Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CCG7D Family

EZ-PD CCG7D Architecture Technical Reference Manual (TRM)

Document No. 002-33193 Rev. **

June 11, 2021

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com
www.infineon.com

Copyrights

© Cypress Semiconductor Corporation, 2021. This document is the property of Cypress Semiconductor Corporation, an Infineon Technologies company, and its affiliates (“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. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product.

CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, “Security Breach”). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. 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. “High-Risk Device” means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. “Critical Component” means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, 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 any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, including its affiliates, and its directors, officers, employees, agents, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress’s published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

Cypress, the Cypress logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, Traveo, WICED, and ModusToolbox are trademarks or registered trademarks of Cypress or a subsidiary of Cypress in the United States or in 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.

Contents Overview

Section A: Overview **12**
1. Introduction 13
2. Getting Started 18
3. Document Construction 19

Section B: CPU System **22**
4. Cortex-M0 CPU 23
5. Interrupts 28

Section C: Memory System **36**
6. Memory Map 37

Section D: System-Wide Resources **39**
7. I/O System 40
8. Clocking System 47
9. Power Supply and Monitoring 52
10. Chip Operational Modes 54
11. Power Modes 55
12. Watchdog Timer 59
13. Reset System 62
14. Device Security 64

Section E: Digital System **65**
15. Serial Communications (SCB) 66
16. Cryptography (TR) Block 103
17. Timer and PWM 109

Section F: USB Power Delivery **124**
18. USB Power Delivery 125

Section G: Program and Debug **170**
19. Program and Debug Interface 171
20. Nonvolatile Memory Programming 178

Glossary **192**

Contents

Section A: Overview 12

1. Introduction 13

1.1 Top Level Architecture 14
1.2 Features 14
1.3 CPU System 15
1.3.1 Processor 15
1.3.2 Interrupt Controller 15
1.4 Memory 15
1.5 USB PD Subsystem 15
1.5.1 USB-PD Physical Layer 15
1.5.2 ADC 15
1.5.3 VCONN FET 15
1.5.4 OCP on VBUS 15
1.5.5 Under-Voltage and Over-Voltage Protection on VBUS 15
1.5.6 VBAT to Ground Protection 15
1.5.7 NFET Gate Driver for VBUS 15
1.5.8 Charger Detection 16
1.5.9 VBUS Discharge 16
1.5.10 Gate Drivers for Input and Output Side of Buck-Boost Switches 16
1.5.11 Error Amplifier 16
1.5.12 High-side Current Sense Amplifier for VIN 16
1.5.13 Pulse Width Modulator 16
1.5.14 High-side Current Sense Amplifier for VBUS 16
1.5.15 VIN Regulator 16
1.5.16 20-V Tolerant SBU and CC Lines 16
1.6 System-Wide Resources 16
1.6.1 Clocking System 16
1.6.2 Power System 17
1.6.3 GPIO 17
1.7 Fixed-Function Digital 17
1.7.1 Timer/Counter/PWM Block 17
1.7.2 Serial Communication Block (SCB) 17
1.8 Program and Debug 17

2. Getting Started 18

2.1 Support 18
2.2 Product Upgrades 18

3. Document Construction 19

3.1 Major Sections 19
3.2 Documentation Conventions 19

3.2.1 Register Conventions 19
3.2.2 Numeric Naming 19
3.2.3 Units of Measure 20
3.2.4 Acronyms 20

Section B: CPU System 22

4. Cortex-M0 CPU 23

4.1 Features 23
4.2 Block Diagram 24
4.3 How It Works 24
4.3.1 Registers 24
4.3.2 Operating Modes 26
4.3.3 Instruction Set 26
4.3.3.1 Address Alignment 27
4.3.3.2 Memory Endianness 27
4.3.4 Systick Timer 27
4.3.5 Debug 27

5. Interrupts 28

5.1 Features 28
5.2 How It Works 28
5.3 Interrupts and Exceptions – Operation 29
5.3.1 Interrupt and Exception Handling 29
5.3.2 Level and Pulse Interrupts 29
5.3.3 Exception Vector Table 30
5.4 Exception Sources 30
5.4.1 Reset Exception 30
5.4.2 Non-Maskable Interrupt (NMI) Exception 31
5.4.3 HardFault Exception 31
5.4.4 Supervisor Call (SVCall) Exception 31
5.4.5 PendSV Exception 31
5.4.6 SysTick Exception 31
5.5 Interrupt Sources 31
5.6 Exception Priority 32
5.7 Enabling/Disabling Interrupts 33
5.8 Exception States 33
5.8.1 Pending Exceptions 33
5.9 Stack Usage for Exceptions 34
5.10 Interrupts and Low-Power Modes 34
5.11 Exception – Initialization and Configuration 34
5.12 Registers 35
5.13 Associated Documents 35

Section C: Memory System 36

6. Memory Map 37

6.1 Features 37
6.2 How It Works 37

Section D: System-Wide Resources 39

7. I/O System 40

7.1 Features....40
7.2 Block Diagram....41
7.3 GPIO Drive Modes....42
7.3.1 High-Impedance Analog....43
7.3.2 High-Impedance Digital....43
7.3.3 Resistive Pull-Up or Resistive Pull-Down....43
7.3.4 Open Drain Drives High and Open Drain Drives Low....43
7.3.5 Strong Drive....43
7.3.6 Resistive Pull-Up and Resistive Pull-Down....43
7.4 Slew Rate Control....44
7.5 CMOS LVTTL Level Control....44
7.6 GPIO-OVT....44
7.7 High-Speed I/O Matrix....44
7.8 Firmware Controlled GPIO....45
7.9 I/O Port Reconfiguration....45
7.10 I/O State on Power Up....45
7.11 Behavior in Low-Power Modes....45
7.12 GPIO Interrupt....45
7.12.1 Features....45
7.12.2 Interrupt Controller Block Diagram....45
7.12.3 Function and Configuration....46
7.13 Registers....46

8. Clocking System 47

8.1 Block Diagram....47
8.2 Clock Sources....48
8.2.1 Internal Main Oscillator....48
8.2.1.1 Startup Behavior....48
8.2.2 Internal Low-speed Oscillator....48
8.2.3 External Clock....48
8.3 Clock Distribution....48
8.3.1 HFCLK Input Selection....48
8.3.2 HFCLK Predivider Configuration....48
8.3.3 SYSCLK Prescaler Configuration....49
8.3.4 Peripheral Clock Divider Configuration....49
8.4 Low-Power Mode Operation....50
8.5 Registers....51

9. Power Supply and Monitoring 52

9.1 Block Diagram....52
9.2 How It Works....53
9.2.1 Regulator Summary....53
9.2.1.1 Active Digital Regulator....53
9.2.1.2 Quiet Regulator....53
9.2.1.3 Deep-Sleep Regulator....53
9.3 Voltage Monitoring....53
9.3.1 Power-On-Reset....53
9.3.1.1 Brownout-Detect....53
9.4 Register List....53

10. Chip Operational Modes **54**
10.1 Boot 54
10.2 User 54
10.3 Privileged 54
10.4 Debug 54

11. Power Modes **55**
11.1 Active Mode 56
11.2 Sleep Mode 56
11.3 Deep Sleep Mode 56
11.4 Power Mode Summary 57
11.5 Low-Power Mode Entry and Exit 57
11.6 Register List 58

12. Watchdog Timer **59**
12.1 Features 59
12.2 Block Diagram 59
12.3 How It Works 59
12.3.1 Enabling and Disabling WDT 60
12.3.2 WDT Interrupts and Low-Power Modes 60
12.3.3 WDT Reset Mode 60
12.4 Register List 61

13. Reset System **62**
13.1 Reset Sources 62
13.1.1 Power-on Reset 62
13.1.2 Brownout Reset 62
13.1.3 Watchdog Reset 62
13.1.4 Software Initiated Reset 62
13.1.5 External Reset 63
13.1.6 Protection Fault Reset 63
13.2 Identifying Reset Sources 63
13.3 Register List 63

14. Device Security **64**
14.1 Features 64
14.2 How It Works 64

Section E: Digital System **65**

15. Serial Communications (SCB) **66**
15.1 Features 66
15.2 Serial Peripheral Interface (SPI) 66
15.2.1 Features 66
15.2.2 General Description 67
15.2.3 SPI Modes of Operation 67
15.2.3.1 Motorola SPI 67
15.2.3.2 Texas Instruments SPI 69
15.2.3.3 National Semiconductors SPI 71
15.2.4 Easy SPI (EZSPI) Protocol 73
15.2.4.1 EZ Address Write 73

15.2.4.2 Memory Array Write 73
15.2.4.3 Memory Array Read 73
15.2.4.4 Configuring SCB for EZSPI Mode 75
15.2.5 SPI Registers 75
15.2.6 SPI Interrupts 76
15.2.7 Enabling and Initializing SPI 76
15.2.8 Internally and Externally Clocked SPI Operations 77
15.2.8.1 Non-EZ Mode of Operation 77
15.2.8.2 EZ Mode of Operation 78
15.3 UART 79
15.3.1 Features 79
15.3.2 General Description 80
15.3.3 UART Modes of Operation 80
15.3.3.1 Standard Protocol 80
15.3.3.2 SmartCard (ISO7816) 83
15.3.3.3 IrDA 84
15.3.4 UART Registers 85
15.3.5 UART Interrupts 85
15.3.6 Enabling and Initializing UART 86
15.4 Inter Integrated Circuit (I2C) 87
15.4.1 Features 87
15.4.2 General Description 87
15.4.3 Terms and Definitions 87
15.4.4 I2C Modes of Operation 88
15.4.4.1 Write Transfer 89
15.4.4.2 Read Transfer 89
15.4.5 EZI2C Protocol 90
15.4.6 Memory Array Write 90
15.4.7 Memory Array Read 90
15.4.8 I2C Registers 91
15.4.9 I2C Interrupts 91
15.4.10 Enabling and Initializing the I2C 92
15.4.11 Internal and External Clock Operation in I2C 93
15.4.11.1 I2C Non-EZ Operation Mode 93
15.4.11.2 EZ Operation Mode 93
15.4.12 Wake up from Sleep 94
15.4.13 Master Mode Transfer Examples 95
15.4.13.1 Master Transmit 95
15.4.13.2 Master Receive 96
15.4.14 Slave Mode Transfer Examples 97
15.4.14.1 Slave Transmit 97
15.4.14.2 Slave Receive 98
15.4.15 EZ Slave Mode Transfer Example 99
15.4.15.1 EZ Slave Transmit 99
15.4.15.2 EZ Slave Receive 100
15.4.16 Multi-Master Mode Transfer Example 101
15.4.16.1 Multi-Master – Slave Not Enabled 101
15.4.16.2 Multi-Master – Slave Enabled 102

16. Cryptography (TR) Block 103

16.1 Features 103
16.2 Block Diagram 103
16.2.1 Block Application Overview 104

16.2.1.1 True Random Number Generator (TRNG) 104

17. Timer and PWM **109**

17.1 Features 109
17.2 Block Diagram 110
17.2.1 Enabling and Disabling Counter in TCPWM Block 110
17.2.2 Clocking 110
17.2.3 Events Based on Trigger Inputs 111
17.2.4 Output Signals 112
17.2.4.1 Signals upon Trigger Conditions 112
17.2.4.2 Interrupts 112
17.2.4.3 Outputs 112
17.2.5 Power Modes 113
17.3 Modes of Operation 113
17.3.1 Timer Mode 114
17.3.1.1 Block Diagram 114
17.3.1.2 How It Works 114
17.3.1.3 Configuring Counter for Timer Mode 116
17.3.2 Pulse Width Modulation Mode 117
17.3.2.1 Block Diagram 117
17.3.2.2 How It Works 118
17.3.2.3 Other Configurations 119
17.3.2.4 Kill Feature 119
17.3.2.5 Configuring Counter for PWM Mode 120
17.3.3 Pulse Width Modulation Pseudo-Random Mode 121
17.3.3.1 Block Diagram 121
17.3.3.2 How It Works 121
17.3.3.3 Configuring Counter for Pseudo-Random PWM Mode 122
17.4 TCPWM Registers 123

Section F: USB Power Delivery **124**

18. USB Power Delivery **125**

18.1 Features 125
18.2 Block Diagram 126
18.3 How It Works 127
18.3.1 Power Delivery PHY 127
18.3.1.1 Analog Baseband PHY 128
18.3.1.2 Digital Baseband PHY 130
18.3.1.3 Protocol Interface Description 130
18.3.2 UVOV Block on VBUS_C 133
18.3.2.1 UVOV Block Architecture and Functional Description 133
18.3.2.2 CCG7D Filtering Comparator Output 134
18.3.3 UVOV Block on VIN 134
18.3.4 SAR Analog to Digital Converter Block 135
18.3.5 REFGEN Block 136
18.3.5.1 REFGEN Functional Description 136
18.3.6 Provider Path NGDO Block 136
18.3.6.1 Provider Path NGDO Block Architecture and Functional Overview 136
18.3.7 VBUS_DISCHG Block 137
18.3.7.1 VBUS_DISCHG Block Architecture and Functional Overview 137
18.3.8 DPSLP_Ref Block 138

18.3.8.1 Architecture and Function Description 138
18.3.9 VCONN FETs 139
18.3.9.1 Architecture and Function Description 139
18.3.10 VBAT to Ground Protection 140
18.3.10.1 Architecture and Function Description 140
18.3.11 Charger Detection 141
18.3.11.1 Architecture and Function Description 141
18.3.12 Gate Drivers for Input and Output Side of Buck-Boost Switches 142
18.3.12.1 Architecture and Function Description for Input 142
18.3.12.2 Architecture and Function Description for Output 144
18.3.13 Error Amplifier 145
18.3.13.1 Architecture and Function Description 145
18.3.14 High-Side Current Sense Amplifier for VIN 150
18.3.14.1 Architecture and Function Description 150
18.3.15 Pulse-Width Modulator 153
18.3.15.1 Architecture and Function Description 153
18.3.16 High-Side Current Sense Amplifier for VBUS 153
18.3.16.1 Architecture and Function Description 153
18.3.17 OCP on VBUS 154
18.3.18 VIN Regulator 154
18.3.18.1 Architecture and Function Description 154
18.3.19 Low Frequency Counter (LF_CNTR) 155
18.3.19.1 Features 155
18.3.19.2 How It Works 155
18.3.19.3 LF_CNTR Interrupts and Low-Power Modes 155
18.4 PD Block Register Usage 156
18.4.1 PD Block Registers and Interrupt Handling Flow 156
18.4.1.1 PD Block Clocking 156
18.4.1.2 PD Block Key Registers 156
18.5 USB PD Interrupts 158
18.5.1 Basic Flow for PD Interrupt Handling 160
18.6 USB PD Block Power Modes 162
18.7 USB PD Functional Modes 162
18.8 Procedures Handled by PD Block - Type-C PD Initialization and Operation 162
18.8.1 Initialization, Enabling, and Disabling the PD Block 162
18.8.2 Deep-Sleep Entry and Exit Procedures 163
18.8.3 Changing Termination on CC-lines 164
18.8.4 Measuring CC-line State 164
18.8.5 Sending a PD Packet 164
18.8.6 Receiving a PD Packet 165
18.8.7 Initialization and Usage of ADC 165
18.8.8 Initialization and Usage of UP/DOWN and ADC Comparator Output Filters 166
18.8.9 Regulation Sequence in Firmware 167
18.8.9.1 Buck-Boost Enable Sequence 168

Section G: Program and Debug 170

19. Program and Debug Interface 171

19.1 Features 171
19.2 Functional Description 171
19.3 Serial Wire Debug (SWD) Interface 172
19.3.1 SWD Timing Details 173
19.3.2 ACK Details 173

19.3.3 Turnaround (Trn) Period Details....173
19.4 Cortex-M0 Debug and Access Port (DAP)....174
19.4.1 Debug Port (DP) Registers....174
19.4.2 Access Port (AP) Registers....174
19.5 Programming the CCG7D Device....175
19.5.1 SWD Port Acquisition....175
19.5.1.1 Primary and Secondary SWD Pin Pairs....175
19.5.1.2 SWD Port Acquire Sequence....175
19.5.2 SWD Programming Mode Entry....175
19.5.3 SWD Programming Routines Executions....175
19.6 CCG7D SWD Debug Interface....175
19.6.1 Debug Control and Configuration Registers....176
19.6.2 Breakpoint Unit (BPU)....176
19.6.3 Data Watchpoint (DWT)....176
19.6.4 Debugging the CCG7D Device....176
19.7 Registers....177

20. Nonvolatile Memory Programming 178

20.1 Features....178
20.2 Functional Description....178
20.3 System Call Implementation....179
20.4 Blocking and Non-Blocking System Calls....179
20.4.1 Performing a System Call....179
20.5 System Calls....180
20.5.1 Silicon ID....181
20.5.2 Configure Clock....181
20.5.3 Load Flash Bytes....182
20.5.4 Write Row....183
20.5.5 Program Row....183
20.5.6 Erase All....184
20.5.7 Checksum....185
20.5.8 Write Protection....185
20.5.9 Non-Blocking Write Row....186
20.5.10 Non-Blocking Program Row....187
20.5.11 Resume Non-Blocking....188
20.6 System Call Status....189
20.7 Non-Blocking System Call Pseudo Code....190

Glossary 192

Section A: Overview

This section encompasses the following chapters:

- Introduction chapter on page 13
- Getting Started chapter on page 18
- Document Construction chapter on page 19

Document Revision History

Table 1-1. Document Revision History

Revision	Issue Date	Description of Change
**	05/06/2021	Initial version.

1. Introduction

EZ-PD™ CCG7D is a dual port USB Type-C PD solution with integrated buck-boost controller. The controller complies with the latest USB Type-C 1.3 and Power Delivery (PD) 3.0 standards. EZ-PD CCG7D is an enhancement to Cypress' EZ-PD CCG6DF Type-C controller with integrated load switch, reverse current protection, and short circuit protection. EZ-PD CCG7D provides a complete USB Type-C and USB-PD port control solution for automotive charger applications such as head-unit charger, rear seat chargers, and rear seat entertainment. EZ-PD CCG7D can be used in downstream facing port (DFP) applications. EZ-PD CCG7D uses Cypress' proprietary M0S8 technology with a 32-bit, 48-MHz Arm® Cortex®-M0 processor with 128-KB flash and integrates two Type-C transceivers including the Type-C termination resistors Rp and Rd. It also includes analog and digital peripherals such as ADC, PWMs, and timers. The inclusion of a fully-programmable MCU with analog and digital peripherals allows the implementation of custom system management functions such as power throttling, load sharing, temperature monitoring, and black box. CCG7D devices have these characteristics:

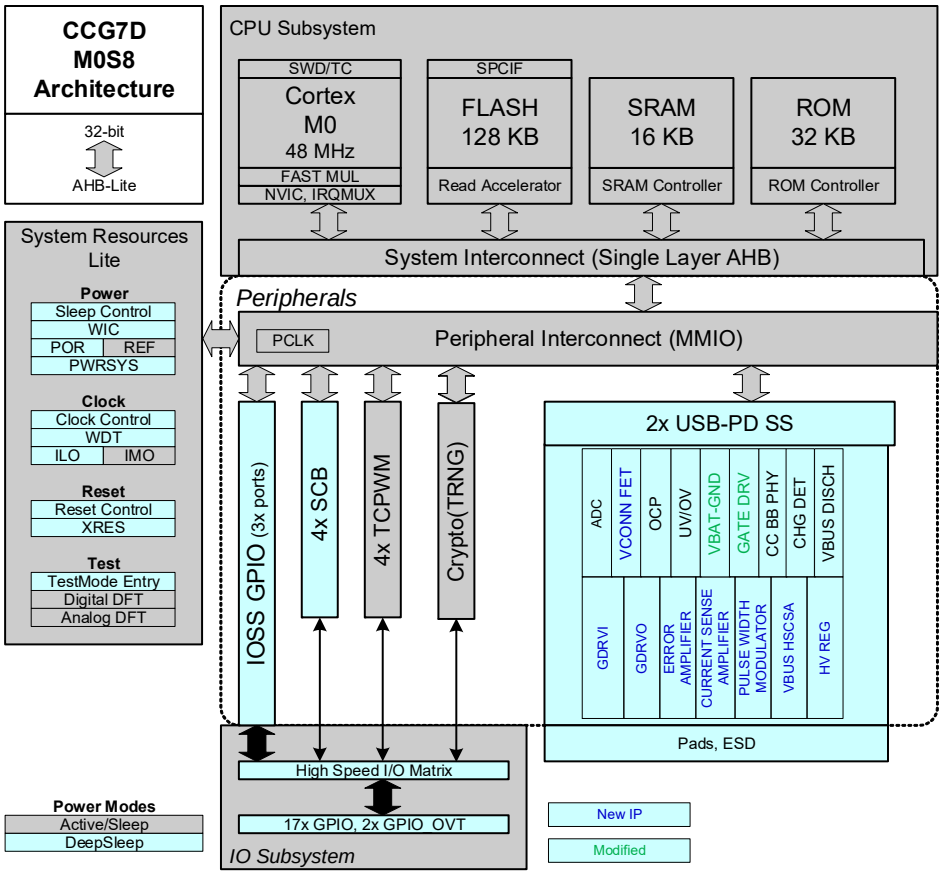
- CCG7D supports two USB Type-C ports
- High-performance, 32-bit single-cycle Cortex-M0 CPU core
- Configurable timer, counter, and PWM blocks
- Configurable serial communication blocks (SCB) to support I^2C, SPI, and UART
- Integrated oscillator eliminating the need for an external clock
- Integrated UFP (Rd) and current sources (Rp) termination resistors
- Integrated legacy charging protocols charger block
- High-voltage protection on configuration channels (CC1 and CC2)
- Integrated VCONN FETs
- Integrated Load Switch in the VBUS provider path supporting fast role swap (FRS) and capable of detecting OC (over current), RC (reverse current), UV (under voltage), OV (over voltage), and SC (short circuit) events
- Integrated buck-boost controller that can be configured to operate in buck-boost mode, buck-only mode or boost-only mode

This document describes each function block of the CCG7D device in detail. This information will help designers to create system-level designs.

1.1 Top Level Architecture

Figure 1-1 shows the major components of the CCG7D architecture.

Figure 1-1. CCG7D Family Block Diagram



1.2 Features

The CCG7D family of USB Type-C PD controllers has these major components:

- 32-bit CPU subsystem
  - 48-MHz Arm Cortex-M0 CPU
  - 128 KB of flash and 16 KB of SRAM
  - 24 KB User SROM programmed with commonly used firmware functions
- Integrated digital blocks
  - Four integrated TCPWM blocks, which can each be configured as a timer, counter, or pulse width modulator
  - Four configurable serial communication blocks (I2C/UART/SPI)
- Type-C support
  - Integrated transceiver (baseband PHY) and support for USB Type-C port
  - Integrated UFP (Rd) termination and current sources for DFP (Rp) for Type-C port
- Power Operating Range
  - 5.5-V to 24-V operation
  - Integrated VCONN FETs to power EMCA cables
  - GPIO power supply generated on chip and decoupled through external cap (VDDIO to be shorted to VDDD)

1.3 CPU System

1.3.1 Processor

The heart of the CCG7D is a 32-bit Cortex-M0 CPU core running up to 48 MHz. It is optimized for low-power operation with extensive clock gating. It uses 16-bit instructions and executes a subset of the Thumb-2 instruction set. This CPU core also includes a hardware multiplier that provides a 32-bit result in one cycle.

1.3.2 Interrupt Controller

The CPU subsystem of CCG7D includes a nested vectored interrupt controller (NVIC) with 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep Sleep mode.

1.4 Memory

The CCG7D memory subsystem consists of a 128 KB flash module with read accelerator, 16 KB SRAM, 8 KB of Boot SROM and 24 KB of User SROM.

1.5 USB PD Subsystem

This subsystem provides the interface to the Type-C USB port and controls the buck-boost converter. This subsystem comprises of USB-PD physical layer, ADC, VCONN FETs, OCP on VBUS, under-voltage and over-voltage protection on VBUS, VBAT to ground protection, NFET gate drivers for VBUS, charger detection, VBUS discharge, gate drivers for non-inverting buck-boost topology switches, error amplifier, high-side current sense amplifier for VIN, pulse-width modulator, high-side current sense amplifier for VBUS, VIN regulator and 20-V VBUS tolerant CC Lines. This subsystem also includes human body model (HBM) electro static discharge (ESD) support on the Type-C port.

1.5.1 USB-PD Physical Layer

The USB-PD sub-system contains the USB-PD physical layer block and supporting circuits. The USB-PD Physical Layer consists of a transmitter and receiver that communicate BMC encoded data over the CC channel per the PD 3.0 standard. All communication is half-duplex. The Physical Layer or PHY practices collision avoidance to minimize communication errors on the channel.

USB-PD block includes termination resistors Rp and Rd as required by the USB Type-C spec. Rp and Rd resistors are required to implement connection detection, plug orientation detection and for establishment of the USB source/sink roles. The Rp resistor is implemented as a current source.

1.5.2 ADC

The ADC is a low-footprint 8-bit SAR ADC that is available for general-purpose A-D conversion applications in the chip. The ADC can be accessed from the GPIOs through an on-chip analog mux. In CCG7D, one ADC shall be instantiated per PD port.

1.5.3 VCONN FET

CCG7D's internal LDO voltage regulator is capable of powering a 100 mW VCONN supply for electronically marked cable assemblies (EMCA), VCONN-powered devices (VPD), and VCONN-powered accessories as defined in the USB Type-C Specification. There are two VCONN FETs in CCG7D to power either CC1 or CC2 pins. These FETs can provide a maximum of 100 mW power over VCONN on the CC1 and CC2 pins for the EMCA cables. At any given time, only one VCONN FET will be enabled on the non-CC line. The FETs have over current protection and support 5 Ω RDS on resistance.

1.5.4 OCP on VBUS

CCG7D supports over-current (OC) protection on the VBUS path using a current sense amplifier. The voltage drop across this precision resistor is monitored to sense the magnitude of current on the VBUS path and compare against a programmable threshold.

1.5.5 Under-Voltage and Over-Voltage Protection on VBUS

CCG7D supports dedicated comparators for detecting over-voltage and under-voltage faults on VBUS. The threshold for both UV and OV are made programmable through firmware.

1.5.6 VBAT to Ground Protection

CCG7D supports VBAT to ground short protection at the connector. The excess current taken from VBAT during a short event at the connector is monitored via the LSSCP. The short protection threshold can be configured. The short protection can be used in either differential or single-ended mode with the help of the VBAT to ground multiplexer.

1.5.7 NFET Gate Driver for VBUS

CCG7D has an integrated high-voltage gate driver to drive the gate of an external high-side NFET on the VBUS provider path. It also supports soft start capability to limit inrush current.

1.5.8 Charger Detection

One charger detection block connected to the DP/DM pins allows CCG7D to detect conventional battery chargers conforming to BC 1.2, and the following proprietary charger specifications: Apple, Qualcomm's Quick-Charge 4.0, 3.0, and Samsung AFC.

1.5.9 VBUS Discharge

CCG7D supports VBUS discharge circuitry inside. After cable removal detection, the device will discharge the residual charge and bring the floating VBUS back to 0.8 V.

1.5.10 Gate Drivers for Input and Output Side of Buck-Boost Switches

CCG7D's buck-boost controller provides four N-channel MOSFET gate drivers: two floating high-side gate drivers at the HG1 and HG2 pins, and two ground referenced low-side drivers at the LG1 and LG2 pin. The high-side gate drivers drive the high-side external FET with a nominal VGS of 5 V. The high-side gate driver has a programmable drive strength to drive external FET. An external capacitor and Schottky diode form a bootstrap network to collect and store the high-voltage source (VIN + ~5 V for HG1 and VBUS + ~5 V for HG2) needed to drive the high-side FET. The low-side gate driver drives the low-side external FET with a nominal VGS of 5 V using energy sourced from CCG7D's internal LDO regulator and stored in the capacitor between PVDD and PGND. The low-side gate driver has programmable drive strength to drive external FET.

1.5.11 Error Amplifier

CCG7D's buck-boost controller contains two error amplifiers for output voltage and current regulation. The error amplifier is a trans-conductance type amplifier with single compensation pin (COMP) to ground for both the voltage and current loops. In voltage regulation, the output voltage is compared with the internal reference voltage and the output of EA is fed to the PWM block. In current regulation, the average current is sensed by VBUS high side current sense amplifier through the external resistor. The output of the VBUS CSA is compared with internal reference in error amplifier block and EA output is fed to the PWM block.

1.5.12 High-side Current Sense Amplifier for VIN

CCG7D features a high-side current sense amplifier (CSA) for peak current sensing through an external resistor of 5 mΩ placed in series with the buck control FET. The CSA is high bandwidth and a very wide common mode amplifier. This block implements slope compensation to avoid sub-harmonic oscillation for the internal current loop. In addition to peak current sensing, it provides a current limit comparator for shutting off the buck-boost converter if the current hits an upper threshold which is programmable.

1.5.13 Pulse Width Modulator

CCG7D device family's PWM block generates the control signals for the gate drivers driving the external FETs in peak current mode control. There are many programmable options for minimum/maximum pulse width, minimum/maximum period, frequency and pulse skip levels to optimize the system design.

1.5.14 High-side Current Sense Amplifier for VBUS

CCG7D device family supports VBUS current measurement and control using an external resistor (5 mΩ) in series with the VBUS path. The voltage drop across this resistor is used to measure the average output current. The same resistor is also used to sense and precisely control the output current in the PPS Current Foldback mode of operation.

1.5.15 VIN Regulator

CCG7D has a VIN regulator to derive the chip operating supply. This regulator features in-rush current detection and control functions. In-rush is detected when load current exceeds 300 mA.

1.5.16 20-V Tolerant SBU and CC Lines

CCG7D supports 20 V tolerant CC lines. For CC short to VBUS through connectors, these lines will be protected internally. The cable interface will be shut-off when 20 V or over-voltage is detected on SBU/CC lines.

1.6 System-Wide Resources

1.6.1 Clocking System

The clock system for the CCG7D controller consists of the internal main oscillator (IMO) and an internal low-speed oscillator (ILO) as internal clocks.

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the CCG7D. The default IMO frequency is 48 MHz. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a 32-kHz low-power, low accuracy oscillator and is used to generate clocks for peripheral operation in Deep Sleep power mode.

1.6.2 Power System

CCG7D operates from VIN power supply. The VIN input supports operation in the range 4 V to 24 V. CCG7D has three power modes - Active (default mode), Sleep, and Deep Sleep. The transitions between these modes are managed by the power system. A separate power domain VDDIO is provided for the GPIOs. The VCCD pin, the output of the core (1.8 V) regulator, is brought out only for connecting a 0.1-µF capacitor for stability. This pin is not supported as a power supply.

In Active mode, the CPU runs with all the peripherals and other subsystems powered. In Deep Sleep mode, the CPU, SRAM, and high-speed logic are in retention (states retained); the main system clock (IMO) is OFF while the low-frequency clock (ILO) is ON and the low-frequency peripherals are in operation. Multiple internal regulators are available in the system to support power supply in different power modes.

The VDDD supply is used to power Type-C cable markers through the VCONN switches and supports operation in the range 4.85 V to 5.5 V.

1.6.3 GPIO

CCG7D has 19 GPIOs that include dedicated GPIOs, I^2C, UART, and SWD pins which can also be used as GPIOs. Every GPIO in CCG7D has the following characteristics:

- Eight drive strength modes
- Input threshold select (CMOS or LVTTL)
- Individual control of input and output disables
- Selectable slew rates for dV/dt related noise control to improve EMI
- Hold mode to latch the previous state (used to retain I/O state in Deep Sleep mode)
- Interrupt generation - edge and level triggered

1.7 Fixed-Function Digital

1.7.1 Timer/Counter/PWM Block

The Timer/Counter/PWM block consists of two 16-bit counters with user-programmable period length. The functionality of these counters can be synchronized. Each block has a capture register, period register, and compare registers.

1.7.2 Serial Communication Block (SCB)

The CCG7D has four SCBs, which can each implement a serial communication interface as I^2C, universal asynchronous receiver/transmitter (UART), or serial peripheral interface (SPI). All the four SCBs are fully configurable to either one of the following I2C / SPI / UART.

