW4 + d8 | ANDL A, RL2 | ANDL A, @RW4 + d8 | ORL A, RL2 | ORL A, @RW4 + d8 | XORL A, RL2 | XORL A, @RW4 + d8 | R4, #8, rel | @RW4 + d8 #8, rel |
| +5 | ADDL A, RL2 | ADDL A, @RW5 + d8 | SUBL A, RL2 | SUBL A, @RW5 + d8 | RW5, #16, rel | @RW5 + d8 #16, rel | CMPL A, RL2 | CMPL A, @RW5 + d8 | ANDL A, RL2 | ANDL A, @RW5 + d8 | ORL A, RL2 | ORL A, @RW5 + d8 | XORL A, RL2 | XORL A, @RW5 + d8 | R5, #8, rel | @RW5 + d8 #8, rel |
| +6 | ADDL A, RL3 | ADDL A, @RW6 + d8 | SUBL A, RL3 | SUBL A, @RW6 + d8 | RW6, #16, rel | @RW6 + d8 #16, rel | CMPL A, RL3 | CMPL A, @RW6 + d8 | ANDL A, RL3 | ANDL A, @RW6 + d8 | ORL A, RL3 | ORL A, @RW6 + d8 | XORL A, RL3 | XORL A, @RW6 + d8 | R6, #8, rel | @RW6 + d8 #8, rel |
| +7 | ADDL A, RL3 | ADDL A, @RW7 + d8 | SUBL A, RL3 | SUBL A, @RW7 + d8 | RW7, #16, rel | @RW7 + d8 #16, rel | CMPL A, RL3 | CMPL A, @RW7 + d8 | ANDL A, RL3 | ANDL A, @RW7 + d8 | ORL A, RL3 | ORL A, @RW7 + d8 | XORL A, RL3 | XORL A, @RW7 + d8 | R7, #8, rel | @RW7 + d8 #8, rel |
| +8 | ADDL A, @RW0 | ADDL A, @RW0 + d16 | SUBL A, @RW0 | SUBL A, @RW0 + d16 | @RW0, #16, rel | @RW0 + d16 #16, rel | CMPL A, @RW0 | CMPL A, @RW0 + d16 | ANDL A, @RW0 | ANDL A, @RW0 + d16 | ORL A, @RW0 | ORL A, @RW0 + d16 | XORL A, @RW0 | XORL A, @RW0 + d16 | @RW0, #8, rel | @RW0 + d16 #8, rel |
| +9 | ADDL A, @RW1 | ADDL A, @RW1 + d16 | SUBL A, @RW1 | SUBL A, @RW1 + d16 | @RW1, #16, rel | @RW1 + d16 #16, rel | CMPL A, @RW1 | CMPL A, @RW1 + d16 | ANDL A, @RW1 | ANDL A, @RW1 + d16 | ORL A, @RW1 | ORL A, @RW1 + d16 | XORL A, @RW1 | XORL A, @RW1 + d16 | @RW1, #8, rel | @RW1 + d16 #8, rel |
| +A | ADDL A, @RW2 | ADDL A, @RW2 + d16 | SUBL A, @RW2 | SUBL A, @RW2 + d16 | @RW2, #16, rel | @RW2 + d16 #16, rel | CMPL A, @RW2 | CMPL A, @RW2 + d16 | ANDL A, @RW2 | ANDL A, @RW2 + d16 | ORL A, @RW2 | ORL A, @RW2 + d16 | XORL A, @RW2 | XORL A, @RW2 + d16 | @RW2, #8, rel | @RW2 + d16 #8, rel |
| +B | ADDL A, @RW3 | ADDL A, @RW3 + d16 | SUBL A, @RW3 | SUBL A, @RW3 + d16 | @RW3, #16, rel | @RW3 + d16 #16, rel | CMPL A, @RW3 | CMPL A, @RW3 + d16 | ANDL A, @RW3 | ANDL A, @RW3 + d16 | ORL A, @RW3 | ORL A, @RW3 + d16 | XORL A, @RW3 | XORL A, @RW3 + d16 | @RW3, #8, rel | @RW3 + d16 #8, rel |
| +C | ADDL A, @RW0+ | ADDL A, @RW0 + RW7 | SUBL A, @RW0+ | SUBL A, @RW0 + RW7 | Do not use | @RW0 + RW7 #16, rel | CMPL A, @RW0+ | CMPL A, @RW0 + RW7 | ANDL A, @RW0+ | ANDL A, @RW0 + RW7 | ORL A, @RW0+ | ORL A, @RW0 + RW7 | XORL A, @RW0+ | XORL A, @RW0 + RW7 | Do not use | @RW0 + RW7 #8, rel |
| +D | ADDL A, @RW1+ | ADDL A, @RW1 + RW7 | SUBL A, @RW1+ | SUBL A, @RW1 + RW7 | Do not use | @RW1 + RW7 #16, rel | CMPL A, @RW1+ | CMPL A, @RW1 + RW7 | ANDL A, @RW1+ | ANDL A, @RW1 + RW7 | ORL A, @RW1+ | ORL A, @RW1 + RW7 | XORL A, @RW1+ | XORL A, @RW1 + RW7 | Do not use | @RW1 + RW7 #8, rel |
| +E | ADDL A, @RW2+ | ADDL A, @PC + d16 | SUBL A, @RW2+ | SUBL A, @PC + d16 | Do not use | @PC + d16, #16, rel | CMPL A, @RW2+ | CMPL A, @PC + d16 | ANDL A, @RW2+ | ANDL A, @PC + d16 | ORL A, @RW2+ | ORL A, @PC + d16 | XORL A, @RW2+ | XORL A, @PC + d16 | Do not use | @PC + d16 #8, rel |
| +F | ADDL A, @RW3+ | ADDL A, addr16 | SUBL A, @RW3+ | SUBL A, addr16 | Do not use | addr16, #16, rel | CMPL A, @RW3+ | CMPL A, addr16 | ANDL A, @RW3+ | ANDL A, addr16 | ORL A, @RW3+ | ORL A, addr16 | XORL A, @RW3+ | XORL A, addr16 | Do not use | addr16 #8, rel |

Table 37-32. ea instruction 2 (first byte = $71_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | JMPP @RL0 | JMPP @@RW0 + d8 | CALLP @RL0 | CALLP @@RW0 + d8 | INCL RL0 | INCL @RW0 + d8 | DECL RL0 | DECL @RW0 + d8 | MOVL A, RL0 | MOVL A, @RW0 + d8 | MOVL RL0, A | MOVL @R W0 + d8,A | MOV R0,#8 | MOV @R W0 + d8,#8 | MOVEA A,RW0 | MOVEAA, @RW0 + d8 |
| +1 | JMPP @RL0 | JMPP @@RW1 + d8 | CALLP @RL0 | CALLP @@RW1 + d8 | INCL RL0 | INCL @RW1 + d8 | DECL RL0 | DECL @RW1 + d8 | MOVL A, RL0 | MOVL A, @RW1 + d8 | MOVL RL0, A | MOVL @R W1 + d8,A | MOV R1,#8 | MOV @R W1 + d8,#8 | MOVEA A,RW1 | MOVEAA, @RW1 + d8 |
| +2 | JMPP @RL1 | JMPP @@RW2 + d8 | CALLP @RL1 | CALLP @@RW2 + d8 | INCL RL1 | INCL @RW2 + d8 | DECL RL1 | DECL @RW2 + d8 | MOVL A, RL1 | MOVL A, @RW2 + d8 | MOVL RL1, A | MOVL @R W2 + d8,A | MOV R2,#8 | MOV @R W2 + d8,#8 | MOVEA A,RW2 | MOVEAA, @RW2 + d8 |
| +3 | JMPP @RL1 | JMPP @@RW3 + d8 | CALLP @RL1 | CALLP @@RW3 + d8 | INCL RL1 | INCL @RW3 + d8 | DECL RL1 | DECL @RW3 + d8 | MOVL A, RL1 | MOVL A, @RW3 + d8 | MOVL RL1, A | MOVL @R W3 + d8,A | MOV R3,#8 | MOV @R W3 + d8,#8 | MOVEA A,RW3 | MOVEAA, @RW3 + d8 |
| +4 | JMPP @RL2 | JMPP @@RW4 + d8 | CALLP @RL2 | CALLP @@RW4 + d8 | INCL RL2 | INCL @RW4 + d8 | DECL RL2 | DECL @RW4 + d8 | MOVL A, RL2 | MOVL A, @RW4 + d8 | MOVL RL2, A | MOVL@R W4 + d8,A | MOV R4,#8 | MOV @R W4 + d8,#8 | MOVEA A,RW4 | MOVEAA, @RW4 + d8 |
| +5 | JMPP @RL2 | JMPP @@RW5 + d8 | CALLP @RL2 | CALLP @@RW5 + d8 | INCL RL2 | INCL @RW5 + d8 | DECL RL2 | DECL @RW5 + d8 | MOVL A, RL2 | MOVL A, @RW5 + d8 | MOVL RL2, A | MOVL@R W5 + d8,A | MOV R5,#8 | MOV @R W5 + d8,#8 | MOVEA A,RW5 | MOVEAA, @RW5 + d8 |
| +6 | JMPP @RL3 | JMPP @@RW6 + d8 | CALLP @RL3 | CALLP @@RW6 + d8 | INCL RL3 | INCL @RW6 + d8 | DECL RL3 | DECL @RW6 + d8 | MOVL A, RL3 | MOVL A, @RW6 + d8 | MOVL RL3, A | MOVL @R W6 + d8,A | MOV R6,#8 | MOV @R W6 + d8,#8 | MOVEA A,RW6 | MOVEAA, @RW6 + d8 |
| +7 | JMPP @RL3 | JMPP @@RW7 + d8 | CALLP @RL3 | CALLP @@RW7 + d8 | INCL RL3 | INCL @RW7 + d8 | DECL RL3 | DECL @RW7 + d8 | MOVL A, RL3 | MOVL A, @RW7 + d8 | MOVL RL3, A | MOVL @R W7 + d8,A | MOV R7,#8 | MOV @R W7 + d8,#8 | MOVEA A,RW7 | MOVEAA, @RW7 + d8 |
| +8 | JMPP @@RW0 | JMPP @ @RW0 + d16 | CALLP @@RW0 | CALLP @ @RW0 + d16 | INCL @RW0 | INCL @RW0 + d16 | DECL @RW0 | DECL @RW0 + d16 | MOVL A, @RW0 | MOVL A, @RW0 + d16 | MOVL @RW0, A | MOVL @R W0 + d16,A | MOV @RW0,#8 | MOV @R W0 + d16,#8 | MOVEA A,@RW0 | MOVEAA, @RW0 + d16 |
| +9 | JMPP @@RW1 | JMPP @ @RW1 + d16 | CALLP @@RW1 | CALLP @ @RW1 + d16 | INCL @RW1 | INCL @RW1 + d16 | DECL @RW1 | DECL @RW1 + d16 | MOVL A, @RW1 | MOVL A, @RW1 + d16 | MOVL @RW1, A | MOVL @R W1 + d16,A | MOV @RW1,#8 | MOV @R W1 + d16,#8 | MOVEA A,@RW1 | MOVEAA, @RW1 + d16 |
| +A | JMPP @@RW2 | JMPP @ @RW2 + d16 | CALLP @@RW2 | CALLP @ @RW2 + d16 | INCL @RW2 | INCL @RW2 + d16 | DECL @RW2 | DECL @RW2 + d16 | MOVL A, @RW2 | MOVL A, @RW2 + d16 | MOVL @RW2, A | MOVL @R W2 + d16,A | MOV @RW2,#8 | MOV @R W2 + d16,#8 | MOVEA A,@RW2 | MOVEAA, @RW2 + d16 |
| +B | JMPP @@RW3 | JMPP @ @RW3 + d16 | CALLP @@RW3 | CALLP @ @RW3 + d16 | INCL @RW3 | INCL @RW3 + d16 | DECL @RW3 | DECL @RW3 + d16 | MOVL A, @RW3 | MOVL A, @RW3 + d16 | MOVL @RW3, A | MOVL @R W3 + d16,A | MOV @RW3,#8 | MOV @R W3 + d16,#8 | MOVEA A,@RW3 | MOVEAA, @RW3 + d16 |
| +C | JMPP @@RW0+ | JMPP @ @RW0+RW7 | CALLP @@RW0+ | CALLP @ @RW0+RW7 | INCL @RW0+ | INCL @RW0+RW7 | DECL @RW0+ | DECL @RW0+RW7 | MOVL A, @RW0+ | MOVL A, @RW0+RW7 | MOVL @RW0+, A | MOVL @R W0+RW7,A | MOV @RW0+,#8 | MOV @R W0+RW7,#8 | MOVEA A,@RW0+ | MOVEAA, @RW0+RW7 |
| +D | JMPP @@RW1+ | JMPP @ @RW1+RW7 | CALLP @@RW1+ | CALLP @ @RW1+RW7 | INCL @RW1+ | INCL @RW1+RW7 | DECL @RW1+ | DECL @RW1+RW7 | MOVL A, @RW1+ | MOVL A, @RW1+RW7 | MOVL @RW1+, A | MOVL @R W1+RW7,A | MOV @RW1+,#8 | MOV @R W1+RW7,#8 | MOVEA A,@RW1+ | MOVEAA, @RW1+RW7 |
| +E | JMPPR @@W2+ | JMPP @@PC + d16 | CALLP @@RW2+ | CALLP @@PC + d16 | INCL @RW2+ | INCL @PC + d16 | DECL @RW2+ | DECL @PC + d16 | MOVL A, @RW2+ | MOVL A, @PC + d16 | MOVL @RW2+, A | MOVL @P C + d16,A | MOV @RW2+,#8 | MOV @P C + d16,#8 | MOVEA A,@RW2+ | MOVEAA, @PC + d16 |
| +F | JMPP @@RW3+ | JMPP @addr16 | CALLP @@RW3+ | CALLP @@addr16 | INCL @RW3+ | INCL addr16 | DECL @RW3+ | DECL addr16 | MOVL A, @RW3+ | MOVL A, addr16 | MOVL @RW3+, A | MOVL addr16,A | MOV @RW3+,#8 | MOV addr16,#8 | MOVEA A,@RW3+ | MOVEAA, addr16 |

Table 37-33. ea instruction 3 (first byte = $72_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | ROLC R0 | ROLC @RW0 + d8 | RORC R0 | RORC @RW0 + d8 | INC R0 | INC @RW0 + d8 | DEC R0 | DEC @RW0 + d8 | MOV A,R0 | MOV A, @RW0 + d8 | MOV R0,A | MOV @R W0 + d8,A | MOVX A,R0 | MOVX A, @RW0 + d8 | XCH A,R0 | XCH A, @RW0 + d8 |
| +1 | ROLC R1 | ROLC @RW1 + d8 | RORC R1 | RORC @RW1 + d8 | INC R1 | INC @RW1 + d8 | DEC R1 | DEC @RW1 + d8 | MOV A,R1 | MOV A, @RW1 + d8 | MOV R1,A | MOV @R W1 + d8,A | MOVX A,R1 | MOVX A, @RW1 + d8 | XCH A,R1 | XCH A, @RW1 + d8 |
| +2 | ROLC R2 | ROLC @RW2 + d8 | RORC R2 | RORC @RW2 + d8 | INC R2 | INC @RW2 + d8 | DEC R2 | DEC @RW2 + d8 | MOV A,R2 | MOV A, @RW2 + d8 | MOV R2,A | MOV @R W2 + d8,A | MOVX A,R2 | MOVX A, @RW2 + d8 | XCH A,R2 | XCH A, @RW2 + d8 |
| +3 | ROLC R3 | ROLC @RW3 + d8 | RORC R3 | RORC @RW3 + d8 | INC R3 | INC @RW3 + d8 | DEC R3 | DEC @RW3 + d8 | MOV A,R3 | MOV A, @RW3 + d8 | MOV R3,A | MOV @R W3 + d8,A | MOVX A,R3 | MOVX A, @RW3 + d8 | XCH A,R3 | XCH A, @RW3 + d8 |
| +4 | ROLC R4 | ROLC @RW4 + d8 | RORC R4 | RORC @RW4 + d8 | INC R4 | INC @RW4 + d8 | DEC R4 | DEC @RW4 + d8 | MOV A,R4 | MOV A, @RW4 + d8 | MOV R4,A | MOV @R W4 + d8,A | MOVX A,R4 | MOVX A, @RW4 + d8 | XCH A,R4 | XCH A, @RW4 + d8 |
| +5 | ROLC R5 | ROLC @RW5 + d8 | RORC R5 | RORC @RW5 + d8 | INC R5 | INC @RW5 + d8 | DEC R5 | DEC @RW5 + d8 | MOV A,R5 | MOV A, @RW5 + d8 | MOV R5,A | MOV @R W5 + d8,A | MOVX A,R5 | MOVX A, @RW5 + d8 | XCH A,R5 | XCH A, @RW5 + d8 |
| +6 | ROLC R6 | ROLC @RW6 + d8 | RORC R6 | RORC @RW6 + d8 | INC R6 | INC @RW6 + d8 | DEC R6 | DEC @RW6 + d8 | MOV A,R6 | MOV A, @RW6 + d8 | MOV R6,A | MOV @R W6 + d8,A | MOVX A,R6 | MOVX A, @RW6 + d8 | XCH A,R6 | XCH A, @RW6 + d8 |
| +7 | ROLC R7 | ROLC @RW7 + d8 | RORC R7 | RORC @RW7 + d8 | INC R7 | INC @RW7 + d8 | DEC R7 | DEC @RW7 + d8 | MOV A,R7 | MOV A, @RW7 + d8 | MOV R7,A | MOV @R W7 + d8,A | MOVX A,R7 | MOVX A, @RW7 + d8 | XCH A,R7 | XCH A, @RW7 + d8 |
| +8 | ROLC @RW0 | ROLC @RW0 + d16 | RORC @RW0 | RORC @RW0 + d16 | INC @RW0 | INC @RW0 + d16 | DEC @RW0 | DEC @RW0 + d16 | MOV A,@RW0 | MOV A, @RW0 + d16 | MOV @RW0, A | MOV @R W0 + d16,A | MOVX A,@RW0 | MOVX A, @RW0 + d16 | XCH A,@RW0 | XCH A, @RW0 + d16 |
| +9 | ROLC @RW1 | ROLC @RW1 + d16 | RORC @RW1 | RORC @RW1 + d16 | INC @RW1 | INC @RW1 + d16 | DEC @RW1 | DEC @RW1 + d16 | MOV A,@RW1 | MOV A, @RW1 + d16 | MOV @RW1, A | MOV @R W1 + d16,A | MOVX A,@RW1 | MOVX A, @RW1 + d16 | XCH A,@RW1 | XCH A, @RW1 + d16 |
| +A | ROLC @RW2 | ROLC @RW2 + d16 | RORC @RW2 | RORC @RW2 + d16 | INC @RW2 | INC @RW2 + d16 | DEC @RW2 | DEC @RW2 + d16 | MOV A,@RW2 | MOV A, @RW2 + d16 | MOV @RW2, A | MOV @R W2 + d16,A | MOVX A,@RW2 | MOVX A, @RW2 + d16 | XCH A,@RW2 | XCH A, @RW2 + d16 |
| +B | ROLC @RW3 | ROLC @RW3 + d16 | RORC @RW3 | RORC @RW3 + d16 | INC @RW3 | INC @RW3 + d16 | DEC @RW3 | DEC @RW3 + d16 | MOV A,@RW3 | MOV A, @RW3 + d16 | MOV @RW3, A | MOV @R W3 + d16,A | MOVX A,@RW3 | MOVX A, @RW3 + d16 | XCH A,@RW3 | XCH A, @RW3 + d16 |
| +C | ROLC @RW0+ | ROLC @RW0+RW7 | RORC @RW0+ | RORC @RW0+RW7 | +INC @RW0 | +INC @RW0+RW7 | DEC @RW0+ | DEC @RW0+RW7 | MOV A,@RW0+ | MOV A, @RW0+RW7 | MOV @RW0+, A | MOV @R W0+RW7,A | MOVX A,@RW0+ | MOVX A, @RW0+RW7 | XCH A,@RW0+ | XCH A, @RW0+RW7 |
| +D | ROLC @RW1+ | ROLC @RW1+RW7 | RORC @RW1+ | RORC @RW1+RW7 | INC @RW1+ | INC @RW1+RW7 | DEC @RW1+ | DEC @RW1+RW7 | MOV A,@RW1+ | MOV A, @RW1+RW7 | MOV @RW1+, A | MOV @R W1+RW7,A | MOVX A,@RW1+ | MOVX A, @RW1+RW7 | XCH A,@RW1+ | XCH A, @RW1+RW7 |
| +E | ROLC @RW2+ | ROLC @PC + d16 | RORC @RW2+ | RORC @PC + d16 | INC @RW2+ | INC @PC + d16 | DEC @RW2+ | DEC @PC + d16 | MOV A,@RW2+ | MOV A, @PC + d16 | MOV @RW2+, A | MOV @P C + d16, A | MOVX A,@RW2+ | MOVX A, @PC + d16 | XCH A,@RW2+ | XCH A, @PC + d16 |
| +F | ROLC @RW3+ | ROLC addr16 | RORC @RW3+ | RORC addr16 | INC @RW3+ | INC addr16 | DEC @RW3+ | DEC addr16 | MOV A,@RW3+ | MOV A, addr16 | MOV @RW3+, A | MOV addr16, A | MOVX A,@RW3+ | MOVX A, addr16 | XCH A,@RW3+ | XCH A, addr16 |

Table 37-34. ea instruction 4 (first byte = $73_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | JMP @RW0 | JMP @@RW0 + d8 | CALL @RW0 | CALL @@RW0 + d8 | INCW RW0 | INCW @RW0 + d8 | DECW RW0 | DECW @RW0 + d8 | MOVW A,RW0 | MOVW A, @RW0 + d8 | MOVW RW0,A | MOVW @R W0 + d8,A | MOVW RW0,#16 | MOVW @RW 0 + d8,#16 | XCHW A, RW0 | XCHW A, @RW0 + d8 |
| +1 | JMP @RW1 | JMP @@RW1 + d8 | CALL @RW1 | CALL @@RW1 + d8 | INCW RW1 | INCW @RW1 + d8 | DECW RW1 | DECW @RW1 + d8 | MOVW A,RW1 | MOVW A, @RW1 + d8 | MOVW RW1,A | MOVW @R W1 + d8,A | MOVW RW1,#16 | MOVW @RW 1 + d8,#16 | XCHW A, RW1 | XCHW A, @RW1 + d8 |
| +2 | JMP @RW2 | JMP @@RW2 + d8 | CALL @RW2 | CALL @@RW2 + d8 | INCW RW2 | INCW @RW2 + d8 | DECW RW2 | DECW @RW2 + d8 | MOVW A,RW2 | MOVW A, @RW2 + d8 | MOVW RW2,A | MOVW @R W2 + d8,A | MOVW RW2,#16 | MOVW @RW 2 + d8,#16 | XCHW A, RW2 | XCHW A, @RW2 + d8 |
| +3 | JMP @RW3 | JMP @@RW3 + d8 | CALL @RW3 | CALL @@RW3 + d8 | INCW RW3 | INCW @RW3 + d8 | DECW RW3 | DECW @RW3 + d8 | MOVW A,RW3 | MOVW A, @RW3 + d8 | MOVW RW3,A | MOVW @R W3 + d8,A | MOVW RW3,#16 | MOVW @RW 3 + d8,#16 | XCHW A, RW3 | XCHW A, @RW3 + d8 |
| +4 | JMP @RW4 | JMP @@RW4 + d8 | CALL @RW4 | CALL @@RW4 + d8 | INCW RW4 | INCW @RW4 + d8 | DECW RW4 | DECW @RW4 + d8 | MOVW A,RW4 | MOVW A, @RW4 + d8 | MOVW RW4,A | MOVW @R W4 + d8,A | MOVW RW4,#16 | MOVW @RW 4 + d8,#16 | XCHW A, RW4 | XCHW A, @RW4 + d8 |
| +5 | JMP @RW5 | JMP @@RW5 + d8 | CALL @RW5 | CALL @@RW5 + d8 | INCW RW5 | INCW @RW5 + d8 | DECW RW5 | DECW @RW5 + d8 | MOVW A,RW5 | MOVW A, @RW5 + d8 | MOVW RW5,A | MOVW @R W5 + d8,A | MOVW RW5,#16 | MOVW @RW 5 + d8,#16 | XCHW A, RW5 | XCHW A, @RW5 + d8 |
| +6 | JMP @RW6 | JMP @@RW6 + d8 | CALL @RW6 | CALL @@RW6 + d8 | INCW RW6 | INCW @RW6 + d8 | DECW RW6 | DECW @RW6 + d8 | MOVW A,RW6 | MOVW A, @RW6 + d8 | MOVW RW6,A | MOVW @R W6 + d8,A | MOVW RW6,#16 | MOVW @RW 6 + d8,#16 | XCHW A, RW6 | XCHW A, @RW6 + d8 |
| +7 | JMP @RW7 | JMP @@RW7 + d8 | CALL @RW7 | CALL @@RW7 + d8 | INCW RW7 | INCW @RW7 + d8 | DECW RW7 | DECW @RW7 + d8 | MOVW A,RW7 | MOVW A, @RW7 + d8 | MOVW RW7,A | MOVW @R W7 + d8,A | MOVW RW7,#16 | MOVW @RW 7 + d8,#16 | XCHW A, RW7 | XCHW A, @RW7 + d8 |
| +8 | JMP @@RW0 | JMP @@RW0 + d16 | CALL @@RW0 | CALL @@RW0 + d16 | INCW @RW0 | INCW @RW0 + d16 | DECW @RW0 | DECW @RW0 + d16 | MOVW A,@RW0 | MOVW A, @RW0 + d16 | MOVW @RW0,A | MOVW @R W0 + d16,A | MOVW @RW0,#16 | MOVW@RW0 + d16,#16 | XCHW A, @RW0 | XCHW A, @RW0 + d16 |
| +9 | JMP @@RW1 | JMP @@RW1 + d16 | CALL @@RW1 | CALL @@RW1 + d16 | INCW @RW1 | INCW @RW1 + d16 | DECW @RW1 | DECW @RW1 + d16 | MOVW A,@RW1 | MOVW A, @RW1 + d16 | MOVW @RW1,A | MOVW @R W1 + d16,A | MOVW @RW1,#16 | MOVW@RW1 + d16,#16 | XCHW A, @RW1 | XCHW A, @RW1 + d16 |
| +A | JMP @@RW2 | JMP @@RW2 + d16 | CALL @@RW2 | CALL @@RW2 + d16 | INCW @RW2 | INCW @RW2 + d16 | DECW @RW2 | DECW @RW2 + d16 | MOVW A,@RW2 | MOVW A, @RW2 + d16 | MOVW @RW2,A | MOVW @R W2 + d16,A | MOVW @RW2,#16 | MOVW@RW2 + d16,#16 | XCHW A, @RW2 | XCHW A, @RW2 + d16 |
| +B | JMP @@RW3 | JMP @@RW3 + d16 | CALL @@RW3 | CALL @@RW3 + d16 | INCW @RW3 | INCW @RW3 + d16 | DECW @RW3 | DECW @RW3 + d16 | MOVW A,@RW3 | MOVW A, @RW3 + d16 | MOVW @RW3,A | MOVW @R W3 + d16,A | MOVW @RW3,#16 | MOVW@RW3 + d16,#16 | XCHW A, @RW3 | XCHW A, @RW3 + d16 |
| +C | JMP @@RW0+ | JMP @@RW0+RW7 | CALL @@RW0+ | CALL @@RW0+RW7 | INCW @RW0+ | INCW @RW0+RW7 | DECW @RW0+ | DECW @RW0+RW7 | MOVW A,@RW0+ | MOVW A, @RW0+RW7 | MOVW @RW0+,A | MOVW @R W0+RW7,A | MOVW @RW0+,#16 | MOVW@RW0 +RW7,#16 | XCHW A, @RW0+ | XCHW A, @RW0+RW7 |
| +D | JMP @@RW1+ | JMP @@RW1+RW7 | CALL @@RW1+ | CALL @@RW1+RW7 | INCW @RW1+ | INCW @RW1+RW7 | DECW @RW1+ | DECW @RW1+RW7 | MOVW A,@RW1+ | MOVW A, @RW1+RW7 | MOVW @RW1+,A | MOVW @R W1+RW7,A | MOVW @RW1+,#16 | MOVW@RW1 +RW7,#16 | XCHW A, @RW1+ | XCHW A, @RW1+RW7 |
| +E | JMP @@RW2+ | JMP @@PC + d16 | CALL @@RW2+ | CALL @@PC + d16 | INCW @RW2+ | INCW @PC + d16 | DECW @RW2+ | DECW @PC + d16 | MOVW A,@RW2+ | MOVW A, @PC + d16 | MOVW @RW2+,A | MOVW @P C + d16,A | MOVW @RW2+,#16 | MOVW @PC + d16,#16 | XCHW A, @RW2+ | XCHW A, @PC + d16 |
| +F | JMP @@RW3+ | JMP @addr16 | CALL @@RW3+ | CALL @addr16 | INCW @RW3+ | INCW addr16 | DECW @RW3+ | DECW addr16 | MOVW A,@RW3+ | MOVW A, addr16 | MOVW @RW3+,A | MOVW addr16,A | MOVW @RW3+,#16 | MOVW ad dr16,#16 | XCHW A, @RW3+ | XCHW A, addr16 |

Table 37-35. ea instruction 5 (first byte = $74_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | ADD A,@RW0 | ADD A,@RW0 + d8 | SUB A,R0 | SUB A,@RW0 + d8 | ADDC A,R0 | ADDC A,@RW0 + d8 | CMP A,R0 | CMP A,@RW0 + d8 | AND A,R0 | AND A,@RW0 + d8 | OR A,R0 | OR A,@RW0 + d8 | XOR A,R0 | XOR A,@RW0 + d8 | DBNZ R0, r | DBNZ @RW0 + d8, r |
| +1 | ADD A,R1 | ADD A,@RW1 + d8 | SUB A,R1 | SUB A,@RW1 + d8 | ADDC A,R1 | ADDC A,@RW1 + d8 | CMP A,R1 | CMP A,@RW1 + d8 | AND A,R1 | AND A,@RW1 + d8 | OR A,R1 | OR A,@RW1 + d8 | XOR A,R1 | XOR A,@RW1 + d8 | DBNZ R1, r | DBNZ @RW1 + d8, r |
| +2 | ADD A,R2 | ADD A,@RW2 + d8 | SUB A,R2 | SUB A,@RW2 + d8 | ADDC A,R2 | ADDC A,@RW2 + d8 | CMP A,R2 | CMP A,@RW2 + d8 | AND A,R2 | AND A,@RW2 + d8 | OR A,R2 | OR A,@RW2 + d8 | XOR A,R2 | XOR A,@RW2 + d8 | DBNZ R2, r | DBNZ @RW2 + d8, r |
| +3 | ADD A,R3 | ADD A,@RW3 + d8 | SUB A,R3 | SUB A,@RW3 + d8 | ADDC A,R3 | ADDC A,@RW3 + d8 | CMP A,R3 | CMP A,@RW3 + d8 | AND A,R3 | AND A,@RW3 + d8 | OR A,R3 | OR A,@RW3 + d8 | XOR A,R3 | XOR A,@RW3 + d8 | DBNZ R3, r | DBNZ @RW3 + d8, r |
| +4 | ADD A,R4 | ADD A,@RW4 + d8 | SUB A,R4 | SUB A,@RW4 + d8 | ADDC A,R4 | ADDC A,@RW4 + d8 | CMP A,R4 | CMP A,@RW4 + d8 | AND A,R4 | AND A,@RW4 + d8 | OR A,R4 | OR A,@RW4 + d8 | XOR A,R4 | XOR A,@RW4 + d8 | DBNZ R4, r | DBNZ @RW4 + d8, r |
| +5 | ADD A,R5 | ADD A,@RW5 + d8 | SUB A,R5 | SUB A,@RW5 + d8 | ADDC A,R5 | ADDC A,@RW5 + d8 | CMP A,R5 | CMP A,@RW5 + d8 | AND A,R5 | AND A,@RW5 + d8 | OR A,R5 | OR A,@RW5 + d8 | XOR A,R5 | XOR A,@RW5 + d8 | DBNZ R5, r | DBNZ @RW5 + d8, r |
| +6 | ADD A,R6 | ADD A,@RW6 + d8 | SUB A,R6 | SUB A,@RW6 + d8 | ADDC A,R6 | ADDC A,@RW6 + d8 | CMP A,R6 | CMP A,@RW6 + d8 | AND A,R6 | AND A,@RW6 + d8 | OR A,R6 | OR A,@RW6 + d8 | XOR A,R6 | XOR A,@RW6 + d8 | DBNZ R6, r | DBNZ @RW6 + d8, r |
| +7 | ADD A,R7 | ADD A,@RW7 + d8 | SUB A,R7 | SUB A,@RW7 + d8 | ADDC A,R7 | ADDC A,@RW7 + d8 | CMP A,R7 | CMP A,@RW7 + d8 | AND A,R7 | AND A,@RW7 + d8 | OR A,R7 | OR A,@RW7 + d8 | XOR A,R7 | XOR A,@RW7 + d8 | DBNZ R7, r | DBNZ @RW7 + d8, r |
| +8 | ADD A,@RW0 | ADD A,@RW0 + d16 | SUB A,@RW0 | SUB A,@RW0 + d16 | ADDC A,@RW0 | ADDC A,@RW0 + d16 | CMP A,@RW0 | CMP A,@RW0 + d16 | AND A,@RW0 | AND A,@RW0 + d16 | OR A,@RW0 | OR A,@RW0 + d16 | XOR A,@RW0 | XOR A,@RW0 + d16 | DBNZ @RW0, r | DBNZ @RW0 + d16, r |
| +9 | ADD A,@RW1 | ADD A,@RW1 + d16 | SUB A,@RW1 | SUB A,@RW1 + d16 | ADDC A,@RW1 | ADDC A,@RW1 + d16 | CMP A,@RW1 | CMP A,@RW1 + d16 | AND A,@RW1 | AND A,@RW1 + d16 | OR A,@RW1 | OR A,@RW1 + d16 | XOR A,@RW1 | XOR A,@RW1 + d16 | DBNZ @RW1, r | DBNZ @RW1 + d16, r |
| +A | ADD A,@RW2 | ADD A,@RW2 + d16 | SUB A,@RW2 | SUB A,@RW2 + d16 | ADDC A,@RW2 | ADDC A,@RW2 + d16 | CMP A,@RW2 | CMP A,@RW2 + d16 | AND A,@RW2 | AND A,@RW2 + d16 | OR A,@RW2 | OR A,@RW2 + d16 | XOR A,@RW2 | XOR A,@RW2 + d16 | DBNZ @RW2, r | DBNZ @RW2 + d16, r |
| +B | ADD A,@RW3 | ADD A,@RW3 + d16 | SUB A,@RW3 | SUB A,@RW3 + d16 | ADDC A,@RW3 | ADDC A,@RW3 + d16 | CMP A,@RW3 | CMP A,@RW3 + d16 | AND A,@RW3 | AND A,@RW3 + d16 | OR A,@RW3 | OR A,@RW3 + d16 | XOR A,@RW3 | XOR A,@RW3 + d16 | DBNZ @RW3, r | DBNZ @RW3 + d16, r |
| +C | ADD A,@RW0+ | ADD A,@RW0+RW7 | SUB A,@RW0+ | SUB A,@RW0+RW7 | ADDC A,@RW0+ | ADDC A,@RW0+RW7 | CMP A,@RW0+ | CMP A,@RW0+RW7 | AND A,@RW0+ | AND A,@RW0+RW7 | OR A,@RW0+ | OR A,@RW0+RW7 | XOR A,@RW0+ | XOR A,@RW0+RW7 | DBNZ @RW0+, r | DBNZ @RW0+RW7, r |
| +D | ADD A,@RW1+ | ADD A,@RW1+RW7 | SUB A,@RW1+ | SUB A,@RW1+RW7 | ADDC A,@RW1+ | ADDC A,@RW1+RW7 | CMP A,@RW1+ | CMP A,@RW1+RW7 | AND A,@RW1+ | AND A,@RW1+RW7 | OR A,@RW1+ | OR A,@RW1+RW7 | XOR A,@RW1+ | XOR A,@RW1+RW7 | DBNZ @RW1+, r | DBNZ @RW1+RW7, r |
| +E | ADD A,@RW2+ | ADD A,@PC + d16 | SUB A,@RW2+ | SUB A,@PC + d16 | ADDC A,@RW2+ | ADDC A,@PC + d16 | CMP A,@RW2+ | CMP A,@PC + d16 | AND A,@RW2+ | AND A,@PC + d16 | OR A,@RW2+ | OR A,@PC + d16 | XOR A,@RW2+ | XOR A,@PC + d16 | DBNZ @RW2+, r | DBNZ @PC + d16, r |
| +F | ADD A,@RW3+ | ADD A,addr16 | SUB A,@RW3+ | SUB A,addr16 | ADDC A,@RW3+ | ADDC A,addr16 | CMP A,@RW3+ | CMP A,addr16 | AND A,@RW3+ | AND A,addr16 | OR A,@RW3+ | OR A,addr16 | XOR A,@RW3+ | XOR A,addr16 | DBNZ @RW3+, r | DBNZ @addr16, r |

Table 37-36. ea instruction 6 (first byte = $75_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | ADD R0,A | ADD @R W0 + d8,A | SUB R0,A | SUB @R W0 + d8,A | SUBC A,R0 | SUBC A, @RW0 + d8 | NEG R0 | NEG @RW0 + d8 | AND R0,A | AND @R W0 + d8,A | OR R0, A | OR @RW0 + d8, A | XOR R0, A | XOR @R W0 + d8, A | NOT R0 | NOT @RW0 + d8 |
| +1 | ADD R1,A | ADD @R W1 + d8,A | SUB R1,A | SUB @R W1 + d8,A | SUBC A,R1 | SUBC A, @RW1 + d8 | NEG R1 | NEG @RW1 + d8 | AND R1,A | AND @R W1 + d8,A | OR R1, A | OR @RW1 + d8, A | XOR R1, A | XOR @R W1 + d8, A | NOT R1 | NOT @RW1 + d8 |
| +2 | ADD R2,A | ADD @R W2 + d8,A | SUB R2,A | SUB @R W2 + d8,A | SUBC A,R2 | SUBC A, @RW2 + d8 | NEG R2 | NEG @RW2 + d8 | AND R2,A | AND @R W2 + d8,A | OR R2, A | OR @RW2 + d8, A | XOR R2, A | XOR @R W2 + d8, A | NOT R2 | NOT @RW2 + d8 |
| +3 | ADD R3,A | ADD @R W3 + d8,A | SUB R3,A | SUB @R W3 + d8,A | SUBC A,R3 | SUBC A, @RW3 + d8 | NEG R3 | NEG @RW3 + d8 | AND R3,A | AND @R W3 + d8,A | OR R3, A | OR @RW3 + d8, A | XOR R3, A | XOR @R W3 + d8, A | NOT R3 | NOT @RW3 + d8 |
| +4 | ADD R4,A | ADD @R W4 + d8,A | SUB R4,A | SUB @R W4 + d8,A | SUBC A,R4 | SUBC A, @RW4 + d8 | NEG R4 | NEG @RW4 + d8 | AND R4,A | AND @R W4 + d8,A | OR R4, A | OR @RW4 + d8, A | XOR R4, A | XOR @R W4 + d8, A | NOT R4 | NOT @RW4 + d8 |
| +5 | ADD R5,A | ADD @R W5 + d8,A | SUB R5,A | SUB @R W5 + d8,A | SUBC A,R5 | SUBC A, @RW5 + d8 | NEG R5 | NEG @RW5 + d8 | AND R5,A | AND @R W5 + d8,A | OR R5, A | OR @RW5 + d8, A | XOR R5, A | XOR @R W5 + d8, A | NOT R5 | NOT @RW5 + d8 |
| +6 | ADD R6,A | ADD @R W6 + d8,A | SUB R6,A | SUB @R W6 + d8,A | SUBC A,R6 | SUBC A, @RW6 + d8 | NEG R6 | NEG @RW6 + d8 | AND R6,A | AND @R W6 + d8,A | OR R6, A | OR @RW6 + d8, A | XOR R6, A | XOR @R W6 + d8, A | NOT R6 | NOT @RW6 + d8 |
| +7 | ADD R7,A | ADD @R W7 + d8,A | SUB R7,A | SUB @R W7 + d8,A | SUBC A,R7 | SUBC A, @RW7 + d8 | NEG R7 | NEG @RW7 + d8 | AND R7,A | AND @R W7 + d8,A | OR R7, A | OR @RW7 + d8, A | XOR R7, A | XOR @R W7 + d8, A | NOT R7 | NOT @RW7 + d8 |
| +8 | ADD @RW0,A | ADD @R W0 + d16,A | SUB @RW0,A | SUB @R W0 + d16,A | SUBC A,@RW0 | SUBC A, @RW0 + d16 | NEG @RW0 | NEG @RW0 + d16 | AND @RW0,A | AND @R W0 + d16,A | OR @RW0, A | OR @RW0 + d16, A | XOR @RW0, A | XOR @R W0 + d16, A | NOT @RW0 | NOT @RW0 + d16 |
| +9 | ADD @RW1,A | ADD @R W1 + d16,A | SUB @RW1,A | SUB @R W1 + d16,A | SUBC A,@RW1 | SUBC A, @RW1 + d16 | NEG @RW1 | NEG @RW1 + d16 | AND @RW1,A | AND @R W1 + d16,A | OR @RW1, A | OR @RW1 + d16, A | XOR @RW1, A | XOR @R W1 + d16, A | NOT @RW1 | NOT @RW1 + d16 |
| +A | ADD @RW2,A | ADD @R W2 + d16,A | SUB @RW2,A | SUB @R W2 + d16,A | SUBC A,@RW2 | SUBC A, @RW2 + d16 | NEG @RW2 | NEG @RW2 + d16 | AND @RW2,A | AND @R W2 + d16,A | OR @RW2, A | OR @RW2 + d16, A | XOR @RW2, A | XOR @R W2 + d16, A | NOT @RW2 | NOT @RW2 + d16 |
| +B | ADD @RW3,A | ADD @R W3 + d16,A | SUB @RW3,A | SUB @R W3 + d16,A | SUBC A, @RW3 | SUBC A, @RW3 + d16 | NEG @RW3 | NEG @RW3 + d16 | AND @RW3,A | AND @R W3 + d16,A | OR @RW3, A | OR @RW3 + d16, A | XOR @RW3, A | XOR @R W3 + d16, A | NOT @RW3 | NOT @RW3 + d16 |
| +C | ADD @RW0+,A | ADD @R W0+RW7,A | SUB @RW0+,A | SUB @R W0+RW7,A | SUBC A,@RW0+ | SUBC A, @RW0+RW7 | NEG @RW0+ | NEG @RW0+RW7 | AND @RW0+,A | AND @R W0+RW7,A | OR @RW0+, A | OR @R W0+RW7, A | XOR @RW0+, A | XOR @R W0+RW7, A | NOT @RW0+ | NOT @RW0+RW7 |
| +D | ADD @RW1+,A | ADD @R W1+RW7,A | SUB @RW1+,A | SUB @R W1+RW7,A | SUBC A,@RW1+ | SUBC A, @RW1+RW7 | NEG @RW1+ | NEG @RW1+RW7 | AND @RW1+,A | AND @R W1+RW7,A | OR @RW1+, A | OR @R W1+RW7, A | XOR @RW1+, A | XOR @R W1+RW7, A | NOT @RW1+ | NOT @RW1+RW7 |
| +E | ADD @RW2+,A | ADD @P C + d16,A | SUB @RW2+,A | SUB @P C + d16,A | SUBC A,@RW2+ | SUBC A, @PC + d16 | NEG @RW2+ | NEG @PC + d16 | AND @RW2+,A | AND @P C + d16,A | OR @RW2+, A | OR @PC + d16, A | XOR @RW2+, A | XOR @P C + d16, A | NOT @RW2+ | NOT @PC + d16 |
| +F | ADD @RW3+,A | ADD addr16,A | SUB @RW3+,A | SUB addr16,A | SUBC A,@RW3+ | SUBC A, addr16 | NEG @RW3+ | NEG addr16 | AND @RW3+,A | AND addr16,A | OR @RW3+, A | OR addr16, A | XOR @RW3+, A | XOR addr16, A | NOT @RW3+ | NOT addr16 |