The features of SCB include:

- Standard I2C multi-master and slave functionality compatible with the standard NXP Semiconductor I2C Specification V3.0
- Standard SPI master and slave function with Motorola, TI, and National Semiconductor (MicroWire) modes
- Standard UART transmitter and receiver function

1.8 Program and Debug

CCG7D devices support programming and debug features of the device via the on-chip SWD interface.

2. Getting Started

2.1 Support

Free support for CCG7D products is available online at www.cypress.com. Resources include application notes, CRM technical support email, knowledge base articles, and application support engineers.

For application assistance, visit www.cypress.com/support or call 1-800-541-4736.

2.2 Product Upgrades

Cypress provides scheduled upgrades and version enhancements for the CCGx Host Software Development Kit (SDK) which includes firmware and tools for the CCG7D family. Upgrades can be downloaded from the Software and Drivers section of www.cypress.com/CCG7D. Critical updates to system documentation are provided in the Documentation section.

3. Document Construction

This document has the following sections:

- Section B: CPU System on page 22
- Section C: Memory System on page 36
- Section D: System-Wide Resources on page 39
- Section E: Digital System on page 65
- Section F: USB Power Delivery on page 124
- Section G: Program and Debug on page 170

3.1 Major Sections

For ease of use, information is organized into sections and chapters that are divided according to device functionality.

- Section – Presents the top-level architecture, how to get started, and conventions and overview information about any particular area that inform the reader about the construction and organization of the product.
- Chapter – Presents the chapters specific to an individual aspect of the section topic. These provide the detailed information about the topic.
- Glossary – Defines the specialized terminology used in this technical reference manual (TRM). Glossary terms are presented in bold, italic font throughout.
- CCG7D Registers Technical Reference Manual – Supplies all device register details summarized in the technical reference manual. These are additional documents.

3.2 Documentation Conventions

This document uses only four distinguishing font types, besides those found in the headings.

- The first is the use of *italics* when referencing a document title or file name.
- The second is the use of ***bold italics*** when referencing a term described in the Glossary of this document.
- The third is the use of Times New Roman font, distinguishing equation examples.
- The fourth is the use of `Courier New` font, distinguishing code examples.

3.2.1 Register Conventions

Register conventions are detailed in the CCG7D Family EZ-PD CCG7D Registers TRM.

3.2.2 Numeric Naming

Hexadecimal numbers are represented with all letters in uppercase with an appended lowercase ‘h’ (for example, ‘14h’ or ‘3Ah’) and *hexadecimal* numbers may also be represented by a ‘0x’ prefix, the *C* coding convention. Binary numbers have an appended lowercase ‘b’ (for example, 01010100b’ or ‘01000011b’). Numbers not indicated by an ‘h’ or ‘b’ are *decimal*.

3.2.3 Units of Measure

This table lists the units of measure used in this document.

Table 3-1. Units of Measure

Symbol	Unit of Measure
°C	degrees Celsius
dB	decibels
fF	femtofarads
Hz	Hertz
k	kilo, 1000
K	kilo, 2^10
KB	1024 bytes, or approximately one thousand bytes
Kbit	1024 bits
kHz	kilohertz
kΩ	kilohms
MHz	megahertz
MΩ	megaohms
µA	microamperes
µF	microfarads
µs	microseconds
µV	microvolts
µVrms	microvolts root-mean-square
mA	milliamperes
ms	milliseconds
mV	millivolts
nA	nanoamperes
ns	nanoseconds
nV	nanovolts
Ω	ohms
pF	picofarads
pp	peak-to-peak
ppm	parts per million
SPS	samples per second
σ	sigma: one standard deviation
V	volts

3.2.4 Acronyms

This table lists the acronyms used in this document

Table 3-2. Acronyms

Acronym	Description
ABUS	analog output bus
AC	alternating current
ADC	analog-to-digital converter
AHB	AMBA (advanced microcontroller bus architecture) high-performance bus, an Arm data transfer bus
API	application programming interface
APOR	analog power-on reset
BMC	Bi-Phase Mark Coding
BR	bit rate
BRA	bus request acknowledge
BRQ	bus request
CAN	controller area network
CC	Configuration Channel
CCG7D	Cypress's sixth generation dual-port USB Type-C controller
CI	carry in
CMP	compare
CO	carry out
CPU	central processing unit
CRC	cyclic redundancy check
CT	continuous time
DAC	digital-to-analog converter
DC	direct current
DFP	Downstream Facing Port
DI	digital or data input
DMA	direct memory access
DNL	differential nonlinearity
DO	digital or data output
DRP	dual-role-power
DSI	digital signal interface
DSM	deep sleep mode
ECO	external crystal oscillator
EEPROM	electrically erasable programmable read only memory
EMCA	Electronically Marked Cable Assembly
EMIF	external memory interface
FB	feedback
FIFO	first in first out
FRS	Fast Role Swap
FSR	full scale range
GPIO	general purpose I/O

Table 3-2. Acronyms *(continued)*

Acronym	Description
HCI	host-controller interface
HFCLK	high-frequency clock
I^2C	inter-integrated circuit
IDE	integrated development environment
ILO	internal low-speed oscillator
IMO	internal main oscillator
INL	integral nonlinearity
I/O	input/output
IOR	I/O read
IOW	I/O write
IRES	initial power on reset
IRA	interrupt request acknowledge
IRQ	interrupt request
ISR	interrupt service routine
IVR	interrupt vector read
LRb	last received bit
LRB	last received byte
LSb	least significant bit
LSB	least significant byte
LUT	lookup table
MISO	master-in-slave-out
MMIO	memory mapped input/output
MOSI	master-out-slave-in
MSb	most significant bit
MSB	most significant byte
PC	program counter
PCH	program counter high
PCL	program counter low
PD	power down
PGA	programmable gain amplifier
PM	power management
POR	power-on reset
PPOR	precision power-on reset
PRS	pseudo random sequence
PSRR	power supply rejection ratio
PSSDC	power system sleep duty cycle
PWM	pulse width modulator
RAM	random-access memory
RETI	return from interrupt
RF	radio frequency
ROM	read only memory
RW	read/write
SAR	successive approximation register

Table 3-2. Acronyms *(continued)*

Acronym	Description
SC	switched capacitor
SCB	serial communication block
SIE	serial interface engine
SIO	special I/O
SE0	single-ended zero
SNR	signal-to-noise ratio
SOF	start of frame
SOI	start of instruction
SOP	Start of Packet
SP	stack pointer
SPD	sequential phase detector
SPI	serial peripheral interconnect
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random-access memory
SROM	supervisory read only memory
SSADC	single slope ADC
SSC	supervisory system call
SYSCLK	system clock
SWD	serial wire debug
TC	terminal count
TD	transaction descriptors
UART	universal asynchronous receiver/transmitter
UFP	Upstream Facing Port
USB	universal serial bus
USBIO	USB I/O
USB PD	USB Power Delivery
WCO	watch crystal oscillator
WDT	watchdog timer
WDR	watchdog reset
XRES	external reset
XRES_N	external reset, active low

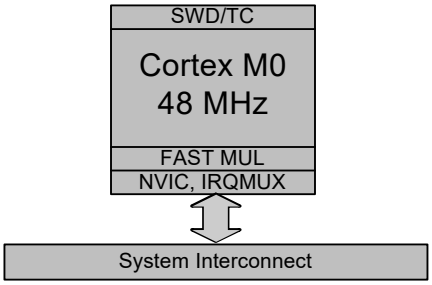
Section B: CPU System

This section encompasses the following chapters:

- Cortex-M0 CPU chapter on page 23
- Interrupts chapter on page 28

Top Level Architecture

CPU System Block Diagram



4. Cortex-M0 CPU

The CCG7D Arm Cortex-M0 core is a 32-bit CPU optimized for low-power operation. It has an efficient three-stage pipeline, a fixed 4 GB memory map, and supports the Armv6-M Thumb instruction set. The Cortex-M0 also features low-latency interrupt service routine (ISR) entry and exit. The Cortex-M0 processor includes a number of other components that are tightly linked to the CPU core. These include an NVIC, a SYSTICK timer, and debug.

This section gives an overview of the Cortex-M0 processor. For more details, see the Arm Cortex-M0 user guide or technical reference manual, both available at www.arm.com.

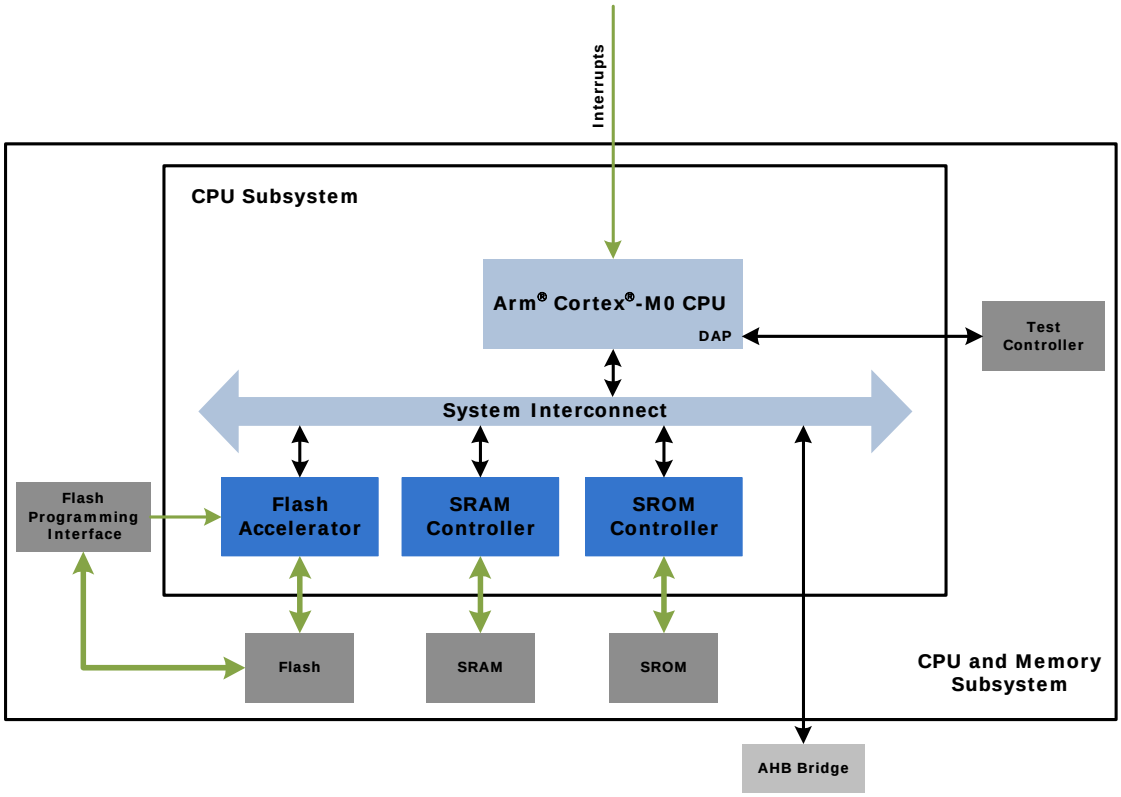
4.1 Features

The CCG7D Arm Cortex-M0 has the following features:

- Easy to use, program, and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.9 DMIPS/MHz; this helps to increase execution speed or reduce power
- Supports Thumb instruction set for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions for rapid and deterministic interrupt response
- Extensive debug support including:
  - Serial wire debug (SWD) port
  - Breakpoints
  - Watchpoints

4.2 Block Diagram

Figure 4-1. CCG7D CPU Subsystem Block Diagram



4.3 How It Works

The Cortex-M0 is a 32-bit processor with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. It supports most 16-bit instructions in the Thumb instruction set and some 32-bit instructions in the Thumb-2 instruction set.

4.3.1 Registers

The Cortex-M0 has 16 32-bit registers, as Table 4-1 shows:

- R0 to R12 – General-purpose registers. R0 to R7 can be accessed by all instructions; the other registers can be accessed by a subset of the instructions.
- R13 – Stack pointer (SP). There are two stack pointers, with only one available at a time. In thread mode[1], the CONTROL register indicates the stack pointer to use, main stack pointer (MSP) or process stack pointer (PSP).
- R14 – Link register. Stores the return program counter during function calls.
- R15 – Program counter. This register can be written to control program flow.

1. Thread mode is an Arm processor mode. It is used to execute the application software. The processor enters this mode when it comes out of reset. For details, see the Cortex-M0 Generic User Guide.

Table 4-1. Cortex-M0 Registers

Name	Type[a]	Reset Value	Description
R0-R12	RW	Unknown	R0-R12 are 32-bit general-purpose registers for data operations.
MSP	RW	[0x00000000]	The stack pointer (SP) is register R13. In Thread mode, bit[1] of the CONTROL register indicates the stack pointer to use: 0 = MSP. This is the reset value. 1 = PSP. On reset, the processor loads the MSP with the value from address 0x00000000.
PSP			
LR	RW	Unknown	The link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC	RW	[0x00000004]	The program counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value from address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.
PSR	RW	Unknown[b]	The program status register (PSR) combines: Application Program Status Register (APSR) Execution Program Status Register (EPSR). Interrupt Program Status Register (IPSR)
APSR	RW	Unknown	The APSR contains the current state of the condition flags from previous instruction executions.
EPSR	RO	Unknown[b]	The EPSR contains the Thumb state bit.
IPSR	RO	0	The IPSR contains the exception number of the current ISR.
PRIMASK	RW	0	The PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	The CONTROL register controls the stack used when the processor is in Thread mode.

a. Describes access type during program execution in thread mode and handler mode. Debug access can differ.
b. Bit[24] is the T bit and is loaded from bit[0] of the reset vector.

Table 4-2 shows how the PSR bits are assigned.

Table 4-2. Cortex-M0 PSR Bit Assignments

Bit	PSR Register	Name	Usage
31	APSR	N	Negative flag
30	APSR	Z	Zero flag
29	APSR	C	Carry or borrow flag
28	APSR	V	Overflow flag
27–25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.
23–6	–	–	Reserved
5–0	IPSR	N/A	Exception number of current ISR: 0 = thread mode 1 = reserved 2 = Non maskable interrupt (NMI) 3 = HardFault 4 – 10 = reserved 11 = SVCall 12, 13 = reserved 14 = PendSV 15 = SysTick 16 = IRQ0 … 47 = IRQ31

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset, are disabled. See the Interrupts chapter on page 28 for a list of exceptions.

4.3.2 Operating Modes

The Cortex-M0 processor supports two operating modes:

- Thread mode – used by all normal applications. In the Thread mode, the MSP or PSP can be used. The CONTROL register bit 1 determines which stack pointer is used:
  - 0 = MSP is the current stack pointer
  - 1 = PSP is the current stack pointer
- Handler mode – used to execute exception handlers. The MSP is always used.

In Thread mode, use the MSR instruction to set the stack pointer bit in the CONTROL register. When changing the stack pointer, use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB execute using the new stack pointer.

In Handler mode, explicit writes to the CONTROL register are ignored, because the MSP is always used. The exception entry and return mechanisms automatically update the CONTROL register.

4.3.3 Instruction Set

The Cortex-M0 implements a version of the Thumb instruction set. For details, see the Cortex-M0 Generic User Guide.

An instruction operand can be an Arm register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. Many instructions are unable to use, or have restrictions on using, the PC or SP for the operands or destination register.

Table 4-3. Thumb Instruction Set

Mnemonic	Brief Description
ADCS	Add with Carry
ADD{S}	Add
ADR	PC-relative Address to Register
ANDS	Bit wise AND
ASRS	Arithmetic Shift Right
B{cc}	Branch {conditionally}
BICS	Bit Clear
BKPT	Breakpoint
BL	Branch with Link
BLX	Branch indirect with Link
BX	Branch indirect
CMN	Compare Negative
CMP	Compare

Table 4-3. Thumb Instruction Set <Italic> (continued)

Mnemonic	Brief Description
CPSID	Change Processor State, Disable Interrupts
CPSIE	Change Processor State, Enable Interrupts
DMB	Data Memory Barrier
DSB	Data Synchronization Barrier
EORS	Exclusive OR
ISB	Instruction Synchronization Barrier
LDM	Load Multiple registers, increment after
LDR	Load Register from PC-relative address
LDRB	Load Register with word
LDRH	Load Register with half-word
LDRSB	Load Register with signed byte
LDRSH	Load Register with signed half-word
LSLS	Logical Shift Left
LSRS	Logical Shift Right
MOV{S}	Move
MRS	Move to general register from special register
MSR	Move to special register from general register
MULS	Multiply, 32-bit result
MVNS	Bit wise NOT
NOP	No Operation
ORRS	Logical OR
POP	Pop registers from stack
PUSH	Push registers onto stack
REV	Byte-Reverse word
REV16	Byte-Reverse packed half-words
REVSH	Byte-Reverse signed half-word
RORS	Rotate Right
RSBS	Reverse Subtract
SBCS	Subtract with Carry
SEV	Send Event
STM	Store Multiple registers, increment after
STR	Store Register as word
STRB	Store Register as byte
STRH	Store Register as half-word
SUB{S}	Subtract
SVC	Supervisor Call
SXTB	Sign extend byte
SXTH	Sign extend half-word
TST	Logical AND based test
UXTB	Zero extend a byte
UXTH	Zero extend a half-word
WFE	Wait For Event
WFI	Wait For Interrupt

4.3.3.1 Address Alignment

An aligned access is an operation where a word-aligned address is used for a word or multiple word access, or where a half-word-aligned address is used for a half-word access. Byte accesses are always aligned.

No support is provided for unaligned accesses on the Cortex-M0 processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

4.3.3.2 Memory Endianness

The CCG7D Arm Cortex-M0 uses little-endian format, where the least-significant byte of a word is stored at the lowest address and the most significant byte is stored at the highest address.

4.3.4 Systick Timer

The Systick timer is integrated with the NVIC and generates the SYSTICK interrupt. This interrupt can be used for task management in a real-time system. The timer has a reload register with 24 bits available to use as a countdown value. The Systick timer uses the Cortex-M0 internal clock as a source and can work in Sleep mode.

4.3.5 Debug

CCG7D Arm Cortex-M0 contains a debug interface based on SWD; it features four breakpoint (address) comparators and two watchpoint (data) comparators.

5. Interrupts

The Arm Cortex-M0 (CM0) CPU in CCG7D supports interrupts and exceptions. Interrupts refer to those events generated by peripherals external to the CPU such as timers, serial communication block, and port pin signals. Exceptions refer to those events that are generated by the CPU such as memory access faults and internal system timer events. Both interrupts and exceptions result in the current program flow being stopped and the exception handler or ISR being executed by the CPU. CCG7D provides a unified exception vector table for both interrupt handlers/ISR and exception handlers.

5.1 Features

CCG7D supports the following interrupt features:

- Supports 20 interrupts
- NVIC integrated with CPU core, yielding low interrupt latency
- Vector table may be placed in either flash or SRAM
- Configurable priority levels from 0 to 3 for each interrupt
- Level-triggered and pulse-triggered interrupt signals

5.2 How It Works

Figure 5-1. Interrupts Block Diagram

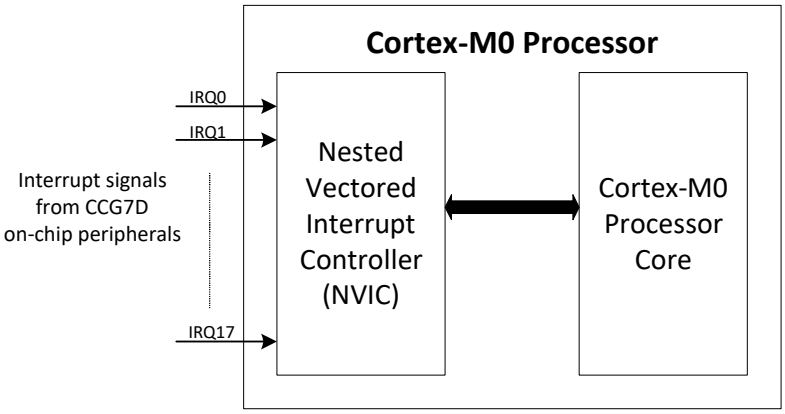


Figure 5-1 shows the interaction between interrupt signals and the Cortex-M0 CPU. CCG7D supports 20 interrupts; these interrupt signals are processed by the NVIC. The NVIC takes care of enabling/disabling individual interrupts, priority resolution, and communication with the CPU core. The exceptions are not shown in Figure 5-1 because they are part of CM0 core generated events, unlike interrupts, which are generated by peripherals external to the CPU.

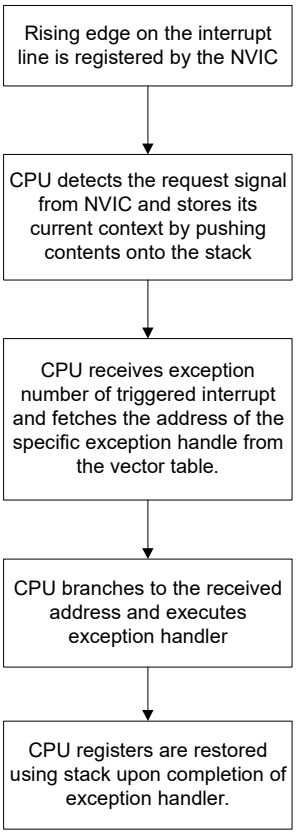
5.3 Interrupts and Exceptions – Operation

5.3.1 Interrupt and Exception Handling

The sequence of events that occur when an interrupt or exception event is triggered is:

1. Assuming that all the interrupt signals are initially low (idle or inactive state) and the processor is executing the main code, a rising edge on any one of the interrupt lines is registered by the NVIC. The interrupt line is now in a pending state waiting to be serviced by the CPU.
2. On detecting the interrupt request signal from the NVIC, the CPU stores its current context by pushing the contents of the CPU registers onto the stack.
3. The CPU also receives the exception number of the triggered interrupt from the NVIC. All interrupts and exceptions in CCG7D have a unique exception number, as given in Table 5-1. By using this exception number, the CPU fetches the address of the specific exception handler from the vector table.
4. The CPU then branches to this address and executes the exception handler that follows.
5. Upon completion of the exception handler, the CPU registers are restored to their original state using stack pop operations; the CPU resumes the main code execution.

Figure 5-2. Interrupt Handling When Triggered



When the NVIC receives an interrupt request while another interrupt is being serviced or receives multiple interrupt requests at the same time, it evaluates the priority of all these interrupts, sending the exception number of the highest priority interrupt to the CPU. Thus, a higher priority interrupt can block the execution of a lower priority ISR at any time.

Exceptions are handled in the same way that interrupts are handled. Each exception event has a unique exception number, which is used by the CPU to execute the appropriate exception handler.

5.3.2 Level and Pulse Interrupts

CCG7D NVIC supports both level and pulse signals on the interrupt lines (IRQ0 to IRQ19). The classification of an interrupt as level or pulse is based on the interrupt source.

Figure 5-3. Level Interrupts

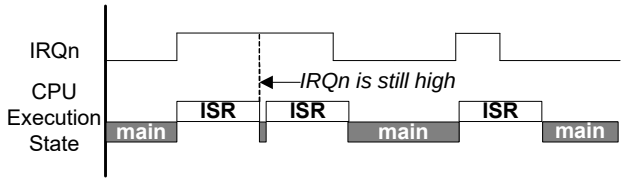


Figure 5-4. Pulse Interrupts

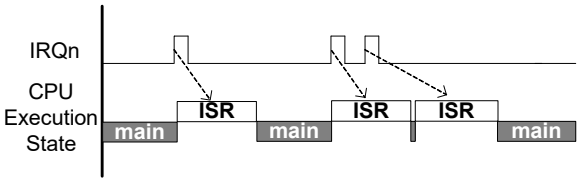


Figure 5-3 and Figure 5-4 show the working of level and pulse interrupts, respectively. Assuming the interrupt signal is initially inactive (logic low), the following sequence of events explains the handling of level and pulse interrupts:

1. On a rising edge event of the interrupt signal, the NVIC registers the interrupt request. The interrupt is now in the pending state, which means the interrupt requests have not yet been serviced by the CPU.
2. The NVIC then sends the exception number along with the interrupt request signal to the CPU. When the CPU starts executing the ISR, the pending state of the interrupt is cleared.
3. When the ISR is being executed by the CPU, one or more rising edges of the interrupt signal are logged as a single pending request. The pending interrupt is serviced again after the current ISR execution is complete (see Figure 5-4 for pulse interrupts).
4. If the interrupt signal is still high after completing the ISR, it will be pending and the ISR is executed again. Figure 5-3 illustrates this for level-triggered interrupts, where the ISR is executed as long as the interrupt signal is high.

5.3.3 Exception Vector Table

The exception vector table (see Table 5-1), stores the entry point addresses for all exception handlers in CCG7D. The CPU fetches the appropriate address based on the exception number.

Table 5-1. CCG7D Exception Vector Table

Exception Number	Exception	Exception Priority	Vector Address
–	Initial Stack Pointer Value	Not applicable (NA)	Base_Address – Can be 0x00000000 (start of flash memory) or 0x20000000 (start of SRAM)[a]
1	Reset	–3, the highest priority	Base_Address + 0x04
2	Non Maskable Interrupt (NMI)	–2	Base_Address + 0x08
3	Hard-Fault	–1	Base_Address + 0x0C
4–10	Reserved	NA	Base_Address + 0x10 to Base_Address + 0x28
11	Supervisory Call (SVCall)	Configurable (0–3)	Base_Address + 0x2C
12–13	Reserved	NA	Base_Address + 0x30 to Base_Address + 0x34
14	PendSupervisory (PendSV)	Configurable (0–3)	Base_Address + 0x38
15	System Timer (SysTick)	Configurable (0–3)	Base_Address + 0x3C
16	External Interrupt (IRQ0)	Configurable (0–3)	Base_Address + 0x40
…	…	Configurable (0–3)	…
35	External Interrupt (IRQ19)	Configurable (0–3)	Base_Address + 0x8C8

a. Note that the reset exception address in SRAM vector table will never be used because the device comes out of reset with the flash vector table selected.

In Table 5-1, the first word (4 bytes) is not marked as exception number zero. This is because the first word in the exception table is used to initialize the main stack pointer (MSP) value on device reset; it is not considered as an exception. In CCG7D, the vector table can be configured to be located either in flash memory (base address of 0x00000000) or SRAM (base address of 0x20000000). This configuration is done by writing to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. When the VECT_IN_RAM bit field is '1', CPU fetches exception handler addresses from the SRAM vector table location. When this bit field is '0' (reset state), the vector table in flash memory is used for exception address fetches. You must set the VECT_IN_RAM bit field as part of the device boot code to configure the vector table to be in SRAM. The advantage of moving the vector table to SRAM is that the exception handler addresses can be dynamically changed by modifying the SRAM vector table contents. However, the nonvolatile flash memory vector table must be modified by a flash memory write.

The exception sources (exception numbers 1 to 15) are explained in 5.4 Exception Sources. The exceptions marked as Reserved in Table 5-1 are not used in CCG7D, though they have addresses reserved for them in the vector table. The interrupt sources (exception numbers 16 to 36) are explained in 5.5 Interrupt Sources.

5.4 Exception Sources

This section describes the different exception sources listed in Table 5-1 (exception numbers 1 to 15).

5.4.1 Reset Exception

Device reset is treated as an exception in CCG7D. It is always enabled with a fixed priority of –3, the highest priority exception. A device reset can occur due to multiple reasons, such as power-on-reset (POR), external reset signal on XRES pin, or watchdog reset. When the device is reset, the initial boot code for configuring the device is executed out of the SROM. The boot code and other data in SROM memory are programmed by Cypress, and are not read/write accessible to external users. After completing the SROM boot sequence, the CPU code execution jumps to flash memory. Flash memory address 0x00000004 (Exception#1 in Table 5-1) stores the location of the startup code in flash memory. The CPU starts executing code out of this address. Note that the reset exception address in SRAM vector table will never be used because the device comes out of reset with the flash vector table selected. The register configuration to select the SRAM vector table can be done only as part of the startup code in flash after the reset is de-asserted.

5.4.2 Non-Maskable Interrupt (NMI) Exception

Non-maskable interrupt (NMI) is the highest priority exception other than reset. It is always enabled with a fixed priority of –2. There are two ways to trigger an NMI exception in CCG7D:

- **NMI exception by setting NMIPENDSET bit (user NMI exception):** NMI exception can be triggered in software by setting the NMIPENDSET bit in the interrupt control state register (CM0_ICSR register). Setting this bit will execute the NMI handler pointed to by the active vector table (flash or SRAM vector table).
- **System Call NMI exception:** This exception is used for nonvolatile programming operations in CCG7D such as flash write operation and flash checksum operation. It is triggered by setting the SYSCALL_REQ bit in the CPUSS_SYSREQ register. An NMI exception triggered by SYSCALL_REQ bit always executes the NMI exception handler code that resides in SROM. Flash or SRAM exception vector table is not used for system call NMI exception. The NMI handler code in SROM is not read/write accessible because it contains nonvolatile programming routines that should not be modified by the user.

5.4.3 HardFault Exception

HardFault is an always-enabled exception that occurs because of an error during normal or exception processing. HardFault has a fixed priority of –1; this means it has higher priority than any exception with configurable priority. HardFault exception is a catch-all exception for different types of fault conditions, which include executing an undefined instruction and accessing an invalid memory addresses. The CM0 CPU does not provide fault status information to the HardFault exception handler, but it does permit the handler to perform an exception return and continue execution in cases where software has the ability to recover from the fault situation.

5.4.4 Supervisor Call (SVCall) Exception

Supervisor Call (SVCall) is an always-enabled exception caused when the CPU executes the SVC instruction as part of the application code. Application software uses the SVC instruction to make a call to an underlying operating system and provide a service. This is known as a supervisor call. The SVC instruction enables the application to issue a supervisor call that requires privileged access to the system. Note that the CM0 in CCG7D uses a privileged mode for the system call NMI exception, which is not related to the SVCall exception. (See the Chip Operational Modes chapter on page 54 for details on privileged mode.) There is no other privileged mode support for SVCall at the architecture level in CCG7D. The application developer must define the SVCall exception handler according to the end application requirements.

The priority of a SVCall exception can be configured to a value between 0 and 3 by writing to the two-bit field PRI_11[31:30] of the System Handler Priority Register 2 (CM0_SHPR2). When the SVC instruction is executed, the SVCall exception enters the pending state and waits to be serviced by the CPU. The SVCALLPENDED bit in the System Handler Control and State Register (CM0_SHCSR) can be used to check or modify the pending status of the SVCall exception.

5.4.5 PendSV Exception

PendSV is another supervisor call related exception similar to SVCall, normally being software-generated. PendSV is always enabled and its priority is configurable. The PendSV exception is triggered by setting the PENDSVSET bit in the Interrupt Control State Register, CM0_ICSR. On setting this bit, the PendSV exception enters the pending state, and waits to be serviced by the CPU. The pending state of a PendSV exception can be cleared by setting the PENDSVCLR bit in the Interrupt Control State Register, CM0_ICSR. The priority of a PendSV exception can be configured to a value between 0 and 3 by writing to the two-bit field PRI_14[23:22] of the System Handler Priority Register 3 (CM0_SHPR3). See the Armv6-M Architecture Reference Manual for more details.

5.4.6 SysTick Exception

CM0 CPU in CCG7D supports a system timer, referred to as SysTick, as part of its internal architecture. SysTick provides a simple, 24-bit decrementing counter for various time keeping purposes such as an RTOS tick timer, high-speed alarm timer, or simple counter. The SysTick timer can be configured to generate an interrupt when its count value reaches zero, which is referred to as SysTick Exception. The exception is enabled by setting the TICKINT bit in the SysTick Control and Status Register (CM0_SYST_CSR). The priority of a SysTick exception can be configured to a value between 0 and 3 by writing to the two-bit field PRI_15[31:30] of the System Handler Priority Register 3 (CM0_SHPR3). The SysTick exception can always be generated in software at any instant by writing a ‘1’ to the PENDSTSETb bit in the Interrupt Control State Register, CM0_ICSR. Similarly, the pending state of the SysTick exception can be cleared by writing a ‘1’ to the PENDSTCLR bit in the Interrupt Control State Register, CM0_ICSR.