Table 37-37. ea instruction 7 (first byte = $76_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | ADDW A, RW0 | ADDW A, @RW0 + d8 | SUBW A, RW0 | SUBW A, @RW0 + d8 | ADDCW A, RW0 | ADDCW A, @RW0 + d8 | CMPW A, RW0 | CMPW A, @RW0 + d8 | ANDW A, RW0 | ANDW A, @RW0 + d8 | ORW A, RW0 | ORW A, @RW0 + d8 | XORW A, RW0 | XORW A, @RW0 + d8 | DWBNZ RW0,r | DWBNZ @ RW0 + d8,r |
| +1 | ADDW A, RW1 | ADDW A, @RW1 + d8 | SUBW A, RW1 | SUBW A, @RW1 + d8 | ADDCW A, RW1 | ADDCW A, @RW1 + d8 | CMPW A, RW1 | CMPW A, @RW1 + d8 | ANDW A, RW1 | ANDW A, @RW1 + d8 | ORW A, RW1 | ORW A, @RW1 + d8 | XORW A, RW1 | XORW A, @RW1 + d8 | DWBNZ RW1,r | DWBNZ @ RW1 + d8,r |
| +2 | ADDW A, RW2 | ADDW A, @RW2 + d8 | SUBW A, RW2 | SUBW A, @RW2 + d8 | ADDCW A, RW2 | ADDCW A, @RW2 + d8 | CMPW A, RW2 | CMPW A, @RW2 + d8 | ANDW A, RW2 | ANDW A, @RW2 + d8 | ORW A, RW2 | ORW A, @RW2 + d8 | XORW A, RW2 | XORW A, @RW2 + d8 | DWBNZ RW2,r | DWBNZ @ RW2 + d8,r |
| +3 | ADDW A, RW3 | ADDW A, @RW3 + d8 | SUBW A, RW3 | SUBW A, @RW3 + d8 | ADDCW A, RW3 | ADDCW A, @RW3 + d8 | CMPW A, RW3 | CMPW A, @RW3 + d8 | ANDW A, RW3 | ANDW A, @RW3 + d8 | ORW A, RW3 | ORW A, @RW3 + d8 | XORW A, RW3 | XORW A, @RW3 + d8 | DWBNZ RW3,r | DWBNZ @ RW3 + d8,r |
| +4 | ADDW A, RW4 | ADDW A, @RW4 + d8 | SUBW A, RW4 | SUBW A, @RW4 + d8 | ADDCW A, RW4 | ADDCW A, @RW4 + d8 | CMPW A, RW4 | CMPW A, @RW4 + d8 | ANDW A, RW4 | ANDW A, @RW4 + d8 | ORW A, RW4 | ORW A, @RW4 + d8 | XORW A, RW4 | XORW A, @RW4 + d8 | DWBNZ RW4,r | DWBNZ @ RW4 + d8,r |
| +5 | ADDW A, RW5 | ADDW A, @RW5 + d8 | SUBW A, RW5 | SUBW A, @RW5 + d8 | ADDCW A, RW5 | ADDCW A, @RW5 + d8 | CMPW A, RW5 | CMPW A, @RW5 + d8 | ANDW A, RW5 | ANDW A, @RW5 + d8 | ORW A, RW5 | ORW A, @RW5 + d8 | XORW A, RW5 | XORW A, @RW5 + d8 | DWBNZ RW5,r | DWBNZ @ RW5 + d8,r |
| +6 | ADDW A, RW6 | ADDW A, @RW6 + d8 | SUBW A, RW6 | SUBW A, @RW6 + d8 | ADDCW A, RW6 | ADDCW A, @RW6 + d8 | CMPW A, RW6 | CMPW A, @RW6 + d8 | ANDW A, RW6 | ANDW A, @RW6 + d8 | ORW A, RW6 | ORW A, @RW6 + d8 | XORW A, RW6 | XORW A, @RW6 + d8 | DWBNZ RW6,r | DWBNZ @ RW6 + d8,r |
| +7 | ADDW A, RW7 | ADDW A, @RW7 + d8 | SUBW A, RW7 | SUBW A, @RW7 + d8 | ADDCW A, RW7 | ADDCW A, @RW7 + d8 | CMPW A, RW7 | CMPW A, @RW7 + d8 | ANDW A, RW7 | ANDW A, @RW7 + d8 | ORW A, RW7 | ORW A, @RW7 + d8 | XORW A, RW7 | XORW A, @RW7 + d8 | DWBNZ RW7,r | DWBNZ @ RW7 + d8,r |
| +8 | ADDW A, @RW0 | ADDW A, @RW0 + d16 | SUBW A, @RW0 | SUBW A, @RW0 + d16 | ADDCW A, @RW0 | ADDCW A, @RW0 + d16 | CMPW A, @RW0 | CMPW A, @RW0 + d16 | ANDW A, @RW0 | ANDW A, @RW0 + d16 | ORW A, @RW0 | ORW A, @RW0 + d16 | XORW A, @RW0 | XORW A, @RW0 + d16 | DWBNZ @RW0,r | DWBNZ @R W0 + d16,r |
| +9 | ADDW A, @RW1 | ADDW A, @RW1 + d16 | SUBW A, @RW1 | SUBW A, @RW1 + d16 | ADDCW A, @RW1 | ADDCW A, @RW1 + d16 | CMPW A, @RW1 | CMPW A, @RW1 + d16 | ANDW A, @RW1 | ANDW A, @RW1 + d16 | ORW A, @RW1 | ORW A, @RW1 + d16 | XORW A, @RW1 | XORW A, @RW1 + d16 | DWBNZ @RW1,r | DWBNZ @R W1 + d16,r |
| +A | ADDW A, @RW2 | ADDW A, @RW2 + d16 | SUBW A, @RW2 | SUBW A, @RW2 + d16 | ADDCW A, @RW2 | ADDCW A, @RW2 + d16 | CMPW A, @RW2 | CMPW A, @RW2 + d16 | ANDW A, @RW2 | ANDW A, @RW2 + d16 | ORW A, @RW2 | ORW A, @RW2 + d16 | XORW A, @RW2 | XORW A, @RW2 + d16 | DWBNZ @RW2,r | DWBNZ @R W2 + d16,r |
| +B | ADDW A, @RW3 | ADDW A, @RW3 + d16 | SUBW A, @RW3 | SUBW A, @RW3 + d16 | ADDCW A, @RW3 | ADDCW A, @RW3 + d16 | CMPW A, @RW3 | CMPW A, @RW3 + d16 | ANDW A, @RW3 | ANDW A, @RW3 + d16 | ORW A, @RW3 | ORW A, @RW3 + d16 | XORW A, @RW3 | XORW A, @RW3 + d16 | DWBNZ @RW3,r | DWBNZ @R W3 + d16,r |
| +C | ADDW A, @RW0+ | ADDW A, @RW0+RW7 | SUBW A, @RW0+ | SUBW A, @RW0+RW7 | ADDCW A, @RW0+ | ADDCW A, @RW0+RW7 | CMPW A, @RW0+ | CMPW A, @RW0+RW7 | ANDW A, @RW0+ | ANDW A, @RW0+RW7 | ORW A, @RW0+ | ORW A, @RW0+RW7 | XORW A, @RW0+ | XORW A, @RW0+RW7 | DWBNZ @RW0+,r | DWBNZ @R W0+RW7,r |
| +D | ADDW A, @RW1+ | ADDW A, @RW1+RW7 | SUBW A, @RW1+ | SUBW A, @RW1+RW7 | ADDCW A, @RW1+ | ADDCW A, @RW1+RW7 | CMPW A, @RW1+ | CMPW A, @RW1+RW7 | ANDW A, @RW1+ | ANDW A, @RW1+RW7 | ORW A, @RW1+ | ORW A, @RW1+RW7 | XORW A, @RW1+ | XORW A, @RW1+RW7 | DWBNZ @RW1+,r | DWBNZ @R W1+RW7,r |
| +E | ADDW A, @RW2+ | ADDW A, @PC + d16 | SUBW A, @RW2+ | SUBW A, @PC + d16 | ADDCW A, @RW2+ | ADDCW A, @PC + d16 | CMPW A, @RW2+ | CMPW A, @PC + d16 | ANDW A, @RW2+ | ANDW A, @PC + d16 | ORW A, @RW2+ | ORW A, @PC + d16 | XORW A, @RW2+ | XORW A, @PC + d16 | DWBNZ @RW2+,r | DWBNZ @ PC + d16,r |
| +F | ADDW A, @RW3+ | ADDW A, addr16 | SUBW A, @RW3+ | SUBW A, addr16 | ADDCW A, @RW3+ | ADDCW A, addr16 | CMPW A, @RW3+ | CMPW A, addr16 | ANDW A, @RW3+ | ANDW A, addr16 | ORW A, @RW3+ | ORW A, addr16 | XORW A, @RW3+ | XORW A, addr16 | DWBNZ @RW3+,r | DWBNZ addr16,r |

Table 37-38. ea instruction 8 (first byte = $77_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | ADDW RW0, A | ADDW @R W0 + d8, A | SUBW RW0, A | SUBW @R W0 + d8, A | SUBCW A, RW0 | SUBCW A, @RW0 + d8 | NEGW RW0 | NEGW @RW0 + d8 | ANDW RW0, A | ANDW @R W0 + d8, A | ORW RW0, A | ORW @R W0 + d8, A | XORW RW0, A | XORW @R W0 + d8, A | NOTW RW0 | NOTW @RW0 + d8 |
| +1 | ADDW RW1, A | ADDW @R W1 + d8, A | SUBW RW1, A | SUBW @R W1 + d8, A | SUBCW A, RW1 | SUBCW A, @RW1 + d8 | NEGW RW1 | NEGW @RW1 + d8 | ANDW RW1, A | ANDW @R W1 + d8, A | ORW RW1, A | ORW @R W1 + d8, A | XORW RW1, A | XORW @R W1 + d8, A | NOTW RW1 | NOTW @RW1 + d8 |
| +2 | ADDW RW2, A | ADDW @R W2 + d8, A | SUBW RW2, A | SUBW @R W2 + d8, A | SUBCW A, RW2 | SUBCW A, @RW2 + d8 | NEGW RW2 | NEGW @RW2 + d8 | ANDW RW2, A | ANDW @R W2 + d8, A | ORW RW2, A | ORW @R W2 + d8, A | XORW RW2, A | XORW @R W2 + d8, A | NOTW RW2 | NOTW @RW2 + d8 |
| +3 | ADDW RW3, A | ADDW @R W3 + d8, A | SUBW RW3, A | SUBW @R W3 + d8, A | SUBCW A, RW3 | SUBCW A, @RW3 + d8 | NEGW RW3 | NEGW @RW3 + d8 | ANDW RW3, A | ANDW @R W3 + d8, A | ORW RW3, A | ORW @R W3 + d8, A | XORW RW3, A | XORW @R W3 + d8, A | NOTW RW3 | NOTW @RW3 + d8 |
| +4 | ADDW RW4, A | ADDW @R W4 + d8, A | SUBW RW4, A | SUBW @R W4 + d8, A | SUBCW A, RW4 | SUBCW A, @RW4 + d8 | NEGW RW4 | NEGW @RW4 + d8 | ANDW RW4, A | ANDW @R W4 + d8, A | ORW RW4, A | ORW @R W4 + d8, A | XORW RW4, A | XORW @R W4 + d8, A | NOTW RW4 | NOTW @RW4 + d8 |
| +5 | ADDW RW5, A | ADDW @R W5 + d8, A | SUBW RW5, A | SUBW @R W5 + d8, A | SUBCW A, RW5 | SUBCW A, @RW5 + d8 | NEGW RW5 | NEGW @RW5 + d8 | ANDW RW5, A | ANDW @R W5 + d8, A | ORW RW5, A | ORW @R W5 + d8, A | XORW RW5, A | XORW @R W5 + d8, A | NOTW RW5 | NOTW @RW5 + d8 |
| +6 | ADDW RW6, A | ADDW @R W6 + d8, A | SUBW RW6, A | SUBW @R W6 + d8, A | SUBCW A, RW6 | SUBCW A, @RW6 + d8 | NEGW RW6 | NEGW @RW6 + d8 | ANDW RW6, A | ANDW @R W6 + d8, A | ORW RW6, A | ORW @R W6 + d8, A | XORW RW6, A | XORW @R W6 + d8, A | NOTW RW6 | NOTW @RW6 + d8 |
| +7 | ADDW RW7, A | ADDW @R W7 + d8, A | SUBW RW7, A | SUBW @R W7 + d8, A | SUBCW A, RW7 | SUBCW A, @RW7 + d8 | NEGW RW7 | NEGW @RW7 + d8 | ANDW RW7, A | ANDW @R W7 + d8, A | ORW RW7, A | ORW @R W7 + d8, A | XORW RW7, A | XORW @R W7 + d8, A | NOTW RW7 | NOTW @RW7 + d8 |
| +8 | ADDW @RW0, A | ADDW @R W0 + d16, A | SUBW @RW0, A | SUBW @R W0 + d16, A | SUBCW A, @RW0 | SUBCW A, @RW0 + d16 | NEGW @RW0 | NEGW @RW0 + d16 | ANDW @RW0, A | ANDW @R W0 + d16, A | ORW @RW0, A | ORW @R W0 + d16, A | XORW @RW0, A | XORW @R W0 + d16, A | NOTW @RW0 | NOTW @RW0 + d16 |
| +9 | ADDW @RW1, A | ADDW @R W1 + d16, A | SUBW @RW1, A | SUBW @R W1 + d16, A | SUBCW A, @RW1 | SUBCW A, @RW1 + d16 | NEGW @RW1 | NEGW @RW1 + d16 | ANDW @RW1, A | ANDW @R W1 + d16, A | ORW @RW1, A | ORW @R W1 + d16, A | XORW @RW1, A | XORW @R W1 + d16, A | NOTW @RW1 | NOTW @RW1 + d16 |
| +A | ADDW @RW2, A | ADDW @R W2 + d16, A | SUBW @RW2, A | SUBW @R W2 + d16, A | SUBCW A, @RW2 | SUBCW A, @RW2 + d16 | NEGW @RW2 | NEGW @RW2 + d16 | ANDW @RW2, A | ANDW @R W2 + d16, A | ORW @RW2, A | ORW @R W2 + d16, A | XORW @RW2, A | XORW @R W2 + d16, A | NOTW @RW2 | NOTW @RW2 + d16 |
| +B | ADDW @RW3, A | ADDW @R W3 + d16, A | SUBW @RW3, A | SUBW @R W3 + d16, A | SUBCW A, @RW3 | SUBCW A, @RW3 + d16 | NEGW @RW3 | NEGW @RW3 + d16 | ANDW @RW3, A | ANDW @R W3 + d16, A | ORW @RW3, A | ORW @R W3 + d16, A | XORW @RW3, A | XORW @R W3 + d16, A | NOTW @RW3 | NOTW @RW3 + d16 |
| +C | ADDW @RW0+, A | ADDW @R W0+RW7, A | SUBW @RW0+, A | SUBW @R W0+RW7, A | SUBCW A, @RW0+ | SUBCW A, @RW0+RW7 | NEGW @RW0+ | NEGW @RW0+RW7 | ANDW @RW0+, A | ANDW @R W0+RW7, A | ORW @RW0+, A | ORW @R W0+RW7, A | XORW @RW0+, A | XORW @R W0+RW7, A | NOTW @RW0+ | NOTW @RW0+RW7 |
| +D | ADDW @RW1+, A | ADDW @R W1+RW7, A | SUBW @RW1+, A | SUBW @R W1+RW7, A | SUBCW A, @RW1+ | SUBCW A, @RW1+RW7 | NEGW @RW1+ | NEGW @RW1+RW7 | ANDW @RW1+, A | ANDW @R W1+RW7, A | ORW @RW1+, A | ORW @R W1+RW7, A | XORW @RW1+, A | XORW @R W1+RW7, A | NOTW @RW1+ | NOTW @RW1+RW7 |
| +E | ADDW @RW2+, A | ADDW @P C + d16, A | SUBW @RW2+, A | SUBW @P C + d16, A | SUBCW A, @RW2+ | SUBCW A, @PC + d16 | NEGW @RW2+ | NEGW @PC + d16 | ANDW @RW2+, A | ANDW @P C + d16, A | ORW @RW2+, A | ORW @P C + d16, A | XORW @RW2+, A | XORW @P C + d16, A | NOTW @RW2+ | NOTW @PC + d16 |
| +F | ADDW @RW3+, A | ADDW addr16, A | SUBW @RW3+, A | SUBW addr16, A | SUBCW A, @RW3+ | SUBCW A, addr16 | NEGW @RW3+ | NEGW addr16 | ANDW @RW3+, A | ANDW addr16, A | ORW @RW3+, A | ORW addr16, A | XORW @RW3+, A | XORW addr16, A | NOTW @RW3+ | NOTW addr16 |

Table 37-39. ea instruction 9 (first byte = $78_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MULU A, R0 | MULU A, @RW0 + d8 | MULUW A, RW0 | MULUW A, @RW0 + d8 | MUL A, R0 | MUL A, @RW0 + d8 | MULW A, RW0 | MULW A, @RW0 + d8 | DIVU A, R0 | DIVU A, @RW0 + d8 | DIVUW A, RW0 | DIVUW A, @RW0 + d8 | DIV A, R0 | DIV A, @RW0 + d8 | DIVW A, RW0 | DIVW A, @RW0 + d8 |
| +1 | MULU A, R1 | MULU A, @RW1 + d8 | MULUW A, RW1 | MULUW A, @RW1 + d8 | MUL A, R1 | MUL A, @RW1 + d8 | MULW A, RW1 | MULW A, @RW1 + d8 | DIVU A, R1 | DIVU A, @RW1 + d8 | DIVUW A, RW1 | DIVUW A, @RW1 + d8 | DIV A, R1 | DIV A, @RW1 + d8 | DIVW A, RW1 | DIVW A, @RW1 + d8 |
| +2 | MULU A, R2 | MULU A, @RW2 + d8 | MULUW A, RW2 | MULUW A, @RW2 + d8 | MUL A, R2 | MUL A, @RW2 + d8 | MULW A, RW2 | MULW A, @RW2 + d8 | DIVU A, R2 | DIVU A, @RW2 + d8 | DIVUW A, RW2 | DIVUW A, @RW2 + d8 | DIV A, R2 | DIV A, @RW2 + d8 | DIVW A, RW2 | DIVW A, @RW2 + d8 |
| +3 | MULU A, R3 | MULU A, @RW3 + d8 | MULUW A, RW3 | MULUW A, @RW3 + d8 | MUL A, R3 | MUL A, @RW3 + d8 | MULW A, RW3 | MULW A, @RW3 + d8 | DIVU A, R3 | DIVU A, @RW3 + d8 | DIVUW A, RW3 | DIVUW A, @RW3 + d8 | DIV A, R3 | DIV A, @RW3 + d8 | DIVW A, RW3 | DIVW A, @RW3 + d8 |
| +4 | MULU A, R4 | MULU A, @RW4 + d8 | MULUW A, RW4 | MULUW A, @RW4 + d8 | MUL A, R4 | MUL A, @RW4 + d8 | MULW A, RW4 | MULW A, @RW4 + d8 | DIVU A, R4 | DIVU A, @RW4 + d8 | DIVUW A, RW4 | DIVUW A, @RW4 + d8 | DIV A, R4 | DIV A, @RW4 + d8 | DIVW A, RW4 | DIVW A, @RW4 + d8 |
| +5 | MULU A, R5 | MULU A, @RW5 + d8 | MULUW A, RW5 | MULUW A, @RW5 + d8 | MUL A, R5 | MUL A, @RW5 + d8 | MULW A, RW5 | MULW A, @RW5 + d8 | DIVU A, R5 | DIVU A, @RW5 + d8 | DIVUW A, RW5 | DIVUW A, @RW5 + d8 | DIV A, R5 | DIV A, @RW5 + d8 | DIVW A, RW5 | DIVW A, @RW5 + d8 |
| +6 | MULU A, R6 | MULU A, @RW6 + d8 | MULUW A, RW6 | MULUW A, @RW6 + d8 | MUL A, R6 | MUL A, @RW6 + d8 | MULW A, RW6 | MULW A, @RW6 + d8 | DIVU A, R6 | DIVU A, @RW6 + d8 | DIVUW A, RW6 | DIVUW A, @RW6 + d8 | DIV A, R6 | DIV A, @RW6 + d8 | DIVW A, RW6 | DIVW A, @RW6 + d8 |
| +7 | MULU A, R7 | MULU A, @RW7 + d8 | MULUW A, RW7 | MULUW A, @RW7 + d8 | MUL A, R7 | MUL A, @RW7 + d8 | MULW A, RW7 | MULW A, @RW7 + d8 | DIVU A, R7 | DIVU A, @RW7 + d8 | DIVUW A, RW7 | DIVUW A, @RW7 + d8 | DIV A, R7 | DIV A, @RW7 + d8 | DIVW A, RW7 | DIVW A, @RW7 + d8 |
| +8 | MULU A, @RW0 | MULU A, @RW0 + d16 | MULUW A, @RW0 | MULUW A, @RW0 + d16 | MUL A, @RW0 | MUL A, @RW0 + d16 | MULW A, @RW0 | MULW A, @RW0 + d16 | DIVU A, @RW0 | DIVU A, @RW0 + d16 | DIVUW A, @RW0 | DIVUW A, @RW0 + d16 | DIV A, @RW0 | DIV A, @RW0 + d16 | DIVW A, @RW0 | DIVW A, @RW0 + d16 |
| +9 | MULU A, @RW1 | MULU A, @RW1 + d16 | MULUW A, @RW1 | MULUW A, @RW1 + d16 | MUL A, @RW1 | MUL A, @RW1 + d16 | MULW A, @RW1 | MULW A, @RW1 + d16 | DIVU A, @RW1 | DIVU A, @RW1 + d16 | DIVUW A, @RW1 | DIVUW A, @RW1 + d16 | DIV A, @RW1 | DIV A, @RW1 + d16 | DIVW A, @RW1 | DIVW A, @RW1 + d16 |
| +A | MULU A, @RW2 | MULU A, @RW2 + d16 | MULUW A, @RW2 | MULUW A, @RW2 + d16 | MUL A, @RW2 | MUL A, @RW2 + d16 | MULW A, @RW2 | MULW A, @RW2 + d16 | DIVU A, @RW2 | DIVU A, @RW2 + d16 | DIVUW A, @RW2 | DIVUW A, @RW2 + d16 | DIV A, @RW2 | DIV A, @RW2 + d16 | DIVW A, @RW2 | DIVW A, @RW2 + d16 |
| +B | MULU A, @RW3 | MULU A, @RW3 + d16 | MULUW A, @RW3 | MULUW A, @RW3 + d16 | MUL A, @RW3 | MUL A, @RW3 + d16 | MULW A, @RW3 | MULW A, @RW3 + d16 | DIVU A, @RW3 | DIVU A, @RW3 + d16 | DIVUW A, @RW3 | DIVUW A, @RW3 + d16 | DIV A, @RW3 | DIV A, @RW3 + d16 | DIVW A, @RW3 | DIVW A, @RW3 + d16 |
| +C | MULU A, @RW0+ | MULU A, @RW0-RW7 | MULUW A, @RW0+ | MULUW A, @RW0-RW7 | MUL A, @RW0+ | MUL A, @RW0-RW7 | MULW A, @RW0+ | MULW A, @RW0-RW7 | DIVU A, @RW0+ | DIVU A, @RW0-RW7 | DIVUW A, @RW0+ | DIVUW A, @RW0-RW7 | DIV A, @RW0+ | DIV A, @RW0-RW7 | DIVW A, @RW0+ | DIVW A, @RW0-RW7 |
| +D | MULU A, @RW1+ | MULU A, @RW1+RW7 | MULUW A, @RW1+ | MULUW A, @RW1+RW7 | MUL A, @RW1+ | MUL A, @RW1+RW7 | MULW A, @RW1+ | MULW A, @RW1+RW7 | DIVU A, @RW1+ | DIVU A, @RW1+RW7 | DIVUW A, @RW1+ | DIVUW A, @RW1+RW7 | DIV A, @RW1+ | DIV A, @RW1+RW7 | DIVW A, @RW1+ | DIVW A, @RW1+RW7 |
| +E | MULU A, @RW2+ | MULU A, @PC + d16 | MULUW A, @RW2+ | MULUW A, @PC + d16 | MUL A, @RW2+ | MUL A, @PC + d16 | MULW A, @RW2+ | MULW A, @PC + d16 | DIVU A, @RW2+ | DIVU A, @PC + d16 | DIVUW A, @RW2+ | DIVUW A, @PC + d16 | DIV A, @RW2+ | DIV A, @PC + d16 | DIVW A, @RW2+ | DIVW A, @PC + d16 |
| +F | MULU A, @RW3+ | MULU A, addr16 | MULUW A, @RW3+ | MULUW A, addr16 | MUL A, @RW3+ | MUL A, addr16 | MULW A, @RW3+ | MULW A, addr16 | DIVU A, @RW3+ | DIVU A, addr16 | DIVUW A, @RW3+ | DIVUW A, addr16 | DIV A, @RW3+ | DIV A, addr16 | DIVW A, @RW3+ | DIVW A, addr16 |

Table 37-40. MOVEA RWi, ea instruction (first byte = $79_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOVEA RW0,RW0 | MOVEA RW0, @RW0 + d8 | MOVEA RW1,RW0 | MOVEA RW1, @RW0 + d8 | MOVEA RW2,RW0 | MOVEA RW2, @RW0 + d8 | MOVEA RW3,RW0 | MOVEA RW3, @RW0 + d8 | MOVEA RW4,RW0 | MOVEA RW4, @RW0 + d8 | MOVEA RW5,RW0 | MOVEA RW5, @RW0 + d8 | MOVEA RW6,RW0 | MOVEA RW6 ,@RW0 + d8 | MOVEA RW7,RW0 | MOVEA RW7, @RW0 + d8 |
| +1 | MOVEA RW0,RW1 | MOVEA RW0, @RW1 + d8 | MOVEA RW1,RW1 | MOVEA RW1, @RW1 + d8 | MOVEA RW2,RW1 | MOVEA RW2, @RW1 + d8 | MOVEA RW3,RW1 | MOVEA RW3, @RW1 + d8 | MOVEA RW4,RW1 | MOVEA RW4, @RW1 + d8 | MOVEA RW5,RW1 | MOVEA RW5, @RW1 + d8 | MOVEA RW6,RW1 | MOVEA RW6 ,@RW1 + d8 | MOVEA RW7,RW1 | MOVEA RW7, @RW1 + d8 |
| +2 | MOVEA RW0,RW2 | MOVEA RW0, @RW2 + d8 | MOVEA RW1,RW2 | MOVEA RW1, @RW2 + d8 | MOVEA RW2,RW2 | MOVEA RW2, @RW2 + d8 | MOVEA RW3,RW2 | MOVEA RW3, @RW2 + d8 | MOVEA RW4,RW2 | MOVEA RW4, @RW2 + d8 | MOVEA RW5,RW2 | MOVEA RW5, @RW2 + d8 | MOVEA RW6,RW2 | MOVEA RW6 ,@RW2 + d8 | MOVEA RW7,RW2 | MOVEA RW7, @RW2 + d8 |
| +3 | MOVEA RW0,RW3 | MOVEA RW0, @RW3 + d8 | MOVEA RW1,RW3 | MOVEA RW1, @RW3 + d8 | MOVEA RW2,RW3 | MOVEA RW2, @RW3 + d8 | MOVEA RW3,RW3 | MOVEA RW3, @RW3 + d8 | MOVEA RW4,RW3 | MOVEA RW4, @RW3 + d8 | MOVEA RW5,RW3 | MOVEA RW5, @RW3 + d8 | MOVEA RW6,RW3 | MOVEA RW6 ,@RW3 + d8 | MOVEA RW7,RW3 | MOVEA RW7, @RW3 + d8 |
| +4 | MOVEA RW0,RW4 | MOVEA RW0, @RW4 + d8 | MOVEA RW1,RW4 | MOVEA RW1, @RW4 + d8 | MOVEA RW2,RW4 | MOVEA RW2, @RW4 + d8 | MOVEA RW3,RW4 | MOVEA RW3, @RW4 + d8 | MOVEA RW4,RW4 | MOVEA RW4, @RW4 + d8 | MOVEA RW5,RW4 | MOVEA RW5, @RW4 + d8 | MOVEA RW6,RW4 | MOVEA RW6 ,@RW4 + d8 | MOVEA RW7,RW4 | MOVEA RW7, @RW4 + d8 |
| +5 | MOVEA RW0,RW5 | MOVEA RW0, @RW5 + d8 | MOVEA RW1,RW5 | MOVEA RW1, @RW5 + d8 | MOVEA RW2,RW5 | MOVEA RW2, @RW5 + d8 | MOVEA RW3,RW5 | MOVEA RW3, @RW5 + d8 | MOVEA RW4,RW5 | MOVEA RW4, @RW5 + d8 | MOVEA RW5,RW5 | MOVEA RW5, @RW5 + d8 | MOVEA RW6,RW5 | MOVEA RW6 ,@RW5 + d8 | MOVEA RW7,RW5 | MOVEA RW7, @RW5 + d8 |
| +6 | MOVEA RW0,RW6 | MOVEA RW0, @RW6 + d8 | MOVEA RW1,RW6 | MOVEA RW1, @RW6 + d8 | MOVEA RW2,RW6 | MOVEA RW2, @RW6 + d8 | MOVEA RW3,RW6 | MOVEA RW3, @RW6 + d8 | MOVEA RW4,RW6 | MOVEA RW4, @RW6 + d8 | MOVEA RW5,RW6 | MOVEA RW5, @RW6 + d8 | MOVEA RW6,RW6 | MOVEA RW6 ,@RW6 + d8 | MOVEA RW7,RW6 | MOVEA RW7, @RW6 + d8 |
| +7 | MOVEA RW0,RW7 | MOVEA RW0, @RW7 + d8 | MOVEA RW1,RW7 | MOVEA RW1, @RW7 + d8 | MOVEA RW2,RW7 | MOVEA RW2, @RW7 + d8 | MOVEA RW3,RW7 | MOVEA RW3, @RW7 + d8 | MOVEA RW4,RW7 | MOVEA RW4, @RW7 + d8 | MOVEA RW5,RW7 | MOVEA RW5, @RW7 + d8 | MOVEA RW6,RW7 | MOVEA RW6 ,@RW7 + d8 | MOVEA RW7,RW7 | MOVEA RW7, @RW7 + d8 |
| +8 | MOVEA RW0,@RW0 | MOVEA RW0, @RW0 + d16 | MOVEA RW1,@RW0 | MOVEA RW1, @RW0 + d16 | MOVEA RW2,@RW0 | MOVEA RW2, @RW0 + d16 | MOVEA RW3,@RW0 | MOVEA RW3, @RW0 + d16 | MOVEA RW4,@RW0 | MOVEA RW4, @RW0 + d16 | MOVEA RW5,@RW0 | MOVEA RW5, @RW0 + d16 | MOVEA RW6,@RW0 | MOVEA RW6 ,@RW0 + d16 | MOVEA RW7,@RW0 | MOVEA RW7, @RW0 + d16 |
| +9 | MOVEA RW0,@RW1 | MOVEA RW0, @RW1 + d16 | MOVEA RW1,@RW1 | MOVEA RW1, @RW1 + d16 | MOVEA RW2,@RW1 | MOVEA RW2, @RW1 + d16 | MOVEA RW3,@RW1 | MOVEA RW3, @RW1 + d16 | MOVEA RW4,@RW1 | MOVEA RW4, @RW1 + d16 | MOVEA RW5,@RW1 | MOVEA RW5, @RW1 + d16 | MOVEA RW6,@RW1 | MOVEA RW6 ,@RW1 + d16 | MOVEA RW7,@RW1 | MOVEA RW7, @RW1 + d16 |
| +A | MOVEA RW0,@RW2 | MOVEA RW0, @RW2 + d16 | MOVEA RW1,@RW2 | MOVEA RW1, @RW2 + d16 | MOVEA RW2,@RW2 | MOVEA RW2, @RW2 + d16 | MOVEA RW3,@RW2 | MOVEA RW3, @RW2 + d16 | MOVEA RW4,@RW2 | MOVEA RW4, @RW2 + d16 | MOVEA RW5,@RW2 | MOVEA RW5, @RW2 + d16 | MOVEA RW6,@RW2 | MOVEA RW6 ,@RW2 + d16 | MOVEA RW7,@RW2 | MOVEA RW7, @RW2 + d16 |
| +B | MOVEA RW0,@RW3 | MOVEA RW0, @RW3 + d16 | MOVEA RW1,@RW3 | MOVEA RW1, @RW3 + d16 | MOVEA RW2,@RW3 | MOVEA RW2, @RW3 + d16 | MOVEA RW3,@RW3 | MOVEA RW3, @RW3 + d16 | MOVEA RW4,@RW3 | MOVEA RW4, @RW3 + d16 | MOVEA RW5,@RW3 | MOVEA RW5, @RW3 + d16 | MOVEA RW6,@RW3 | MOVEA RW6 ,@RW3 + d16 | MOVEA RW7,@RW3 | MOVEA RW7, @RW3 + d16 |
| +C | MOVEA R W0,@RW0+ | MOVEA RW0, @RW0+RW7 | MOVEA R W1,@RW0+ | MOVEA RW1, @RW0+RW7 | MOVEA R W2,@RW0+ | MOVEA RW2, @RW0+RW7 | MOVEA R W3,@RW0+ | MOVEA RW3, @RW0+RW7 | MOVEA R W4,@RW0+ | MOVEA RW4, @RW0+RW7 | MOVEA R W5,@RW0+ | MOVEA RW5, @RW0+RW7 | MOVEA R W6,@RW0+ | MOVEA RW6 ,@RW0+RW7 | MOVEA R W7,@RW0+ | MOVEA RW7, @RW0+RW7 |
| +D | MOVEA R W0,@RW1+ | MOVEA RW0, @RW1+RW7 | MOVEA R W1,@RW1+ | MOVEA RW1, @RW1+RW7 | MOVEA R W2,@RW1+ | MOVEA RW2, @RW1+RW7 | MOVEA R W3,@RW1+ | MOVEA RW3, @RW1+RW7 | MOVEA R W4,@RW1+ | MOVEA RW4, @RW1+RW7 | MOVEA R W5,@RW1+ | MOVEA RW5, @RW1+RW7 | MOVEA R W6,@RW1+ | MOVEA RW6 ,@RW1+RW7 | MOVEA R W7,@RW1+ | MOVEA RW7, @RW1+RW7 |
| +E | MOVEA R W0,@RW2+ | MOVEA RW0, @PC + d16 | MOVEA R W1,@RW2+ | MOVEA RW1, @PC + d16 | MOVEA R W2,@RW2+ | MOVEA RW2, @PC + d16 | MOVEA R W3,@RW2+ | MOVEA RW3, @PC + d16 | MOVEA R W4,@RW2+ | MOVEA RW4, @PC + d16 | MOVEA R W5,@RW2+ | MOVEA RW5, @PC + d16 | MOVEA R W6,@RW2+ | MOVEA RW6 ,@PC + d16 | MOVEA R W7,@RW2+ | MOVEA RW7, @PC + d16 |
| +F | MOVEA R W0,@RW3+ | MOVEA RW0, addr16 | MOVEA R W1,@RW3+ | MOVEA RW1, addr16 | MOVEA R W2,@RW3+ | MOVEA RW2, addr16 | MOVEA R W3,@RW3+ | MOVEA RW3, addr16 | MOVEA R W4,@RW3+ | MOVEA RW4, addr16 | MOVEA R W5,@RW3+ | MOVEA RW5, addr16 | MOVEA R W6,@RW3+ | MOVEA RW6 ,addr16 | MOVEA R W7,@RW3+ | MOVEA RW7, addr16 |

Table 37-41. MOV Ri, ea instruction (first byte = $7A_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOV R0,R0 | MOV R0, @RW0 + d8 | MOV R1,R0 | MOV R1. @RW0 + d8 | MOV R2,R0 | MOV R2. @RW0 + d8 | MOV R3,R0 | MOV R3. @RW0 + d8 | MOV R4,R0 | MOV R4. @RW0 + d8 | MOV R5,R0 | MOV R5. @RW0 + d8 | MOV R6,R0 | MOV R6. @RW0 + d8 | MOV R7,R0 | MOV R7. @RW0 + d8 |
| +1 | MOV R0,R1 | MOV R0. @RW1 + d8 | MOV R1,R1 | MOV R1. @RW1 + d8 | MOV R2,R1 | MOV R2. @RW1 + d8 | MOV R3,R1 | MOV R3. @RW1 + d8 | MOV R4,R1 | MOV R4. @RW1 + d8 | MOV R5,R1 | MOV R5. @RW1 + d8 | MOV R6,R1 | MOV R6. @RW1 + d8 | MOV R7,R1 | MOV R7. @RW1 + d8 |
| +2 | MOV R0,R2 | MOV R0. @RW2 + d8 | MOV R1,R2 | MOV R1. @RW2 + d8 | MOV R2,R2 | MOV R2. @RW2 + d8 | MOV R3,R2 | MOV R3. @RW2 + d8 | MOV R4,R2 | MOV R4. @RW2 + d8 | MOV R5,R2 | MOV R5. @RW2 + d8 | MOV R6,R2 | MOV R6. @RW2 + d8 | MOV R7,R2 | MOV R7. @RW2 + d8 |
| +3 | MOV R0,R3 | MOV R0. @RW3 + d8 | MOV R1,R3 | MOV R1. @RW3 + d8 | MOV R2,R3 | MOV R2. @RW3 + d8 | MOV R3,R3 | MOV R3. @RW3 + d8 | MOV R4,R3 | MOV R4. @RW3 + d8 | MOV R5,R3 | MOV R5. @RW3 + d8 | MOV R6,R3 | MOV R6. @RW3 + d8 | MOV R7,R3 | MOV R7. @RW3 + d8 |
| +4 | MOV R0,R4 | MOV R0. @RW4 + d8 | MOV R1,R4 | MOV R1. @RW4 + d8 | MOV R2,R4 | MOV R2. @RW4 + d8 | MOV R3,R4 | MOV R3. @RW4 + d8 | MOV R4,R4 | MOV R4. @RW4 + d8 | MOV R5,R4 | MOV R5. @RW4 + d8 | MOV R6,R4 | MOV R6. @RW4 + d8 | MOV R7,R4 | MOV R7. @RW4 + d8 |
| +5 | MOV R0,R5 | MOV R0. @RW5 + d8 | MOV R1,R5 | MOV R1. @RW5 + d8 | MOV R2,R5 | MOV R2. @RW5 + d8 | MOV R3,R5 | MOV R3. @RW5 + d8 | MOV R4,R5 | MOV R4. @RW5 + d8 | MOV R5,R5 | MOV R5. @RW5 + d8 | MOV R6,R5 | MOV R6. @RW5 + d8 | MOV R7,R5 | MOV R7. @RW5 + d8 |
| +6 | MOV R0,R6 | MOV R0. @RW6 + d8 | MOV R1,R6 | MOV R1. @RW6 + d8 | MOV R2,R6 | MOV R2. @RW6 + d8 | MOV R3,R6 | MOV R3. @RW6 + d8 | MOV R4,R6 | MOV R4. @RW6 + d8 | MOV R5,R6 | MOV R5. @RW6 + d8 | MOV R6,R6 | MOV R6. @RW6 + d8 | MOV R7,R6 | MOV R7. @RW6 + d8 |
| +7 | MOV R0,R7 | MOV R0. @RW7 + d8 | MOV R1,R7 | MOV R1. @RW7 + d8 | MOV R2,R7 | MOV R2. @RW7 + d8 | MOV R3,R7 | MOV R3. @RW7 + d8 | MOV R4,R7 | MOV R4. @RW7 + d8 | MOV R5,R7 | MOV R5. @RW7 + d8 | MOV R6,R7 | MOV R6. @RW7 + d8 | MOV R7,R7 | MOV R7. @RW7 + d8 |
| +8 | MOV R0,@RW0 | MOV R0. @RW0 + d16 | MOV R1,@RW0 | MOV R1. @RW0 + d16 | MOV R2,@RW0 | MOV R2. @RW0 + d16 | MOV R3,@RW0 | MOV R3. @RW0 + d16 | MOV R4,@RW0 | MOV R4. @RW0 + d16 | MOV R5,@RW0 | MOV R5. @RW0 + d16 | MOV R6,@RW0 | MOV R6. @RW0 + d16 | MOV R7,@RW0 | MOV R7. @RW0 + d16 |
| +9 | MOV R0,@RW1 | MOV R0. @RW1 + d16 | MOV R1,@RW1 | MOV R1. @RW1 + d16 | MOV R2,@RW1 | MOV R2. @RW1 + d16 | MOV R3,@RW1 | MOV R3. @RW1 + d16 | MOV R4,@RW1 | MOV R4. @RW1 + d16 | MOV R5,@RW1 | MOV R5. @RW1 + d16 | MOV R6,@RW1 | MOV R6. @RW1 + d16 | MOV R7,@RW1 | MOV R7. @RW1 + d16 |
| +A | MOV R0,@RW2 | MOV R0. @RW2 + d16 | MOV R1,@RW2 | MOV R1. @RW2 + d16 | MOV R2,@RW2 | MOV R2. @RW2 + d16 | MOV R3,@RW2 | MOV R3. @RW2 + d16 | MOV R4,@RW2 | MOV R4. @RW2 + d16 | MOV R5,@RW2 | MOV R5. @RW2 + d16 | MOV R6,@RW2 | MOV R6. @RW2 + d16 | MOV R7,@RW2 | MOV R7. @RW2 + d16 |
| +B | MOV R0,@RW3 | MOV R0. @RW3 + d16 | MOV R1,@RW3 | MOV R1. @RW3 + d16 | MOV R2,@RW3 | MOV R2. @RW3 + d16 | MOV R3,@RW3 | MOV R3. @RW3 + d16 | MOV R4,@RW3 | MOV R4. @RW3 + d16 | MOV R5,@RW3 | MOV R5. @RW3 + d16 | MOV R6,@RW3 | MOV R6. @RW3 + d16 | MOV R7,@RW3 | MOV R7. @RW3 + d16 |
| +C | MOV R0, @RW0+ | MOV R0. @RW0 + RW7 | MOV R1, @RW0+ | MOV R1. @RW0 + RW7 | MOV R2, @RW0+ | MOV R2. @RW0 + RW7 | MOV R3, @RW0+ | MOV R3. @RW0 + RW7 | MOV R4, @RW0+ | MOV R4. @RW0 + RW7 | MOV R5, @RW0+ | MOV R5. @RW0 + RW7 | MOV R6, @RW0+ | MOV R6. @RW0 + RW7 | MOV R7, @RW0+ | MOV R7. @RW0 + RW7 |
| +D | MOV R0, @RW1+ | MOV R0. @RW1 + RW7 | MOV R1, @RW1+ | MOV R1. @RW1 + RW7 | MOV R2, @RW1+ | MOV R2. @RW1 + RW7 | MOV R3, @RW1+ | MOV R3. @RW1 + RW7 | MOV R4, @RW1+ | MOV R4. @RW1 + RW7 | MOV R5, @RW1+ | MOV R5. @RW1 + RW7 | MOV R6, @RW1+ | MOV R6. @RW1 + RW7 | MOV R7, @RW1+ | MOV R7. @RW1 + RW7 |
| +E | MOV R0, @RW2+ | MOV R0. @PC + d16 | MOV R1, @RW2+ | MOV R1. @PC + d16 | MOV R2, @RW2+ | MOV R2. @PC + d16 | MOV R3, @RW2+ | MOV R3. @PC + d16 | MOV R4, @RW2+ | MOV R4. @PC + d16 | MOV R5, @RW2+ | MOV R5. @PC + d16 | MOV R6, @RW2+ | MOV R6. @PC + d16 | MOV R7, @RW2+ | MOV R7. @PC + d16 |
| +F | MOV R0, @RW3+ | MOV R0, addr16 | MOV R1, @RW3+ | MOV R1. addr16 | MOV R2, @RW3+ | MOV R2. addr16 | MOV R3, @RW3+ | MOV R3. addr16 | MOV R4, @RW3+ | MOV R4. addr16 | MOV R5, @RW3+ | MOV R5. addr16 | MOV R6, @RW3+ | MOV R6. addr16 | MOV R7, @RW3+ | MOV R7. addr16 |