5.5 Interrupt Sources

CCG7D supports 19 interrupts (IRQ0–IRQ18, or exception numbers 16–34) from peripherals. Table 5-2 lists the source of each interrupt. CCG7D provides flexible sourcing options for each of the 20 interrupt lines. The interrupts include standard interrupts from the on-chip peripherals such as TCPWM, serial communication block, and interrupts from GPIO ports. The interrupt generated is usually the logical OR of the different peripheral states. The peripheral status register should be read in the ISR to detect which condition generated the interrupt. These interrupts are usually level interrupts, which require that the peripheral status register be read in the ISR to clear the interrupt. If the status register is not read in the ISR, the interrupt will remain asserted and the ISR will be executed continuously.

See the I/O System chapter on page 40 for details on GPIO interrupts.

Table 5-2. List of CCG7D Interrupt Sources

Interrupt No.	Cortex-M0 Exception No.	Interrupt Source
NMI	2	
IRQ0	16	GPIO Interrupt- Port 0
IRQ1	17	GPIO Interrupt- Port 1
IRQ2	18	GPIO Interrupt- Port 2
IRQ3	19	GPIO Interrupt- Port 3
IRQ4	20	GPIO All Ports
IRQ5	21	Ganged USBPD[0] Interrupt
IRQ6	22	Ganged USBPD[1] Interrupt
IRQ7	23	WDT or Temp (WDT only in Deep Sleep)
IRQ8	24	SCB0 (Serial Communication Block 0)
IRQ9	25	SCB1 (Serial Communication Block 1)
IRQ10	26	SCB2 (Serial Communication Block 2)
IRQ11	27	SCB3 (Serial Communication Block 3)
IRQ12	28	SPC (System Performance Controller)
IRQ13	29	Synchronous USBPD[0] Port Interrupts
IRQ14	30	Synchronous USBPD[1] Port Interrupts
IRQ15	31	TCPWM0 (Timer/Counter/PWM 0)
IRQ16	32	TCPWM1 (Timer/Counter/PWM 1)
IRQ17	33	TCPWM2 (Timer/Counter/PWM 2)
IRQ18	34	TCPWM3 (Timer/Counter/PWM 3)
IRQ19	35	Crypto (TRNG) block

5.6 Exception Priority

Exception priority is useful for exception arbitration when there are multiple exceptions that need to be serviced by the CPU. CCG7D provides flexibility in choosing priority values for different exceptions. All exceptions except Reset, NMI, and HardFault can be assigned a configurable priority level. The Reset, NMI, and HardFault exceptions have a fixed priority of –3, –2, and –1 respectively. In CCG7D, lower priority numbers represent higher priorities. This means that the Reset, NMI, and HardFault exceptions have the highest priorities. The other exceptions can be assigned a configurable priority level between 0 and 3.

CCG7D supports nested exceptions in which a higher priority exception can obstruct (interrupt) the currently active exception handler. This pre-emption does not happen if the incoming exception priority is the same as active exception. The CPU resumes execution of the lower priority exception handler after servicing the higher priority exception. The CM0 CPU in CCG7D allows nesting of up to four exceptions. When the CPU receives two or more exceptions requests of the same priority, the lowest exception number is serviced first.

The registers to configure the priority of exception numbers 1 to 15 are explained in Exception Sources on page 30.

The priority of the 20 interrupts (IRQ0–IRQ19) can be configured by writing to the Interrupt Priority registers (CM0_IPR). This is a group of five 32-bit registers with each register storing the priority values of four interrupts, as given in Table 5-3. The other bit fields in the register are not used.

Table 5-3. Interrupt Priority Register Bit Definitions

Bits	Name	Description
7:6	PRI_N0	Priority of interrupt number N.
15:14	PRI_N1	Priority of interrupt number N+1.
23:22	PRI_N2	Priority of interrupt number N+2.
31:30	PRI_N3	Priority of interrupt number N+3.

5.7 Enabling/Disabling Interrupts

The NVIC provides registers to individually enable and disable the 20 interrupts in software. If an interrupt is not enabled, the NVIC will not process the interrupt requests on that interrupt line. The Interrupt Set-Enable Register (CM0_ISER) and the Interrupt Clear-Enable Register (CM0_ICER) are used to enable and disable the interrupts respectively. These registers are 32-bit wide and each bit corresponds to the same numbered interrupt. These registers can also be read in software to get the enable status of the interrupts. Table 5-4 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 5-4. Interrupt Enable/Disable Registers

Register	Operation	Bit Value	Comment
Interrupt Set Enable Register (CM0_ISER)	Write	1	To enable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled
Interrupt Clear Enable Register (CM0_ICER)	Write	1	To disable the interrupt
		0	No effect
	Read	1	Interrupt is enabled
		0	Interrupt is disabled

The CM0_ISER and CM0_ICER registers are applicable only for the interrupts (IRQ0–IRQ18). These registers cannot be used to enable or disable the exception numbers 1 to 15. The 15 exceptions have their own support for enabling and disabling, as explained in Exception Sources on page 30.

The PRIMASK register in Cortex-M0 (CM0) CPU can be used as a global exception enable register to mask all the configurable priority exceptions irrespective of whether they are enabled. Configurable priority exceptions include all the exceptions except Reset, NMI, and HardFault listed in Table 5-1. They can be configured to a priority level between 0 and 3, 0 being the highest priority and 3 being the lowest priority. When the PM bit (bit 0) in PRIMASK register is set, none of the configurable priority exceptions can be serviced by the CPU, though they can be in the pending state waiting to be serviced by the CPU after the PM bit is cleared.

5.8 Exception States

Each exception can be in one of the following states.

Table 5-5. Exception States

Exception State	Meaning
Inactive	The exception is not active and not pending. Either the exception is disabled or the enabled exception has not triggered.
Pending	The exception request is received by the CPU/ NVIC and the exception is waiting to be serviced by the CPU.
Active	An exception that is being serviced by the CPU but whose exception handler execution is not yet complete. A high-priority exception can interrupt the execution of lower priority exception. In this case, both the exceptions are in the active state.
Active and Pending	The exception is being serviced by the processor and there is a pending request from the same source during its exception handler execution.

The Interrupt Control State Register (CM0_ICSR) contains status bits describing the various exceptions states.

- The VECTACTIVE bits ([8:0]) in the CM0_ICSR store the exception number for the current executing exception. This value is zero if the CPU is not executing any exception handler (CPU is in thread mode). Note that the value in VECTACTIVE bit fields is the same as the value in bits [8:0] of the Interrupt Control State Register (ICSR), which is also used to store the active exception number.
- The VECTPENDING bits ([20:12]) in the CM0_ICSR store the exception number of the highest priority pending exception. This value is zero if there are no pending exceptions.
- The ISRPENDING bit (bit 22) in the CM0_ICSR indicates if a NVIC generated interrupt (IRQ0 to IRQ18) is in a pending state.

5.8.1 Pending Exceptions

When a peripheral generates an interrupt request signal to the NVIC or an exception event occurs, the corresponding exception enters the pending state. When the CPU starts executing the corresponding exception handler routine, the exception is changed from the pending state to the active state.

The NVIC allows software pending of the 19 interrupt lines by providing separate register bits for setting and clearing the pending states of the interrupts. The Interrupt Set-Pending Register (CM0_ISPR) and the Interrupt Clear-Pending Register (CM0_ICPR) are used to set and clear the pending status of the interrupt lines. These registers are 32 bits wide, and each bit corresponds to the same numbered interrupt. Table 5-6 shows the register access properties for these two registers. Note that writing zero to these registers has no effect.

Table 5-6. Interrupt Set Pending/Clear Pending Registers

Register	Operation	Bit Value	Comment
Interrupt Set-Pending Register (CM0_ISPR)	Write	1	To put an interrupt to pending state
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending
Interrupt Clear-Pending Register (CM0_ICPR)	Write	1	To clear a pending interrupt
		0	No effect
	Read	1	Interrupt is pending
		0	Interrupt is not pending

Setting the pending bit when the same bit is already set results in only one execution of the ISR. The pending bit can be updated regardless of whether the corresponding interrupt is enabled. If the interrupt is not enabled, the interrupt line will not move to the pending state until it is enabled by writing to the CM0_ISER register.

Note that the CM0_ISPR and CM0_ICPR registers are used only for the 20 peripheral interrupts (exception numbers 16–35). These registers cannot be used for pending the exception numbers 1 to 15. These 15 exceptions have their own support for pending, as explained in Exception Sources on page 30.

5.9 Stack Usage for Exceptions

When the CPU executes the main code (in thread mode) and an exception request occurs, the CPU stores the state of its general-purpose registers in the stack. It then starts executing the corresponding exception handler (in handler mode). The CPU pushes the contents of the eight 32-bit internal registers into the stack. These registers are the Program and Status Register (PSR), ReturnAddress, Link Register (LR or R14), R12, R3, R2, R1, and R0. Cortex-M0 has two stack pointers – MSP and PSP. Only one of the stack pointers can be active at a time. When in thread mode, the Active Stack Pointer bit in the Control register is used to define the current active stack pointer. When in handler mode, the MSP is always used as the stack pointer. The stack pointer in Cortex-M0 always grows downwards and points to the address that has the last pushed data.

When the CPU is in thread mode and an exception request comes, the CPU uses the stack pointer defined in the control register to store the general-purpose register contents. After the stack push operations, the CPU enters handler mode to execute the exception handler. When another higher priority exception occurs while executing the current exception, the MSP is used for stack push/pop operations, because the CPU is already in handler mode. See the Cortex-M0 CPU chapter on page 23 for details.

The Cortex-M0 uses two techniques, tail chaining and late arrival, to reduce latency in servicing exceptions. These techniques are not visible to the external user and are done as part of the internal processor architecture (infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0419c/index.html).

5.10 Interrupts and Low-Power Modes

CCG7D allows device wakeup from low-power modes when certain peripheral interrupt requests are generated. The Wakeup Interrupt Controller (WIC) block generates a wakeup signal that causes the device to enter Active mode when one or more wakeup sources generate an interrupt signal. After entering Active mode, the ISR of the peripheral interrupt is executed.

The Wait For Interrupt (WFI) instruction, executed by the CM0 CPU, triggers the transition into Sleep and Deep Sleep modes. The sequence of entering the different low-power modes is detailed in the Power Modes chapter on page 55. The device low-power modes have two categories of fixed-function interrupt sources:

- Fixed-function interrupt sources that are available in the Active and Deep Sleep modes (watchdog timer interrupt, I^2C interrupts, USB PD wakeup and GPIO interrupts)
- Fixed-function interrupt sources that are available only in the Active mode (all other fixed-function interrupts)

5.11 Exception – Initialization and Configuration

This section covers the steps involved in initializing and configuring exceptions in CCG7D.

1. Configuring the Exception Vector Table Location: The first step in using exceptions is to configure the vector table location as required – either in flash memory or SRAM. This configuration is done by writing either a ‘1’ (SRAM vector table) or ‘0’ (flash vector table) to the VECT_IN_RAM bit field (bit 0) in the CPUSS_CONFIG register. This register write is done as part of the device initialization code.

   It is recommended that the vector table be available in SRAM if the application needs to change the vector addresses dynamically. If the table is located in flash, then a flash write operation is required to modify the vector table contents.
2. Configuring Individual Exceptions: The next step is to configure individual exceptions required in an application.

a. Configure the exception or interrupt source; this includes setting up the interrupt generation conditions. The register configuration depends on the specific exception required.
b. Define the exception handler function and write the address of the function to the exception vector table. Table 5-1 gives the exception vector table format; the exception handler address should be written to the appropriate exception number entry in the table.
c. Set up the exception priority, as explained in Exception Priority on page 32.
d. Enable the exception, as explained in Enabling/Disabling Interrupts on page 33.

5.12 Registers

Table 5-7. List of Registers

Register Name	Description
CM0_ISER	Interrupt Set-Enable Register
CM0_ICER	Interrupt Clear Enable Register
CM0_ISPR	Interrupt Set-Pending Register
CM0_ICPR	Interrupt Clear-Pending Register
CM0_IPR	Interrupt Priority Registers
CM0_ICSR	Interrupt Control State Register
CM0_AIRCR	Application Interrupt and Reset Control Register
CM0_SCR	System Control Register
CM0_CCR	Configuration and Control Register
CM0_SHPR2	System Handler Priority Register 2
CM0_SHPR3	System Handler Priority Register 3
CM0_SHCSR	System Handler Control and State Register
CM0_SYST_CSR	Systick Control and Status
CPUSS_CONFIG	CPU Subsystem Configuration
CPUSS_SYSREQ	System Request Register

5.13 Associated Documents

- Armv6-M Architecture Reference Manual – This document describes the Arm Cortex-M0 architecture, including the instruction set, NVIC architecture, and CPU register descriptions.

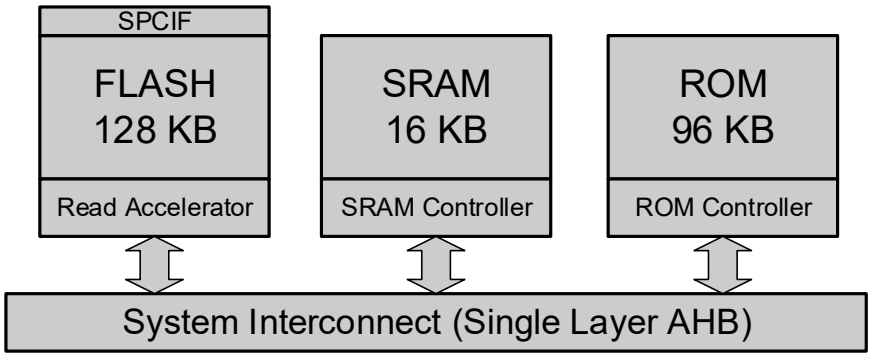
Section C: Memory System

This section presents the following chapter:

- Memory Map chapter on page 37

Top Level Architecture

Memory System Block Diagram



6. Memory Map

All CCG7D memory (flash, SRAM, and SROM) and all registers are accessible by the CPU and in most cases by the debug system. This chapter contains an overall map of the addresses of the memories and registers.

6.1 Features

The CCG7D memory system has the following features:

- 128K bytes flash, 16K bytes SRAM
- 8K byte boot SROM containing boot and configuration routines
- 24K byte user SROM containing commonly used firmware routines
- Arm Cortex-M0 32-bit linear address space, with regions for code, SRAM, peripherals, and CPU internal registers
- Flash is mapped to the Cortex-M0 code region
- SRAM is mapped to the Cortex-M0 SRAM region
- Peripheral registers are mapped to the Cortex-M0 peripheral region
- The Cortex-M0 Private Peripheral Bus (PPB) region includes registers implemented in the CPU core. These include registers for NVIC, SysTick timer, and SCB. For more information, see the Cortex-M0 CPU chapter on page 23.

6.2 How It Works

The CCG7D memory map is detailed in the following tables. For additional information, refer to the CCG7D Family EZ-PD CCG7D Registers TRM.

The Arm Cortex-M0 has a fixed address map allowing access to memory and peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in Table 6-1. Note that code can be executed from the code and SRAM regions.

Table 6-1. Cortex-M0 Address Map

Address Range	Name	Use
0x00000000–0x1FFFFFFF	Code	Executable region for program code. You can also put data here. Includes the exception vector table, which starts at address 0.
0x20000000–0x3FFFFFFF	SRAM	Executable region for data. You can also put code here.
0x40000000–0x5FFFFFFF	Peripheral	All peripheral registers. Code cannot be executed out of this region.
0x60000000–0xDFFFFFFF	–	Not used
0xE0000000–0xE00FFFFF	PPB	Peripheral registers within the CPU core.
0xE0100000–0xFFFFFFFF	Device	CCG7D implementation-specific.

Table 6-2 shows the CCG7D address map.

Table 6-2. CCG7D Address Map

Address Range	Use
0x00000000–0x0000FFFF	128 KB flash
0x0FFFF000–0x0FFFF3FF	2 KB supervisory flash
0x10000000-0x10001FFF	8 KB Boot SROM (no direct read/execute access)
0x10002000-0x10017FFF	24 KB User SROM (read/execute access allowed)
0x20000000–0x20003FFF	16 KB SRAM
0x40010000–0x4001FFFF	Peripheral Interconnect (PERI) registers
0x40020000–0x40023FFF	I/O port control (high-speed I/O matrix) registers
0x40030000–0x4003FFFF	Power, clock, reset control registers
0x40040000–0x40043FFF	I/O port registers
0x40050000–0x4008FFFF	SCB registers
0x40090000–0x4009FFFF	TCPWM registers
0x400A0000–0x400BFFFF	USB Power Delivery Controller registers
0x400C0000-0x400CFFFF	Crypto (TRNG) component registers
0x40100000–0x4011FFFF	CPU subsystem registers
0xE0000000–0xE00FFFFF	Cortex-M0 PPB registers
0xF0000000–0xF0000FFF	CoreSight ROM

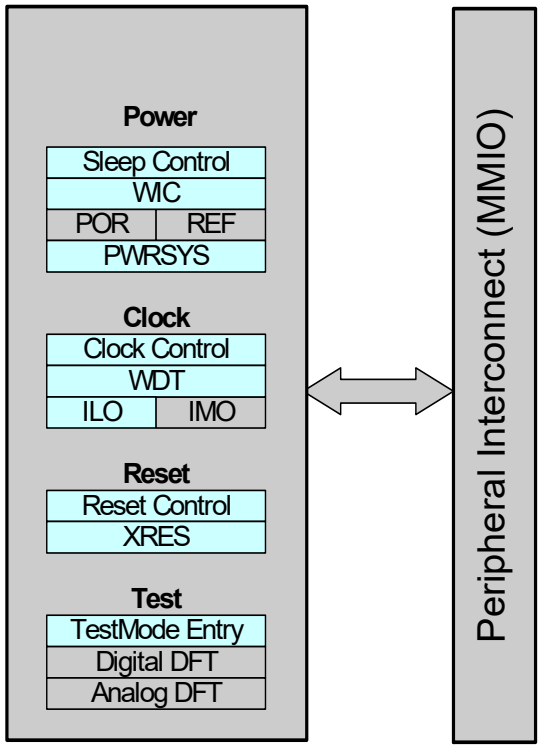
Section D: System-Wide

This section encompasses the following chapters:

- I/O System chapter on page 40
- Clocking System chapter on page 47
- Power Supply and Monitoring chapter on page 52
- Chip Operational Modes chapter on page 54
- Power Modes chapter on page 55
- Watchdog Timer chapter on page 59
- Reset System chapter on page 62
- Device Security chapter on page 64

Top Level Architecture

System-Wide Resources Block Diagram



7. I/O System

This chapter explains the CCG7D I/O system, its features, architecture, operating modes, and interrupts. The general-purpose I/O (GPIOs) pins in CCG7D are grouped into ports; a port can have a maximum of eight GPIOs. The CCG7D device has a maximum of 19 GPIOs arranged in 4 ports.

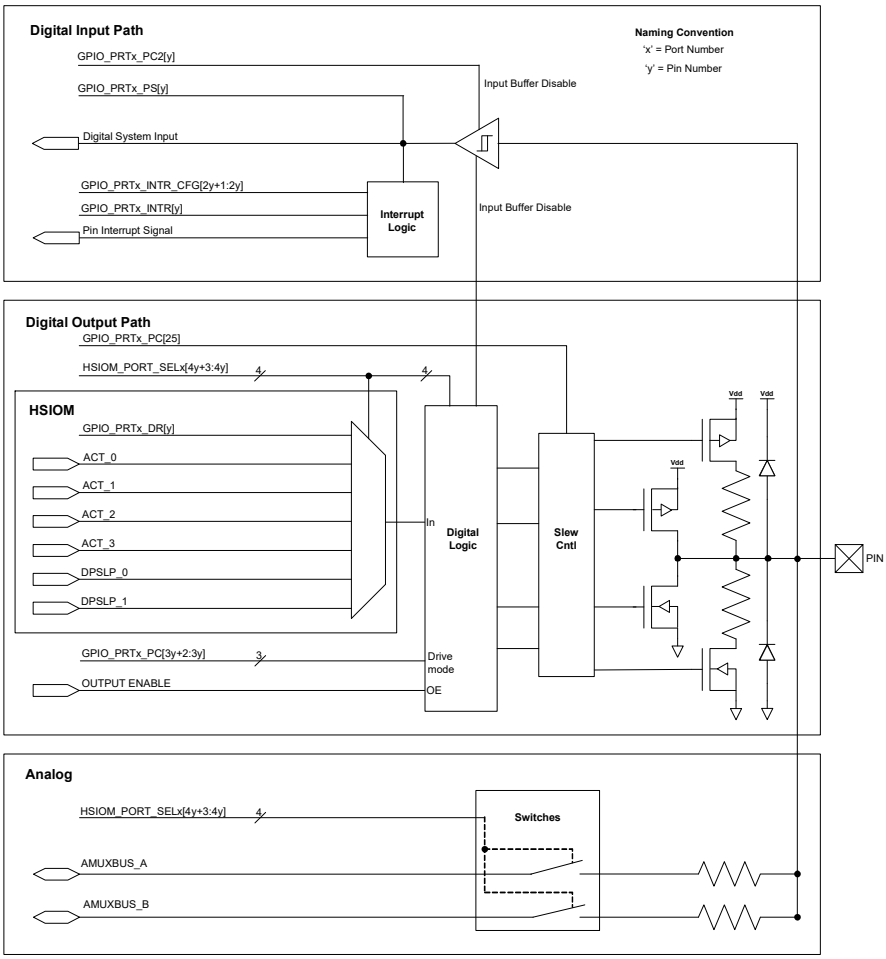
7.1 Features

The CCG7D GPIOs have these features:

- Analog and digital input and output capabilities
- 10-mA sink and 4-mA source current in digital mode
- Separate port read (PS) and write (DR) data registers to avoid read-modify-write errors
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on pin basis
- Slew rate control
- Selectable CMOS and low-voltage LVTTL input buffer mode
- Two over-voltage tolerant GPIOs (I^2C pins from SCB0)

7.2 Block Diagram

Figure 7-1. GPIO Block Diagram



Note The GPIO features shown in this image may not be available on all the pins.

7.3 GPIO Drive Modes

Each I/O is individually configurable into one of the eight drive modes listed in Table 7-1. Figure 7-2 is a simplified pin diagram that shows the pin view based on each of the eight drive modes.

Two port configuration registers are used to configure GPIOs in CCG7D:

Port Configuration Register (GPIO_PRTx_PC) and Port Secondary Configuration Register (GPIO_PRTx_PC2). All CCG7D devices have ports with dedicated GPIO_PRTx_PC and GPIO_PRTx_PC2 registers.

GPIO_PRTx_PC is used to configure the following properties of a port:

- Output drive mode of each pin (three bits select a particular drive mode for a pin)
- Slew rate of the whole port (see Slew Rate Control on page 44)
- Input threshold selection of the whole port (see CMOS LVTTL Level Control on page 44)

GPIO_PRTx_PC2 is used to enable/disable the input buffer of each pin on the port, irrespective of the drive mode configured in GPIO_PRTx_PC. When analog signals are present on the pin, input buffer should be disabled by setting the bit to '1'.

Figure 7-2. I/O Drive Mode Block Diagram

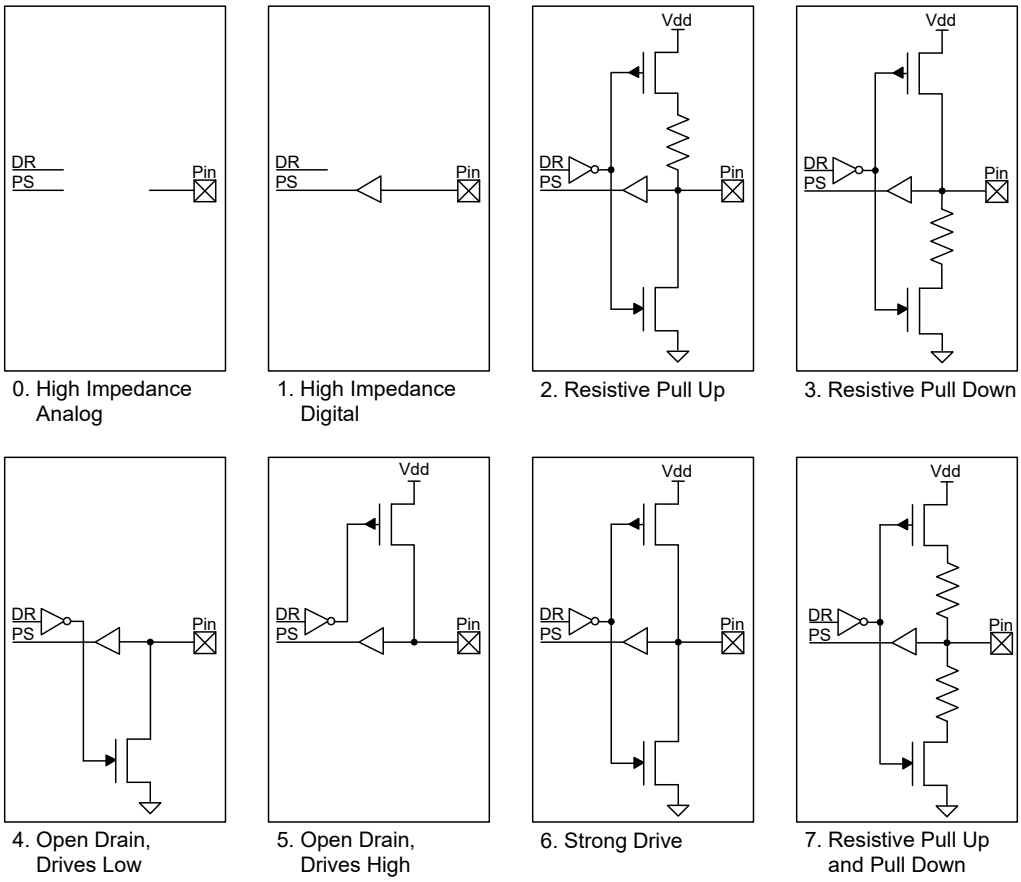


Table 7-1. Drive Mode Settings

GPIO_PRTx_PC ('x' denotes port number and 'y' denotes pin number)				
Bits	**Drive Mode**	**Value**	**Data = 1**	**Data = 0**
3y+2: 3y	SEL'y'	Selects Drive Mode for Pin 'y' ($0 \le y \le 7$)		
	High-Impedance Analog	0	High Z	High Z
	High-impedance Digital	1	High Z	High Z
	Resistive Pull Up	2	Weak 1	Strong 0
	Resistive Pull Down	3	Strong 1	Weak 0
	Open Drain, Drives Low	4	High Z	Strong 0
	Open Drain, Drives High	5	Strong 1	High Z
	Strong Drive	6	Strong 1	Strong 0
	Resistive Pull Up and Down	7	Weak 1	Weak 0

Table 7-2. Input Buffer Disable (Port Configuration 2)

GPIO_PRTx_PC2 ('x' denotes port number and 'y' denotes pin number)		
Bits	**Name**	**Description**
7:0	INP_DIS	Disables the input buffer independent of the port control drive mode. This bit should be set when analog signals are present on the pin.

7.3.1 High-Impedance Analog

High-impedance analog mode is the default reset state; both output driver and digital input buffer are turned off. This state prevents an external voltage from causing a current to flow into the digital input buffer. This drive mode is recommended for pins that are floating or that support an analog voltage. High-impedance analog pins cannot be used for digital inputs. Reading the pin state register returns a 0x00 regardless of the data register value.

To achieve the lowest device current in low-power modes, unused GPIOs must be configured to the high-impedance analog mode.

7.3.2 High-Impedance Digital

High-impedance digital mode is the standard high-impedance (High-Z) state recommended for digital inputs. In this state, the input buffer is enabled for digital input signals.

7.3.3 Resistive Pull-Up or Resistive Pull-Down

Resistive modes provide a series resistance in one of the data states and strong drive in the other. Pins can be used for either digital input or digital output in these modes. If resistive pull-up is required, a '1' must be written to that pin's Data Register bit. If resistive pull-down is required, a '0' must be written to that pin's Data Register. Interfacing mechanical switches is a common application of these drive modes. The resistive modes are also used to interface CCG7D with open drain drive lines. Resistive pull-up is used when input is open drain low and resistive pull-down is used when input is open drain high.

7.3.4 Open Drain Drives High and Open Drain Drives Low

Open drain modes provide high impedance in one of the data states and strong drive in the other. The pins can be used as digital input or output in these modes. Therefore, these modes are widely used in bi-directional digital communication. Open drain drive high mode is used when signal is externally pulled down and open drain drive low is used when signal is externally pulled high.

A common application for open drain drives low mode is driving I^2C bus signal lines.

7.3.5 Strong Drive

The strong drive mode is the standard digital output mode for pins; it provides a strong CMOS output drive in both high and low states. Strong drive mode pins must not be used as inputs under normal circumstances. This mode is often used for digital output signals or to drive external transistors.

7.3.6 Resistive Pull-Up and Resistive Pull-Down

This mode is similar to the drive modes explained in 7.3.3 Resistive Pull-Up or Resistive Pull-Down. In the resistive pull-up and resistive pull-down mode, the GPIO will have a series resistance in both logic 1 and logic 0 output states. The high data state is pulled up while the low data state is pulled down. This mode is used when the bus is driven by other signals that may cause shorts.

7.4 Slew Rate Control

GPIO pins have fast and slow output slew rate options in strong drive mode; this can be configured using the GPIO_PRTx_PC[25] bit. Slew rate is individually configurable for each port. This bit is cleared by default and the port works in fast slew mode. This bit can be set if a slow slew rate is required. The fast slew rate is recommended for signals higher than 1 MHz. Slower slew rate results in reduced EMI and crosstalk; hence, the slow option is recommended for signals that are not speed critical – generally less than 1 MHz.

7.5 CMOS LVTTL Level Control

I/O pins can work at two voltage levels. These levels can be selected by writing to the GPIO_PRTx_PC[19] bit.

Input level is individually configurable for each port. This bit is cleared by default and the port works in CMOS mode. This bit can be set to reconfigure the port to LVTTL mode.

CMOS mode can be used in most cases, whereas LVTTL can be used for custom interface requirements, which works at lower voltage levels. See the device datasheet for the input voltage thresholds (VIH and VIL) for the modes.

7.6 GPIO-OVT

CCG7D device has two over-voltage tolerant (OVT) pins – I^2C pins from SCB0. These are similar to regular GPIOs with the following additional features:

- Over-voltage tolerant
- Provides better pull-down drive strength
- SCB when configured as I^2C and its lines routed to GPIO-OVT pins; it meets the following I^2C specifications:
  - Fast Mode hot-swap
  - Fast Mode Plus IOL Specification
  - Fast Mode and Fast Mode Plus Hysteresis and minimum fall time specifications

See the device datasheet for specifications.

7.7 High-Speed I/O Matrix

High-speed I/O matrix (HSIOM) is a group of high-speed switches that routes GPIOs to the resources inside CCG7D. These resources include TCPWMs, SCB, and the USB PD. The HSIOM selects Active and Deep Sleep power domain sources for a pin. HSIOM_PORT_SELx are 32-bit wide registers that control the routing of GPIOs. Each register controls one port; four dedicated bits are assigned to each GPIO in the port. This provides up to 16 different options for GPIO routing. This selection provides different pin functions, as listed in Table 7-3.

Table 7-3. HSIOM Port Settings

HSIOM_PORT_SELx ('x' denotes port number and 'y' denotes pin number)			
Bits	**Name (SEL 'y')**	**Value**	**Description (Selects pin 'y' source (0 ≤ y ≤ 7))**
4y+3: 4y	DR	0	Pin is regular firmware-controlled GPIO.
	AMUXA	6	Pin is connected to AMUXBUS-A.
	AMUXB	7	Pin is connected to AMUXBUS-B.
	ACT_0	8	Pin-specific Active source # 0 (TCPWM, EXT CLOCK).
	ACT_1	9	Pin-specific Active source #1 (TCPWM).
	ACT_2	10	Pin-specific Active source #2 (SCB–UART).
	ACT_3	11	Pin-specific Active source #3 (TCPWM).
	DPSLP_0	12	Pin-specific Deep Sleep source #0 (SWD, USBPD)
	DPSLP_1	13	Pin-specific Deep Sleep source #1 (SCB–SPI)
	DPSLP_2	14	Pin-specific Deep Sleep source #2 (SCB–SPI)
	DPSLP_3	15	Pin-specific Deep Sleep source #3 (SCB–I2C)

Note The active and deep sleep sources are pin dependent. See the “Pinouts” section of the device datasheet for more details on the features supported by each pin.