Table 37-42. MOVW RWi, ea instruction (first byte = $7B_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOVW RW0, RW0 | MOVW RW0, @RW0 + d8 | MOVW RW1, RW0 | MOVW RW1 @RW0 + d8 | MOVW RW2, @RW0 | MOVW RW2, @RW0 + d8 | MOVW RW3, RW0 | MOVW RW3, @RW0 + d8 | MOVW RW4, RW0 | MOVW RW4, @RW0 + d8 | MOVW RW5, RW0 | MOVW RW5, @RW0 + d8 | MOVW RW6, RW0 | MOVW RW6, @RW0 + d8 | MOVW RW7, RW0 | MOVW RW7, @RW0 + d8 |
| +1 | MOVW RW0, RW1 | MOVW RW0, @RW1 + d8 | MOVW RW1, RW1 | MOVW RW1 @RW1 + d8 | MOVW RW2, @RW1 | MOVW RW2, @RW1 + d8 | MOVW RW3, RW1 | MOVW RW3, @RW1 + d8 | MOVW RW4, RW1 | MOVW RW4, @RW1 + d8 | MOVW RW5, RW1 | MOVW RW5, @RW1 + d8 | MOVW RW6, RW1 | MOVW RW6, @RW1 + d8 | MOVW RW7, RW1 | MOVW RW7, @RW1 + d8 |
| +2 | MOVW RW0, RW2 | MOVW RW0, @RW2 + d8 | MOVW RW1, RW2 | MOVW RW1 @RW2 + d8 | MOVW RW2, @RW2 | MOVW RW2, @RW2 + d8 | MOVW RW3, RW2 | MOVW RW3, @RW2 + d8 | MOVW RW4, RW2 | MOVW RW4, @RW2 + d8 | MOVW RW5, RW2 | MOVW RW5, @RW2 + d8 | MOVW RW6, RW2 | MOVW RW6, @RW2 + d8 | MOVW RW7, RW2 | MOVW RW7, @RW2 + d8 |
| +3 | MOVW RW0, RW3 | MOVW RW0, @RW3 + d8 | MOVW RW1, RW3 | MOVW RW1, @RW3 + d8 | MOVW RW2, @RW3 | MOVW RW2, @RW3 + d8 | MOVW RW3, RW3 | MOVW RW3, @RW3 + d8 | MOVW RW4, RW3 | MOVW RW4, @RW3 + d8 | MOVW RW5, RW3 | MOVW RW5, @RW3 + d8 | MOVW RW6, RW3 | MOVW RW6, @RW3 + d8 | MOVW RW7, RW3 | MOVW RW7, @RW3 + d8 |
| +4 | MOVW RW0, RW4 | MOVW RW0, @RW4 + d8 | MOVW RW1, RW4 | MOVW RW1, @RW4 + d8 | MOVW RW2, @RW4 | MOVW RW2, @RW4 + d8 | MOVW RW3, RW4 | MOVW RW3, @RW4 + d8 | MOVW RW4, RW4 | MOVW RW4, @RW4 + d8 | MOVW RW5, RW4 | MOVW RW5, @RW4 + d8 | MOVW RW6, RW4 | MOVW RW6, @RW4 + d8 | MOVW RW7, RW4 | MOVW RW7, @RW4 + d8 |
| +5 | MOVW RW0, RW5 | MOVW RW0, @RW5 + d8 | MOVW RW1, RW5 | MOVW RW1, @RW5 + d8 | MOVW RW2, @RW5 | MOVW RW2, @RW5 + d8 | MOVW RW3, RW5 | MOVW RW3, @RW5 + d8 | MOVW RW4, RW5 | MOVW RW4, @RW5 + d8 | MOVW RW5, RW5 | MOVW RW5, @RW5 + d8 | MOVW RW6, RW5 | MOVW RW6, @RW5 + d8 | MOVW RW7, RW5 | MOVW RW7, @RW5 + d8 |
| +6 | MOVW RW0, RW6 | MOVW RW0, @RW6 + d8 | MOVW RW1, RW6 | MOVW RW1, @RW6 + d8 | MOVW RW2, @RW6 | MOVW RW2, @RW6 + d8 | MOVW RW3, RW6 | MOVW RW3, @RW6 + d8 | MOVW RW4, RW6 | MOVW RW4, @RW6 + d8 | MOVW RW5, RW6 | MOVW RW5, @RW6 + d8 | MOVW RW6, RW6 | MOVW RW6, @RW6 + d8 | MOVW RW7, RW6 | MOVW RW7, @RW6 + d8 |
| +7 | MOVW RW0, RW7 | MOVW RW0, @RW7 + d8 | MOVW RW1, RW7 | MOVW RW1 @RW7 + d8 | MOVW RW2, @RW7 | MOVW RW2, @RW7 + d8 | MOVW RW3, RW7 | MOVW RW3, @RW7 + d8 | MOVW RW4, RW7 | MOVW RW4, @RW7 + d8 | MOVW RW5, RW7 | MOVW RW5, @RW7 + d8 | MOVW RW6, RW7 | MOVW RW6, @RW7 + d8 | MOVW RW7, RW7 | MOVW RW7, @RW7 + d8 |
| +8 | MOVW RW0,@RW0 | MOVW RW0, @RW0 + d16 | MOVW RW1, @RW0 | MOVW RW1, @RW0 + d16 | MOVW RW2, @RW0 | MOVW RW2, @RW0 + d16 | MOVW RW3,@RW0 | MOVW RW3, @RW0 + d16 | MOVW RW4,@RW0 | MOVW RW4, @R0 + d16 | MOVW RW5,@RW0 | MOVW RW5, @RW0 + d16 | MOVW RW6,@RW0 | MOVW RW6, @RW0 + d16 | MOVW RW7,@RW0 | MOVW RW7, @RW0 + d16 |
| +9 | MOVW RW0,@RW1 | MOVW RW0, @RW1 + d16 | MOVW RW1, @RW1 | MOVW RW1, @RW1 + d16 | MOVW RW2, @RW1 | MOVW RW2, @RW1 + d16 | MOVW RW3,@RW1 | MOVW RW3, @RW1 + d16 | MOVW RW4,@RW1 | MOVW RW4, @RW1 + d16 | MOVW RW5,@RW1 | MOVW RW5, @RW1 + d16 | MOVW RW6,@RW1 | MOVW RW6, @RW1 + d16 | MOVW RW7,@RW1 | MOVW RW7, @RW1 + d16 |
| +A | MOVW RW0,@RW2 | MOVW RW0, @RW2 + d16 | MOVW RW1, @RW2 | MOVW RW1, @RW2 + d16 | MOVW RW2, @RW2 | MOVW RW2, @RW2+ d16 | MOVW RW3,@RW2 | MOVW RW3, @RW2 + d16 | MOVW RW4,@RW2 | MOVW RW4, @RW2 + d16 | MOVW RW5,@RW2 | MOVW RW5, @RW2 + d16 | MOVW RW6,@RW2 | MOVW RW6, @RW2 + d16 | MOVW RW7,@RW2 | MOVW RW7, @RW2 + d16 |
| +B | MOVW RW0,@RW3 | MOVW RW0, @RW3 + d16 | MOVW RW1, @RW3 | MOVW RW1, @RW3 + d16 | MOVW RW2, @RW3 | MOVW RW2, @RW3 + d16 | MOVW RW3,@RW3 | MOVW RW3, @RW3 + d16 | MOVW RW4,@RW3 | MOVW RW4, @RW3 + d16 | MOVW RW5,@RW3 | MOVW RW5, @RW3 + d16 | MOVW RW6,@RW3 | MOVW RW6, @RW3 + d16 | MOVW RW7,@RW3 | MOVW RW7, @RW3 + d16 |
| +C | MOVW RW0,@RW0+ | MOVW RW0, @RW0 + RW7 | MOVW RW1, @RW0+ | MOVW RW1, @RW0 + RW7 | MOVW RW2, @RW0+ | MOVW RW2, @RW0 + RW7 | MOVW RW3,@RW0+ | MOVW RW3, @RW0 + RW7 | MOVW RW4,@RW0+ | MOVW RW4, @RW0 + RW7 | MOVW RW5,@RW0+ | MOVW RW5, @RW0 + RW7 | MOVW RW6,@RW0+ | MOVW RW6, @RW0 + RW7 | MOVW RW7,@RW0+ | MOVW RW7, @RW0 + RW7 |
| +D | MOVW RW0,@RW1+ | MOVW RW0, @RW1 + RW7 | MOVW RW1, @RW1+ | MOVW RW1, @RW1 + RW7 | MOVW RW2, @RW1+ | MOVW RW2, @RW1 + RW7 | MOVW RW3,@RW1+ | MOVW RW3, @RW1 + RW7 | MOVW RW4,@RW1+ | MOVW RW4, @RW1 + RW7 | MOVW RW5,@RW1+ | MOVW RW5, @RW1 + RW7 | MOVW RW6,@RW1+ | MOVW RW6, @RW1 + RW7 | MOVW RW7,@RW1+ | MOVW RW7, @RW1 + RW7 |
| +E | MOVW RW0,@RW2+ | MOVW RW0, @PC + d16 | MOVW RW1,@RW2+ | MOVW RW1, @PC + d16 | MOVW RW2, @RW2+ | MOVW RW2, @PC + d16 | MOVW RW3,@RW2+ | MOVW RW3, @PC + d16 | MOVW RW4,@RW2+ | MOVW RW4, @PC + d16 | MOVW RW5,@RW2+ | MOVW RW5, @PC + d16 | MOVW RW6,@RW2+ | MOVW RW6, @PC + d16 | MOVW RW7,@RW2+ | MOVW RW7, @PC + d16 |
| +F | MOVW RW0,@RW3+ | MOVW RW0, adtdr16 | MOVW RW1,@RW3+ | MOVW RW1, adtdr16 | MOVW RW2, @RW3+ | MOVW RW2, adtdr16 | MOVW RW3,@RW3+ | MOVW RW3, adtdr16 | MOVW RW4,@RW3+ | MOVW RW4, adtdr16 | MOVW RW5,@RW3+ | MOVW RW5, adtdr16 | MOVW RW6,@RW3+ | MOVW RW6, adtdr16 | MOVW RW7,@RW3+ | MOVW RW7 adtdr16 |

Table 37-43. MOV ea, Ri instruction (first byte = $7C_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOV R0,R0 | MOV @R W0 + d8,R0 | MOV R0,R1 | MOV @R W0 + d8,R1 | MOV R0,R2 | MOV @R W0 + d8,R2 | MOV R0,R3 | MOV @R W0 + d8,R3 | MOV R0,R4 | MOV @R W0 + d8,R4 | MOV R0,R5 | MOV @R W0 + d8,R5 | MOV R0,R6 | MOV @R W0 + d8,R6 | MOV R0,R7 | MOV @R W0 + d8,R7 |
| +1 | MOV R1,R0 | MOV @R W1 + d8,R0 | MOV R1,R1 | MOV @R W1 + d8,R1 | MOV R1,R2 | MOV @R W1 + d8,R2 | MOV R1,R3 | MOV @R W1 + d8,R3 | MOV R1,R4 | MOV @R W1 + d8,R4 | MOV R1,R5 | MOV @R W1 + d8,R5 | MOV R1,R6 | MOV @R W1 + d8,R6 | MOV R1,R7 | MOV @R W1 + d8,R7 |
| +2 | MOV R2,R0 | MOV @R W2 + d8,R0 | MOV R2,R1 | MOV @R W2 + d8,R1 | MOV R2,R2 | MOV @R W2 + d8,R2 | MOV R2,R3 | MOV @R W2 + d8,R3 | MOV R2,R4 | MOV @R W2 + d8,R4 | MOV R2,R5 | MOV @R W2 + d8,R5 | MOV R2,R6 | MOV @R W2 + d8,R6 | MOV R2,R7 | MOV @R W2 + d8,R7 |
| +3 | MOV R3,R0 | MOV @R W3 + d8,R0 | MOV R3,R1 | MOV @R W3 + d8,R1 | MOV R3,R2 | MOV @R W3 + d8,R2 | MOV R3,R3 | MOV @R W3 + d8,R3 | MOV R3,R4 | MOV @R W3 + d8,R4 | MOV R3,R5 | MOV @R W3 + d8,R5 | MOV R3,R6 | MOV @R W3 + d8,R6 | MOV R3,R7 | MOV @R W3 + d8,R7 |
| +4 | MOV R4,R0 | MOV @R W4 + d8,R0 | MOV R4,R1 | MOV @R W4 + d8,R1 | MOV R4,R2 | MOV @R W4 + d8,R2 | MOV R4,R3 | MOV @R W4 + d8,R3 | MOV R4,R4 | MOV @R W4 + d8,R4 | MOV R4,R5 | MOV @R W4 + d8,R5 | MOV R4,R6 | MOV @R W4 + d8,R6 | MOV R4,R7 | MOV @R W4 + d8,R7 |
| +5 | MOV R5,R0 | MOV @R W5 + d8,R0 | MOV R5,R1 | MOV @R W5 + d8,R1 | MOV R5,R2 | MOV @R W5 + d8,R2 | MOV R5,R3 | MOV @R W5 + d8,R3 | MOV R5,R4 | MOV @R W5 + d8,R4 | MOV R5,R5 | MOV @R W5 + d8,R5 | MOV R5,R6 | MOV @R W5 + d8,R6 | MOV R5,R7 | MOV @R W5 + d8,R7 |
| +6 | MOV R6,R0 | MOV @R W6 + d8,R0 | MOV R6,R1 | MOV @R W6 + d8,R1 | MOV R6,R2 | MOV @R W6 + d8,R2 | MOV R6,R3 | MOV @R W6 + d8,R3 | MOV R6,R4 | MOV @R W6 + d8,R4 | MOV R6,R5 | MOV @R W6 + d8,R5 | MOV R6,R6 | MOV @R W6 + d8,R6 | MOV R6,R7 | MOV @R W6 + d8,R7 |
| +7 | MOV R7,R0 | MOV @R W7 + d8,R0 | MOV R7,R1 | MOV @R W7 + d8,R1 | MOV R7,R2 | MOV @R W7 + d8,R2 | MOV R7,R3 | MOV @R W7 + d8,R3 | MOV R7,R4 | MOV @R W7 + d8,R4 | MOV R7,R5 | MOV @R W7 + d8,R5 | MOV R7,R6 | MOV @R W7 + d8,R6 | MOV R7,R7 | MOV @R W7 + d8,R7 |
| +8 | MOV @RW0,R0 | MOV @RW 0 + d16,R0 | MOV @RW0,R1 | MOV @RW 0 + d16,R1 | MOV @RW0,R2 | MOV @RW 0 + d16,R2 | MOV @RW0,R3 | MOV @RW 0 + d16,R3 | MOV @RW0,R4 | MOV @RW 0 + d16,R4 | MOV @RW0,R5 | MOV @RW 0 + d16,R5 | MOV @RW0,R6 | MOV @RW 0 + d16,R6 | MOV @RW0,R7 | MOV @RW 0 + d16,R7 |
| +9 | MOV @RW1,R0 | MOV @RW 1 + d16,R0 | MOV @RW1,R1 | MOV @RW 1 + d16,R1 | MOV @RW1,R2 | MOV @RW 1 + d16,R2 | MOV @RW1,R3 | MOV @RW 1 + d16,R3 | MOV @RW1,R4 | MOV @RW 1 + d16,R4 | MOV @RW1,R5 | MOV @RW 1 + d16,R5 | MOV @RW1,R6 | MOV @RW 1 + d16,R6 | MOV @RW1,R7 | MOV @RW 1 + d16,R7 |
| +A | MOV @RW2,R0 | MOV @RW 2 + d16,R0 | MOV @RW2,R1 | MOV @RW 2 + d16,R1 | MOV @RW2,R2 | MOV @RW 2 + d16,R2 | MOV @RW2,R3 | MOV @RW 2 + d16,R3 | MOV @RW2,R4 | MOV @RW 2 + d16,R4 | MOV @RW2,R5 | MOV @RW 2 + d16,R5 | MOV @RW2,R6 | MOV @RW 2 + d16,R6 | MOV @RW2,R7 | MOV @RW 2 + d16,R7 |
| +B | MOV @RW3,R0 | MOV @RW 3 + d16,R0 | MOV @RW3,R1 | MOV @RW 3 + d16,R1 | MOV @RW3,R2 | MOV @RW 3 + d16,R2 | MOV @RW3,R3 | MOV @RW 3 + d16,R3 | MOV @RW3,R4 | MOV @RW 3 + d16,R4 | MOV @RW3,R5 | MOV @RW 3 + d16,R5 | MOV @RW3,R6 | MOV @RW 3 + d16,R6 | MOV @RW3,R7 | MOV @RW 3 + d16,R7 |
| +C | MOV @RW0+,R0 | MOV @RW 0 + RW7,R0 | MOV @RW0+,R1 | MOV @RW 0 + RW7,R1 | MOV @RW0+,R2 | MOV @RW 0 + RW7,R2 | MOV @RW0+,R3 | MOV @RW 0 + RW7,R3 | MOV @RW0+,R4 | MOV @RW 0 + RW7,R4 | MOV @RW0+,R5 | MOV @RW 0 + RW7,R5 | MOV @RW0+,R6 | MOV @RW 0 + RW7,R6 | MOV @RW0+,R7 | MOV @RW 0 + RW7,R7 |
| +D | MOV @RW1+,R0 | MOV @RW 1 + RW7,R0 | MOV @RW1+,R1 | MOV @RW 1 + RW7,R1 | MOV @RW1+,R2 | MOV @RW 1 + RW7,R2 | MOV @RW1+,R3 | MOV @RW 1 + RW7,R3 | MOV @RW1+,R4 | MOV @RW 1 + RW7,R4 | MOV @RW1+,R5 | MOV @RW 1 + RW7,R5 | MOV @RW1+,R6 | MOV @RW 1 + RW7,R6 | MOV @RW1+,R7 | MOV @RW 1 + RW7,R7 |
| +E | MOV @RW2+,R0 | MOV @RW 2 + d16,R0 | MOV @RW2+,R1 | MOV @RW PC + d16,R1 | MOV @RW2+,R2 | MOV @RW PC + d16,R2 | MOV @RW2+,R3 | MOV @RW PC + d16,R3 | MOV @RW2+,R4 | MOV @RW PC + d16,R4 | MOV @RW2+,R5 | MOV @RW PC + d16,R5 | MOV @RW2+,R6 | MOV @RW PC + d16,R6 | MOV @RW2+,R7 | MOV @RW PC + d16,R7 |
| +F | MOV @RW3+,R0 | MOV @RW addr16,R0 | MOV @RW3+,R1 | MOV addr16,R1 | MOV @RW3+,R2 | MOV addr16,R2 | MOV @RW3+,R3 | MOV addr16,R3 | MOV @RW3+,R4 | MOV addr16,R4 | MOV @RW3+,R5 | MOV addr16,R5 | MOV @RW3+,R6 | MOV addr16,R6 | MOV @RW3+,R7 | MOV d addr16,R7 |

Table 37-44. MOVW ea, Rwi instruction (first byte = $7D_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | MOVW RW0,RW0 | MOVW @RW 0 + d8,RW0 | MOVW RW0,RW1 | MOVW @RW 0 + d8,RW1 | MOVW RW0,RW2 | MOVW @RW 0 + d8,RW2 | MOVW RW0,RW3 | MOVW @RW 0 + d8,RW3 | MOVW RW0,RW4 | MOVW @RW 0 + d8,RW4 | MOVW RW0,RW5 | MOVW @RW 0 + d8,RW5 | MOVW RW0,RW6 | MOVW @RW 0 + d8,RW6 | MOVW RW0,RW7 | MOVW @RW 0 + d8,RW7 |
| +1 | MOVW RW1,RW0 | MOVW @RW 1 + d8,RW0 | MOVW RW1,RW1 | MOVW @RW 1 + d8,RW1 | MOVW RW1,RW2 | MOVW @RW 1 + d8,RW2 | MOVW RW1,RW3 | MOVW @RW 1 + d8,RW3 | MOVW RW1,RW4 | MOVW @RW 1 + d8,RW4 | MOVW RW1,RW5 | MOVW @RW 1 + d8,RW5 | MOVW RW1,RW6 | MOVW @RW 1 + d8,RW6 | MOVW RW1,RW7 | MOVW @RW 1 + d8,RW7 |
| +2 | MOVW RW2,RW0 | MOVW @RW 2 + d8,RW0 | MOVW RW2,RW1 | MOVW @RW 2 + d8,RW1 | MOVW RW2,RW2 | MOVW @RW 2 + d8,RW2 | MOVW RW2,RW3 | MOVW @RW 2 + d8,RW3 | MOVW RW2,RW4 | MOVW @RW 2 + d8,RW4 | MOVW RW2,RW5 | MOVW @RW 2 + d8,RW5 | MOVW RW2,RW6 | MOVW @RW 2 + d8,RW6 | MOVW RW2,RW7 | MOVW @RW 2 + d8,RW7 |
| +3 | MOVW RW3,RW0 | MOVW @RW 3 + d8,RW0 | MOVW RW3,RW1 | MOVW @RW 3 + d8,RW1 | MOVW RW3,RW2 | MOVW @RW 3 + d8,RW2 | MOVW RW3,RW3 | MOVW @RW 3 + d8,RW3 | MOVW RW3,RW4 | MOVW @RW 3 + d8,RW4 | MOVW RW3,RW5 | MOVW @RW 3 + d8,RW5 | MOVW RW3,RW6 | MOVW @RW 3 + d8,RW6 | MOVW RW3,RW7 | MOVW @RW 3 + d8,RW7 |
| +4 | MOVW RW4,RW0 | MOVW @RW 4 + d8,RW0 | MOVW RW4,RW1 | MOVW @RW 4 + d8,RW1 | MOVW RW4,RW2 | MOVW @RW 4 + d8,RW2 | MOVW RW4,RW3 | MOVW @RW 4 + d8,RW3 | MOVW RW4,RW4 | MOVW @RW 4 + d8,RW4 | MOVW RW4,RW5 | MOVW @RW 4 + d8,RW5 | MOVW RW4,RW6 | MOVW @RW 4 + d8,RW6 | MOVW RW4,RW7 | MOVW @RW 4 + d8,RW7 |
| +5 | MOVW RW5,RW0 | MOVW @RW 5 + d8,RW0 | MOVW RW5,RW1 | MOVW @RW 5 + d8,RW1 | MOVW RW5,RW2 | MOVW @RW 5 + d8,RW2 | MOVW RW5,RW3 | MOVW @RW 5 + d8,RW3 | MOVW RW5,RW4 | MOVW @RW 5 + d8,RW4 | MOVW RW5,RW5 | MOVW @RW 5 + d8,RW5 | MOVW RW5,RW6 | MOVW @RW 5 + d8,RW6 | MOVW RW5,RW7 | MOVW @RW 5 + d8,RW7 |
| +6 | MOVW RW6,RW0 | MOVW @RW 6 + d8,RW0 | MOVW RW6,RW1 | MOVW @RW 6 + d8,RW1 | MOVW RW6,RW2 | MOVW @RW 6 + d8,RW2 | MOVW RW6,RW3 | MOVW @RW 6 + d8,RW3 | MOVW RW6,RW4 | MOVW @RW 6 + d8,RW4 | MOVW RW6,RW5 | MOVW @RW 6 + d8,RW5 | MOVW RW6,RW6 | MOVW @RW 6 + d8,RW6 | MOVW RW6,RW7 | MOVW @RW 6 + d8,RW7 |
| +7 | MOVW RW7,RW0 | MOVW @RW 7 + d8,RW0 | MOVW RW7,RW1 | MOVW @RW 7 + d8,RW1 | MOVW RW7,RW2 | MOVW @RW 7 + d8,RW2 | MOVW RW7,RW3 | MOVW @RW 7 + d8,RW3 | WOVW RW7,RW4 | MOVW @RW 7 + d8,RW4 | WOVW RW7,RW5 | MOVW @RW 7 + d8,RW5 | WOVW RW7,RW6 | MOVW @RW 7 + d8,RW6 | WOVW RW7,RW7 | MOVW @RW 7 + d8,RW7 |
| +8 | MOVW @RW0,RW0 | MOVW @RW0 + d16,RW0 | MOVW @RW0,RW1 | MOVW @RW0 + d16,RW1 | MOVW @RW0,RW2 | MOVW @RW0 + d16,RW2 | MOVW @RW0,RW3 | MOVW @RW0 + d16,RW3 | MOVW @RW0,RW4 | MOVW @RW0 + d16,RW4 | MOVW @RW0,RW5 | MOVW @RW0 + d16,RW5 | MOVW @RW0,RW6 | MOVW @RW0 + d16,RW6 | MOVW @RW0,RW7 | MOVW @RW0 + d16,RW7 |
| +9 | MOVW @RW1,RW0 | MOVW @RW1 + d16,RW0 | MOVW @RW1,RW1 | MOVW @RW1 + d16,RW1 | MOVW @RW1,RW2 | MOVW @RW1 + d16,RW2 | MOVW @RW1,RW3 | MOVW @RW1 + d16,RW3 | MOVW @RW1,RW4 | MOVW @RW1 + d16,RW4 | MOVW @RW1,RW5 | MOVW @RW1 + d16,RW5 | MOVW @RW1,RW6 | MOVW @RW1 + d16,RW6 | MOVW @RW1,RW7 | MOVW @RW1 + d16,RW7 |
| +A | MOVW @RW2,RW0 | MOVW @RW2 + d16,RW0 | MOVW @RW2,RW1 | MOVW @RW2 + d16,RW1 | MOVW @RW2,RW2 | MOVW @RW2 + d16,RW2 | MOVW @RW2,RW3 | MOVW @RW2 + d16,RW3 | MOVW @RW2,RW4 | MOVW @RW2 + d16,RW4 | MOVW @RW2,RW5 | MOVW @RW2 + d16,RW5 | MOVW @RW2,RW6 | MOVW @RW2 + d16,RW6 | MOVW @RW2,RW7 | MOVW @RW2 + d16,RW7 |
| +B | MOVW @RW3,RW0 | MOVW @RW3 + d16,RW0 | MOVW @RW3,RW1 | MOVW @RW3 + d16,RW1 | MOVW @RW3,RW2 | MOVW @RW3 + d16,RW2 | MOVW @RW3,RW3 | MOVW @RW3 + d16,RW3 | MOVW @RW3,RW4 | MOVW @RW3 + d16,RW4 | MOVW @RW3,RW5 | MOVW @RW3 + d16,RW5 | MOVW @RW3,RW6 | MOVW @RW3 + d16,RW6 | MOVW @RW3,RW7 | MOVW @RW3 + d16,RW7 |
| +C | MOVW @RW0+,RW0 | MOVW @RW0 + RW7,RW0 | MOVW @RW0+,RW1 | MOVW @RW0 + RW7,RW1 | MOVW @RW0+,RW2 | MOVW @RW0 + RW7,RW2 | MOVW @RW0+,RW3 | MOVW @RW0 + RW7,RW3 | MOVW @RW0+,RW4 | MOVW @RW0 + RW7,RW4 | MOVW @RW0+,RW5 | MOVW @RW0 + RW7,RW5 | MOVW @RW0+,RW6 | MOVW @RW0 + RW7,RW6 | MOVW @RW0+,RW7 | MOVW @RW0 + RW7,RW7 |
| +D | MOVW @RW1+,RW0 | MOVW @RW1 + RW7,RW0 | MOVW @RW1+,RW1 | MOVW @RW1 + RW7,RW1 | MOVW @RW1+,RW2 | MOVW @RW1 + RW7,RW2 | MOVW @RW1+,RW3 | MOVW @RW1 + RW7,RW3 | MOVW @RW1+,RW4 | MOVW @RW1 + RW7,RW4 | MOVW @RW1+,RW5 | MOVW @RW1 + RW7,RW5 | MOVW @RW1+,RW6 | MOVW @RW1 + RW7,RW6 | MOVW @RW1+,RW7 | MOVW @RW1 + RW7,RW7 |
| +E | MOVW @RW2+,RW0 | MOVW @PC+ d16,RW0 | MOVW @RW2+,RW1 | MOVW @PC+ d16,RW1 | MOVW @RW2+,RW2 | MOVW @PC+ d16,RW2 | MOVW @RW2+,RW3 | MOVW @PC+ d16,RW3 | MOVW @RW2+,RW4 | MOVW @PC+ d16,RW4 | MOVW @RW2+,RW5 | MOVW @PC+ d16,RW5 | MOVW @RW2+,RW6 | MOVW @PC+ d16,RW6 | MOVW @RW2+,RW7 | MOVW @PC+ d16,RW7 |
| +F | MOVW @RW3+,RW0 | MOVW addr 16,@RW0 | MOVW @RW3+,RW1 | MOVW addr 16,@RW1 | MOVW @RW3+,RW2 | MOVW addr 16,@RW2 | MOVW @RW3+,RW3 | MOVW addr 16,@RW3 | MOVW @RW3+,RW4 | MOVW addr 16,@RW4 | MOVW @RW3+,RW5 | MOVW addr 16,RW5 | MOVW @RW3+,RW6 | MOVW addr 16,RW6 | MOVW @RW3+,RW7 | MOVW addr 16,RW7 |

Table 37-45. XCH Ri, ea instruction (first byte = $7E_H$)

| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | XCH R0,R0 | XCH R0, @RW0 + d8 | XCH R1,R0 | XCH R1, @RW0 + d8 | XCH R2,R0 | XCH R2, @RW0 + d8 | XCH R3,R0 | XCH R3, @RW0 + d8 | XCH R4,R0 | XCH R4, @RW0 + d8 | XCH R5,R0 | XCH R5, @RW0 + d8 | XCH R6,R0 | XCH R6, @RW0 + d8 | XCH R7,R0 | XCH R7, @RW0 + d8 |
| +1 | XCH R0,R1 | XCH R0, @RW1 + d8 | XCH R1,R1 | XCH R1, @RW1 + d8 | XCH R2,R1 | XCH R2, @RW1 + d8 | XCH R3,R1 | XCH R3, @RW1 + d8 | XCH R4,R1 | XCH R4, @RW1 + d8 | XCH R5,R1 | XCH R5, @RW1 + d8 | XCH R6,R1 | XCH R6, @RW1 + d8 | XCH R7,R1 | NOTW R7, @RW1 + d8 |
| +2 | XCH R0,R2 | XCH R0, @RW2 + d8 | XCH R1,R2 | XCH R1, @RW2 + d8 | XCH R2,R2 | XCH R2, @RW2 + d8 | XCH R3,R2 | XCH R3, @RW2 + d8 | XCH R4,R2 | XCH R4, @RW2 + d8 | XCH R5,R2 | XCH R5, @RW2 + d8 | XCH R6,R2 | XCH R6, @RW2 + d8 | XCH R7,R2 | XCH R7, @RW2 + d8 |
| +3 | XCH R0,R3 | XCH R0, @RW3 + d8 | XCH R1,R3 | XCH R1, @RW3 + d8 | XCH R2,R3 | XCH R2, @RW3 + d8 | XCH R3,R3 | XCH R3, @RW3 + d8 | XCH R4,R3 | XCH R4, @RW3 + d8 | XCH R5,R3 | XCH R5, @RW3 + d8 | XCH R6,R3 | XCH R6, @RW3 + d8 | XCH R7,R3 | NOTW R7, @RW3 + d8 |
| +4 | XCH R0,R4 | XCH R0, @RW4 + d8 | XCH R1,R4 | XCH R1, @RW4 + d8 | XCH R2,R4 | XCH R2, @RW4 + d8 | XCH R3,R4 | XCH R3, @RW4 + d8 | XCH R4,R4 | XCH R4, @RW4 + d8 | XCH R5,R4 | XCH R5, @RW4 + d8 | XCH R6,R4 | XCH R6, @RW4 + d8 | XCH R7,R4 | NOTW R7, @RW4 + d8 |
| +5 | XCH R0,R5 | XCH R0, @RW5 + d8 | XCH R1,R5 | XCH R1, @RW5 + d8 | XCH R2,R5 | XCH R2, @RW5 + d8 | XCH R3,R5 | XCH R3, @RW5 + d8 | XCH R4,R5 | XCH R4, @RW5 + d8 | XCH R5,R5 | XCH R5, @RW5 + d8 | XCH R6,R5 | XCH R6, @RW5 + d8 | XCH R7,R5 | XCH R7, @RW5 + d8 |
| +6 | XCH R0,R6 | XCH R0, @RW6 + d8 | XCH R1,R6 | XCH R1, @RW6 + d8 | XCH R2,R6 | XCH R2, @RW6 + d8 | XCH R3,R6 | XCH R3, @RW6 + d8 | XCH R4,R6 | XCH R4, @RW6 + d8 | XCH R5,R6 | XCH R5, @RW6 + d8 | XCH R6,R6 | XCH R6, @RW6 + d8 | XCH R7,R6 | XCH R7, @RW6 + d8 |
| +7 | XCH R0,R7 | XCH R0, @RW7 + d8 | XCH R1,R7 | XCH R1, @RW7 + d8 | XCH R2,R7 | XCH R2, @RW7 + d8 | XCH R3,R7 | XCH R3, @RW7 + d8 | XCH R4,R7 | XCH R4, @RW7 + d8 | XCH R5,R7 | XCH R5, @RW7 + d8 | XCH R6,R7 | XCH R6, @RW7 + d8 | XCH R7,R7 | XCH R7, @RW7 + d8 |
| +8 | XCH R0,@RW0 | XCH R0, @RW0 + d16 | XCH R1,@RW0 | XCH R1, @RW0 + d16 | XCH R2,@RW0 | XCH R2, @RW0 + d16 | XCH R3,@RW0 | XCH R3, @RW0 + d16 | XCH R4,@RW0 | XCH R4, @RW0 + d16 | XCH R5,@RW0 | XCH R5, @RW0 + d16 | XCH R6,@RW0 | XCH R6, @RW0 + d16 | XCH R7,@RW0 | XCH R7, @RW0 + d16 |
| +9 | XCH R0,@RW1 | XCH R0, @RW1 + d16 | XCH R1,@RW1 | XCH R1, @RW1 + d16 | XCH R2,@RW1 | XCH R2, @RW1 + d16 | XCH R3,@RW1 | XCH R3, @RW1 + d16 | XCH R4,@RW1 | XCH R4, @RW1 + d16 | XCH R5,@RW1 | XCH R5, @RW1 + d16 | XCH R6,@RW1 | XCH R6, @RW1 + d16 | XCH R7,@RW1 | XCH R7, @RW1 + d16 |
| +A | XCH R0,@RW2 | XCH R0, W2 + d16,A | XCH R1,@RW2 | XCH R1, W2 + d16,A | XCH R2,@RW2 | XCH R2, W2 + d16,A | XCH R3,@RW2 | XCH R3, W2 + d16,A | XCH R4,@RW2 | XCH R4, W2 + d16,A | XCH R5,@RW2 | XCH R5, W2 + d16,A | XCH R6,@RW2 | XCH R6, W2 + d16,A | XCH R7,@RW2 | XCH R7, W2 + d16,A |
| +B | XCH R0,@RW3 | XCH R0, @RW3 + d16 | XCH R1,@RW3 | XCH R1, @RW3 + d16 | XCH R2,@RW3 | XCH R2, @RW3 + d16 | XCH R3,@RW3 | XCH R3, @RW3 + d16 | XCH R4,@RW3 | XCH R4, @RW3 + d16 | XCH R5,@RW3 | XCH R5, @RW3 + d16 | XCH R6,@RW3 | XCH R6, @RW3 + d16 | XCH R7,@RW3 | XCH R7, @RW3 + d16 |
| +C | XCH R0,@RW0+ | XCH R0, @RW0 + RW7 | XCH R1,@RW0+ | XCH R1, @RW0 + RW7 | XCH R2,@RW0+ | XCH R2, @RW0 + RW7 | XCH R3,@RW0+ | XCH R3, @RW0 + RW7 | XCH R4,@RW0+ | XCH R4, @RW0 + RW7 | XCH R5,@RW0+ | XCH R5, @RW0 + RW7 | XCH R6,@RW0+ | XCH R6, @RW0 + RW7 | XCH R7,@RW0+ | XCH R7, @RW0 + RW7 |
| +D | XCH R0,@RW1+ | XCH R0, @RW1 + RW7 | XCH R1,@RW1+ | XCH R1, @RW1 + RW7 | XCH R2,@RW1+ | XCH R2, @RW1 + RW7 | XCH R3,@RW1+ | XCH R3, @RW1 + RW7 | XCH R4,@RW1+ | XCH R4, @RW1 + RW7 | XCH R5,@RW1+ | XCH R5, @RW1 + RW7 | XCH R6,@RW1+ | XCH R6, @RW1 + RW7 | XCH R7,@RW1+ | XCH R7, @RW1 + RW7 |
| +E | XCH R0,@RW2+ | XCH R0, @PC + d16 | XCH R1,@RW2+ | XCH R1, @PC + d16 | XCH R2,@RW2+ | XCH R2, @PC + d16 | XCH R3,@RW2+ | XCH R3, @PC + d16 | XCH R4,@RW2+ | XCH R4, @PC + d16 | XCH R5,@RW2+ | XCH R5, @PC + d16 | XCH R6,@RW2+ | XCH R6, @PC + d16 | XCH R7,@RW2+ | XCH R7, @PC + d16 |
| +F | XCH R0,@RW3+ | XCH R0, addr16 | XCH R1,@RW3+ | XCH R1, addr16 | XCH R2,@RW3+ | XCH R2, addr16 | XCH R3,@RW3+ | XCH R3, addr16 | XCH R4,@RW3+ | XCH R4, addr16 | XCH R5,@RW3+ | XCH R5, addr16 | XCH R6,@RW3+ | XCH R6, addr16 | XCH R7,@RW3+ | XCH R7, addr16 |

Table 37-46. XCHW RWi, ea instruction (first byte = $7F_H$)