7.8 Firmware Controlled GPIO

See Table 7-3 to know the HSIOM settings for a firmware controlled GPIO. GPIO_PRTx_DR is the data register used to read and write the output data for the GPIOs. A write operation to this register changes the GPIO output to the written value. Note that a read operation reflects the output data written to this register and not the current state of the GPIOs. Using this register, read-modify-write sequences can be safely performed on a port that has both input and output GPIOs.

In addition to the data register, two other registers – GPIO_PRTx_DR_SET and GPIO_PRTx_DR_CLR – are provided to set or clear the output data of specific GPIOs in port. Additionally, the GPIO_PRTx_DR_INV register can be used to invert the output data of a specific GPIO. GPIO_PRTx_PS is the port I/O pad register that provides the state of the GPIOs when read. Writes to this register have no effect.

See the CCG7D Family EZ-PD CCG7D Registers TRM for details of these registers.

7.9 I/O Port Reconfiguration

Drive mode and GPIO can be reconfigured in runtime by changing the value of the GPIO_PRTx_PC and HSIOM_PORT_SELx registers. Take care to retain the pin state during reconfiguration of pins when they are connected directly to a digital peripheral. If the ports are driven by the data registers, state maintenance is automatic. During port configuration, the current configuration should be saved as follows:

1. Read the GPIO pin state – GPIO_PRTx_PS in software.
2. Write the GPIO_PRTx_PS value into the data registers – GPIO_PRTx_DR.
3. Change the corresponding field in HSIOM_PORT_SELx to drive the pin by the data register – GPIO_PRTx_DR.

7.10 I/O State on Power Up

By default, during power up all GPIOs are in high-impedance analog state and input buffers are disabled. When the device is powered, its GPIOs can be configured according to the required application, by writing to the associated registers.

7.11 Behavior in Low-Power Modes

The GPIOs maintain the current pin state during Sleep mode. In Sleep mode, all the GPIOs are active and can be driven by active peripherals, such as USB PD, TCPWM, and SCB.

In Deep Sleep mode, all the pin states are latched and the pin signals are retained, except the SCB pins, which remain functional and can wake up the processor on I^2C address matching event. GPIO interrupts are also available in Deep Sleep mode with wakeup ability.

For more details, see the Power Modes chapter on page 55, Interrupts chapter on page 28 and Serial Communications (SCB) chapter on page 66.

To achieve the lowest device current in low-power modes, unused I/Os must be configured in the high-impedance analog mode.

7.12 GPIO Interrupt

This section describes the interrupt functionality of the CCG7D GPIOs.

7.12.1 Features

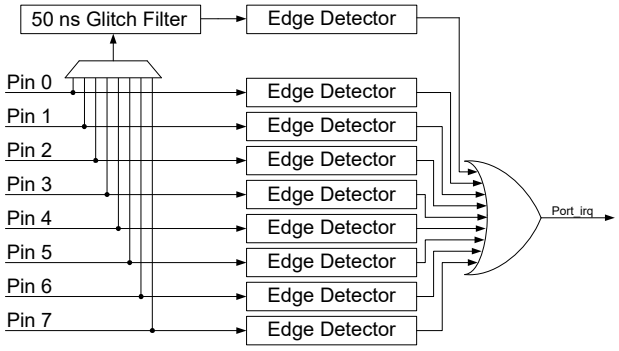
The features of the GPIO interrupt are:

- All eight pins in each port interface have an interrupt and an associated interrupt vector
- Pin status bits provide easy determination of interrupt source down to the pin level
- Rising, falling, or both edge-triggered interrupts are handled
- Pin interrupts can be individually enabled or disabled
- AHB interfaces for read and write into its registers
- Sends out a single port interrupt request (PIRQ) signal, derived from all GPIOs in a port, to the interrupt controller

7.12.2 Interrupt Controller Block Diagram

Each port has its own individual interrupt request and associated interrupt request (IRQ) vector and interrupt service routine (ISR). Additionally, one pin can be selected on each port that is routed through a 50-ns glitch filter to form a glitch-tolerant interrupt for the port. The details are shown in Figure 7-3.

Figure 7-3. Interrupt Generator



7.12.3 Function and Configuration

Each pin of the port can be configured independently to generate an interrupt on the rising edge, falling edge, or on both edges by writing to the GPIO_PRTx_INTR_CFG register. Level-sensitive interrupts are not supported. GPIO_PRTx_INTR_CFG is also used to route a specific channel to the glitch filter and generate a ninth glitch-tolerant interrupt.

When a GPIO interrupt is triggered by a signal on an interrupt-enabled port pin, the GPIO_PRTx_INTR register (Port Interrupt Status Register) is updated. The firmware can read this register to determine which GPIO triggered the interrupt. Firmware can then clear the IRQ bit by writing a '1' to its corresponding bit.

Additionally, when the Port Interrupt Control Status Register is read at the same time an interrupt is occurring on the corresponding port, it can result in the interrupt not being properly detected. Therefore, when using GPIO interrupts, it is recommended to read the status register only inside the corresponding interrupt service routine and not in any other part of the code.

See GPIO_PRTx_INTR_CFG and GPIO_PRTx_INTR in the CCG7D Family EZ-PD CCG7D Registers TRM for details.

7.13 Registers

Table 7-4. I/O Registers

Name	Description
GPIO_PRTx_DR	Port Output Data Register
GPIO_PRTx_DR_SET	Port Output Data Set Register
GPIO_PRTx_DR_CLR	Port Output Data Clear Register
GPIO_PRTx_DR_INV	Port Output Data Inverting Register
GPIO_PRTx_PS	Port Pin State Register – Used to read logical pin state of I/O
GPIO_PRTx_PC	Port Configuration Register – Configures the output drive mode, input threshold, and slew rate
GPIO_PRTx_PC2	Port Secondary Configuration Register – Configures the input buffer of I/O pin
GPIO_PRTx_INTR_CFG	Port Interrupt Configuration Register
GPIO_PRTx_INTR	Port Interrupt Status Register
HSIOM_PORT_SELx	HSIOM Port Selection Register

Note The 'x' in the register name denotes the port number. For example, GPIO_PRT1_DR is the port 1 output data register.

8. Clocking System

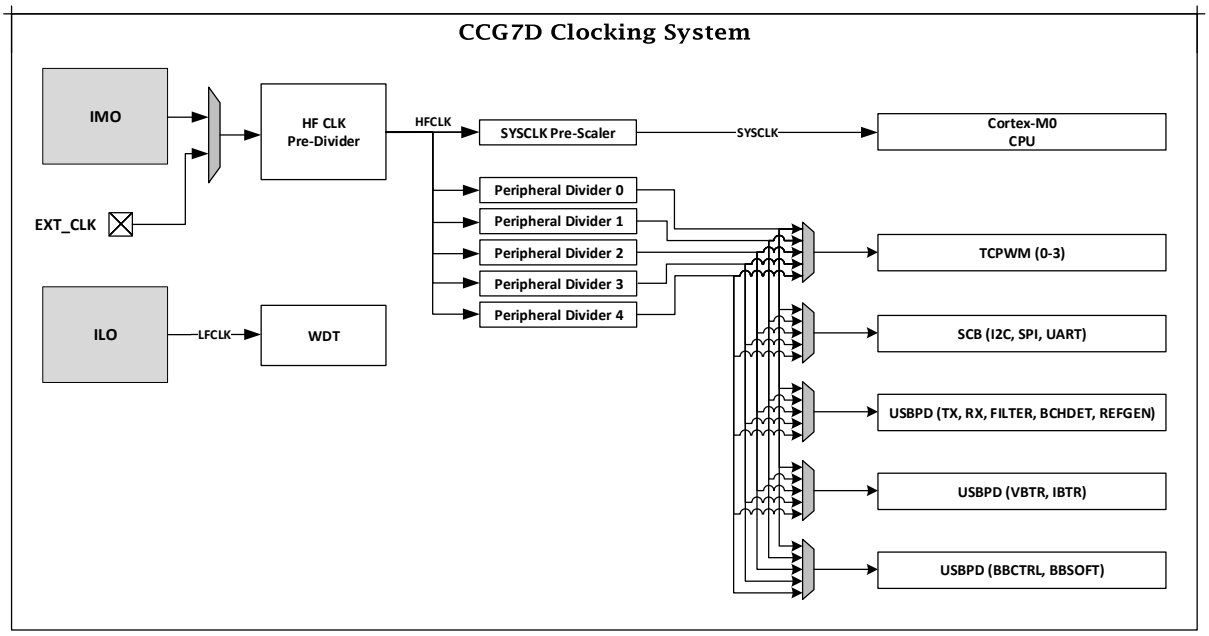
The CCG7D clock system includes these clock resources:

- Two internal clock sources:
  - 24–48 MHz internal main oscillator (IMO) ±2 percent across all frequencies with trim
  - 32-kHz internal low-speed oscillator (ILO)
- High-frequency clock (HFCLK) of up to 48 MHz selected from IMO or external clock
  - Dedicated prescaler for HFCLK
- Low-frequency clock (LFCLK) sourced by ILO
- Dedicated prescaler for system clock (SYSCLK) of up to 48 MHz sourced by HFCLK
- CCG7D device has twenty clock dividers, ten clocks with 8-bit dividers, six clocks with 16-bit dividers, and four clocks with 16.5-bit dividers, used for 29 outputs in total

8.1 Block Diagram

Figure 8-1 gives a generic view of the clocking system in CCG7D devices.

Figure 8-1. Clocking System Block Diagram



The three clock sources in the device are shown in Figure 8-1. The HFCLK mux selects the HFCLK source from an external clock source or the IMO. The HFCLK predivider divides the HFCLK input. The SYSCLK prescaler generates the SYSCLK and the peripheral dividers generate the individual peripheral clocks. The ILO sources the LFCLK.

Each of the SCB and TCPWM blocks used in the application need a clock generated by one of the peripheral dividers. In addition, each of the USB PD blocks require multiple clocks as shown in Table 8-7.

8.2 Clock Sources

8.2.1 Internal Main Oscillator

The internal main oscillator operates with no external components and outputs a stable clock at frequencies spanning 24-48 MHz in 4-MHz increments. Frequencies are selected by setting the frequency range in the CLK_IMO_SELECT register, and setting the IMO trim in the CLK_IMO_TRIM1 register. Each device has an IMO trim measured during manufacturing to meet datasheet specifications; the trim is stored in the manufacturing configuration data in SFLASH. These values may be loaded at startup to achieve the desired configuration. Firmware can retrieve these trim values and reconfigure the device to change the frequency at runtime.

8.2.1.1 Startup Behavior

After reset, the IMO is configured for 24-MHz operation. During the “boot” portion of startup, trim values are read from flash and the IMO is configured to achieve datasheet specified accuracy. The HFCLK predivider is initially set to a divide value of ‘4’ to reduce current consumption at startup.

8.2.2 Internal Low-speed Oscillator

The internal low-speed oscillator operates with no external components and outputs a stable clock at 32-kHz nominal. The ILO is relatively low power and low accuracy. It is available in all power modes. The ILO is always used as the system low-frequency clock, LFCLK, in CCG7D. The ILO is recommended to be always ON, because it is the source of the WDT, which is required for reliable system operation. The ILO can be disabled by clearing the ENABLE bit in the CLK_ILO_CONFIG register. The WDT reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the WDT reset will automatically enable the ILO.

Note Disabling the ILO reset is not recommended if:

- WDT protection is required against firmware crashes.
- WDT protection is required against the power supply events that produce sudden brownout events, which may in turn compromise the CPU functionality.

See the Watchdog Timer chapter on page 59 for details.

8.2.3 External Clock

The external clock is a MHz range clock that can be generated from a signal on a designated CCG7D pin. This clock may be used instead of the IMO as the source of the system high-frequency clock, HFCLK. The allowable range of external clock frequencies is 0–48 MHz. CCG7D always starts up using the IMO, and the external clock must be enabled in user mode, so the device cannot be started from a reset clocked by the external clock.

8.3 Clock Distribution

CCG7D clocks are developed and distributed throughout the device, as shown in Figure 8-1. The distribution configuration options are as follows:

- HFCLK input selection
- HFCLK predivider configuration
- SYSCLK prescaler configuration
- Peripheral divider configuration

8.3.1 HFCLK Input Selection

HFCLK in CCG7D has two input options: IMO and EXTCLK. The HFCLK input is selected using the CLK_SELECT register’s HFCLK_SEL bits, as described in Table 8-1.

Table 8-1. HFCLK Input Selection Bits HFCLK_SEL

Name	Description
HFCLK_SEL[2:0]	HFCLK input clock selection 0: IMO. Uses the IMO as the source of the HFCLK 1: EXTCLK. Uses the EXTCLK as the source of the HFCLK 2–7: Reserved. Do not use

When manually configuring a pin as the input to the EXTCLK, the drive mode of the pin must be set to high-impedance digital to enable the digital input buffer. See the I/O System chapter on page 40 for more details.

8.3.2 HFCLK Predivider Configuration

The HFCLK predivider allows the device to divide the HFCLK selection mux input before use as HFCLK. The predivider is capable of dividing the HFCLK by powers of two between 1 and 8. The predivider value is set using register CLK_SELECT bits HFCLK_DIV, as described in Table 8-2. The HFCLK predivider is set to a divide value of ‘4’ during startup to reduce current consumption.

Table 8-2. HFCLK Predivider Value Bits HFCLK_DIV

Name	Description
HFCLK_DIV[1:0]	HFCLK predivider value 0: 1. no divider on HFCLK 1: 2. divides HFCLK by 2 2: 4. divides HFCLK by 4 3: 8. divides HFCLK by 8

8.3.3 SYSCLK Prescaler Configuration

The SYSCLK prescaler allows the device to divide the predivided HFCLK before use as SYSCLK, which allows for non-integer relationships between peripheral clocks and the system clock. SYSCLK must be equal to or faster than all other clocks in the device that are derived from HFCLK. The SYSCLK prescaler is capable of dividing the HFCLK by powers of two between 1 and 8. The prescaler divide value is set using register CLK_SELECT bits SYSCLK_DIV, as described in Table 8-3. The prescaler is initially configured to divide by 1.

Table 8-3. SYSCLK Prescaler Divide Value Bits SYSCLK_DIV

Name	Description
SYSCLK_DIV[1:0]	SYSCLK prescaler divide value 0: 1. SYSCLK = HFCLK 1: 2. SYSCLK = HFCLK/2 2: 4. SYSCLK = HFCLK/4 3: 8. SYSCLK = HFCLK/8

8.3.4 Peripheral Clock Divider Configuration

The sixteen peripheral clocks are derived from the HFCLK using the 8-bit, 16-bit, or 16.5-bit peripheral clock dividers. Each of the sixteen dividers (eight 8-bit, four 16-bit and four 16.5 bit) is controlled by a PERI_DIV_8_CTL, PERI_DIV_16_CTL, or PERI_DIV_16_5_CTL register, whose mapping is listed in Table 8-4, Table 8-5, and Table 8-6 respectively.

Table 8-4. Peripheral Clock Divider Control Register PERI_DIV_8_CTLx

Bits	Name	Description
0	EN	Enables or disables the divider 0: Divider disabled 1: Divider enabled
15:8	INT8_DIV	Divide value of the divider. Output = Input/(INT8_DIV + 1)

Table 8-5. Peripheral Clock Divider Control Register PERI_DIV_16_CTLx

Bits	Name	Description
0	EN	Enables or disables the divider 0: Divider disabled 1: Divider enabled
23:8	INT16_DIV	Divide value for the divider. Output = Input/(INT16_DIV + 1). Acceptable divide values range from 0 to 65,536.

Table 8-6. Peripheral Clock Divider Control Register PERI_DIV_16_5_CTLx

Bits	Name	Description
0	EN	Enables or disables the divider 0: Divider disabled 1: Divider enabled
7:3	FRAC5_DIV	Fractional division by (FRAC5_DIV/32)
23:8	INT16_DIV	Integer division by (1+INT16_DIV). Acceptable divide values range from 0 to 65536.

The PERI_DIV_CMD register can be used to enable, disable, and select the type of clock dividers for all peripheral clock dividers. See the PERI_DIV_CMD in the CCG7D Family EZ-PD CCG7D Registers TRM for more details.

Input clocks to the peripherals are selected by PERI_PCLK_CTLx registers. Table 8-7 shows the peripheral clocks and their respective registers. See the CCG7D Family EZ-PD CCG7D Registers TRM for more details.

Table 8-7. Selecting Peripheral Clocks

Clock	Register
SCB0	PERI_PCLK_CTL0
SCB1	PERI_PCLK_CTL1
SCB2	PERI_PCLK_CTL2
SCB3	PERI_PCLK_CTL3
TCPWM0	PERI_PCLK_CTL4
TCPWM1	PERI_PCLK_CTL5
TCPWM2	PERI_PCLK_CTL6
TCPWM3	PERI_PCLK_CTL7
USBPD[0] (RX)	PERI_PCLK_CTL8
USBPD[0] (TX)	PERI_PCLK_CTL9
USBPD[0] (SAR)	PERI_PCLK_CTL10
USBPD[0] (FILTER1)	PERI_PCLK_CTL11
USBPD[0] (FILTER2)	PERI_PCLK_CTL12
USBPD[0] (REFGEN)	PERI_PCLK_CTL13
USBPD[0] (CHGDET)	PERI_PCLK_CTL14
USBPD[0] (VBTR)	PERI_PCLK_CTL15
USBPD[0] (BBCTRL)	PERI_PCLK_CTL16
USBPD[0] (BBCTRLSOFT)	PERI_PCLK_CTL17
USBPD[0] (IBTR)	PERI_PCLK_CTL18
USBPD[0] (RX)	PERI_PCLK_CTL19
USBPD[0] (TX)	PERI_PCLK_CTL20
USBPD[0] (SAR)	PERI_PCLK_CTL21
USBPD[0] (FILTER1)	PERI_PCLK_CTL22
USBPD[0] (FILTER2)	PERI_PCLK_CTL23
USBPD[0] (CHGDET)	PERI_PCLK_CTL24
USBPD[0] (VBTR)	PERI_PCLK_CTL25
USBPD[0] (BBCTRL)	PERI_PCLK_CTL26
USBPD[0] (BBCTRLSOFT)	PERI_PCLK_CTL27
USBPD[0] (IBTR)	PERI_PCLK_CTL28

8.4 Low-Power Mode Operation

The CCG7D clock behavior is different in different power modes. The MHz frequency clocks including the IMO, EXTCLK, HFCLK, SYSCLK, and peripheral clocks operate only in Active and Sleep modes. The ILO and LFCLK operate in all power modes.

8.5 Registers

Table 8-8. Clocking System Register List

Register Name	Description
CLK_IMO_TRIM1	IMO Trim Register – This register contains IMO trim, allowing fine manipulation of its frequency.
CLK_IMO_SELECT	IMO Frequency Selection Register – This register controls the frequency range of the IMO, allowing gross manipulation of its frequency.
CLK_ILO_CONFIG	ILO Configuration Register – This register controls the ILO configuration.
CLK_SELECT	Clock Select – This register controls clock tree configuration, selecting different sources for the system clocks.
PERI_DIV_8_CTLx/ PERI_DIV_16_CTLx/ PERI_DIV_16_5_CTLx	Peripheral Clock Divider Control Registers – These registers configure each of the peripheral clock dividers, enabling or disabling the divider and setting the integer divide value.
PERI_PCLK_CTLx	Programmable clock control registers – These registers are used to select the input clocks to peripherals.
PERI_DIV_CMD	Peripheral Divider Command --This register enables the PCLK divider.

9. Power Supply and Monitoring

CCG7D devices operate from VIN supply, which can vary from 4 V to 24 V.

9.1 Block Diagram

Figure 9-1. CCG7D Power System Block Diagram

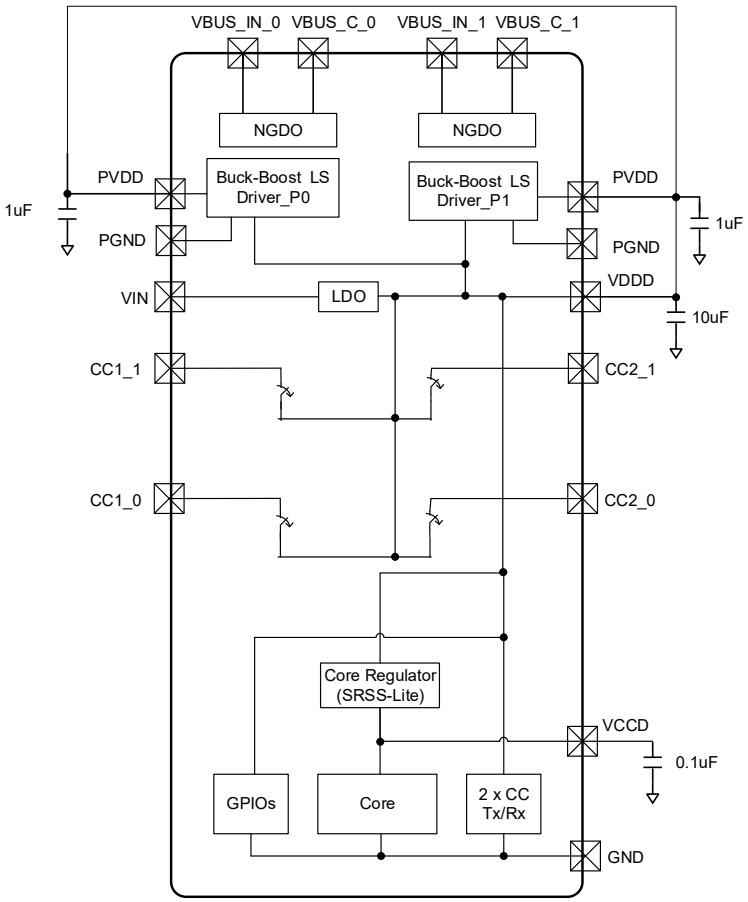


Figure 9-1 shows an overview of the CCG7D power system. CCG7D can operate from VIN supply, which can vary from 4 V to 24 V. CCG7D family of devices have three different power modes: Active, Sleep, and Deep Sleep, transitions between which are managed by the power system. A separate power domain VDDIO is provided for the GPIOs and VDDD, which generates 5 V from the internal regulator. VDDD can be shorted to VDDIO. The VCCD pin, the output of the core (1.8 V) regulator, is brought out for connecting a 0.1-µF capacitor for the regulator stability only. This pin is not supported as a power supply.

The VDDD supply is used to power Type-C cable markers through the VCONN switches and supports operation in the range 4.85 V to 5.5 V.

9.2 How It Works

The regulators in Figure 9-1 power the various domains of the device. All the core regulators and digital I/Os draw their input power from the VDDD pin supply. Digital I/Os are supplied from VDDD. The VDDIO pin, provides a separate voltage domain for the GPIOs. See the device datasheet for details.

9.2.1 Regulator Summary

The Active digital regulator and Quiet regulator are enabled during the Active or Sleep power modes. They are turned off in the Deep Sleep mode.

Table 9-1. Regulator Status in Different Power Modes

Mode	Active Regulator	Quiet Regulator
Deep Sleep	Off	Off
Sleep	On	On
Active	On	On

9.2.1.1 Active Digital Regulator

For external supplies from 2.75 V to 5.5 V, the Active digital regulator provides the main digital logic in Active and Sleep modes. This regulator has its output connected to a pin (VCCD) and requires an external low ESR decoupling capacitor (0.1 µF X7R).

9.2.1.2 Quiet Regulator

In Active and Sleep modes, this regulator supplies analog circuits such as the bandgap reference and capacitive sensing subsystem, which require a quiet supply, free of digital switching noise and power supply noise. This regulator has a high-power supply rejection ratio. The Quiet regulator is available only in Active and Sleep power modes.

9.2.1.3 Deep-Sleep Regulator

This regulator supplies the circuits that remain powered in Deep Sleep mode, such as the ILO and SCB. The deep-sleep regulator is available in all power modes. In Active and Sleep power modes, the main output of this regulator is connected to the output of the digital regulator (VCCD). This regulator also has a separate replica output that provides a stable voltage for the ILO. This output is not connected to VCCD in Active and Sleep modes. In deep sleep mode, the VDDD regulator is always ON. The VDDD regulator either takes VSYS or VBUS supply (with priority to VSYS).

9.3 Voltage Monitoring

The voltage monitoring system includes power-on-reset (POR) and brownout detection (BOD).

9.3.1 Power-On-Reset

POR circuits provide a reset pulse during the initial power ramp. POR circuits monitor VCCD voltage. Typically, the POR circuits are not very accurate with respect to trip-point.

POR circuits are used during initial device power-up and then disabled.

9.3.1.1 Brownout-Detect

The BOD circuit protects the operating or retaining logic from possibly unsafe supply conditions by applying reset to the device. BOD circuit monitors the VCCD voltage. The BOD circuit generates a reset if a voltage excursion dips below the minimum VCCD voltage required for safe operation (see the device datasheet for details). The system will not come out of RESET until the supply is detected to be valid again. To ensure the reliable operation of the device, the watchdog timer should be used in all designs. Watchdog timer provides protection against abnormal brownout conditions that may compromise the CPU functionality. See the Watchdog Timer chapter on page 59 for more details.

9.4 Register List

Table 9-2. Power Supply and Monitoring Register List

Register Name	Description
PWR_CONTROL	Power Mode Control Register – This register allows configuration of device power modes and regulator activity.

10. Chip Operational Modes

CCG7D can execute firmware in four different modes. These modes dictate execution from different locations in Flash and ROM, with different levels of hardware privileges. Only three of these modes are used in end-applications; debug mode is used exclusively to debug designs during firmware development.

CCG7D's operational modes are:

- Boot
- User
- Privileged
- Debug

10.1 Boot

Boot mode is an operational mode where the device is configured by instructions hard-coded in the device SROM. This mode is entered after the end of a reset, provided no debug-acquire sequence is received by the device. Boot mode is a privileged mode; interrupts are disabled in this mode so that the boot firmware can set up the device for operation without being interrupted. During the power-up phase, hardware trim settings are loaded from nonvolatile (NV) latches to guarantee proper operation during power-up. When boot concludes, the device enters user mode and code execution from flash begins.

10.2 User

User mode is an operational mode where normal user firmware from flash is executed. User mode cannot execute code from SROM. Firmware execution in this mode includes the automatically generated firmware by the PSoC Creator IDE and the firmware written by the user. The automatically generated firmware can govern both the firmware startup and portions of normal operation. The boot process transfers control to this mode after it has completed its tasks.

10.3 Privileged

Privileged mode is an operational mode, which allows execution of special subroutines that are stored in the device SROM. These subroutines cannot be modified by the user and are used to execute proprietary code that is not meant to be interrupted or observed. Debugging is not allowed in privileged mode.

The CPU can transition to privileged mode through the execution of a system call. For more information on how to perform a system call, see Performing a System Call on page 179. Exit from this mode returns the device to user mode.

10.4 Debug

Debug mode is an operational mode that allows observation of the CCG7D operational parameters. This mode is used to debug the firmware during development. The debug mode is entered when an SWD debugger connects to the device during the acquire time window, which occurs during the device reset. Debug mode allows IDEs such as PSoC Creator, Eclipse and Arm MDK to debug the firmware. Debug mode is only available on devices in open mode (one of the four protection modes). For more details on the debug interface, see the Program and Debug Interface chapter on page 171.

For more details on protection modes, see the Device Security chapter on page 64.

11. Power Modes

The CCG7D provides a number of power modes, intended at minimizing the average power consumption for a given application. The power modes, in the order of decreasing power consumption, are:

- Active
- Sleep
- Deep-Sleep

Active, Sleep, and Deep-Sleep are standard Arm-defined power modes, supported by the Arm CPUs and instruction set architecture (ISA).

The power consumption in different power modes is controlled by using the following methods:

- Enabling/disabling clocks to peripherals
- Powering on/off internal regulators
- Powering on/off clock sources
- Powering on/off other portions of the CCG7D

Figure 11-1 illustrates the various power modes and the possible transitions between them.

Figure 11-1. Power Mode Transitions State Diagram

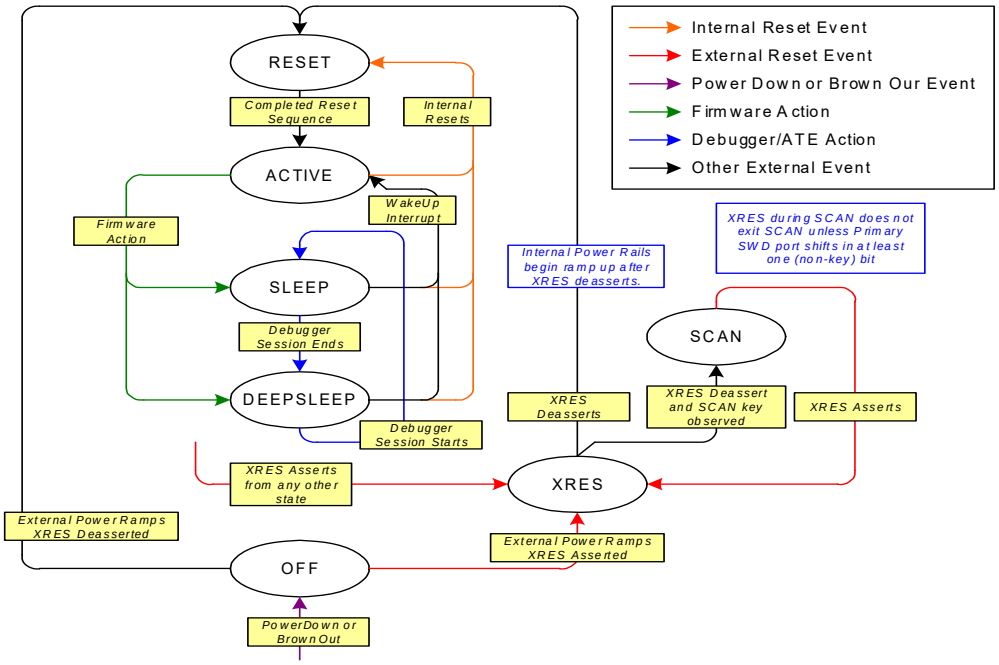


Table 11-1 illustrates the power modes offered by CCG7D

Table 11-1. CCG7D Power Modes

Power Mode	Description	Entry Condition	Wakeup Sources	Active Clocks	Wakeup Action	Available Regulators
Active	Primary mode of operation; all peripherals are available (programmable).	Wakeup from other power modes, internal and external resets, brownout, power on reset	Not applicable	Any (programmable)	Not applicable	All regulators are available. The active digital regulator can be disabled if external regulation is used.
Sleep	CPU enters Sleep mode and SRAM is in retention; all peripherals are available (programmable).	Manual register write	Any interrupt	Any (programmable)	Interrupt	All regulators are available. The active digital regulator can be disabled if external regulation is used.
Deep-Sleep	All internal supplies are driven from the Deep-Sleep regulator. IMO and high-speed peripherals are off. Only the low-frequency (~32 kHz) clock is available. Interrupts from low-speed, asynchronous, or USB PD can cause a wakeup.	Manual register write	GPIO interrupt, SCB, watchdog timer and USB PD	ILO (~32 kHz)	Interrupt	Deep-Sleep regulator

In addition to the wakeup sources mentioned in Table 11-1, external reset (XRES) and brownout reset bring the device to Active mode from any power mode.

11.1 Active Mode

Active mode is the primary power mode of the CCG7D device. This mode provides the option to use every possible subsystem/peripheral in the device. In this mode, the CPU is running and all the peripherals are powered. The firmware may be configured to disable specific peripherals that are not in use to reduce power consumption.

11.2 Sleep Mode

This is a CPU-centric power mode. In this mode, the Cortex-M0 CPU enters Sleep mode and its clock is disabled. It is a mode that the CCG7D can use to accomplish low power consumption in states where Deep Sleep mode cannot be used. It is identical to Active mode from a peripheral point of view. Any enabled interrupt can cause wakeup from Sleep mode.