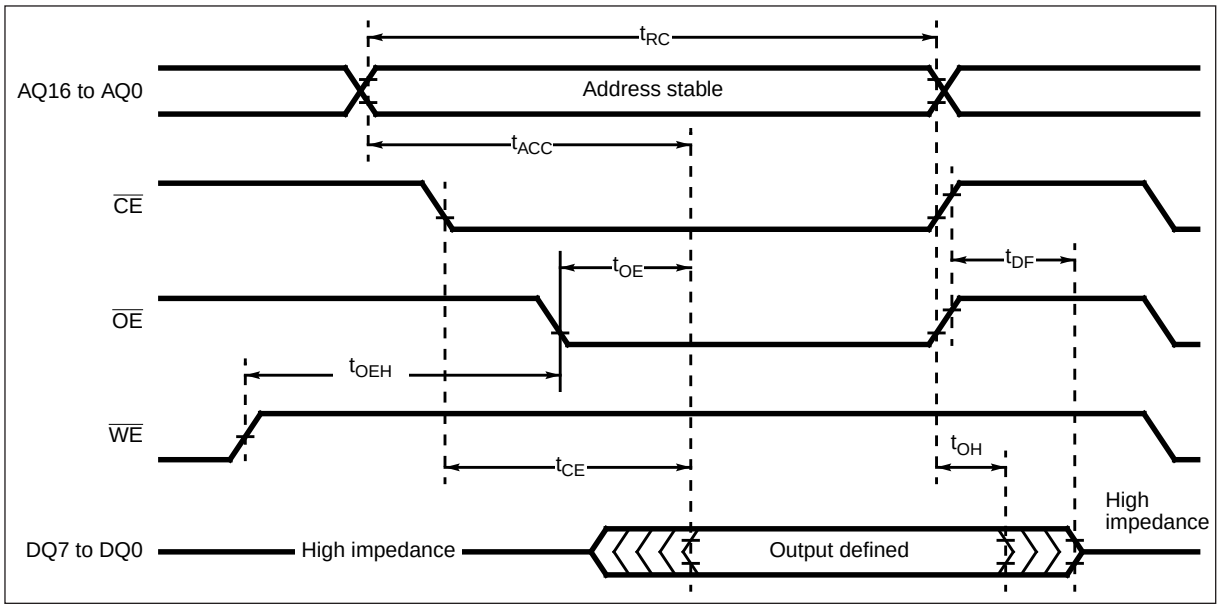
| | 00 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | A0 | B0 | C0 | D0 | E0 | F0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| +0 | XCHW RW0, RW0 | XCHW RW0, @RW0 + d8 | XCHW RW1,RW0 | XCHW RW1, @RW0 + d8 | XCHW RW2,RW0 | XCHW RW2, @RW0 + d8 | XCHW RW3,RW0 | XCHW RW3, @RW0 + d8 | XCHW RW4,RW0 | XCHW RW4, @RW0 + d8 | XCHW RW5,RW0 | XCHW RW5, @RW0 + d8 | XCHW RW6,RW0 | XCHW RW6, @RW0 + d8 | XCHW RW7,RW0 | XCHW RW7, @RW0 + d8 |
| +1 | XCHW RW0,RW1 | XCHW RW0, @RW1 + d8 | XCHW RW1,RW1 | XCHW RW1, @RW1 + d8 | XCHW RW2,RW1 | XCHW RW2, @RW1 + d8 | XCHW RW3,RW1 | XCHW RW3, @RW1 + d8 | XCHW RW4,RW1 | XCHW RW4, @RW1 + d8 | XCHW RW5,RW1 | XCHW RW5, @RW1 + d8 | XCHW RW6,RW1 | XCHW RW6, @RW1 + d8 | XCHW RW7,RW1 | XCHW RW7, @RW1 + d8 |
| +2 | XCHW RW0,RW2 | XCHW RW0, @RW2 + d8 | XCHW RW1,RW2 | XCHW RW1, @RW2 + d8 | XCHW RW2,RW2 | XCHW RW2, @RW2 + d8 | XCHW RW3,RW2 | XCHW RW3, @RW2 + d8 | XCHW RW4,RW2 | XCHW RW4, @RW2 + d8 | XCHW RW5,RW2 | XCHW RW5, @RW2 + d8 | XCHW RW6,RW2 | XCHW RW6, @RW2 + d8 | XCHW RW7,RW2 | XCHW RW7, @RW2 + d8 |
| +3 | XCHW RW0,RW3 | XCHW RW0, @RW3 + d8 | XCHW RW1,RW3 | XCHW RW1, @RW3 + d8 | XCHW RW2,RW3 | XCHW RW2, @RW3 + d8 | XCHW RW3,RW3 | XCHW RW3, @RW3 + d8 | XCHW RW4,RW3 | XCHW RW4, @RW3 + d8 | XCHW RW5,RW3 | XCHW RW5, @RW3 + d8 | XCHW RW6,RW3 | XCHW RW6, @RW3 + d8 | XCHW RW7,RW3 | XCHW RW7, @RW3 + d8 |
| +4 | XCHW RW0,RW4 | XCHW RW0, @RW4 + d8 | XCHW RW1,RW4 | XCHW RW1, @RW4 + d8 | XCHW RW2,RW4 | XCHW RW2, @RW4 + d8 | XCHW RW3,RW4 | XCHW RW3, @RW4 + d8 | XCHW RW4,RW4 | XCHW RW4, @RW4 + d8 | XCHW RW5,RW4 | XCHW RW5, @RW4 + d8 | XCHW RW6,RW4 | XCHW RW6, @RW4 + d8 | XCHW RW7,RW4 | XCHW RW7, @RW4 + d8 |
| +5 | XCHW RW0,RW5 | XCHW RW0, @RW5 + d8 | XCHW RW1,RW5 | XCHW RW1, @RW5 + d8 | XCHW RW2,RW5 | XCHW RW2, @RW5 + d8 | XCHW RW3,RW5 | XCHW RW3, @RW5 + d8 | XCHW RW4,RW5 | XCHW RW4, @RW5 + d8 | XCHW RW5,RW5 | XCHW RW5, @RW5 + d8 | XCHW RW6,RW5 | XCHW RW6, @RW5 + d8 | XCHW RW7,RW5 | XCHW RW7, @RW5 + d8 |
| +6 | XCHW RW0,RW6 | XCHW RW0, @RW6 + d8 | XCHW RW1,RW6 | XCHW RW1, @RW6 + d8 | XCHW RW2,RW6 | XCHW RW2, @RW6 + d8 | XCHW RW3,RW6 | XCHW RW3, @RW6 + d8 | XCHW RW4,RW6 | XCHW RW4, @RW6 + d8 | XCHW RW5,RW6 | XCHW RW5, @RW6 + d8 | XCHW RW6,RW6 | XCHW RW6, @RW6 + d8 | XCHW RW7,RW6 | XCHW RW7, @RW6 + d8 |
| +7 | XCHW RW0,RW7 | XCHW RW0, @RW7 + d8 | XCHW RW1,RW7 | XCHW RW1, @RW7 + d8 | XCHW RW2,RW7 | XCHW RW2, @RW7 + d8 | XCHW RW3,RW7 | XCHW RW3, @RW7 + d8 | XCHW RW4,RW7 | XCHW RW4, @RW7 + d8 | XCHW RW5,RW7 | XCHW RW5, @RW7 + d8 | XCHW RW6,RW7 | XCHW RW6, @RW7 + d8 | XCHW RW7,RW7 | XCHW RW7, @RW7 + d8 |
| +8 | XCHW RW0,@RW0 | XCHW RW0, @RW0 + d16 | XCHW RW1,@RW0 | XCHW RW1, @RW0 + d16 | XCHW RW2,@RW0 | XCHW RW2, @RW0 + d16 | XCHW RW3,@RW0 | XCHW RW3, @RW0 + d16 | XCHW RW4,@RW0 | XCHW RW4, @RW0 + d16 | XCHW RW5,@RW0 | XCHW RW5, @RW0 + d16 | XCHW RW6,@RW0 | XCHW RW6, @RW0 + d16 | XCHW RW7,@RW0 | XCHW RW7, @RW0 + d16 |
| +9 | XCHW RW0,@RW1 | XCHW RW0, @RW1 + d16 | XCHW RW1,@RW1 | XCHW RW1, @RW1 + d16 | XCHW RW2,@RW1 | XCHW RW2, @RW1 + d16 | XCHW RW3,@RW1 | XCHW RW3, @RW1 + d16 | XCHW RW4,@RW1 | XCHW RW4, @RW1 + d16 | XCHW RW5,@RW1 | XCHW RW5, @RW1 + d16 | XCHW RW6,@RW1 | XCHW RW6, @RW1 + d16 | XCHW RW7,@RW1 | XCHW RW7, @RW1 + d16 |
| +A | XCHW RW0,@RW2 | XCHW RW0, @RW2 + d16 | XCHW RW1,@RW2 | XCHW RW1, @RW2 + d16 | XCHW RW2,@RW2 | XCHW RW2, @RW2 + d16 | XCHW RW3,@RW2 | XCHW RW3, @RW2 + d16 | XCHW RW4,@RW2 | XCHW RW4, @RW2 + d16 | XCHW RW5,@RW2 | XCHW RW5, @RW2 + d16 | XCHW RW6,@RW2 | XCHW RW6, @RW2 + d16 | XCHW RW7,@RW2 | XCHW RW7, @RW2 + d16 |
| +B | XCHW RW0,@RW3 | XCHW RW0, @RW3 + d16 | XCHW RW1,@RW3 | XCHW RW1, @RW3 + d16 | XCHW RW2,@RW3 | XCHW RW2, @RW3 + d16 | XCHW RW3,@RW3 | XCHW RW3, @RW3 + d16 | XCHW RW4,@RW3 | XCHW RW4, @RW3 + d16 | XCHW RW5,@RW3 | XCHW RW5, @RW3 + d16 | XCHW RW6,@RW3 | XCHW RW6, @RW3 + d16 | XCHW RW7,@RW3 | XCHW RW7, @RW3 + d16 |
| +C | XCHW RW0, @RW0+ | XCHW RW0, @RW0 + RW7 | XCHW RW1, @RW0+ | XCHW RW1, @RW0 + RW7 | XCHW RW2, @RW0+ | XCHW RW2, @RW0 + RW7 | XCHW RW3, @RW0+ | XCHW RW3, @RW0 + RW7 | XCHW RW4, @RW0+ | XCHW RW4, @RW0 + RW7 | XCHW RW5, W5,@RW0+ | XCHW RW5, @RW0 + RW7 | XCHW RW6, @RW0+ | XCHW RW6, @RW0 + RW7 | XCHW RW7, @RW0+ | XCHW RW7, @RW0 + RW7 |
| +D | XCHW RW0, @RW1+ | XCHW RW0, @RW1 + RW7 | XCHW RW1, @RW1+ | XCHW RW1, @RW1 + RW7 | XCHW RW2, @RW1+ | XCHW RW2, @RW1 + RW7 | XCHW RW3, @RW1+ | XCHW RW3, @RW1 + RW7 | XCHW RW4, @RW1+ | XCHW RW4, @RW1 + RW7 | XCHW RW5 W5,@RW1+ | XCHW RW5, @RW1 + RW7 | XCHW RW6, @RW1+ | XCHW RW6, @RW1 + RW7 | XCHW RW7, @RW1+ | XCHW RW7, @RW1 + RW7 |
| +E | XCHW RW0, @RW2+ | XCHW RW0, @PC + d16 | XCHW RW1, @RW2+ | XCHW RW1, @PC + d16 | XCHW RW2, @RW2+ | XCHW RW2, @PC + d16 | XCHW RW3, @RW2+ | XCHW RW3, @PC + d16 | XCHW RW4, @RW2+ | XCHW RW4, @PC + d16 | XCHW RW5, @RW2+ | XCHW RW5, @PC + d16 | XCHW RW6, @RW2+ | XCHW RW6, @PC + d16 | XCHW RW7, @RW2+ | XCHW RW7, @PC + d16 |
| +F | XCHW RW0, @RW3+ | XCHW RW0, addr16 | XCHW RW1, @RW3+ | XCHW RW1, addr16 | XCHW RW2, @RW3+ | XCHW RW2, addr16 | XCHW RW3, @RW3+ | XCHW RW3, addr16 | XCHW RW4, @RW3+ | XCHW RW4, addr16 | XCHW RW5, @RW3+ | XCHW RW5, addr16 | XCHW RW6, @RW3+ | XCHW RW6, addr16 | XCHW RW7, @RW3+ | XCHW RW7, addr16 |

## 37.3 Timing Diagrams in Flash Memory Mode

Each timing diagram for the external pins of the Flash devices in MB96300 Super series during Flash Memory mode is shown below.

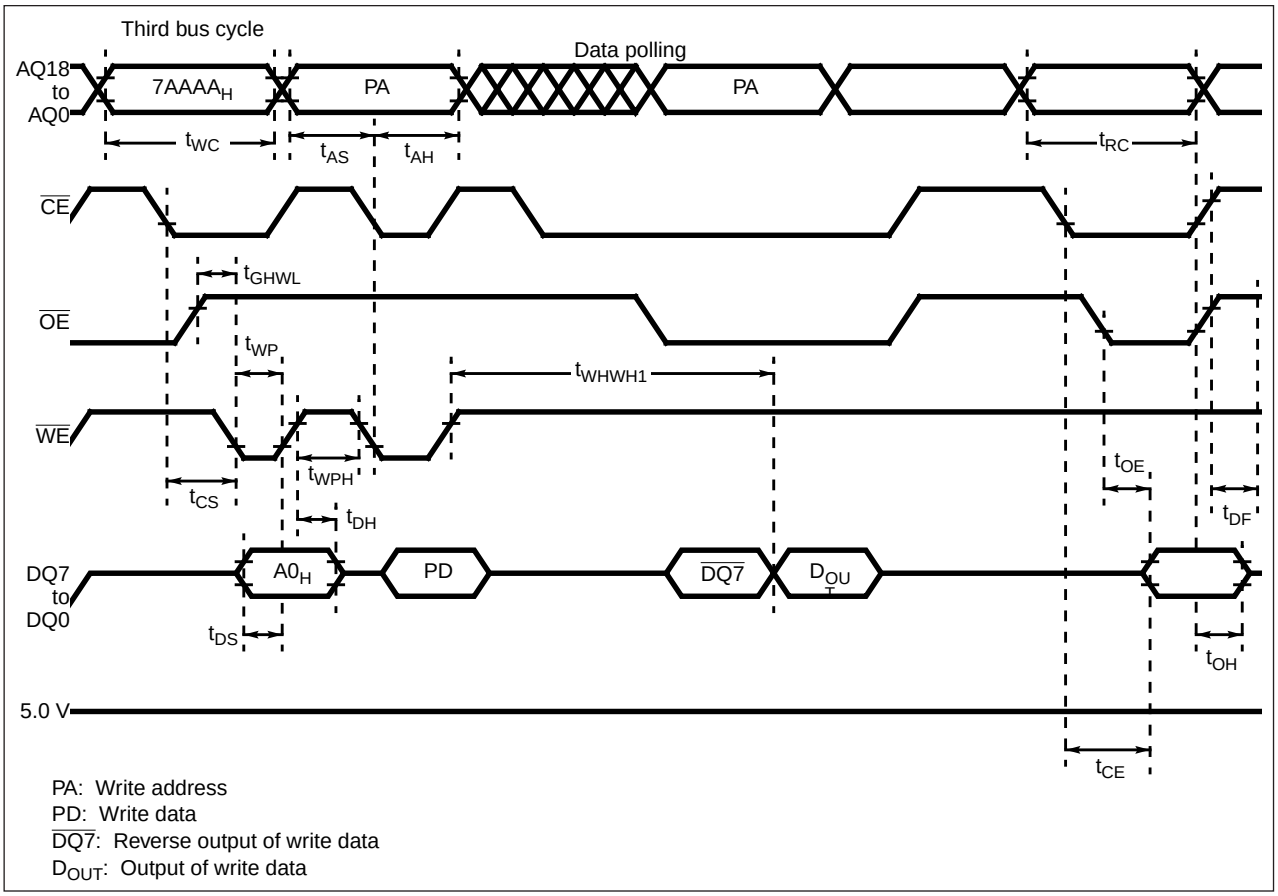
### Data read by read access

Figure 37-27. Timing diagram for read access



### Write, data polling, read (WE control)

Figure 37-28. Write, data polling, read (WE control)

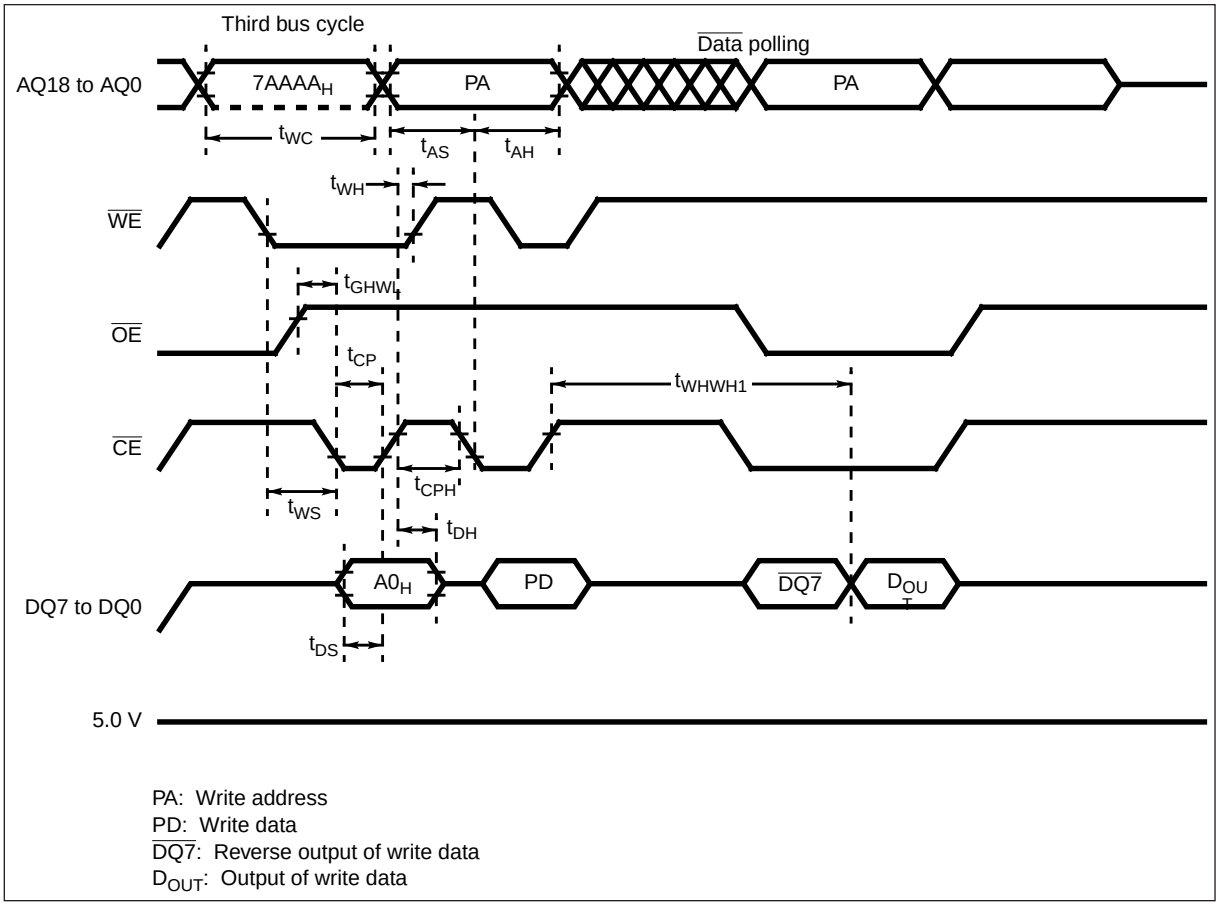



**Note:**

The last two bus cycle sequences out of the four are described.

### Write, data polling, read (CE control)

Figure 37-29. Timing diagram for write access (CE control)

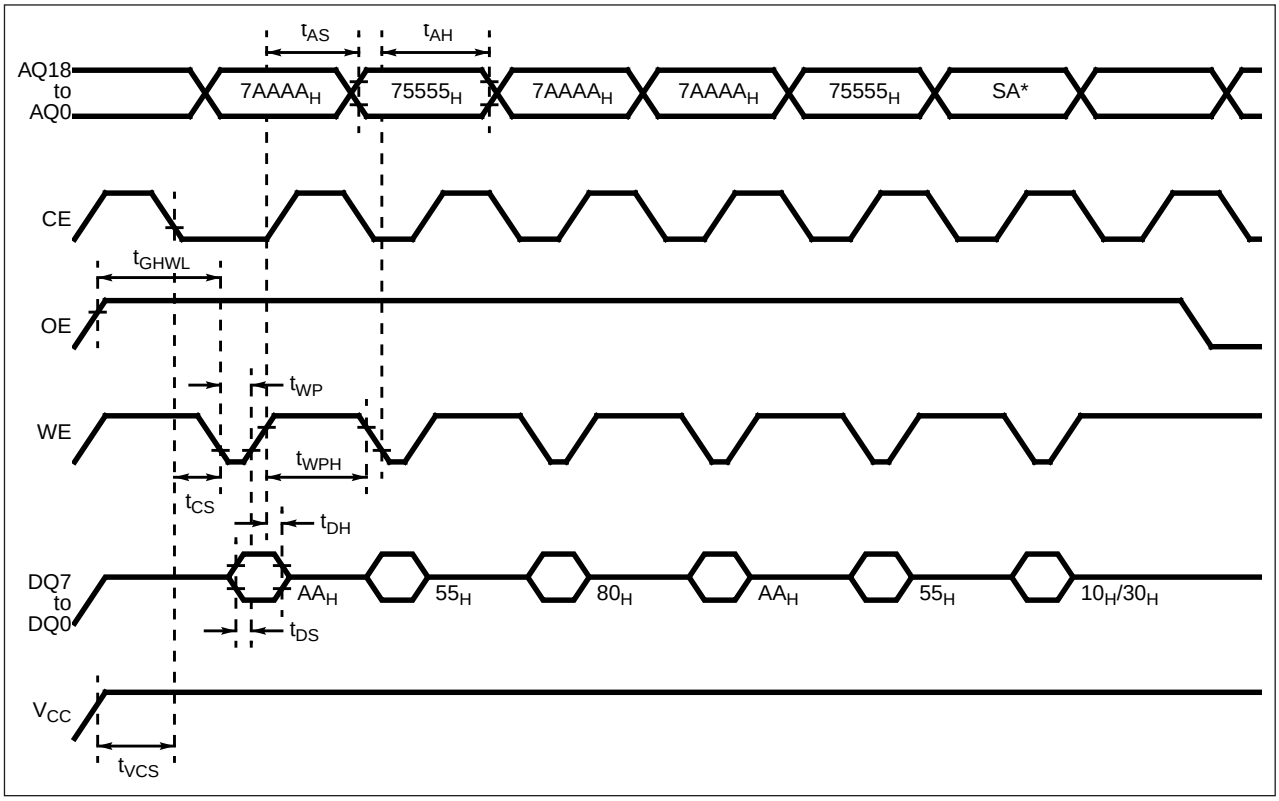



**Note:**

The last two bus cycle sequences out of the four are described.

### Chip erase/sector erase command sequence

Figure 37-30. Timing diagram for write access (chip erasing/sector erasing)

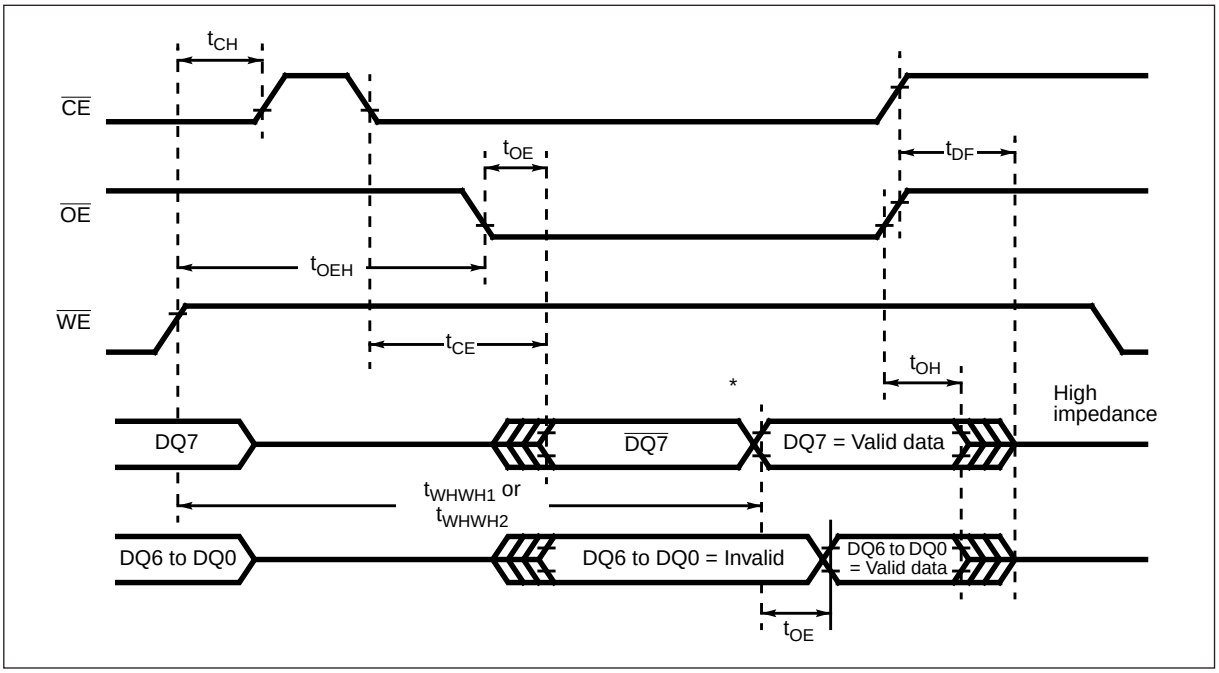



**Note:**

SA is the sector address at sector erasing. $7AAAA_H$ (or $6AAAA_H$) is the address at chip erasing.

### Data polling

Figure 37-31. Timing diagram for data polling

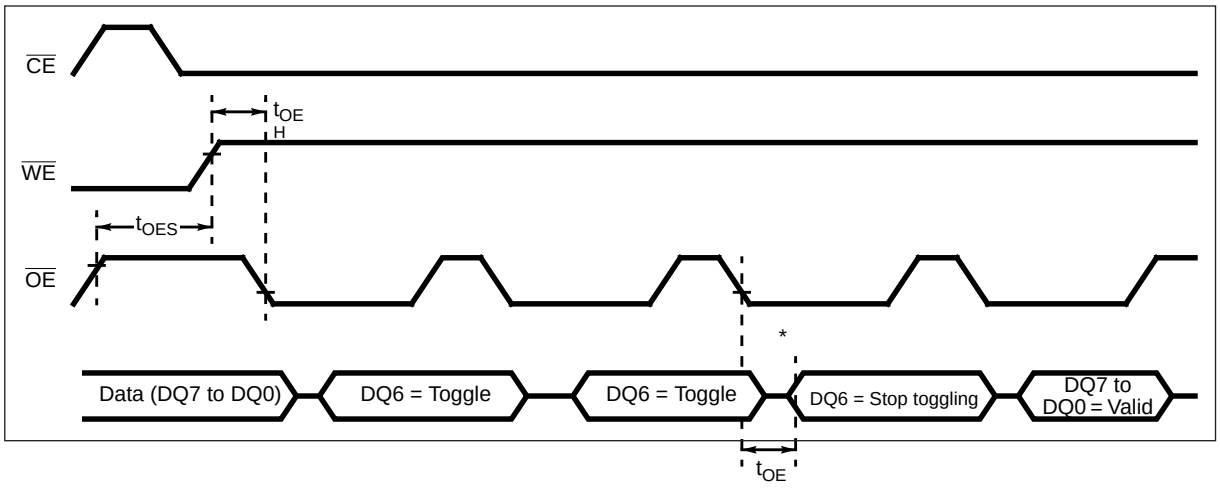



**Note:**

DQ7 is valid data (The device terminates automatic operation).

### Toggle bit

Figure 37-32. Timing diagram for toggle bit

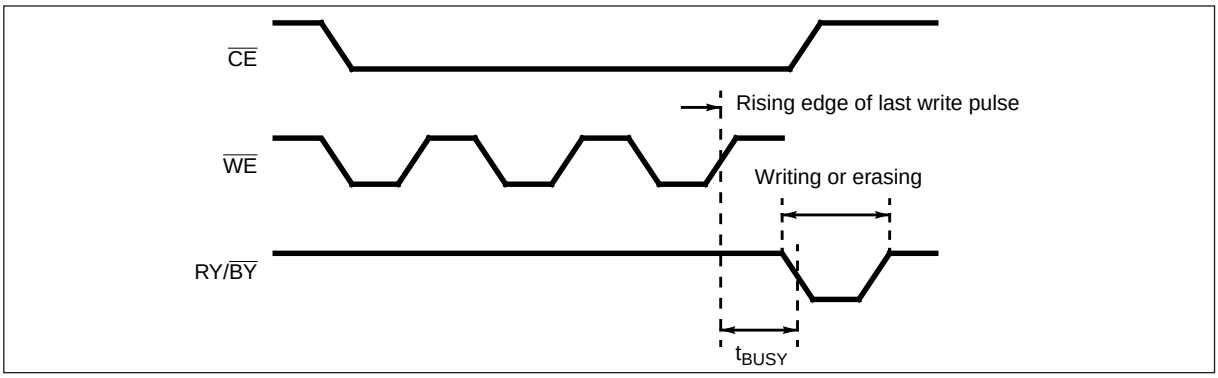



**Note:**

DQ6 stops toggling (The device terminates automatic operation).

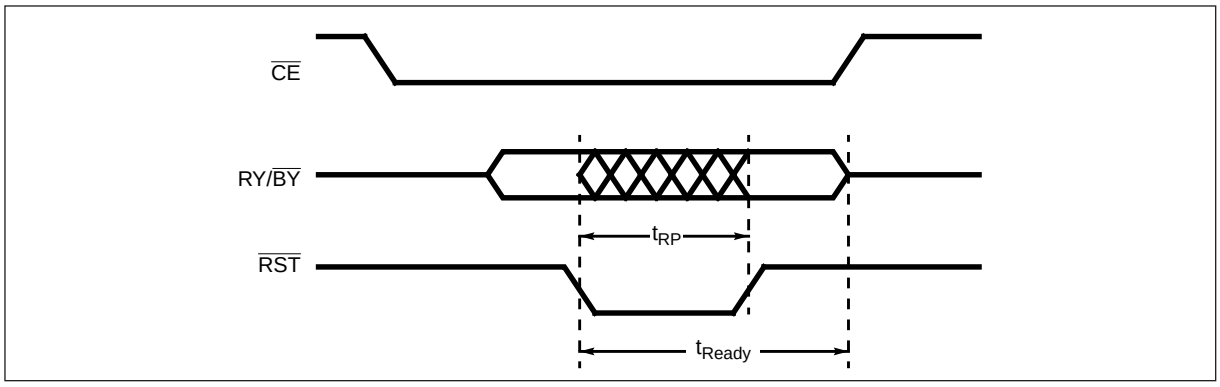
### RY/BY timing during writing/erasing

Figure 37-33. Timing diagram for output of RY/BY signal during writing/erasing



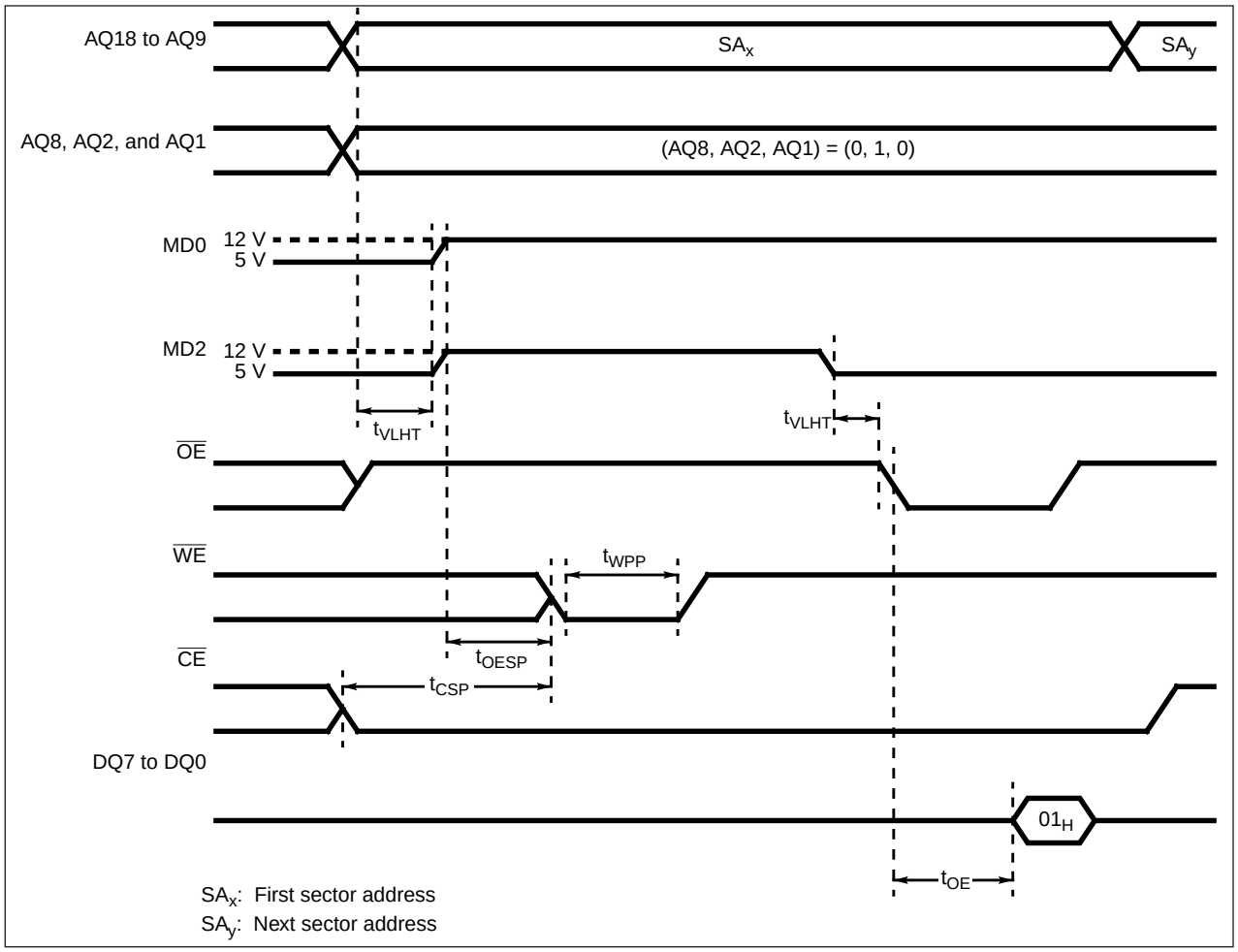

### RST and RY/BY timing

Figure 37-34. Timing diagram for output of RY/BY signal at hardware reset



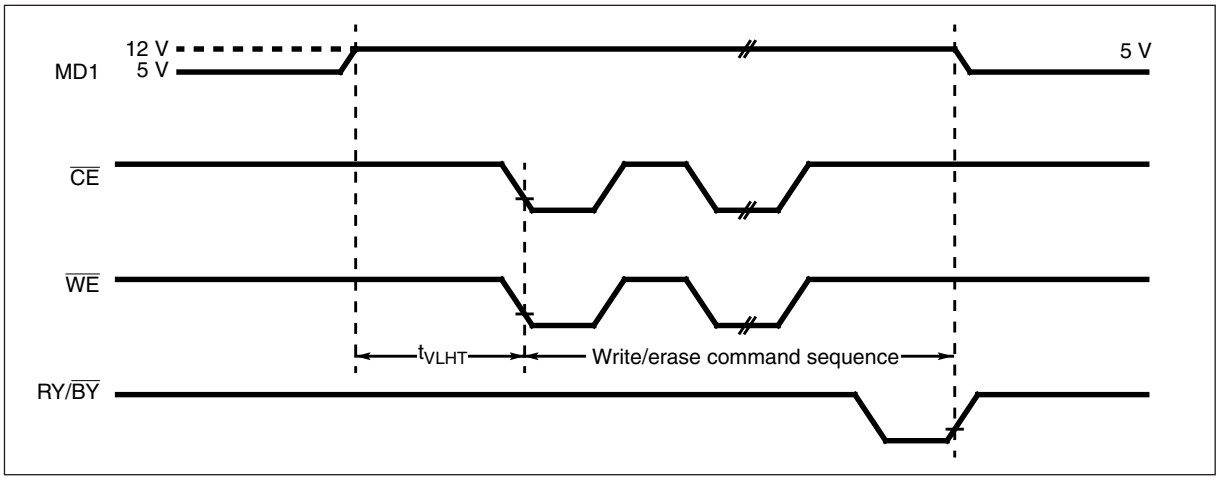
### Enable sector protect/verify sector protect

Figure 37-35. Enable sector protect/verify sector protect




### Temporary sector protect cancellation

Figure 37-36. Temporary sector protect cancellation



## 37.4 Revision History

| Revision | Date | Modification |
|---|---|---|
| 4 | 2006-08-04 | Added MB96F348R/Y, MB96F348H/T<br>Table 1.2-1 "Product Lineup"<br>Figure 1.6-2 "Pin assignment of MB96(F)34x (QFP package)"<br>Figure 1.6-3 "Pin assignment of MB96(F)34x (LQFP package)"<br>Table 1.8-1<br>Table 1.14-1<br>APPENDIX A "I/O Map of MB96V300" |
| 4 | 2006-08-04 | Added Flash memory control signal AQ19 to AQ21<br>Table 1.8-1 "Flash memory control signals MB96F34X"to<br>Table 1.8-4 "Flash memory control signals MB96F38X" |
| 4 | 2006-08-04 | Changed description of Flash programming to be compliant with MBM29LV200 description<br>Table 34.5-1 |
| 5 | 2006-08-07 | Corrected number of USARTs in product line-up<br>Table 1.2-1<br>Table D-2 "Interrupt vector table MB96(F)34x" |
| 5 | 2006-08-07 | Corrected availability of Satellite Flash memory interrupt |
| 5 | 2006-08-08 | Corrected number of SMC channels in MB96F38x.<br>Table 1.2-1 "Product Lineup" |
| 5 | 2006-08-08 | Corrected description of Free Running Timer: software clear and clock selection.<br>Figure 15.3-1 "16-bit Free-Running Timer block diagram"<br>Table 15.3-2 |
| 5 | 2006-08-08 | Corrected description of OCU on MB96F38x<br>Table 1.12-2 "Connection between Output Compare Units and Free Running Timers" |
| 5 | 2006-08-08 | Added note, that port input is disabled when MCU enters Stop or Timer mode.<br>Section 14.2.4 "Port Input Enable Register (PIERnn)" |
| 5 | 2006-08-08 | Added CHAPTER 8 "CLOCK MODULATOR" |
| 5 | 2006-08-08 | Corrected description of USART Baud Rate Generator<br>Figure 21.4-8 "Baud rate generator register (BGRn)" |
| 5 | 2006-08-08 | Added description of USART ECCR:INV bit.<br>Section 21.4.6 "Extended Communication Control Register (ECCRn)" |
| 5 | 2006-08-10 | Added detailed description of reset cause for each register.<br>Section 25.2 "Real Time Clock Registers"<br>Added description of WOT pin function<br>Figure 25.3-2 "WOT pin operation" |
| 5 | 2006-08-10 | Corrected Sound generator description<br>Figure 29.1-1 "Block diagram of Sound Generator"<br>Figure 29.2-2 "Configuration of the Sound Generator Control Register (SGCRHn)" |
| 5 | 2006-08-10 | Corrected Alarm comparator mode description |
| 5 | 2006-08-11 | Corrected Stepping Motor Controller description<br>28.6 "PWM1 and PWM2 Selection Registers (PWS1n, PWS2n)" |
| 5 | 2006-08-14 | USART chapter: corrected some formatting issues.<br>Corrected description of ESIR:AICD bit<br>Table 21.4-4 "Configuration of the extended status/control register (ESCRn)" |
| 5 | 2006-08-14 | Changed naming: fixed vector mode -> internal vector mode<br>CHAPTER 2 "CPU"<br>CHAPTER 9 "RESETS AND STARTUP"<br>CHAPTER 13 "EXTERNAL BUS INTERFACE" |
| 6 | 2006-08-18 | Added explanation of USART ESIR:AICD.<br>CHAPTER 21 "USART" |

| Revision | Date | Modification |
|---|---|---|
| 6 | 2006-08-18 | Changed Alarm comparator input pin name to ALARMn.<br>Figure 20.1-1 "Block diagram of Alarm Comparator" |
| 6 | 2006-08-21 | Corrected pin-out of MB96(F)36x, pin 35, 36<br>Figure 1.6-5 "Pin assignment of MB96(F)36x"<br>Table 1.8-1 |
| 6 | 2006-08-21 | Corrected product line-up, feature CAN, part numbers.<br>Table 1.2-1 "Product Lineup" |
| 6 | 2006-08-21 | Corrected reset value of PLLCR<br>Figure 6.2-6 "Configuration of the PLL Control Register (PLLCR)" |
| 6 | 2006-08-22 | Added RTC for MB96(F)34x |
| 6 | 2006-08-25 | Changed max. operation frequency to 56MHz<br>MB96F38x: - added two additional UARTs, added 1ch SG, added 4ch ICU |
| 6 | 2006-10-06 | LCD: changed prescaler settings for Subclock and RC-Clock |
| 6 | 2006-10-06 | Flash: table with recommended setting changed |
| 6 | 2006-10-09 | register name changed of CANx output enable register |
| 6 | 2006-10-16 | Overview: INTxR renamed to INTx_R, INT3_R1 added to all Cantrol Pin assignments |
| 6 | 2006-10-16 | Overview: IO circuit types corrected according to DS: 4mA -> 5/2mA, I2C cell added with 3mA |
| 7 | 2006-11-08 | RTC: Corrected bit position description of register WTCKSR.<br>Figure 25.2-1 "Real Time Clock registers"<br>Figure 25.2-4 "Clock Select Register (WTCKSR)" |
| 7 | 2006-12-05 | External bus: Setting the clock to inactive level during external bus pause is possible only for divided ECLK clock (DIV[2:0]≠"000") |
| 7 | 2006-12-11 | Added USART 7_R, 8_R, RTC for MB96F35x<br>Table 1.2-1 "Product Lineup"<br>Figure 1.6-4 "Pin assignment of MB96(F)35x"<br>Table 1.8-1 |
| 7 | 2006-12-11 | Corrected package type of MB96F35x in product line-up.<br>Table 1.2-1 "Product Lineup" |
| 7 | 2006-12-11 | Corrected Flash characteristics in product line-up.<br>Table 1.2-1 "Product Lineup" |
| 8 | 2006-12-12 | Removed devices with less than 128kB ROM/Flash.<br>Table 1.2-1 "Product Lineup" |
| 9 | 2006-12-13 | Changed external bus i/f naming AD0 - AD9 -> AD01 -> AD09, A0 - A9 -> A00 -> A09. |
| 10 | 2007-01-22 | Changed Figure 28.2-1 "Overview of the Stepping Motor Controller registers". |
| 10 | 2007-02-12 | Corrected "Interrupt vector table MB96(F)32x" |
| 10 | 2007-02-12 | Corrected "Interrupt vector table MB96(F)38x" |
| 10 | 2007-02-12 | Added MB96384 in product line-up Table 1.2-1 "Product Lineup" |
| 10 | 2007-02-12 | Added ADC reference voltage switching in Table 1.2-1 "Product Lineup", Table 1.8-1, Table 1.8-2, block diagrams and CHAPTER 19 |
| 10 | 2007-02-12 | Updated Table "MB96300 I/O map" |
| 11 | 2007-03-08 | Added MB96F348TSB/HSB/TWB/HWB and made corresponding changes for MB96F348TSA/HSA/TWA/HWA |
| 11 | 2007-04-02 | Added chapter CHAPTER 7 "VOLTAGE REGULATOR CONTROL"<br>Added 1.9V operation in CHAPTER 34 "FLASH MEMORY".<br>Added register VRCR in APPENDIX A "I/O Map of MB96V300".<br>Introduced consistent naming of resources in APPENDIX A "I/O Map of MB96V300".<br>Made revision history a separate chapter (removed it from APPENDIX). |
| 11 | 2007-04-13 | Changed description of initial value of EDSU:TIE bit from 'X' to '0'.<br>Figure 32.2-4 "EDSU extension register (EDSU)" |
| 11 | 2007-04-13 | Removed Jump Queue from chapter 2.1 "Outline of the CPU". |
| 11 | 2007-04-13 | Removed sentence in CAN chapter 23.3 "Register Description" that IF data registers are provided big endian and little endian. |

| Revision | Date | Modification |
|---|---|---|
| 11 | 2007-04-13 | Corrected Figure 16.2-1 "16-bit Reload Timer register" grouping of TMR register. |
| 11 | 2007-04-17 | Removed some sections in chapter 1, which are described in datasheet:<br>Block diagram of Flash/Mask ROM MCU<br>Pin assignment drawings<br>Pin function description<br>I/O cell type<br>Handling devices<br>Flash memory interface port connection |
| 11 | 2007-04-17 | Removed Appendix D: List of Interrupt vectors, because it is shown in the datasheet. |
| 11 | 2007-04-19 | LCD, RTC, Calibration unit: added note about using clock selection |
| 11 | 2007-04-24 | Corrected some drawing in CHAPTER 36 "EXAMPLES OF SERIAL PROGRAMMING CONNECTION". |
| 12 | 2007-05-15 | Clock Modulator, RTC and Reset chapters updates |
| 12 | 2007-06-16 | SMC update, EDSU2 add-on, corrections in External Bus register description |
| 13 | 2007-08-06 | Overview: corrected address of PRRR9 in figure 1.14-10 |
| 13 | 2007-09-28 | Overview: change block diagram to include USB and CLKP3. Super series lineup table update<br>clocks: Add CLKP3 information<br>Clock ouput function: Add CLKP3 information<br>USB chapter modified. |
| 14 | 2007-11-20 | Overview modified to include CLKP3 clock domain and MB96330 series.<br>Chapter 10 updated to include some small corrections and improvements of the regulator control description.<br>New USB chapters.<br>Fig 21.4-8 BGR has 15bits not 14 as previously described.<br>Chapter 10: typos error corrected.<br>IOPORT chapter: precision added for PSR and PDR regarding the IER settings.<br>Flash chapter: table 34.4-1: table title corrected in content and layout. Note on DQ2 bit added. Flash programming example added<br>Mask-ROM Interface chapter added<br>Relaod Timer chapter: TIN pulse length corrected<br>ROM/Flash security chapter modified to include ROM security features. typo corrections.<br>USART: correction in SCDE bit description. Cross-reference added. Typos<br>SMC. typos corrections.<br>Annex: IOMAP modified to include GPIO18 &19. |
| 15 | 2008-02-01 | Overview: Extended and updated features, updated block diagram.<br>DMA: Corrected initial value of DISELx registers.<br>Flash/ROM: Changed register naming.<br>Watchdog: Changed interval times.<br>LCD: Removed limitation of COMEN setting<br>CAN: Fixed initial value of IF2MSK2Hn register.<br>USART, I2C, Soundgenerator: Fixed initial value shading of registers.<br>Fixed several typos. |
| 16 | 2008-11-07 | DMA: Added description of IOABK register<br>Source Clock Timers: Added handling note for interrupt flags<br>ADC: Fixed description of PAUS bit and conversion data protection function<br>Real Time Clock: Fixed sample values<br>USB: Reworked entire chapter<br>Flash: Added Data Flash descriptions<br>ROM/Flash Security: Added Data Flash descriptions |

# Revision History

Document History

| Document Title: F$^2$MC-16FX 16-Bit Microcontroller MB96300 Series Hardware Manual<br>Document Number: 002-19737 | | | | |
|---|---|---|---|---|
| **Revision** | **ECN#** | **Issue Date** | **Origin of Change** | **Description of Change** |
| ** | 5747313 | 05/23/2017 | ANMA | Migrated to Cypress and assigned document number 002-19737.<br>No change to document contents. |