11.3 Deep Sleep Mode

In Deep Sleep mode, the CPU, SRAM, and high-speed logic are in retention. The MHz range clocks, including HFCLK and SYSCLK, are disabled. Optionally, the internal low-frequency (32 kHz) oscillator remains on and low-frequency peripherals continue to operate. Digital peripherals that do not need a clock or receive a clock from their external interface (for example, I^2C slave) continue to operate. Interrupts from low-speed, asynchronous or USB PD can cause a wakeup from Deep Sleep mode.

The available wakeup sources are listed in Table 11-3.

11.4 Power Mode Summary

Table 11-2 illustrates the peripherals available in each low-power mode; Table 11-3 illustrates the wakeup sources available in each power mode.

Table 11-2. Available Peripherals

Peripheral	Active	Sleep	Deep-Sleep
CPU	On	Retention[a]	Retention
SRAM	On	Retention	Retention
High-speed peripherals	On	On	Retention
Low-speed peripherals	On	On	On (optional)
Internal main oscillator (IMO)	On	On	Off
Internal low-speed oscillator (ILO, 32 kHz)	On	On	On (optional)
Asynchronous peripherals	On	On	On
Power-on-reset, brownout detection	On	On	On
Regular analog peripherals	On	On	Off
GPIO output state	On	On	On

a. The configuration and state of the peripheral is retained. Peripheral continues its operation when the device enters Active mode.

Table 11-3. Wakeup Sources

Power Mode	Wakeup Source	Wakeup Action
Sleep	Any interrupt source	Interrupt
	Any reset source	Reset
Deep-Sleep	GPIO interrupt	Interrupt
	I2C address match	Interrupt
	Watchdog timer	Interrupt/Reset
	USB PD	Interrupt
	XRES (external reset pin)[a], Brownout	Reset

a. XRES triggers a full system restart. All the states including frozen GPIOs are lost. In this case, the cause of wakeup is not readable after the device restarts.

11.5 Low-Power Mode Entry and Exit

A Wait For Interrupt (WFI) instruction from the Cortex-M0 (CM0) triggers the transitions into Sleep and Deep Sleep mode. The Cortex-M0 can delay the transition into a low-power mode until the lowest priority ISR is exited (if the SLEEPONEXIT bit in the CM0 System Control Register is set).

The transition to Sleep and Deep Sleep modes are controlled by the flags SLEEPDEEP in the CM0 System Control Register (CM0_SCR):

- Sleep is entered when the WFI instruction is executed, SLEEPDEEP = 0.
- Deep-Sleep is entered when the WFI instruction is executed, SLEEPDEEP = 1.

The LPM READY bit in the PWR_CONTROL register shows the status of Deep-Sleep regulator. If the firmware tries to enter Deep Sleep mode before the regulators are ready, then CCG7D goes to Sleep mode first; when the regulators are ready, the device enters Deep Sleep mode. This operation is automatically done in hardware.

In Sleep and Deep Sleep modes, a selection of peripherals are available (see Table 11-3), and firmware can either enable or disable their associated interrupts. Enabled interrupts can cause wakeup from low-power mode to Active mode. Additionally, any RESET returns the system to Active mode.

11.6 Register List

Table 11-4. Power Mode Register List

Register Name	Description
CM0_SCR	System Control Register – Sets or returns system control data.
PWR_CONTROL	Power Mode Control – Controls the device power mode options and allows observation of current state.

12. Watchdog Timer

The watchdog timer (WDT) is used to automatically reset the device in the event of an unexpected firmware execution path or a brownout that compromises the CPU functionality. The WDT runs from the LFCLK (32-kHz clock), generated by the ILO. The timer must be serviced periodically in firmware to avoid a reset. Otherwise, the timer will elapse and generate a device reset. The WDT can be used as an interrupt source or a wakeup source in low-power modes.

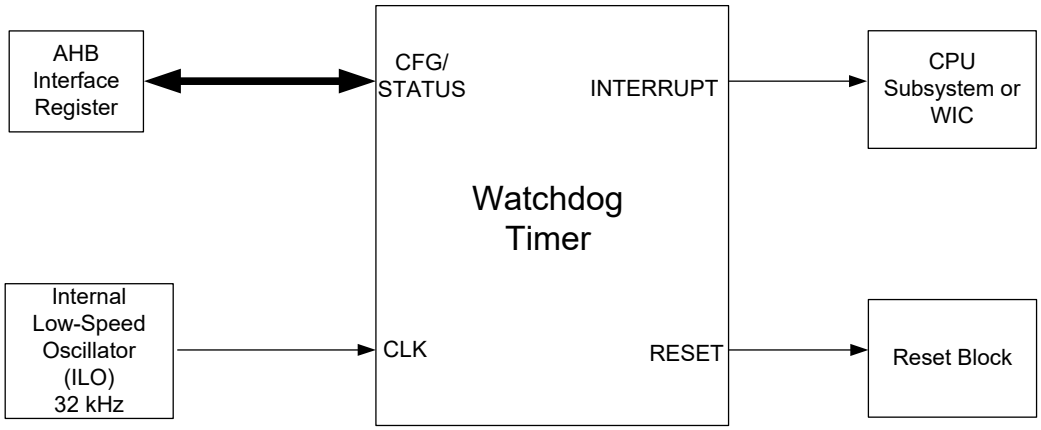
12.1 Features

The WDT has these features:

- Configurable timer period
- Can generate an interrupt in Sleep and Deep-Sleep power modes to wake up the device
- Can generate an interrupt in Active mode after a specified interval

12.2 Block Diagram

Figure 12-1. Watchdog Timer Block Diagram



12.3 How It Works

The WDT asserts a hardware reset to the device after three WDT interrupts each of programmable interval of up to 2048 ms, unless it is periodically serviced in firmware. The WDT is a 16-bit free-running wraparound up-counter.

The WDT_COUNTER register provides the count value of the WDT. The WDT generates an interrupt when the count value in WDT_COUNTER equals the match value stored in the WDT_MATCH register, but it does not reset the count to '0'. Instead, the WDT keeps counting until it overflows and rolls back to 0. When the count value reaches the match value again, another interrupt is generated.

A bit named WDT_MATCH in the SRSS_INTR register is set whenever the WDT interrupt occurs. This interrupt must be cleared by writing a '1' to the WDT_MATCH bit in SRSS_INTR to feed the watchdog timer. If the firmware does not feed the WDT for two consecutive interrupts, the third match event will generate a hardware reset. For details, see the

WDT_COUNTER, WDT_MATCH, and SRSS_INTR registers in the Register List on page 61.

When the WDT is used to protect against system crashes, clearing the WDT interrupt bit to feed the watchdog must be done from a portion of the code that is not directly associated with the WDT interrupt. Otherwise, even if the main function of the firmware crashes or is in an endless loop, the WDT interrupt vector can still be intact and feed the WDT periodically.

The safest way to use the WDT against system crashes is:

- Feed the watchdog by clearing the interrupt bit regularly in the main body of the firmware code.
- Guarantee that the interrupt is cleared at least once every WDT period.
- Use the WDT ISR only as a timer to trigger certain actions and to change the next WDT_MATCH value. Do not feed WDT in this ISR.

Follow these steps to use WDT as a periodic interrupt generator:

1. Write the desired match value to the WDT_MATCH register.
2. Clear the WDT_MATCH bit in SRSS_INTR to clear any pending WDT interrupt.
3. Enable the WDT interrupt by setting the WDT_MATCH bit in SRSS_INTR_MASK.
4. In the ISR, clear the WDT interrupt and add the desired match value to the existing match value. By doing so, another periodic interrupt will be generated when the counter reaches the new match value.
5. The IGNORE_BITS in the WDT_MATCH register can be used to reduce the entire WDT counter period. The ignore bits can specify the number of MSbs that needs to be discarded. For example, if the IGNORE_BITS value is 3, then WDT counter becomes a 13-bit counter.

12.3.1 Enabling and Disabling WDT

The WDT is a free-running counter that cannot be disabled. However, it is possible to disable the WDT reset by writing a key '0xACED8865' to the WDT_DISABLE_KEY register. Writing any other value to this register will enable the WDT reset. If WDT reset is disabled, the firmware does not need to periodically feed the WDT to avoid a reset. The WDT can still be used as an interrupt source or wakeup source. The only way to stop WDT from generating interrupts and wakeup events is to disable the ILO by clearing the ENABLE bit in the CLK_ILO_CONFIG register. The WDT reset must be disabled before disabling the ILO. Otherwise, any register write to disable the ILO will be ignored. Enabling the WDT reset will automatically enable the ILO.

Note Disabling the WDT reset is not recommended if:

- Protection is required against firmware crashes
- The power supply can produce sudden brownout events that may compromise the CPU functionality

12.3.2 WDT Interrupts and Low-Power Modes

The WDT counter sends the interrupt requests to the CPU in Active power mode and to the WakeUp Interrupt Controller (WIC) in Sleep and Deep-Sleep power modes. It works as follows:

- **Active Mode:** In Active mode, the WDT sends the interrupt to the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.
- **Sleep or Deep Sleep Mode:** In this mode, the CPU subsystem is powered down. Therefore, the interrupt request from the WDT is directly sent to the WIC, which will then wake up the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.

For more details, see the Power Modes chapter on page 55.

12.3.3 WDT Reset Mode

The RESET_WDT bit in the RES_CAUSE register indicates the reset generated by the WDT. This bit remains set until cleared or until a POR, BOD, or XRES occurs. All other resets leave this bit untouched.

For more details, see the Reset System chapter on page 62.

12.4 Register List

Table 12-1. WDT Registers

Register Name	Description
WDT_DISABLE_KEY	Disables the WDT when 0XACED8865 is written, for any other value WDT works normally
WDT_COUNTER	Provides the count value of the WDT
WDT_MATCH	Stores the match value of the WDT
SRSS_INTR	Feeds the WDT to avoid reset

13. Reset System

CCG7D supports several types of resets that guarantee error-free operation during power up and allow the device to reset based on user-supplied external hardware or internal software reset signals. CCG7D also contains hardware to enable the detection of certain resets.

The reset system has these sources:

- Power-on reset (POR) to hold the device in reset while the power supply ramps up
- Brownout reset (BOD) to reset the device if the power supply falls below specifications during operation
- Watchdog reset (WRES) to reset the device if firmware execution fails to service the watchdog timer
- Software initiated reset (SRES) to reset the device on demand using firmware
- External reset (XRES) to reset the device using an electrical signal external to the CCG7D
- Protection fault reset (PROT_FAULT) to reset the device if unauthorized operating conditions occur

13.1 Reset Sources

The following sections provide a description of the reset sources available in CCG7D.

13.1.1 Power-on Reset

Power-on reset is provided for system reset at power-up. POR holds the device in reset until the supply voltage, VDDD, is according to the datasheet specification. The POR activates automatically at power-up.

POR events do not set a reset cause status bit, but can be partially inferred by the absence of any other reset source. If no other reset event is detected, then the reset is caused by POR, BOD, or XRES.

13.1.2 Brownout Reset

Brownout reset monitors the digital voltage supply VCCD and generates a reset if VCCD is below the minimum logic operating voltage specified in the device datasheet. BOD is available in all power modes and cannot be distinguished from a POR or XRES reset.

13.1.3 Watchdog Reset

Watchdog reset (WRES) detects errant code by causing a reset if the watchdog timer is not cleared within the user-specified time limit. Watchdog reset is enabled by default. To disable it, refer to Enabling and Disabling WDT chapter on page 60.

The RESET_WDT status bit of the RES_CAUSE register is set when a watchdog reset occurs. This bit remains set until cleared or until a POR or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

For more details, see the Watchdog Timer chapter on page 59

13.1.4 Software Initiated Reset

Software initiated reset (SRES) is a mechanism that allows a software-driven reset. The Cortex-M0 application interrupt and reset control register (CM0_AIRCR) forces a device reset when a ‘1’ is written into the SYSRESETREQ bit. CM0_AIRCR requires a value of 05FA written to the top two bytes for writes. Therefore, write 05FA0004 to trigger the reset.

The RESET_SOFT status bit of the RES_CAUSE register is set when a software reset occurs. This bit remains set until cleared or until a POR or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

13.1.5 External Reset

External reset (XRES) is a user-supplied reset that causes immediate system reset when asserted. The XRES_N pin is ***active low*** – a high voltage on the pin causes no behavior and a low voltage causes a reset. The pin is pulled high inside the device. XRES is available as a dedicated pin in most of the devices. For detailed pinout, refer to the pinout section of the device datasheet.

The XRES pin holds the device in reset while held active. When the pin is released, the device goes through a normal boot sequence. The logical thresholds for XRES and other electrical characteristics, are listed in the Electrical Specifications section of the device datasheet.

XRES events do not set a reset cause status bit, but can be partially inferred by the absence of any other reset source. If no other reset event is detected, then the reset is caused by POR, BOD, or XRES.

13.1.6 Protection Fault Reset

Protection fault reset (PROT_FAULT) detects unauthorized protection violations and causes a device reset if they occur. One example of a protection fault is if a debug breakpoint is reached while executing privileged code. For details about privilege code, see Privileged on page 54.

The RESET_PROT_FAULT bit of the RES_CAUSE register is set when a protection fault occurs. This bit remains set until cleared or until a POR or BOD reset; for example, in the case of a device power cycle. All other resets leave this bit untouched.

13.2 Identifying Reset Sources

When the device comes out of reset, it is often useful to know the cause of the most recent or even older resets. This is achieved in the device primarily through the RES_CAUSE register. This register has specific status bits allocated for some of the reset sources. The RES_CAUSE register supports detection of watchdog reset, software reset, and protection fault reset. It does not record the occurrences of POR, BOD, or XRES. The bits are set on the occurrence of the corresponding reset and remain set after the reset, until cleared or a loss of retention, such as a POR reset or brownout below the logic retention voltage.

If the RES_CAUSE register cannot detect the cause of the reset, then it can be one of the non-recorded and non-retention resets: BOD, POR, or XRES. These resets cannot be distinguished using on-chip resources.

13.3 Register List

Table 13-1. Reset System Register List

Register Name	Description
CM0_AIRCR	Cortex-M0 Application Interrupt and Reset Control Register – This register allows initiation of software resets, among other Cortex-M0 functions.
RES_CAUSE	Reset Cause Register – This register captures the cause of recent resets.

14. Device Security

CCG7D offers a number of options for protecting user designs from unauthorized access or copying. Disabling debug features and robust flash protection provide a high level of security.

The debug circuits are enabled by default and can only be disabled in firmware. If disabled, the only way to re-enable them is to erase the entire device, clear flash protection, and reprogram the device with new firmware that enables debugging. Additionally, all device interfaces can be permanently disabled for applications concerned about phishing attacks due to a maliciously reprogrammed device or attempts to defeat security by starting and interrupting flash programming sequences. Permanently disabling interfaces is not recommended for most applications because the designer cannot access the device. For more information, as well as a discussion of flash row and chip protection, see the CYPDXXXX Programming Specifications.

Note Because all programming, debug, and test interfaces are disabled when maximum device security is enabled, CCG7D devices with full device security enabled may not be returned for failure analysis.

14.1 Features

The CCG7D device security system has the following features:

- User-selectable levels of protection
- In the most secure case provided, the chip can be “locked” such that it cannot be acquired for test/debug and it cannot enter erase cycles. Interrupting erase cycles is a known way for hackers to leave chips in an undefined state and open to observation.
- CPU execution in a privileged mode by use of the non-maskable interrupt (NMI). When in privileged mode, NMI remains asserted to prevent any inadvertent return from interrupt instructions causing a security leak.

14.2 How It Works

The CPU operates in normal user mode or in privileged mode, and the device operates in one of four protection modes: BOOT, OPEN, PROTECTED, and KILL. Each mode provides specific capabilities for the CPU software and debug. You can change the mode by writing to the CPUSS_PROTECTION register.

- **BOOT mode**: The device comes out of reset in BOOT mode. It stays there until its protection state is copied from supervisor flash to the protection control register (CPUSS_PROTECTION). The debug-access port is stalled until this has happened. BOOT is a transitory mode required to set the part to its configured protection state. During BOOT mode, the CPU always operates in privileged mode.
- **OPEN mode**: This is the factory default. The CPU can operate in user mode or privileged mode. In user mode, flash can be programmed and debugger features are supported. In privileged mode, access restrictions are enforced.
- **PROTECTED mode**: The user may change the mode from OPEN to PROTECTED. This disables all debug access to user code or memory. Access to most registers is still available; debug access to registers to reprogram flash is not available. The mode can be set back to OPEN but only after completely erasing the flash.
- **KILL mode**: The user may change the mode from OPEN to KILL. This removes all debug access to user code or memory, and the flash cannot be erased. Access to most registers is still available; debug access to registers to reprogram flash is not available. The part cannot be taken out of KILL mode; devices in KILL mode may not be returned for failure analysis.

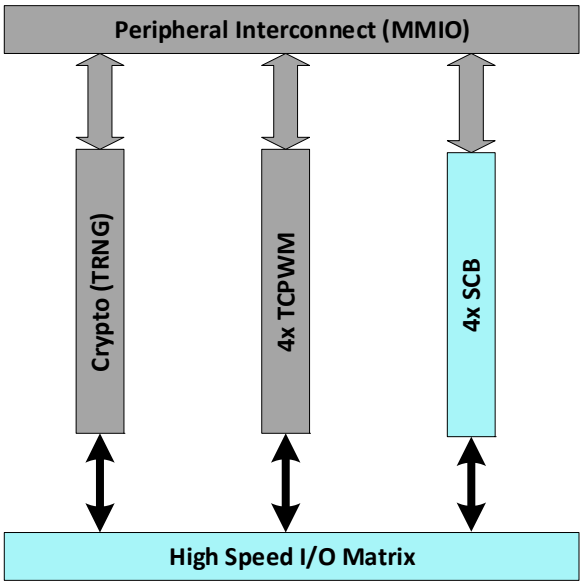
Section E: Digital System

This section encompasses the following chapters:

- Serial Communications (SCB) chapter on page 66
- Cryptography (TR) Block chapter on page 103
- Timer and PWM chapter on page 109

Top Level Architecture

Digital System Block Diagram



15. Serial Communications (SCB)

The Serial Communications Block (SCB) of CCG7D supports three serial interface protocols: SPI, UART, and I^2C. Only one of the protocols is supported by an SCB at any given time. CCG7D has four SCBs. CCG7D 96-pin BGA package supports four configurable SCBs. See the device datasheet for packaging specifications.

15.1 Features

This block supports the following features:

- Standard SPI master and slave functionality with Motorola, Texas Instruments, and National Semiconductor protocols
- Standard UART functionality with SmartCard reader, local interconnect network (LIN), and IrDA protocols
- Standard I^2C master and slave functionality
- SPI and I^2C EZ mode, which allows operating without CPU intervention
- Low-power (Deep-Sleep) power mode for SPI and I^2C protocols (using external clocking)

Each of the three protocols is explained in the following sections.

15.2 Serial Peripheral Interface (SPI)

The SPI protocol is a synchronous serial interface protocol. Devices operate in either master or slave mode. The master initiates the data transfer. The SCB supports single-master-multiple-slaves topology for SPI. Multiple slaves are supported with individual slave select lines.

You can use the SPI master mode when the CCG7D needs to communicate with one or more SPI slave devices. The SPI slave mode can be used when the CCG7D needs to communicate with an SPI master device.

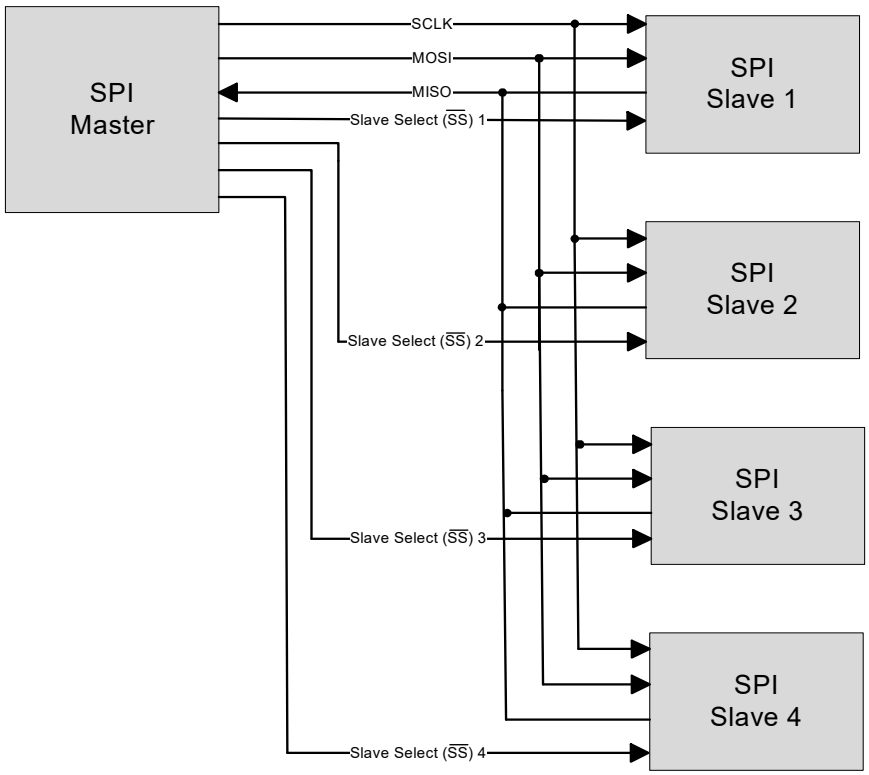
15.2.1 Features

- Supports master and slave functionality
- Supports three types of SPI protocols:
  - Motorola SPI – modes 0, 1, 2, and 3
  - TI SPI, with coinciding and preceding data frame indicator for mode 1
  - National (MicroWire) SPI for mode 0
- Data frame size programmable from 4 bits to 16 bits
- Interrupts or polling CPU interface
- Programmable oversampling
- Supports EZ mode of operation (Easy SPI (EZSPI) Protocol (applicable only for SPI slave functionality))
- Supports externally clocked slave operation:
  - In this mode, the slave operates in Active, Sleep, and Deep-Sleep system power modes
  - EZSPI mode allows for operation without CPU intervention

15.2.2 General Description

Figure 15-1 illustrates an example of SPI master with four slaves.

Figure 15-1. SPI Example



A standard SPI interface consists of four signals as follows.

- SCLK: Serial clock (clock output from the master, input to the slave).
- MOSI: Master-out-slave-in (data output from the master, input to the slave).
- MISO: Master-in-slave-out (data input to the master, output from the slave).
- Slave Select ($\overline{SS}$): Typically, an active low signal (output from the master, input to the slave).

A simple SPI data transfer involves the following: the master selects a slave by driving its SS line, then it drives data on the MOSI line and a clock on the SCLK line, The slave uses the edges of SCLK to capture the data on the MOSI line; it also drives data on the MISO line, which is captured by the master.

By default, the SPI interface supports a data frame size of eight bits (1 byte). The data frame size can be configured to any value in the range 4 to 16 bits. The serial data can be transmitted either most significant bit (MSB) first or least significant bit (LSB) first.

Three different variants of the SPI protocol are supported by the SCB:

- Motorola SPI: This is the original SPI protocol.
- Texas Instruments SPI: A variation of the original SPI protocol, in which data frames are identified by a pulse on the $\overline{SS}$ line.
- National Semiconductors SPI: A half-duplex variation of the original SPI protocol.

15.2.3 SPI Modes of Operation

15.2.3.1 Motorola SPI

The original SPI protocol was defined by Motorola. It is a full duplex protocol. Multiple data transfers may happen with the SS line held at '0'. As a result, slave devices must keep track of the progress of data transfers to separate individual data frames. When not transmitting data, the SS line is held at '1' and SCLK is typically off.

Modes of Motorola SPI

The Motorola SPI protocol has four different modes based on how data is driven and captured on the MOSI and MISO lines. These modes are determined by clock polarity (CPOL) and clock phase (CPHA).

Clock polarity determines the value of the SCLK line when not transmitting data. CPOL = 0 indicates that SCLK is '0' when not transmitting data. CPOL = 1 indicates that SCLK is '1' when not transmitting data.

Clock phase determines when data is driven and captured. CPHA = 0 means sample (capture data) on the leading (first) clock edge, while CPHA = 1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. When CPHA is 0, the data must be stable for setup time before the first clock cycle.

- Mode 0: CPOL is '0', CPHA is '0': Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK.
- Mode 1: CPOL is '0', CPHA is '1': Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK.
- Mode 2: CPOL is '1', CPHA is '0': Data is driven on a rising edge of SCLK. Data is captured on a falling edge of SCLK.
- Mode 3: CPOL is '1', CPHA is '1': Data is driven on a falling edge of SCLK. Data is captured on a rising edge of SCLK.

Figure 15-2 illustrates driving and capturing of MOSI/MISO data as a function of CPOL and CPHA.

Figure 15-2. SPI Motorola, 4 Modes

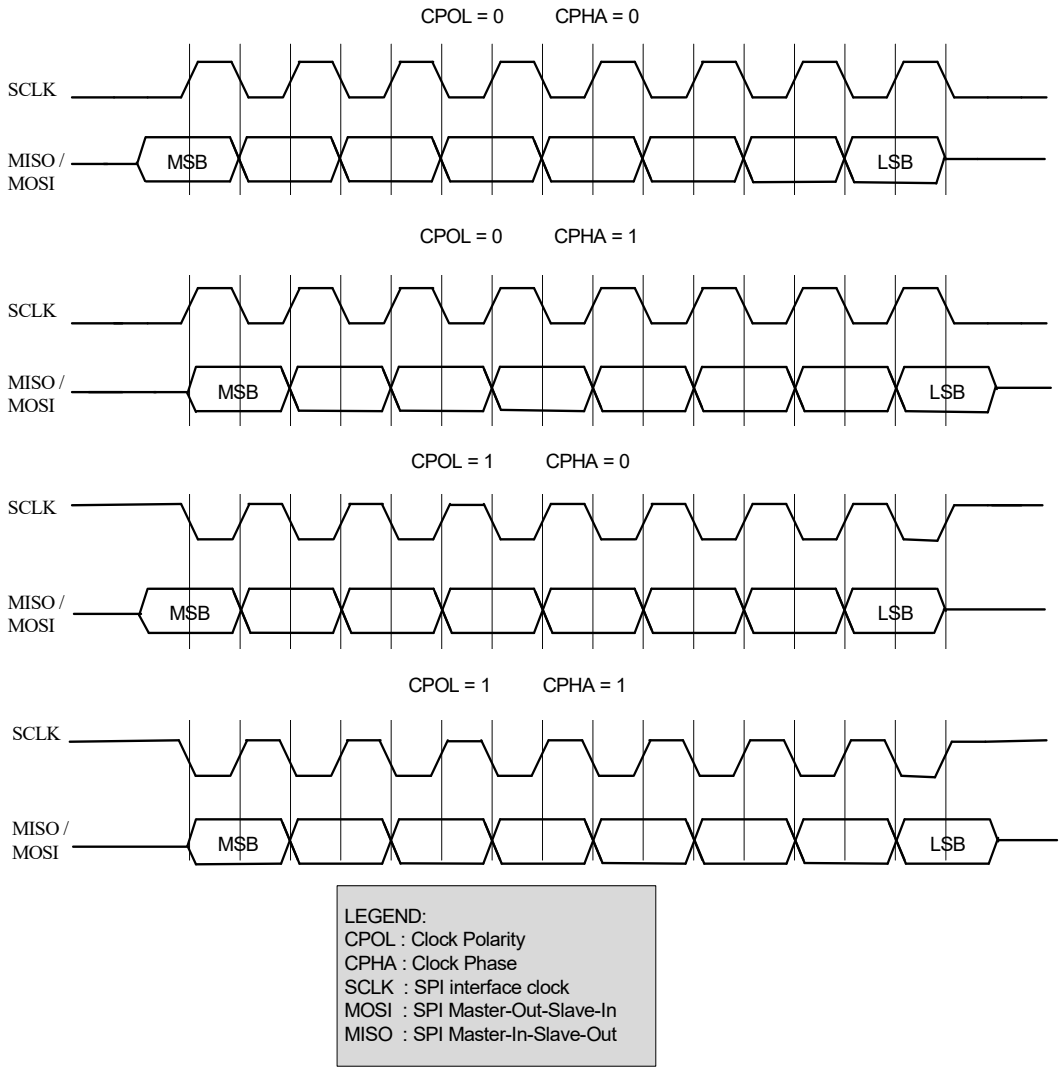
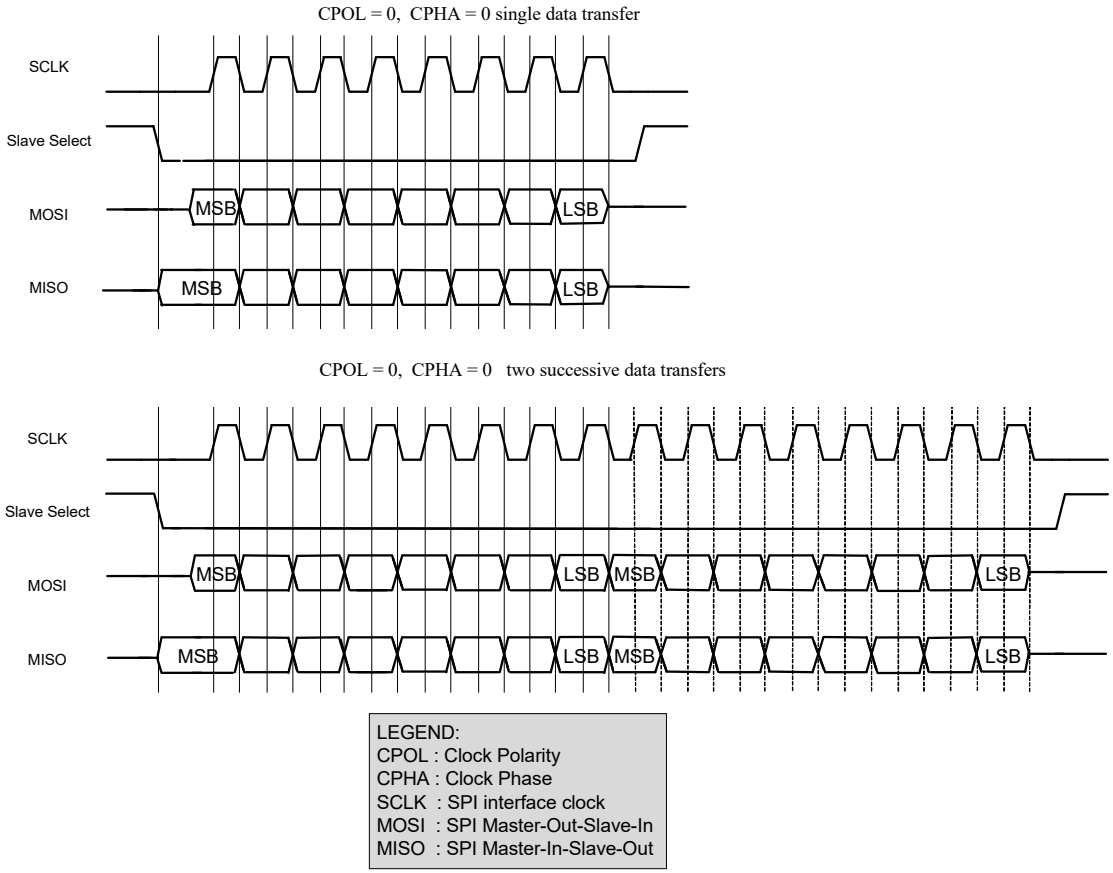


Figure 15-3 illustrates a single 8-bit data transfer and two successive 8-bit data transfers in mode 0 (CPOL is '0', CPHA is '0').

Figure 15-3. SPI Motorola Data Transfer Example



Configuring SCB for SPI Motorola Mode

To configure the SCB for SPI Motorola mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Select SPI Motorola mode by writing '00' to the MODE (bits [25:24]) of the SCBx_SPI_CTRL register.
3. Select the mode of operation in Motorola by writing to the SCB_CPHA and SCB_CPOL fields (bits 2 and 3 respectively) of the SCBx_SPI_CTRL register.
4. Follow steps 2 to 4 from Enabling and Initializing SPI on page 76.

For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

15.2.3.2 Texas Instruments SPI

The Texas Instruments' SPI protocol redefines the use of the $\overline{SS}$ signal. It uses the signal to indicate the start of a data transfer, rather than a low active slave select signal, as in the case of Motorola SPI. As a result, slave devices need not keep track of the progress of data transfers to separate individual data frames. The start of a transfer is indicated by a high active pulse on slave select signal for a single bit transfer period. This pulse can be configured to occur one cycle before the transmission of the first data bit, or coincide with the transmission of the first data bit. The TI SPI protocol supports only mode 1 (CPOL is '0' and CPHA is '1'): data is driven on a rising edge of SCLK and data is captured on a falling edge of SCLK.

Figure 15-4 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse precedes the first data bit. Note how the SELECT pulse of the second data transfer coincides with the last data bit of the first data transfer.

Figure 15-4. SPI TI Data Transfer Example

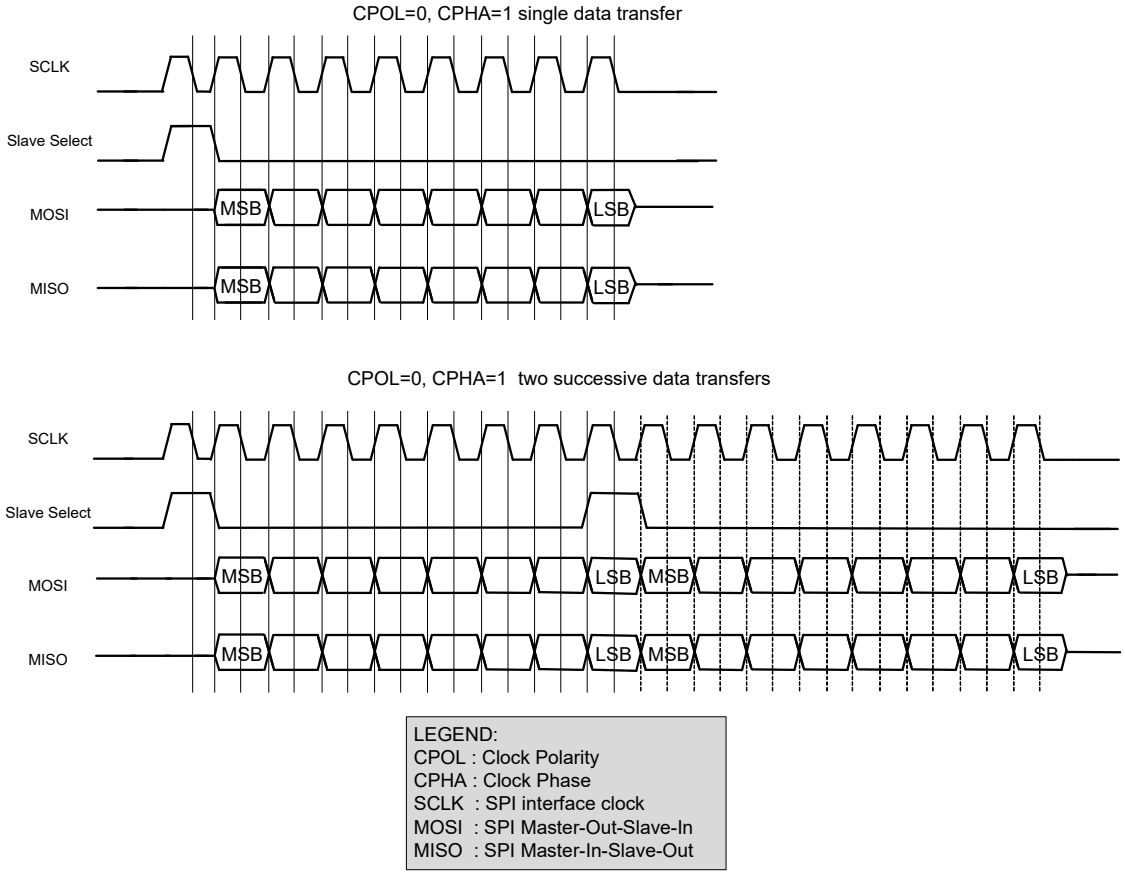
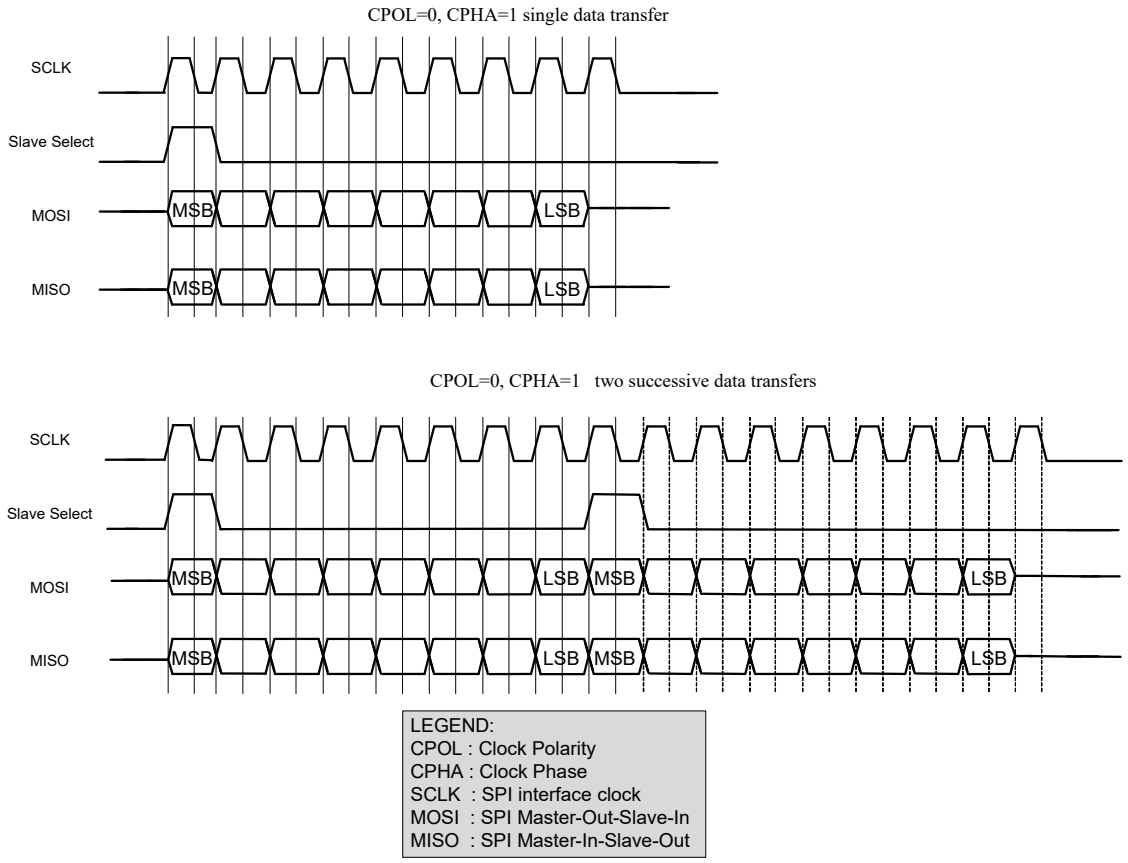


Figure 15-5 illustrates a single 8-bit data transfer and two successive 8-bit data transfers. The SELECT pulse coincides with the first data bit of a frame.

Figure 15-5. SPI TI Data Transfer Example



Configuring the SCB for SPI TI Mode

To configure the SCB for SPI TI mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Select SPI TI mode by writing '01' to the MODE (bits [25:24]) of the SCBx_SPI_CTRL register.
3. Select the mode of operation in TI by writing to the SELECT_PRECEDE field (bit 1) of the SCBx_SPI_CTRL register ('1' configures the SELECT pulse to precede the first bit of next frame and '0' otherwise).
4. Follow steps 2 to 4 from Enabling and Initializing SPI on page 76.

For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

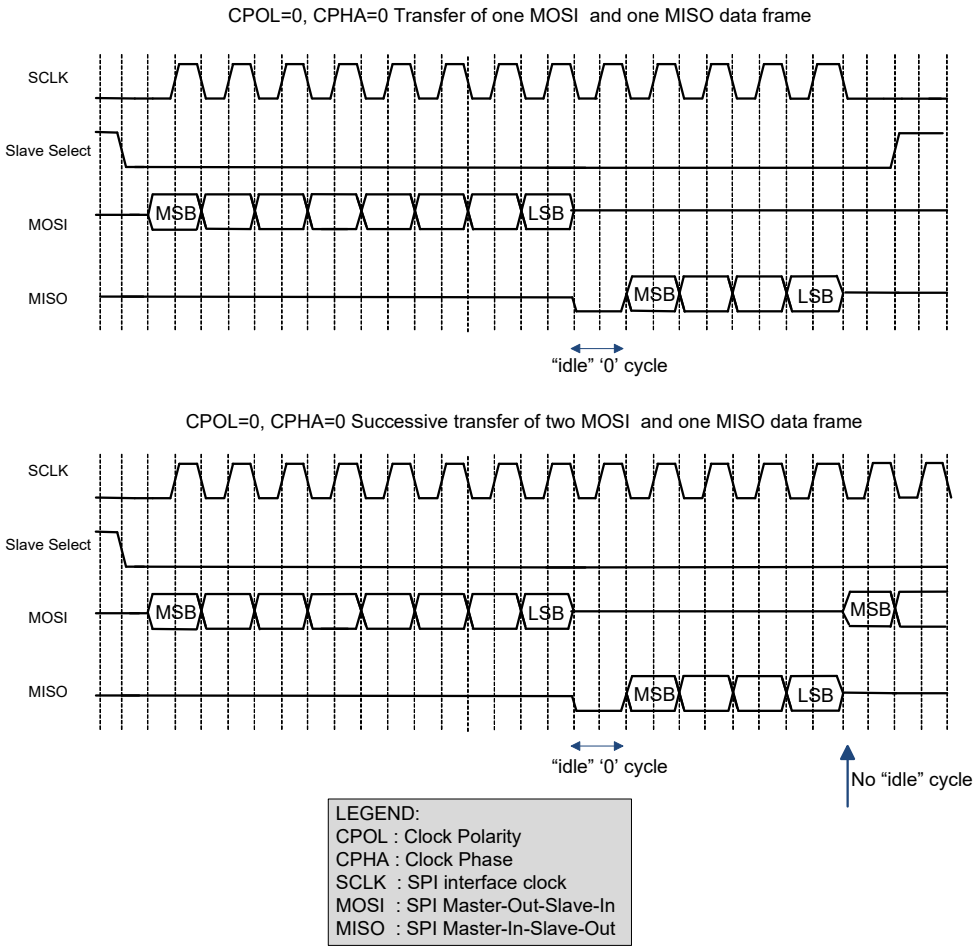
15.2.3.3 National Semiconductors SPI

The National Semiconductors' SPI protocol is a half-duplex protocol. Rather than transmission and reception occurring at the same time, they take turns. The transmission and reception data sizes may differ. A single “idle” bit transfer period separates transmission from reception. However, the successive data transfers are NOT separated by an “idle” bit transfer period.

The National Semiconductors SPI protocol only supports mode 0 (CPOL is ‘0’ and CPHA is ‘0’): data is driven on a falling edge of SCLK and data is captured on a rising edge of SCLK.

Figure 15-6 illustrates a single data transfer and two successive data transfers. In both cases, the transmission data transfer size is eight bits and the reception data transfer size is four bits.

Figure 15-6. SPI NS Data Transfer Example



Configuring the SCB for SPI NS Mode

To configure the SCB for SPI NS mode, set various register bits in the following order:

1. Select SPI by writing '01' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Select SPI NS mode by writing '10' to the MODE (bits [25:24]) of the SCBx_SPI_CTRL register.
3. Follow steps 2 to 4 from Enabling and Initializing SPI on page 76.

For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

15.2.4 Easy SPI (EZSPI) Protocol

The easy SPI (EZSPI) protocol is based on the Motorola SPI operating in mode 0 (CPOL is ‘0’ and CPHA is ‘0’). EZSPI mode is applicable only for slave functionality in a single slave topology. It allows communication between a master and a single slave without the need for CPU intervention at the level of individual frames.

The EZSPI protocol defines an 8-bit EZ address that indexes a memory array (32-entry array of eight bit per entry is supported) located on the slave device. To address these 32 locations, the lower five bits of the EZ address are used. All EZSPI data transfers have 8-bit data frames.

Note The SCB has a FIFO memory, which is a 16-word by 16-bit SRAM, with byte write enable. The access methods for EZ and non-EZ functions are different. In non-EZ mode, the FIFO is split into TXFIFO and RXFIFO. Each has eight entries with 16 bits per entry. The 16-bit width per entry is used to accommodate configurable data width. In EZ mode, it is used as a single 32x8 bit EZFIFO because only a fixed 8-bit width data is used in EZ mode.

EZSPI has three types of transfers: a write of the EZ address from the master to the slave, a write of data from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

15.2.4.1 EZ Address Write

A write of the EZ address starts with a command byte (0x00) on the MOSI line indicating the master's intent to write the EZ address. The slave then drives a reply byte on the MISO line to indicate that the command is observed (0xFE) or not (0xFF). The second byte on the MOSI line is the EZ address.

15.2.4.2 Memory Array Write

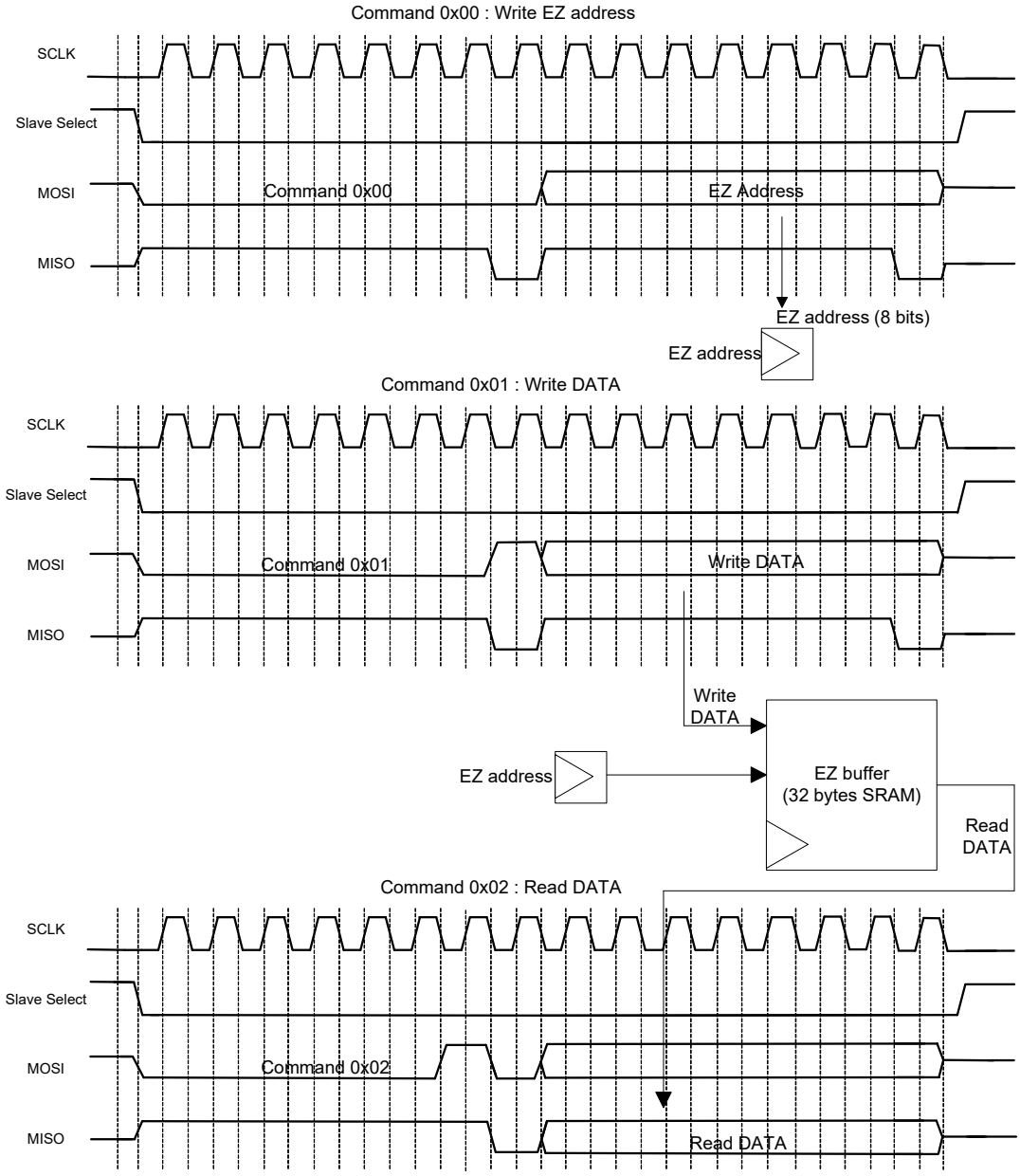
A write to a memory array index starts with a command byte (0x01) on the MOSI line indicating the master's intent to write to the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was observed (0xFE) or not (0xFF). Any additional write data bytes on the MOSI line are written to the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are written into the memory array. When the EZ address exceeds the maximum number of memory entries (32), it wraps around to 0.

15.2.4.3 Memory Array Read

A read from a memory array index starts with a command byte (0x02) on the MOSI line indicating the master's intent to read from the memory array. The slave then drives a reply byte on the MISO line to indicate that the command was observed (0xFE) or not (0xFF). Any additional read data bytes on the MISO line are read from the memory array at locations indicated by the communicated EZ address. The EZ address is automatically incremented by the slave as bytes are read from the memory array. When the EZ address exceeds the maximum number of memory entries (32), it wraps around to 0.

Figure 15-7 illustrates the write of EZ address, write to a memory array and read from a memory array operations in the EZSPI protocol.

Figure 15-7. EZSPI Example



15.2.4.4 Configuring SCB for EZSPI Mode

By default, the SCB is configured for non-EZ mode of operation. To configure the SCB for EZSPI mode, set various register bits in the following order:

1. Select EZ mode by writing '1' to the EZ_MODE bit (bit 10) of the SCBx_CTRL register.
2. Follow steps 2 to 4 from Enabling and Initializing SPI on page 76.
3. Use continuous transmission mode for transmitter by writing '1' to the CONTINUOUS bit of SCBx_SPI_CTRL register.
4. EZSPI mode is applicable only for slave functionality (write '0' to the MASTER_MODE field, bit 31 of SCBx_SPI_CTRL register).
5. Set the data frame width eight bits long (write ‘0111’ to the DATA_WIDTH field, bits [3:0] of SCBx_TX_CTRL and SCBx_RX_CTRL registers).
6. Set the shift direction as MSB first (write '1' to the MSB_FIRST field, bit 8 of SCBx_TX_CTRL and SCBx_RX_CTRL registers).

For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

15.2.5 SPI Registers

The SPI interface is controlled using a set of 32-bit control and status registers listed in Table 15-1. For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

Table 15-1. SPI Registers

Register Name	Operation
SCBx_CTRL	Enables the SCB, selects the type of serial interface (SPI, UART, I2C), and selects internally and externally clocked operation, EZ and non-EZ modes of operation.
SCBx_STATUS	In EZ mode, this register indicates whether the externally clocked logic is potentially using the EZ memory.
SCBx_SPI_CTRL	Configures the SPI as either a master or a slave, selects SPI protocols (Motorola, TI, National) and clock-based submodes in Motorola SPI (modes 0,1,2,3), selects the type of SELECT signal in TI SPI.
SCBx_SPI_STATUS	Indicates whether the SPI bus is busy and sets the SPI slave EZ address in the internally clocked mode.
SCBx_TX_CTRL	Enables the transmitter, specifies the data frame width, and specifies whether MSB or LSB is the first bit in transmission.
SCBx_RX_CTRL	Performs the same function as that of the SCBx_TX_CTRL register, but for the receiver.
SCBx_TX_FIFO_CTRL	Specifies the trigger level, clears the transmitter FIFO and shift registers, and performs the FREEZE operation of the transmitter FIFO.
SCBx_RX_FIFO_CTRL	Performs the same function as that of the SCBx_TX_FIFO_CTRL register, but for the receiver.
SCBx_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCBx_RX_FIFO_RD	Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO – behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCBx_RX_FIFO_RD_SILENT	Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCBx_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCBx_RX_FIFO_STATUS	Performs the same function as that of the SCBx_TX_FIFO_STATUS register, but for the receiver.
SCBx_EZ_DATAn	Holds the data in EZ memory location (n: 0–31)

15.2.6 SPI Interrupts

The SPI supports both internal and external interrupt requests. The internal interrupt events are listed here.

The SPI predefined interrupts can be classified as TX interrupts and RX interrupts. The TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The RX interrupt output is the logical OR of the group of all possible RX interrupt sources. This signal goes high when any of the enabled RX interrupt sources are true. Various interrupt registers are used to determine the actual source of the interrupt.

The SPI supports interrupts on the following events:

- SPI transfer done
- SPI is Idle
- TX FIFO is not full
- TX FIFO is empty
- SPI byte/word transfer is complete
- RX FIFO is empty
- RX FIFO is not empty
- Attempt to write to a full RX FIFO.
- RX FIFO is full

15.2.7 Enabling and Initializing SPI

The SPI must be programmed in the following order:

1. Program protocol specific information using the SCBx_SPI_CTRL register, according to Table 15-2. This includes selecting the submodes of the protocol and selecting master-slave functionality.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL registers, as shown in Table 15-3:
   a. Specify the data frame width.
   b. Specify whether MSB or LSB is the first bit to be transmitted/received.
   c. Enable the transmitter and receiver.
3. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL registers respectively, as shown in Table 15-4:
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFO.
4. Program the SCBx_CTRL register to enable the SCB block. Also select the mode of operation. These register bits are shown in Table 15-5.

.

Table 15-2. SCBx_SPI_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	SPI Motorola submode
		01	SPI Texas Instruments submode
		10	SPI National Semiconductors submode
		11	Reserved
31	MASTER_MODE	0	Master mode
		1	Slave mode

Table 15-3. SCBx_TX_CTRL/SCBx_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_WIDTH	'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits.
8	MSB_FIRST	1= MSB first 0= LSB first

Table 15-4. SCBx_TX_FIFO_CTRL/ SCBx_RX_FIFO_CTRL Registers

Bits	Name	Description
[2:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer.

Table 15-5. SCBx_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	I2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block enabled
		1	SCB block disabled

After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from Motorola mode to TI mode) or to go from externally to internally clocked operation (explained in Internally and Externally Clocked SPI Operations on page 77). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example, FIFO content).

The last step of initialization should always be to enable the block (write a '1' to the ENABLED bit of the SCBx_CTRL register).

15.2.8 Internally and Externally Clocked SPI Operations

The SCB supports both internally and externally clocked operations for SPI and I^2C functions. An internally clocked operation uses a clock provided by the device. An externally clocked operation uses a clock provided by the serial interface. Externally clocked operation enables operation in the Deep-Sleep system power mode, in which an on-chip internal clock is provided to the block.

Internally clocked operation uses the high-frequency clock of the system. For more information on system clocking, see the Clocking System chapter on page 47. It also supports oversampling. Oversampling is implemented with respect to the high-frequency clock. The OVS (bits [3:0]) of the SCBx_CTRL register specify the oversampling.

In SPI master mode, the valid range for oversampling is 4 to 16. Hence, the maximum bit rate is 12 Mbps. However, if you consider the I/O cell and routing delays, the effective oversampling range becomes 6 to 16. So, the maximum bit rate is 8 Mbps. **Note** LATE_MISO_SAMPLE must be set to '1' in SPIM mode.

In SPI slave mode, the oversampling field (bits [3:0]) of SCBx_CTRL register is not used. However, there is a frequency requirement for the SCB clock with respect to the interface clock (SCLK). This requirement is expressed in terms of the ratio (SCB clock/SCLK). This ratio is dependent on two fields: MEDIAN of SCBx_RX_CTRL register and LATE_MISO_SAMPLE of SCBx_CTRL register. With the MEDIAN bit set to ‘0’ and LATE_MISO_SAMPLE bit set to ‘1’, the SCB can achieve a maximum bit rate of 16 Mbps. However, if you consider the I/O cell and routing delays, the maximum data rate that can be achieved becomes 8 Mbps. Based on these bits, the maximum bit rates are given in Table 15-6.

Table 15-6. SPI Slave Maximum Data Rates

Median of SCBx_RX_CTRL	LATE_MISO_SAMPLE of SCBx_CTRL	Ratio Requirement	Maximum Bit Rate at Peripheral Clock of 48 MHz
0	0	≥12	4 Mbps
0	1	≥6	8 Mbps
1	0	≥16	3 Mbps
1	1	≥8	6 Mbps

Externally clocked operation is limited to:

- Slave functionality.
- EZ functionality. EZ functionality uses the block's SRAM as a memory structure. Non-EZ functionality uses the block's SRAM as TX and RX FIFOs; FIFO support is not available in externally clocked operation.
- Motorola mode 0 (in the case of SPI slave functionality).

Externally clocked EZ mode of operation can support a data rate of 48 Mbps (at a peripheral clock of 48 MHz).

Internally and externally clocked operation is determined by two register fields of the SCBx_CTRL register:

- EC_AM_MODE: Indicates whether SPI slave selection is internally ('0') or externally ('1') clocked. SPI slave selection comprises the first part of the protocol.
- EC_OP_MODE: Indicates whether the remaining protocol operation (besides SPI slave selection) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does NOT support non-EZ functionality.

These two register fields determine the functional behavior of SPI. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep system power mode. Improper setting may result in faulty behavior in certain system power modes. Table 15-7 and Table 15-8 describe the settings for SPI (in EZ and non-EZ mode).

15.2.8.1 Non-EZ Mode of Operation

In non-EZ mode there are two possible settings. As externally clocked operation is not supported for non-EZ functionality (no FIFO support), EC_OP_MODE should always be set to '0'. However, EC_AM_MODE can be set to '0' or '1'. Table 15-7 gives an overview of the possibilities. The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

Table 15-7. SPI Non-EZ Mode

SPI, Standard (non-EZ) Mode				
System Power Mode	EC_OP_MODE = 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in the Sleep mode, the MASK bit can be configured by the user. After that, selection using internal clock. Operation using internal clock.	Not supported	Invalid configuration
Deep-Sleep	Not supported	Selection using external clock: Wakeup interrupt cause in enabled (MASK = 1). Generate 0xff bytes.	Not supported	

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active and Sleep system power modes. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active and Sleep system power modes and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by both the internally and externally clocked logic: in Active mode both are active and in Deep Sleep mode only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active mode, the CPU and the block's internally clocked slave selection logic are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in the Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked logic takes care of the ongoing SPI transfer.
- In Deep Sleep mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged (‘1’ bits or “0xFF” bytes are send out on the MISO line) and the internally clocked logic takes care of the next SPI transfer when it is woken up.

15.2.8.2 EZ Mode of Operation

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. Table 15-8 gives an overview of the possibilities. The gray cells indicate a possible, yet not recommended, setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

Table 15-8. SPI EZ Mode

SPI, EZ Mode				
System Power Mode	EC_OP_MODE = 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
Active and Sleep	Selection using internal clock. Operation using internal clock.	Selection using external clock: Wakeup interrupt cause is disabled in Active mode (MASK = 0) and in Sleep mode, the mask bit can be configured by the user. After that, selection using internal clock. Operation using internal clock.	Selection using external clock. Operation using external clock.	Invalid configuration
Deep-Sleep	Not supported	Selection using external clock: Wakeup interrupt cause is enabled (MASK = 1). Generate 0xff bytes.	Selection using external clock. Operation using external clock.	

EC_OP_MODE is '0' and EC_AM_MODE is '0': This setting only works in Active system power mode. The entire block's functionality is provided in the internally clocked domain.

EC_OP_MODE is '0' and EC_AM_MODE is '1': This setting works in Active system power mode and provides limited (wake up) functionality in Deep-Sleep system power mode. SPI slave selection is performed by both the internally and externally clocked logic: in Active mode both are active and in Deep Sleep mode only the externally clocked logic is active. When the externally clocked logic detects slave selection, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wake up the CPU.

- In Active mode, the CPU and the block's internally clocked slave selection logic are active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). But in Sleep mode, wakeup interrupt cause can be either enabled or disabled (MASK bit can be either '1' or '0') based on the application. The remaining operations in the Sleep mode are same as that of the Active mode. The internally clocked logic takes care of the ongoing SPI transfer.
- In Deep Sleep mode, the CPU needs to be woken up and the wakeup interrupt cause is enabled (MASK bit is '1'). Waking up takes time, so the ongoing SPI transfer is negatively acknowledged ('1' bits or "0xFF" bytes are send out on the MISO line) and the internally clocked logic takes care of the next SPI transfer when it is woken up.

EC_OP_MODE is '1' and EC_AM_MODE is '1': This setting works in Active and Deep-Sleep system power modes. The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The field FIFO_BLOCK of the SCBx_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

15.3 UART

The Universal Asynchronous Receiver/Transmitter (UART) protocol is an asynchronous serial interface protocol. UART communication is typically point-to-point. The UART interface consists of two signals:

- TX: Transmitter output
- RX: Receiver input

15.3.1 Features

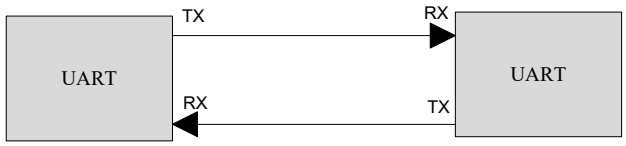
- Asynchronous transmitter and receiver functionality
- Supports a maximum data rate of 1 Mbps
- Supports UART protocol
  - Standard UART
  - SmartCard (ISO7816) reader.
  - IrDA
- Supports local interconnect network (LIN)
  - Break detection
  - Baud rate detection
  - Collision detection (ability to detect that a driven bit value is not reflected on the bus, indicating that another component is driving the same bus).
- Multi-processor mode
- Data frame size programmable from 4 bits to 16 bits.
- Programmable number of STOP bits, which can be set to 1, 1.5, or 2 data bits

- Parity support (odd and even parity)
- Interrupt or polling CPU interface
- Programmable oversampling

15.3.2 General Description

Figure 15-8 illustrates a standard UART TX and RX.

Figure 15-8. UART Example



A typical UART transfer consists of a "Start Bit" followed by multiple "Data Bits", optionally followed by a "Parity Bit" and finally completed by one or more "Stop Bits". The Start and Stop bits indicate the start and end of data transmission. The Parity bit is sent by the transmitter and is used be the receiver to detect single bit errors. As the interface does not have a clock (asynchronous), the transmitter and receiver use their own clocks; also, they need to agree upon the period of a bit transfer.

Three different serial interface protocols are supported:

- Standard UART protocol
  - Multi-Processor mode
  - LIN
- SmartCard, similar to UART, but with a possibility to send a negative acknowledgement
- IrDA, modification to the UART with a modulation scheme

By default, UART supports a data frame width of eight bits. However, this can be configured to any value in the range of 4 to 9. This does not include start, stop, and parity bits. The number of stop bits can be in the range of 1 to 3. The parity bit can be either enabled or disabled. If enabled, the type of parity can be set to either even parity or odd parity. The option of using the parity bit is available only in the Standard UART and SmartCard UART modes. For IrDA UART mode, the parity bit is automatically disabled. Figure 15-9 depicts the default configuration of the UART interface of the SCB.

Note UART interface does not support external clocking operation. Hence, UART operates only in the Active and Sleep power modes.

15.3.3 UART Modes of Operation

15.3.3.1 Standard Protocol

A typical UART transfer consists of a start bit followed by multiple data bits, optionally followed by a parity bit and finally completed by one or more stop bits. The start bit value is always '0', the data bits values are dependent on the data transferred, the parity bit value is set to a value guaranteeing an even or odd parity over the data bits, and the stop bits value is '1'. The parity bit is generated by the transmitter and can be used by the receiver to detect single bit transmission errors. When not transmitting data, the TX line is '1' – the same value as the stop bits.

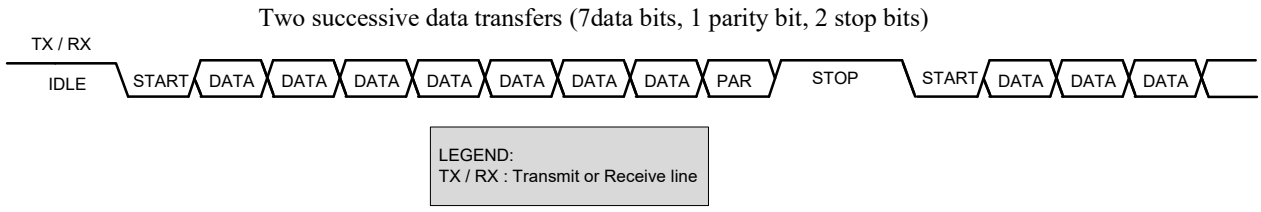
Because the interface does not have a clock, the transmitter and receiver need to agree upon the period of a bit transfer. The transmitter and receiver have their own internal clocks. The receiver clock runs at a higher frequency than the bit transfer frequency, such that the receiver may oversample the incoming signal.

The transition of a stop bit to a start bit is represented by a change from '1' to '0' on the TX line. This transition can be used by the receiver to synchronize with the transmitter clock. Synchronization at the start of each data transfer allows error-free transmission even in the presence of frequency drift between transmitter and receiver clocks. The required clock accuracy is dependent on the data transfer size.

The stop period or the amount of stop bits between successive data transfers is typically agreed upon between transmitter and receiver, and is typically in the range of 1 to 3-bit transfer periods.

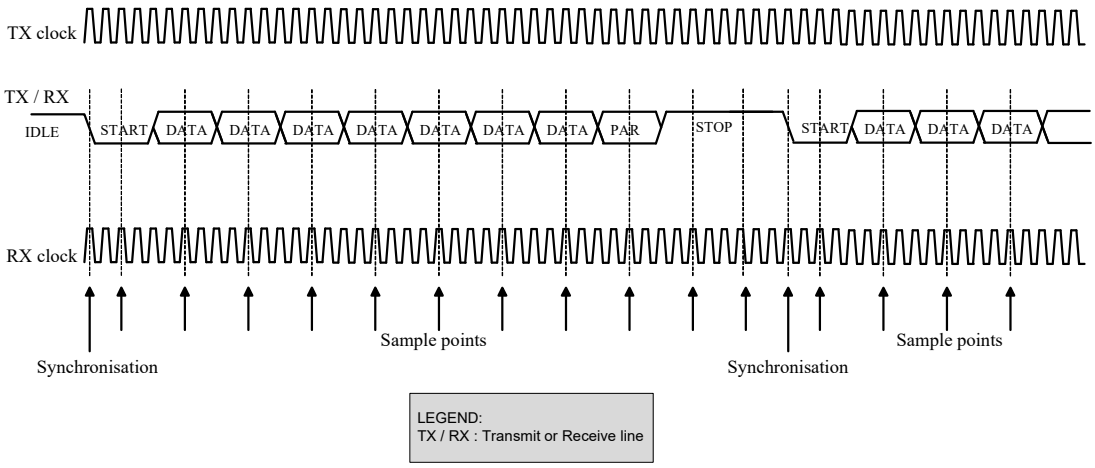
Figure 15-9 illustrates the UART protocol.

Figure 15-9. UART, Standard Protocol Example



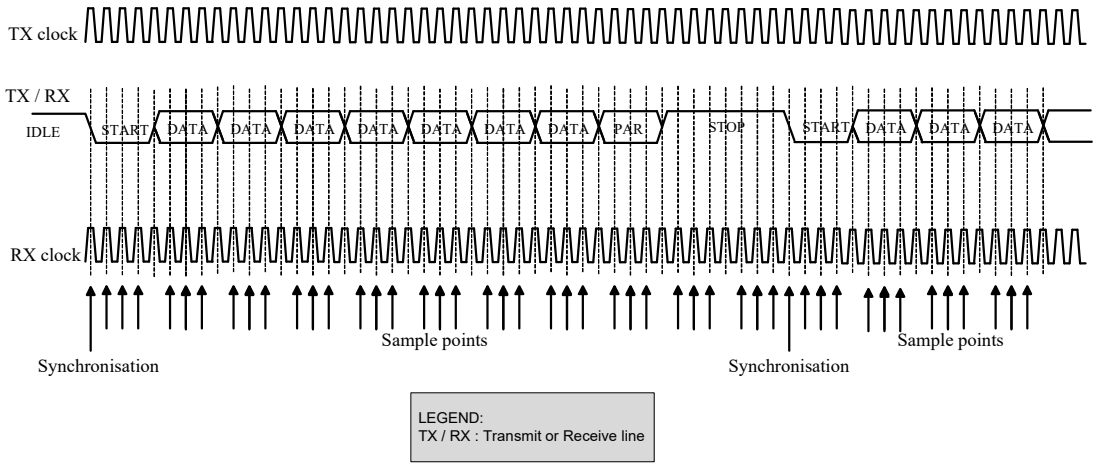
The receiver oversamples the incoming signal; the value of the sample point in the middle of the bit transfer period (on the receiver's clock) is used. Figure 15-10 illustrates this.

Figure 15-10. UART, Standard Protocol Example (Single Sample)



Alternatively, three samples around the middle of the bit transfer period (on the receiver's clock) are used for a majority vote to increase accuracy. Figure 15-11 illustrates this.

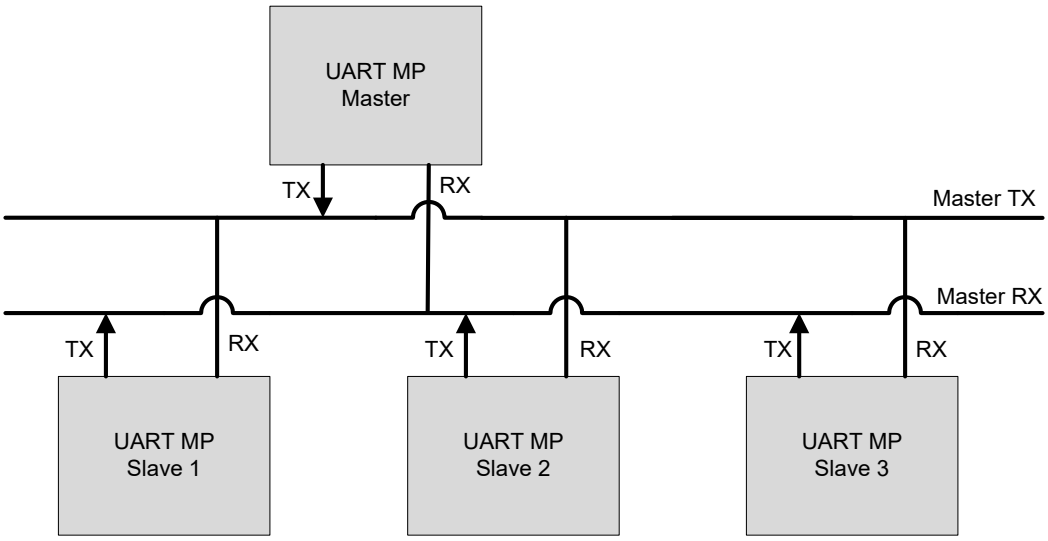
Figure 15-11. UART, Standard Protocol (Multiple Samples)



UART Multi-Processor Mode

The UART_MP (multi-processor) mode is defined with single-master-multi-slave topology, as shown in Figure 15-12. This mode is also known as UART 9-bit protocol because the data field is nine bits wide. UART_MP is part of Standard UART mode.

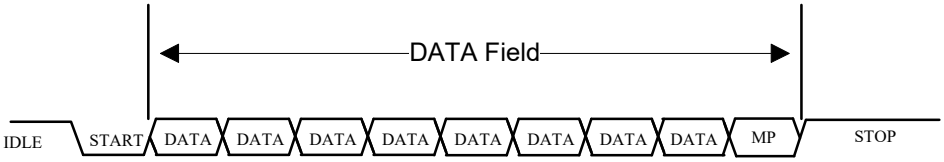
Figure 15-12. UART MP Mode Bus Connections



The main properties of UART_MP mode are:

- Single master with multiple slave concept (multi-drop network)
- Each slave is identified by a unique address
- Using 9-bit data field, with the ninth bit as address/data flag (MP bit). When set high, it indicates an address byte; when set low it indicates a data byte. A data frame is illustrated in Figure 15-13
- Parity bit is disabled

Figure 15-13. UART MP Data Frame

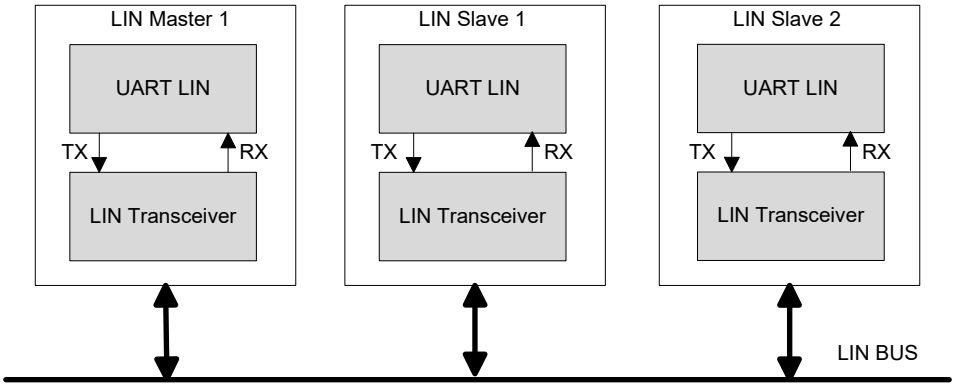


The SCB can be used as either master or slave device in UART_MP mode. Both SCBx_TX_CTRL and SCBx_RX_CTRL registers should be set to 9-bit data frame size. When the SCB works as UART_MP master device, the firmware changes the MP flag for every address or data frame. When it works as UART_MP slave device, the MP_MODE field of the SCBx_UART_RX_CTRL register should be set to '1'. The SCBx_RX_MATCH register should be set for the slave address and address mask. The matched address is written in the RX_FIFO when SCB_ADDRESS_ACCEPT field of the SCBx_CTRL register is set to '1'. If received address does not match its own address, then the interface ignores the following data, until next address is received for compare.

UART LIN Mode

The LIN protocol is supported by the SCB as part of the standard UART. LIN is designed with single-master-multi-slave topology. There is one master node and multiple slave nodes on the LIN bus. The SCB UART supports both LIN master and slave functionality. Figure 15-14 illustrates the UART_LIN and LIN transceiver.

Figure 15-14. UART_LIN and LIN Transceiver

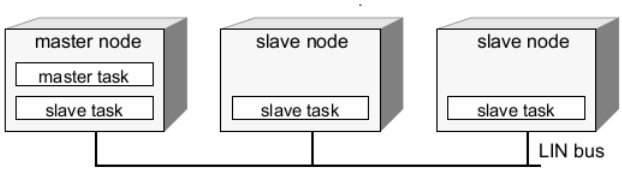


LIN protocol defines two tasks:

- Master task: This task involves sending a header packet to initiate a LIN transfer.
- Slave task: This task involves transmitting or receiving a response.

The master node supports master task and slave task; the slave node supports only slave task, as shown in Figure 15-15.

Figure 15-15. LIN Bus Nodes and Tasks



LIN is based on the transmission of frames at predetermined moments of time. A frame is divided into header and response fields.

- The header field consists of:
  - Break field (at least 13 bit periods with the value '0').
  - Sync field (a 0x55 byte frame). A sync field can be used to synchronize the clock of the slave task with that of the master task.
  - Identifier field (a frame specifying a specific slave).
- The response field consists of data and checksum.

The UART LIN of SCB supports slave task, receiving the header and transmitting the response. It provides baud rate detection (using sync field – 0x55) operation. Apart from the break field, a frame transmission (both header and response) consist of one or multiple byte frame transmissions, with each byte transmission consisting of a start bit, eight data bits and one or more stop bits (on both the UART TX and RX lines).

To support LIN, a dedicated (off-chip) line driver/receiver is required. Supply voltage range on the LIN bus is 7 V to 18 V. Typically, LIN line drivers will drive the LIN line with the value provided on the SCB TX line and present the value on the LIN line to the SCB RX line. By comparing TX and RX lines in the SCB, bus collisions can be detected (indicated by the UART_ARB_LOST field of the SCBx_INTR_TX register).

Configuring the SCB as Standard UART interface

To configure the SCB as a standard UART interface, set various register bits in the following order:

1. Configure the SCB as UART interface by writing '10' to the MODE field (bits [25:24]) of the SCBx_CTRL register.
2. Configure the UART interface to operate as a Standard protocol by writing '00' to the MODE field (bits [25:24]) of the SCBx_UART_CTRL register.
3. To enable the UART MP Mode or UART LIN Mode, write '1' to the MP_MODE (bit 10) or SCB_LIN_MODE (bit 12) respectively of the SCBx_UART_RX_CTRL register.
4. Follow steps 2 to 4 described in Enabling and Initializing UART on page 86.

For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

15.3.3.2 SmartCard (ISO7816)

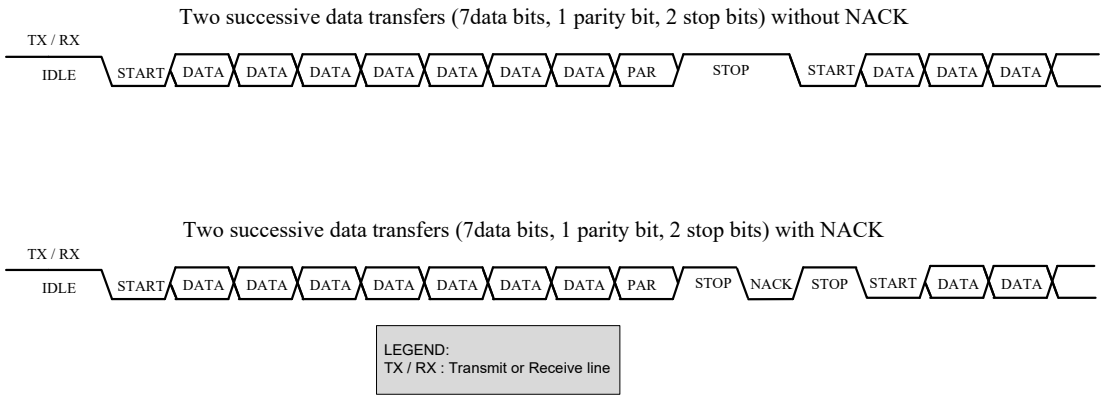
ISO7816 is asynchronous serial interface, defined with single-master-single slave topology. ISO7816 defines both Reader (master) and Card (slave) functionality. For more information, refer to the ISO7816 Specification. Only master (reader) function is supported by the SCB. This block provides the basic physical layer support with asynchronous character transmission. UART_TX line is connected to SmartCard I/O line, by internally multiplexing between UART_TX and UART_RX control modules.

The SmartCard transfer is similar to a UART transfer, with the addition of a negative acknowledgement (NACK) that may be sent from the receiver to the transmitter. A NACK is always '0'. Both master and slave may drive the same line, although never at the same time.

A SmartCard transfer has the transmitter drive the start bit and data bits (and optionally a parity bit). After these bits, it enters its stop period by releasing the bus. Releasing results in the line being '1' (the value of a stop bit). After one bit transfer period into the stop period, the receiver may drive a NACK on the line (a value of '0') for one bit transfer period. This NACK is observed by the transmitter, which reacts by extending its stop period by one bit transfer period. For this protocol to work, the stop period should be longer than one bit transfer period. Note that a data transfer with a NACK takes one bit transfer period longer, than a data transfer without a NACK. Typically, implementations use a tristate driver with a pull-up resistor, such that when the line is not transmitting data or transmitting the Stop bit, its value is '1'.

Figure 15-16 illustrates the SmartCard protocol.

Figure 15-16. SmartCard Example



The communication Baud rate for ISO7816 is given as:

Baud rate= f7816 × (D/F)

Where f7816 is the clock frequency, F is the clock rate conversion integer, and D is the baud rate adjustment integer.

By default, F = 372, D = f1, and the maximum clock frequency is 5 MHz. Thus, maximum baud rate is 13.4 Kbps. Typically, a 3.57-MHz clock is selected. The typical value of the baud rate is 9.6 Kbps.

Configuring SCB as UART SmartCard Interface

To configure the SCB as a UART SmartCard interface, set various register bits in the following order. For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Configure the UART interface to operate as a SmartCard protocol by writing '01' to the MODE (bits [25:24]) of the SCBx-_UART_CTRL register.
3. Follow steps 2 to 4 described in Enabling and Initializing UART on page 86.

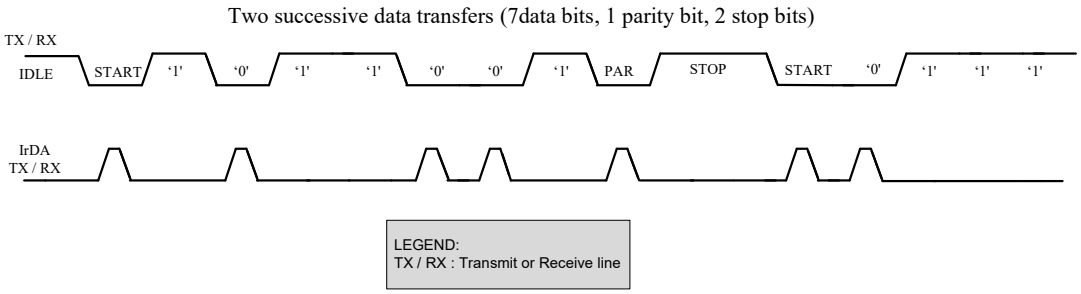
15.3.3.3 IrDA

The SCB supports the Infrared Data Association (IrDA) protocol for data rates of up to 115.2 Kbits/s using the UART interface. It supports only the basic physical layer of IrDA protocol with rates less than 115.2 Kbps. Hence, the system instantiating this block must consider how to implement a complete IrDA communication system with other available system resources.

The IrDA protocol adds a modulation scheme to the UART signaling. At the transmitter, bits are modulated. At the receiver, bits are demodulated. The modulation scheme uses a Return-to-Zero-Inverted (RZI) format. A bit value of '0' is signaled by a short '1' pulse on the line and a bit value of '1' is signaled by holding the line to '0'. For these data rates (<=115.2 Kbps), the RZI modulation scheme is used and the pulse duration is 3/16 of the bit period. The sampling clock frequency should be set 16 times the selected baud rate, by configuring the OVS field of the SCBx_CTRL register.

Different communication speeds under 115.2 Kb/s can be achieved by configuring corresponding block clock frequency. Additional allowable rates are 2.4 Kbps, 9.6 Kbps, 19.2 Kbps, 38.4 Kbps, and 57.6 Kbps. An IrDA serial infrared interface operates at 9.6 Kbps. Figure 15-17 shows how a UART transfer is IrDA modulated.

Figure 15-17. IrDA Example



Configuring the SCB as UART IrDA Interface

To configure the SCB as a UART IrDA interface, set various register bits in the following order. For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

1. Configure the SCB as UART interface by writing '10' to the MODE (bits [25:24]) of the SCBx_CTRL register.
2. Configure the UART interface to operate as IrDA protocol by writing '10' to the MODE (bits [25:24]) of the SCBx_UART_CTRL register.
3. Configure the SCB as described in Enabling and Initializing UART on page 86.

15.3.4 UART Registers

The UART interface is controlled using a set of 32-bit registers listed in Table 15-9. For more information on these registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

Table 15-9. UART Registers

Register Name	Operation
SCBx_UART_CTRL	Used to select the sub-modes of UART (standard UART, SmartCard, IrDA), also used for local loop back control.
SCBx_UART_TX_CTRL	Used to specify the number of stop bits, enable parity, select the type of parity, and enable retransmission on NACK.
SCBx_UART_RX_CTRL	Performs same function as SCBx_UART_TX_CTRL but is also used for enabling multi processor mode, LIN mode drop on parity error, and drop on frame error.
SCBx_TX_CTRL	Used to enable the transmitter, also to specify the data frame width and to specify whether MSB or LSB is the first bit in transmission.
SCBx_RX_CTRL	Performs the same function as that of the SCBx_TX_CTRL register, but for the receiver.

15.3.5 UART Interrupts

The UART supports both internal and external interrupt requests. The internal interrupt events are listed in this section.

The UART predefined interrupts can be classified as TX interrupts and RX interrupts. The TX interrupt output is the logical OR of the group of all possible TX interrupt sources. This signal goes high when any of the enabled TX interrupt sources are true. The RX interrupt output is the logical OR of the group of all possible RX interrupt sources. This signal goes high when any of the enabled RX interrupt sources are true. The UART provides interrupts on the following events:

- UART transmission is complete
- UART TX received a NACK in SmartCard mode
- UART arbitration lost (in LIN or SmartCard modes)
- Frame error in received data frame
- Parity error in received data frame
- LIN baud rate detection is complete
- LIN break detection is successful

15.3.6 Enabling and Initializing UART

The UART must be programmed in the following order:

1. Program protocol specific information using the SCBx_UART_CTRL register, according to Table 15-10. This includes selecting the submodes of the protocol, transmitter-receiver functionality, and so on.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL registers, as shown in Table 15-11.
   a. Specify the data frame width.
   b. Specify whether MSB or LSB is the first bit to be transmitted or received.
   c. Enable the transmitter and receiver.
3. Program the transmitter and receiver FIFOs using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL registers respectively, as shown in Table 15-12.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
   c. Freeze the TX and RX FIFOs.
4. Program the SCBx_CTRL register to enable the SCB block. Also select the mode of operation, as shown in Table 15-13.

Table 15-10. SCBx_UART_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	Standard UART
		01	SmartCard
		10	IrDA
		11	Reserved
16	LOOP_BACK	Loop back control. This allows a SCB UART transmitter to communicate with its receiver counterpart.	

Table 15-11. SCBx_TX_CTRL/SCBx_RX_CTRL Registers

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the no. of bits in the transmitted or received data frame. The valid range is [3, 15]. This does not include start, stop, and parity bits.
8	MSB_FIRST	1= MSB first 0= LSB first

Table 15-12. SCBx_TX_FIFO_CTRL/SCBx_RX_FIFO_CTRL Registers

Bits	Name	Description
[2:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared/invalidated.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze will not advance the TX or RX FIFO read/write pointer.

Table 15-13. SCBx_CTRL Register

Bits	Name	Value	Description
[25:24]	MODE	00	I2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block enabled
		1	SCB block disabled

After the block is enabled, control bits should not be changed. Changes should be made after disabling the block; for example, to modify the operation mode (from SmartCard to IrDA). The change takes effect only after the block is re-enabled. Note that re-enabling the block causes re-initialization and the associated state is lost (for example FIFO content).

The last step of initialization should always be to enable the block (write a '1' to the ENABLED bit of the SCBx_CTRL register).

15.4 Inter Integrated Circuit (I^2C)

CCG7D contains an SCB configured to operate as a I^2C block. This section explains the I^2C implementation in CCG7D. For more information on the I^2C protocol specification, refer to the I^2C-bus specification available on the NXP website.

15.4.1 Features

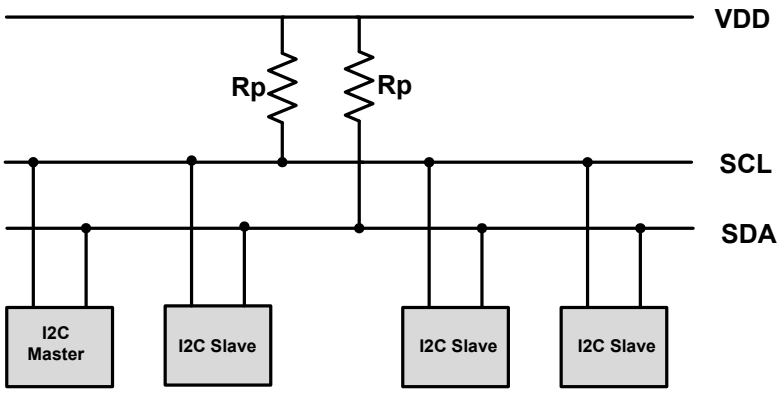
CCG7D supports the following I^2C features:

- Master, slave, and master/slave mode
- Slow-mode (50 Kbps), standard-mode (100 Kbps), fast-mode (400 Kbps), and fast-mode plus (1000 Kbps) data-rates
- 7- or 10-bit slave addressing (10-bit addressing requires firmware support)
- Clock stretching and collision detection
- Programmable oversampling of I2C clock signal (SCL)
- Error reduction using an digital median filter on the input path of the I^2C data signal (SDA)
- Glitch-free signal transmission with an analog glitch filter
- Interrupt or polling CPU interface
- Supports EZ mode of operation (EZI2C protocol – applicable only for I^2C slave functionality)

15.4.2 General Description

Figure 15-18 illustrates an example of an I^2C communication network.

Figure 15-18. I^2C Interface Block Diagram



The standard I^2C bus is a two wire interface with the following lines:

- Serial Data (SDA)
- Serial Clock (SCL)

I^2C devices are connected to these lines using open collector or open-drain output stages, with pull-up resistors (Rp). A simple master/slave relationship exists between devices. Masters and slaves can operate as either transmitter or receiver. Each slave device connected to the bus is software addressable by a unique 7-bit address.

15.4.3 Terms and Definitions

Table 15-14 describes the commonly used terms in an I^2C communication network.

Table 15-14. Definition of I^2C Bus Terminology

Term	Description
Transmitter	The device that sends data to the bus
Receiver	The device that receives data from the bus
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer

Table 15-14. Definition of I²C Bus Terminology <Italic>

Term	Description
Slave	The device addressed by a master
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

Bus Stalling (Clock Stretching)

When a slave device is not yet ready to process data, it may drive a '0' on the SCL line to hold it down. Due to the implementation of the I/O signal interface, the SCL line value will be '0', independent of the values that any other master or slave may be driving on the SCL line. This is known as clock stretching and is the only situation in which a slave drives the SCL line. The master device monitors the SCL line and detects it when it cannot generate a positive clock pulse ('1') on the SCL line. It then reacts by postponing the generation of a positive edge on the SCL line, effectively synchronizing with the slave device that is stretching the clock.

Bus Arbitration

The I²C protocol is a multi-master, multi-slave interface. Bus arbitration is implemented on master devices by monitoring the SDA line. Bus collisions are detected when the master observes an SDA line value that is not the same as the value it is driving on the SDA line. For example, when master 1 is driving the value '1' on the SDA line and master 2 is driving the value '0' on the SDA line, the actual line value will be '0' due to the implementation of the I/O signal interface. Master 1 detects the inconsistency and loses control of the bus. Master 2 does not detect any inconsistency and keeps control of the bus.

15.4.4 I²C Modes of Operation

I²C is a synchronous single master, multi-master, multi-slave serial interface. Devices operate in either master mode, slave mode, or master/slave mode. In master/slave mode, the device switches from master to slave mode when it is addressed. Only a single master may be active during a data transfer. The active master is responsible for driving the clock on the SCL line.

Table 15-15 illustrates the I²C modes of operation.

Table 15-15. I²C Modes

Mode	Description
Slave	Slave only operation (default)
Master	Master only operation
Multi-master	Supports more than one master on the bus
Multi-master-slave	Simultaneous slave and multi-master operation

Data transfer through the I²C bus follows a specific format. Table 15-16 lists some common bus events that are part of an I²C data transfer. The Write Transfer and Read Transfer sections explain the format of bits on an I²C bus during data transfer.

Table 15-16. I²C Bus Events Terminology

Bus Event	Description
START	A HIGH to LOW transition on the SDA line while SCL is HIGH
STOP	A LOW to HIGH transition on the SDA line while SCL is HIGH
ACK	The receiver pulls the SDA line LOW and it remains LOW during the HIGH period of the clock pulse after the transmitter transmits each byte.
NACK	The receiver does not pull the SDA line LOW and it remains HIGH during the HIGH period of clock pulse after the transmitter transmits each byte.
Repeated START	START condition generated by master at the end of a transfer instead of a STOP condition

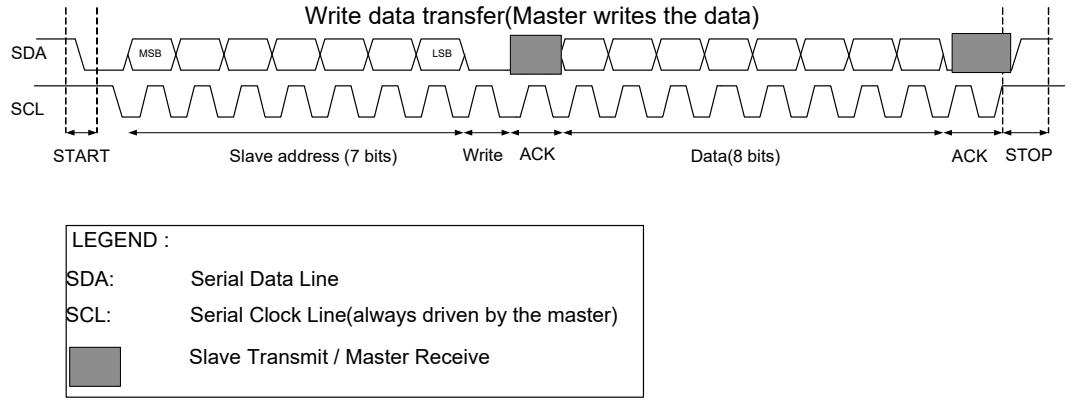
When operating in multi-master mode, the bus should always be checked to see if it is busy; another master may already be communicating with a slave. In this case, the master must wait until the current operation is complete before issuing a START signal (see Table 15-16, Figure 15-19 and Figure 15-20). The master looks for a STOP signal as an indicator that it can start its data transmission.

When operating in multi-master-slave mode, if the master loses arbitration during data transmission, the hardware reverts to slave mode and the received byte generates a slave address interrupt.

With all of these modes, there are two types of transfer – read and write. In write transfer, the master sends data to slave; in read transfer, the master receives data from slave. Write and read transfer examples are available in Master Mode Transfer Examples on page 95, Slave Mode Transfer Examples on page 97, and Multi-Master Mode Transfer Example on page 101.

15.4.4.1 Write Transfer

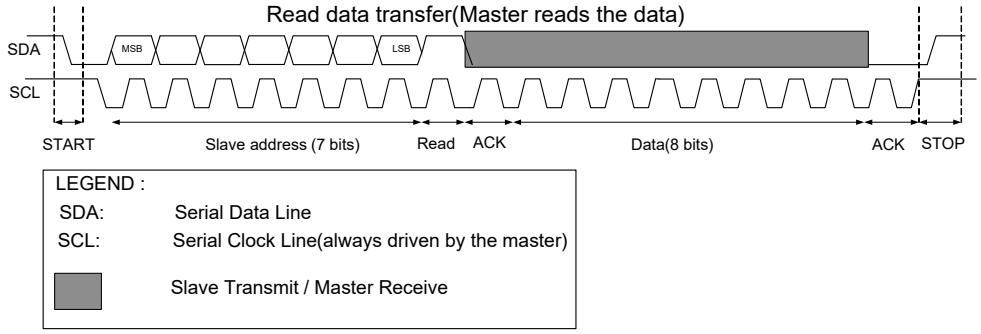
Figure 15-19. Master Write Data Transfer



- A typical write transfer begins with the master generating a START condition on the I^2C bus. The master then writes a 7-bit I^2C slave address and a write indicator ('0') after the START condition. The addressed slave transmits an acknowledgement byte by pulling the data line low during the ninth bit time.
- If the slave address does not match any of the slave devices or if the addressed device does not want to acknowledge the request, it transmits a no acknowledgement (NACK). The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the write transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- The master may transmit write data to the bus if it receives an acknowledgement. The addressed slave transmits an acknowledgement to confirm the receipt of the write data. Upon receipt of this acknowledgement, the master may transmit another data byte.
- When the transfer is complete, the master generates a STOP condition.

15.4.4.2 Read Transfer

Figure 15-20. Master Read Data Transfer



- A typical read transfer begins with the master generating a START condition on the I^2C bus. The master then writes a 7-bit I^2C slave address and a read indicator ('1') after the START condition. The addressed slave transmits an acknowledgement by pulling the data line low during the ninth bit time.
- If the slave address does not match that of the connected slave device or if the addressed device does not want to acknowledge the request, a no acknowledgement (NACK) is transmitted. The absence of an acknowledgement, results in an SDA line value of '1' due to the pull-up resistor implementation.
- If no acknowledgement is transmitted by the slave, the master may end the read transfer with a STOP event. The master can also generate a repeated START condition for a retry attempt.
- If the slave acknowledges the address, it starts transmitting data after the acknowledgement signal. The master transmits an acknowledgement to confirm the receipt of each data byte sent by the slave. Upon receipt of this

acknowledgement, the addressed slave may transmit another data byte.

- The master can send a NACK signal to the slave to stop the slave from sending data bytes. This completes the read transfer.
- When the transfer is complete, the master generates a STOP condition.

15.4.5 EZI2C Protocol

The Easy I2C (EZI2C) protocol is a unique communication scheme built on top of the I^{2}C protocol by Cypress. EZI2C mode is applicable only for slave functionality. It uses a software wrapper around the standard I^{2}C protocol to communicate to an I^{2}C slave using indexed memory transfers. This removes the need for CPU intervention at the level of individual frames.

The EZI2C protocol defines an 8-bit EZ address that indexes a memory array (8-bit wide 32 locations) located on the slave device. Five lower bits of the EZ address is used to address these 32 locations. The number of bytes transferred to or from the EZI2C memory array can be found by comparing the EZ address at the START event and the EZ address at the STOP event.

The I^{2}C block has a hardware FIFO memory, which is 16 bits wide and 16 locations deep with byte write enable. The access methods for EZ and non-EZ functions are different. In non-EZ mode, the FIFO is split into TXFIFO and RXFIFO. Each has 16-bit wide eight locations. In EZ mode, the FIFO is used as a single memory unit with 8-bit wide 32 locations.

EZI2C has two types of transfers: an EZ write of data from the master to an addressed slave memory location, and a read by the master from an addressed slave memory location.

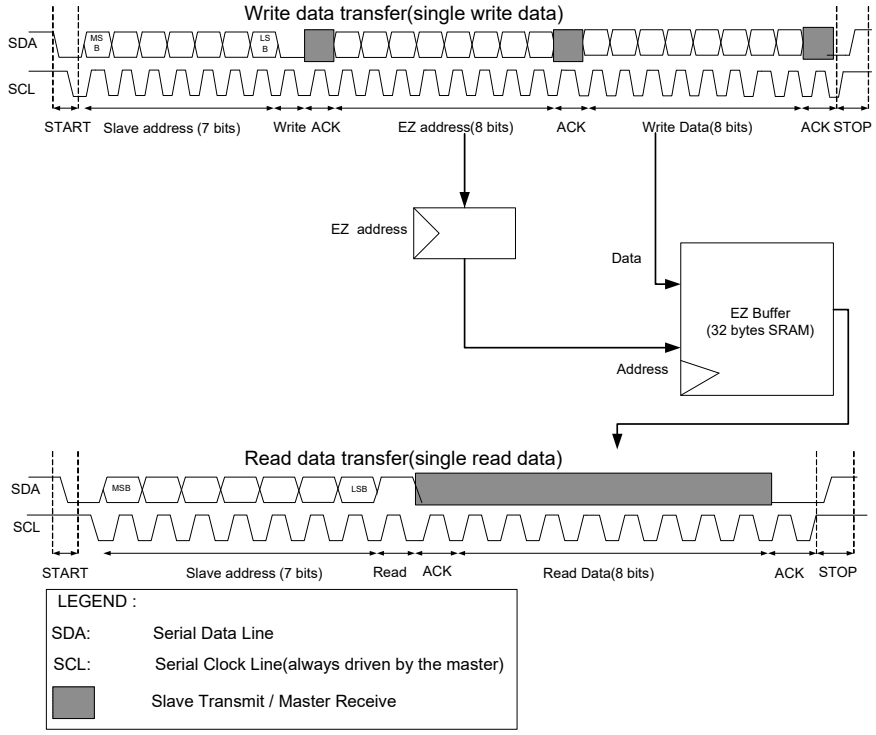
15.4.6 Memory Array Write

An EZ write to a memory array index is by means of an I^{2}C write transfer. The first transmitted write data is used to send an EZ address from the master to the slave. The five lowest significant bits of the write data are used as the "new" EZ address at the slave. Additional write data elements in the write transfer are bytes that are written to the memory array. The EZ address is automatically incremented by the slave as bytes are written into the memory array. When the EZ address exceeds 32 memory entries, it wraps around to 0.

15.4.7 Memory Array Read

An EZ read from a memory array index is by means of an I^{2}C read transfer. The EZ read relies on an earlier EZ write to have set the EZ address at the slave. The first received read data is the byte from the memory array at the EZ address memory location. The EZ address is automatically incremented as bytes are read from the memory array. When the EZ address exceeds the 32 memory entries, it wraps around to 0.

Figure 15-21. EZI2C Write and Read Data Transfer



See EZ Slave Mode Transfer Example on page 99 for examples.

15.4.8 I²C Registers

The I²C interface is controlled by reading and writing a set of configuration, control, and status registers, as listed in Table 15-17.

Table 15-17. I²C Registers

Register	Function
SCBx_CTRL	Enables the I2C block and selects the type of serial interface (SPI, UART, I2C). Also used to select internally and externally clocked operation and EZ and non-EZ modes of operation.
SCBx_I2C_CTRL	Selects the mode (master, slave) and sends an ACK or NACK signal based on receiver FIFO status.
SCBx_I2C_STATUS	Indicates bus busy status detection, read/write transfer status of the slave/master, and stores the EZ slave address.
SCBx_I2C_M_CMD	Enables the master to generate START, STOP, and ACK/NACK signals.
SCBx_I2C_S_CMD	Enables the slave to generate ACK/NACK signals.
SCBx_STATUS	Indicates whether the externally clocked logic is using the EZ memory. This bit can be used by software to determine whether it is safe to issue a software access to the EZ memory.
SCBx_I2C_CFG	Configures filters, which removes glitches from the SDA and SCL lines.
SCBx_TX_CTRL	Enables the transmitter and specifies the data frame width; also used to specify whether MSB or LSB is the first bit in transmission.
SCBx_TX_FIFO_CTRL	Specifies the trigger level, clearing of the transmitter FIFO and shift registers, and FREEZE operation of the transmitter FIFO.
SCBx_TX_FIFO_STATUS	Indicates the number of bytes stored in the transmitter FIFO, the location from which a data frame is read by the hardware (read pointer), the location from which a new data frame is written (write pointer), and decides if the transmitter FIFO holds the valid data.
SCBx_TX_FIFO_WR	Holds the data frame written into the transmitter FIFO. Behavior is similar to that of a PUSH operation.
SCBx_RX_CTRL	Performs the same function as that of the SCBx_TX_CTRL register, but for the receiver.
SCBx_RX_FIFO_CTRL	Performs the same function as that of the SCBx_TX_FIFO_CTRL register, but for the receiver.
SCBx_RX_FIFO_STATUS	Performs the same function as that of the SCBx_TX_FIFO_STATUS register, but for the receiver.
SCBx_RX_FIFO_RD	Holds the data read from the receiver FIFO. Reading a data frame removes the data frame from the FIFO; behavior is similar to that of a POP operation. This register has a side effect when read by software: a data frame is removed from the FIFO.
SCBx_RX_FIFO_RD_SILENT	Holds the data read from the receiver FIFO. Reading a data frame does not remove the data frame from the FIFO; behavior is similar to that of a PEEK operation.
SCBx_RX_MATCH	Stores slave device address and is also used as slave device address MASK.
SCBx_EZ_DATAn	Holds the data in an EZ memory location (n: 0–31).

Note Detailed descriptions of the I²C register bits are available in the CCG7D Family EZ-PD CCG7D Registers TRM.

15.4.9 I²C Interrupts

The I²C block generates interrupts for the following conditions.

- Arbitration lost
- Slave address match
- I²C bus stop/start condition detected
- I²C bus error detected
- I²C byte/word transfer complete
- I²C TX FIFO not full
- I²C TX FIFO empty
- I²C RX FIFO empty
- I²C RX FIFO not empty
- I²C RX FIFO overrun
- I²C RX FIFO full

The I²C interrupt signal is hard-wired to the Cortex-M0 NVIC and cannot be routed to external pins.

The interrupt output is the logical OR of the group of all possible interrupt sources. The interrupt is triggered when any of the enabled interrupt conditions are met. Interrupt status registers are used to determine the actual source of the interrupt. For more information on interrupt registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

15.4.10 Enabling and Initializing the I²C

The following section describes the method to configure the I²C block for standard (non-EZ) mode and EZI2C mode.

Configuring for I²C Standard (Non-EZ) Mode

The I²C interface must be programmed in the following order.

1. Program protocol specific information using the SCBx_I2C_CTRL register according to Table 15-18. This includes selecting master-slave functionality.
2. Program the generic transmitter and receiver information using the SCBx_TX_CTRL and SCBx_RX_CTRL registers, as shown in Table 15-19.
   a. Specify the data frame width.
   b. Specify whether MSB or LSB is the first bit to be transmitted/received.
   c. Enable the transmitter and receiver.
3. Program transmitter and receiver FIFO using the SCBx_TX_FIFO_CTRL and SCBx_RX_FIFO_CTRL registers, respectively, as shown in Table 15-20.
   a. Set the trigger level.
   b. Clear the transmitter and receiver FIFO and Shift registers.
4. Program the SCBx_CTRL register to enable the SCB block and select the I²C mode. These register bits are shown in Table 15-21. For a complete description of the I²C registers, see the CCG7D Family EZ-PD CCG7D Registers TRM.

Table 15-18. SCBx_I2C_CTRL Register

Bits	Name	Value	Description
30	SLAVE_MODE	1	Slave mode
31	MASTER_MODE	1	Master mode

Table 15-19. SCBx_TX_CTRL/SCB_RX_CTRL Register

Bits	Name	Description
[3:0]	DATA_ WIDTH	'DATA_WIDTH + 1' is the number of bits in the transmitted or received data frame. The valid range is [3, 15].
8	MSB_FIRST	1= MSB first
		0= LSB first

Table 15-20. SCBx_TX_FIFO_CTRL/ SCBx_RX_FIFO_CTRL

Bits	Name	Description
[2:0]	TRIGGER_LEVEL	Trigger level. When the transmitter FIFO has less entries or the receiver FIFO has more entries than the value of this field, a transmitter or receiver trigger event is generated in the respective case.
16	CLEAR	When '1', the transmitter or receiver FIFO and the shift registers are cleared.
17	FREEZE	When '1', hardware reads/writes to the transmitter or receiver FIFO have no effect. Freeze does not advance the TX or RX FIFO read/write pointer.

Table 15-21. SCBx_CTRL Registers

Bits	Name	Value	Description
[25:24]	MODE	00	I2C mode
		01	SPI mode
		10	UART mode
		11	Reserved
31	ENABLED	0	SCB block enabled
		1	SCB block disabled

Configuring for EZI2C Mode

To configure the SCB block for EZI2C mode, set the following I²C register bits

1. Select the EZI2C mode by writing '1' to the EZ_MODE bit (bit 10) of the SCBx_CTRL register.
2. Follow steps 2 to 4 from Configuring for I2C Standard (Non-EZ) Mode.
3. Set the S_READY_ADDR_ACK (bit 12) and S_READY_DATA_ACK (bit 13) bits of the SCBx_I2C_CTRL register.

15.4.11 Internal and External Clock Operation in I²C

The I2C supports both internally and externally clocked operation for data-rate generation. Internally clocked operations use a clock signal derived from the CCG7D system bus clock. Externally clocked operations use a clock provided by the user. Externally clocked operation allows limited functionality in the Deep-Sleep power mode, in which on-chip clocks are not active. For more information on system clocking, see the Clocking System chapter on page 47.

Externally clocked operation is limited to the following cases:

- Slave functionality.
- EZ functionality. TX and RX FIFOs do not support externally clocked operation; therefore, it is not used for non-EZ functionality.

Internally and externally clocked operations are determined by two register fields of the SCBx_CTRL register:

- **EC_AM_MODE (Externally Clocked Address Matching Mode)**: Indicates whether I²C address match is internally ('0') or externally ('1') clocked.
- **EC_OP_MODE (Externally Clocked Operation Mode)**: Indicates whether the remaining protocol operation (besides I²C address match) is internally ('0') or externally ('1') clocked. As mentioned earlier, externally clocked operation does not support non-EZ functionality.

These two register fields determine the functional behavior of I²C. The register fields should be set based on the required behavior in Active, Sleep, and Deep-Sleep power modes. Improper setting may result in faulty behavior in certain power modes. Table 15-22 and Table 15-22 describe the settings for I²C in EZ and non-EZ mode.

15.4.11.1 I²C Non-EZ Operation Mode

Externally clocked operation is not supported for non-EZ functionality because there is no FIFO support for this mode. So, the EC_OP_MODE should always be set to '0' for non-EZ mode. However, EC_AM_MODE can be set to '0' or '1'. Table 15-9 gives an overview of the possibilities. The combination EC_AM_MODE = 0 and EC_OP_MODE = 1 is invalid and the block will not respond.

EC_AM_MODE is '0' and EC_OP_MODE is '0'. This setting only works in Active and Sleep power modes. All the I²C's functionality is provided in the internally clocked domain.

EC_AM_MODE is '1' and EC_OP_MODE is '0'. This setting works in Active and Deep-Sleep power modes. I²C address match is performed by the externally clocked logic in both these modes. When the externally clocked logic matches the address, it sets a wakeup interrupt cause bit, which can be used to generate an interrupt to wakeup the CPU.

- In Active mode, the CPU is active and the wakeup interrupt cause is disabled (associated MASK bit is '0'). The externally clocked logic takes care of the address match and the internally locked logic takes care of the rest of the I²C transfer.
- In Sleep mode, wakeup interrupt cause can be either enabled or disabled based on the application. The remaining operations are similar to the Active mode.
- In Deep Sleep mode, the CPU is shut down and will wake up on I²C activity if the wakeup interrupt cause is enabled. CPU wakeup takes time and the ongoing I2C transfer is either negatively acknowledged (NACK) or the clock is stretched. In the case of a NACK, the internally clocked logic takes care of the first I²C transfer after it wakes up. For clock stretching, the internally clocked logic takes care of the ongoing/stretched transfer when it wakes up. The register bit S_NOT_READY_ADDR_NACK (bit 14) of the SCBx_I2C_CTRL register determines whether the externally clocked logic performs a negative acknowledge ('1') or clock stretch ('0').

15.4.11.2 EZ Operation Mode

EZ mode has three possible settings. EC_AM_MODE can be set to '0' or '1' when EC_OP_MODE is '0' and EC_AM_MODE must be set to '1' when EC_OP_MODE is '1'. Table 15-22 gives an overview of the possibilities. The cells in grey indicate a possible, yet not recommended setting because it involves a switch from the externally clocked logic (slave selection) to the internally clocked logic (rest of the operation). The combination EC_AM_MODE=0 and EC_OP_MODE=1 is invalid and the block will not respond.

Table 15-22. I²C EZ Mode

I2C, EZ Mode				
System Power Mode	EC_OP_MODE= 0		EC_OP_MODE = 1	
	EC_AM_MODE = 0	EC_AM_MODE = 1	EC_AM_MODE = 1	EC_AM_MODE=0
Active and Sleep	Address match using internal clock Operation using internal clock	Address match using external clock Operation using internal clock	Address match using external clock Operation using external clock	Invalid configuration
Deep-Sleep	Not supported	Address match using external clock Operation using internal clock	Address match using external clock Operation using external clock	

- **EC_AM_MODE is '0' and EC_OP_MODE is '0'**. This setting only works in Active and Sleep power modes.
- **EC_AM_MODE is '1' and EC_OP_MODE is '0'**. This setting works similar to the I²C non-EZ mode.
- **EC_AM_MODE is '1' and EC_OP_MODE is '1'**. This setting works in Active and Deep-Sleep power modes.

The SCB functionality is provided in the externally clocked domain. Note that this setting results in externally clocked accesses to the block's SRAM. These accesses may conflict with internally clocked accesses from the device. This may cause wait states or bus errors. The FIFO_BLOCK field (bit 17) of the SCBx_CTRL register determines whether wait states ('1') or bus errors ('0') are generated.

15.4.12 Wake up from Sleep

The system wakes up from Sleep or Deep-Sleep power modes when an I²C address match occurs. The I²C block performs either of two actions after address match: Address ACK or Address NACK.

Address ACK – The I²C slave executes clock stretching and waits until the device wakes up and ACKs the address.

Address NACK – The I²C slave NACKs the address immediately. The master must poll the slave again after the device wakeup time is passed. This option is only valid in the slave or multi-master-slave modes.

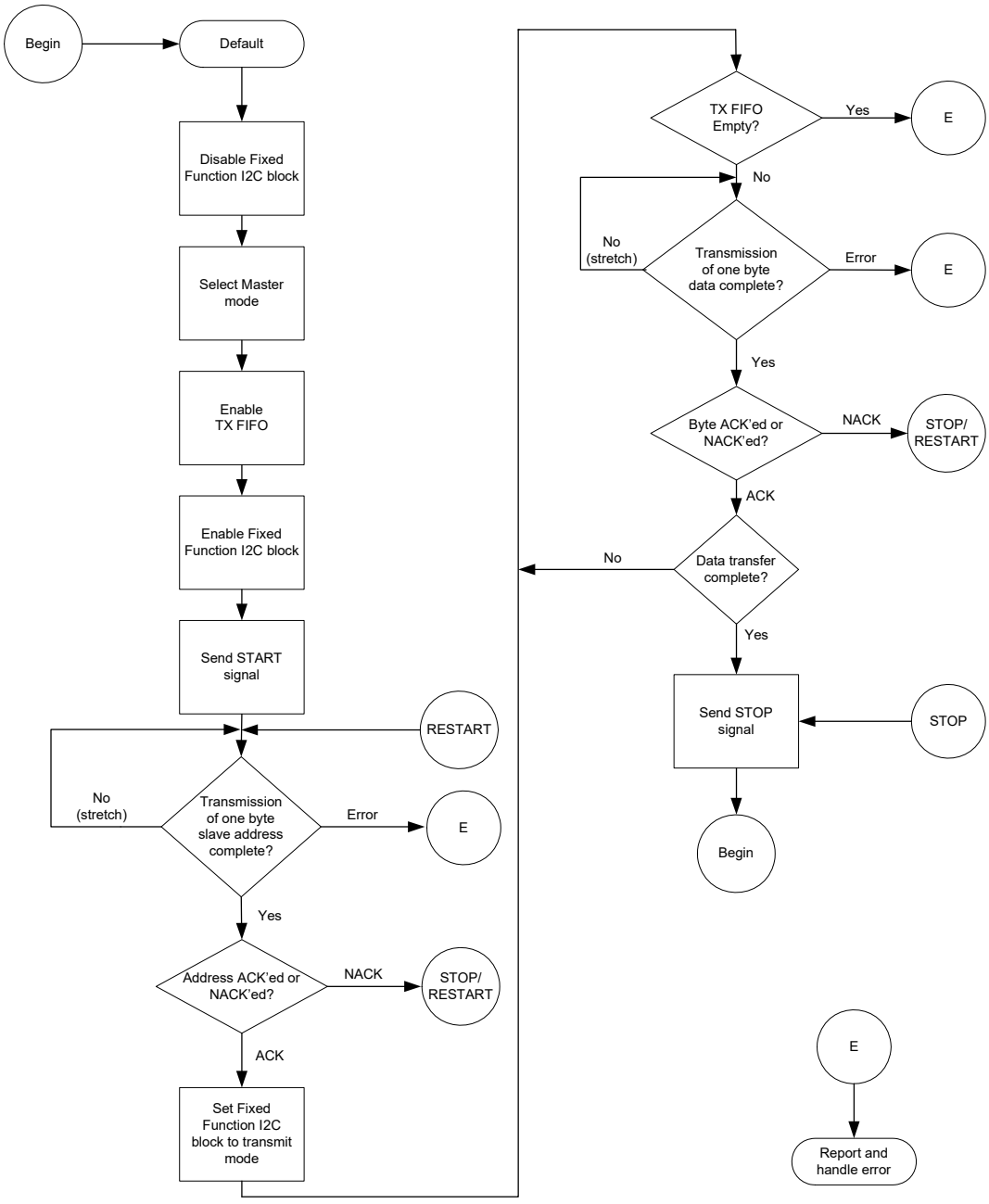
Note The interrupt bit WAKEUP (bit 0) of the INTR_I2C_EC register must be enabled for the I²C to wake up the device on slave address match while switching to the Sleep mode.

15.4.13 Master Mode Transfer Examples

Master mode transmits or receives data.

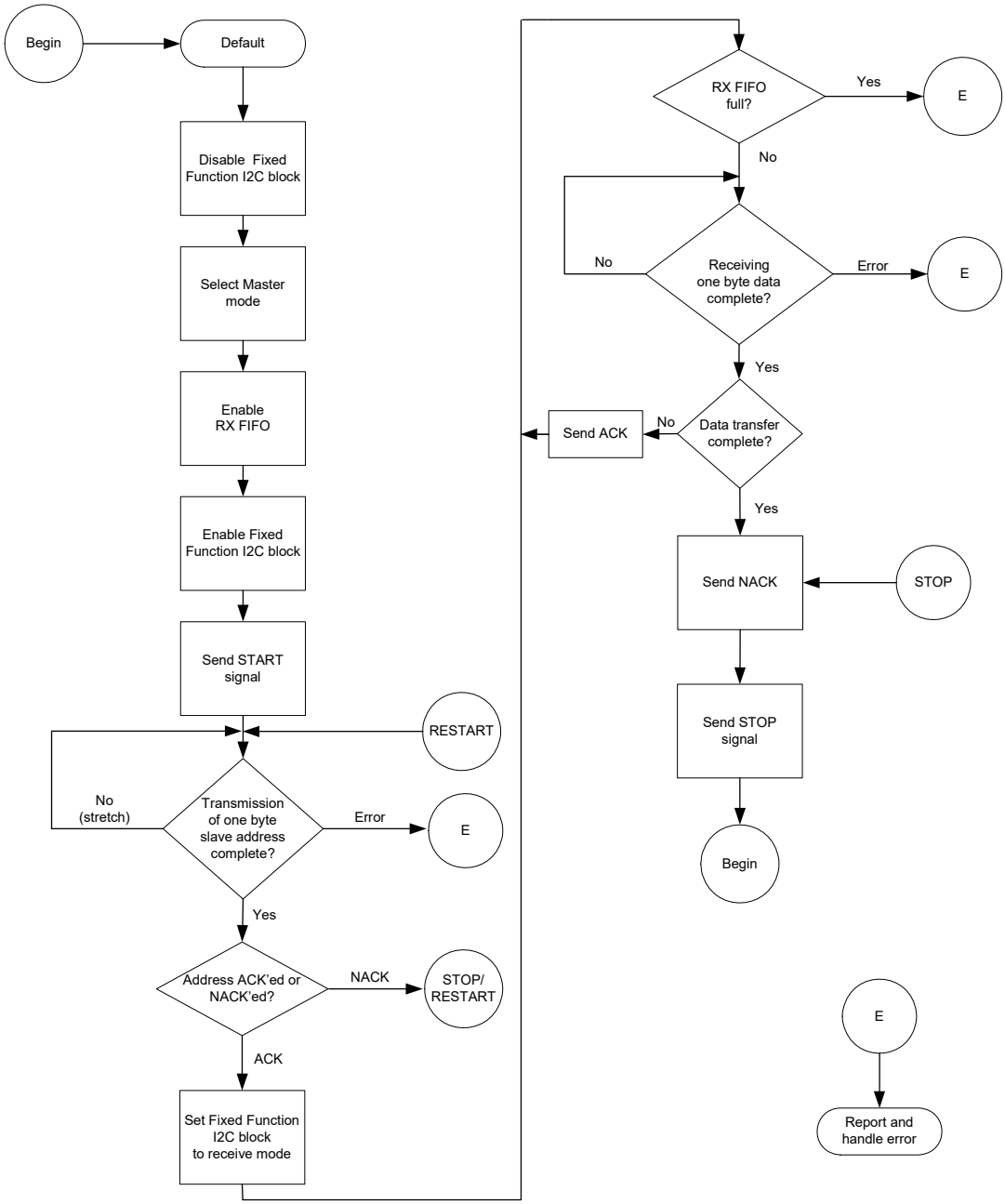
15.4.13.1 Master Transmit

Figure 15-22. Single Master Mode Write Operation Flow Chart



15.4.13.2 Master Receive

Figure 15-23. Single Master Mode Read Operation Flow Chart

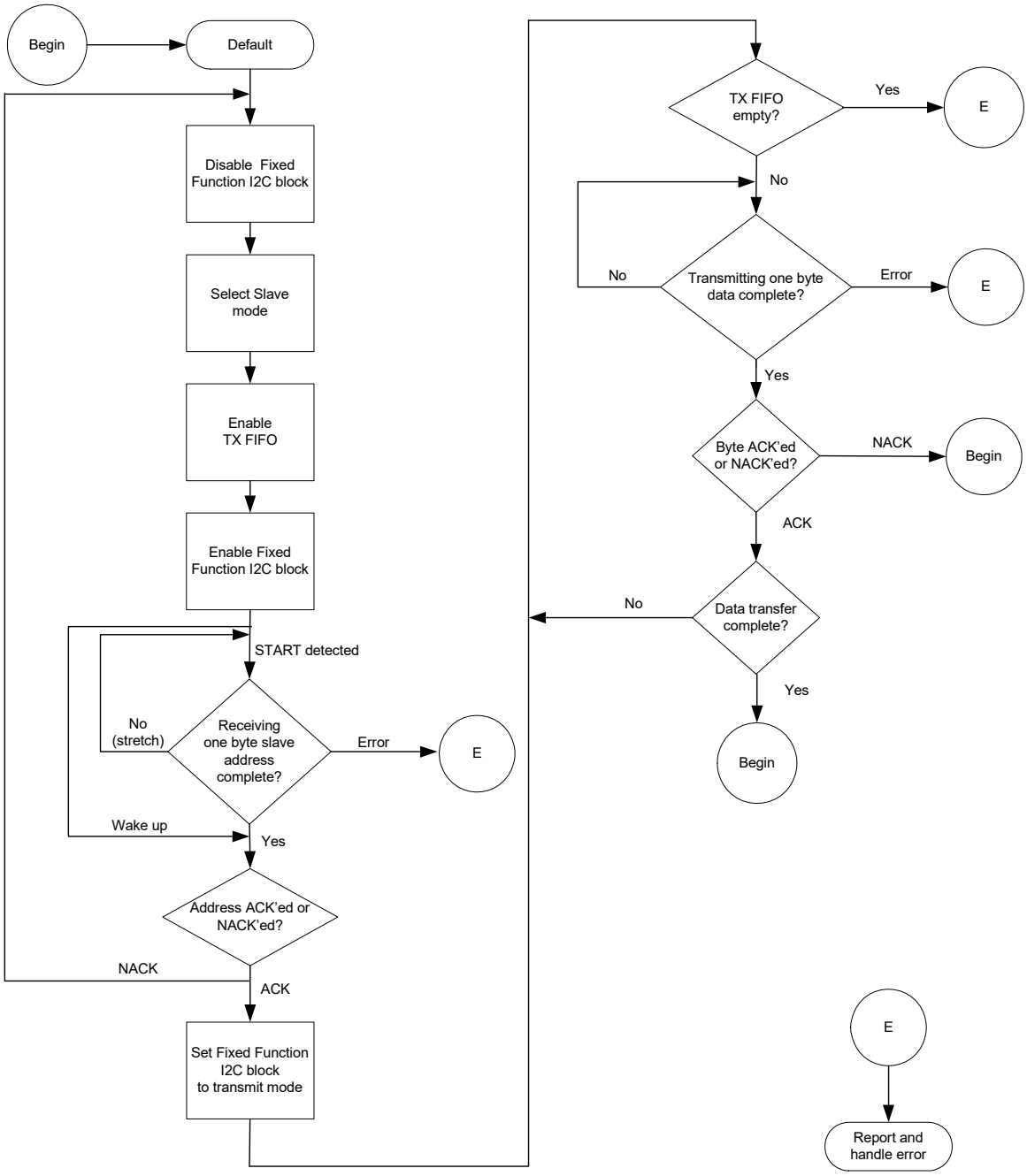


15.4.14 Slave Mode Transfer Examples

Slave mode transmits or receives data.

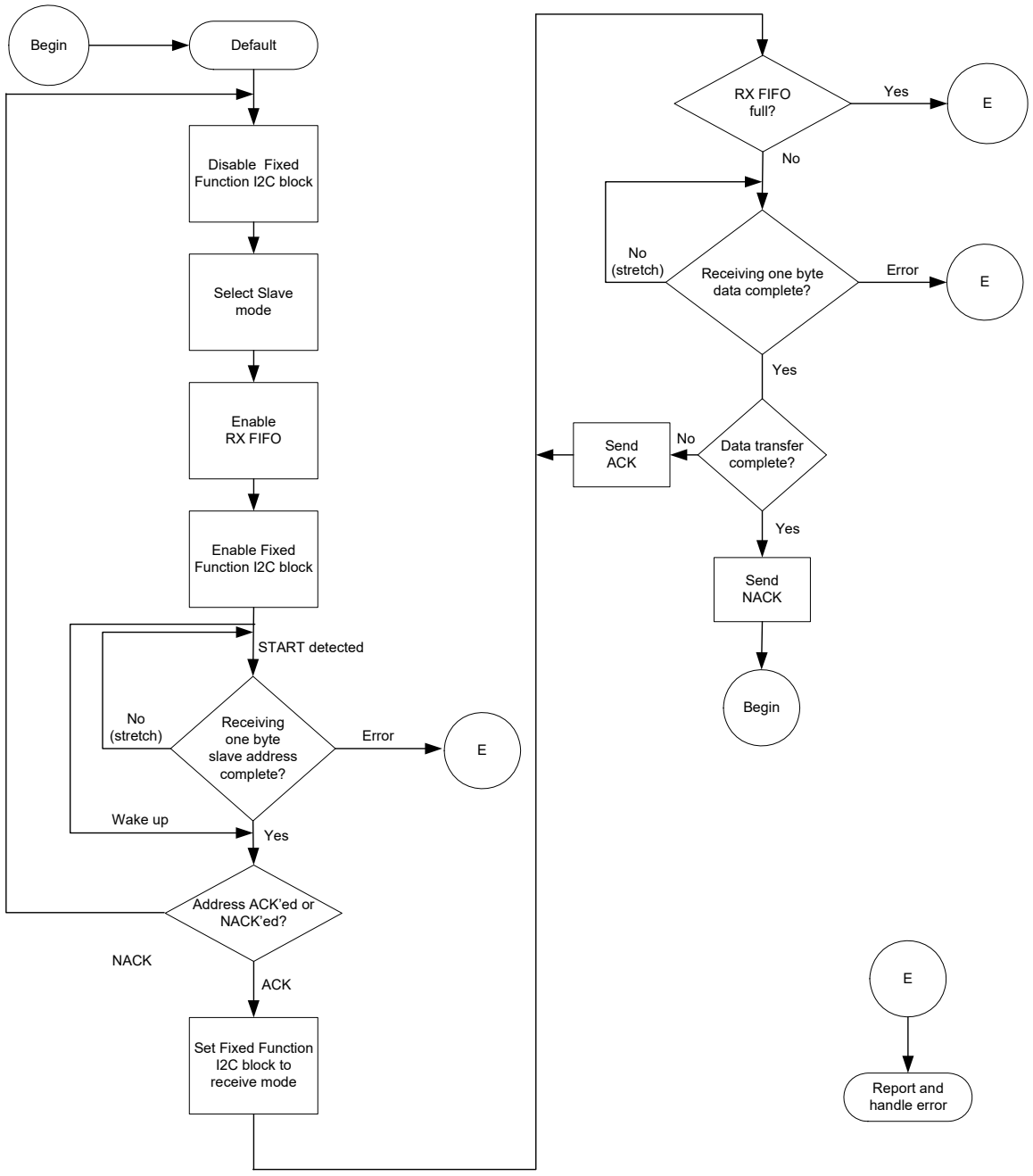
15.4.14.1 Slave Transmit

Figure 15-24. Slave Mode Write Operation Flow Chart



15.4.14.2 Slave Receive

Figure 15-25. Slave Mode Read Operation Flow Chart

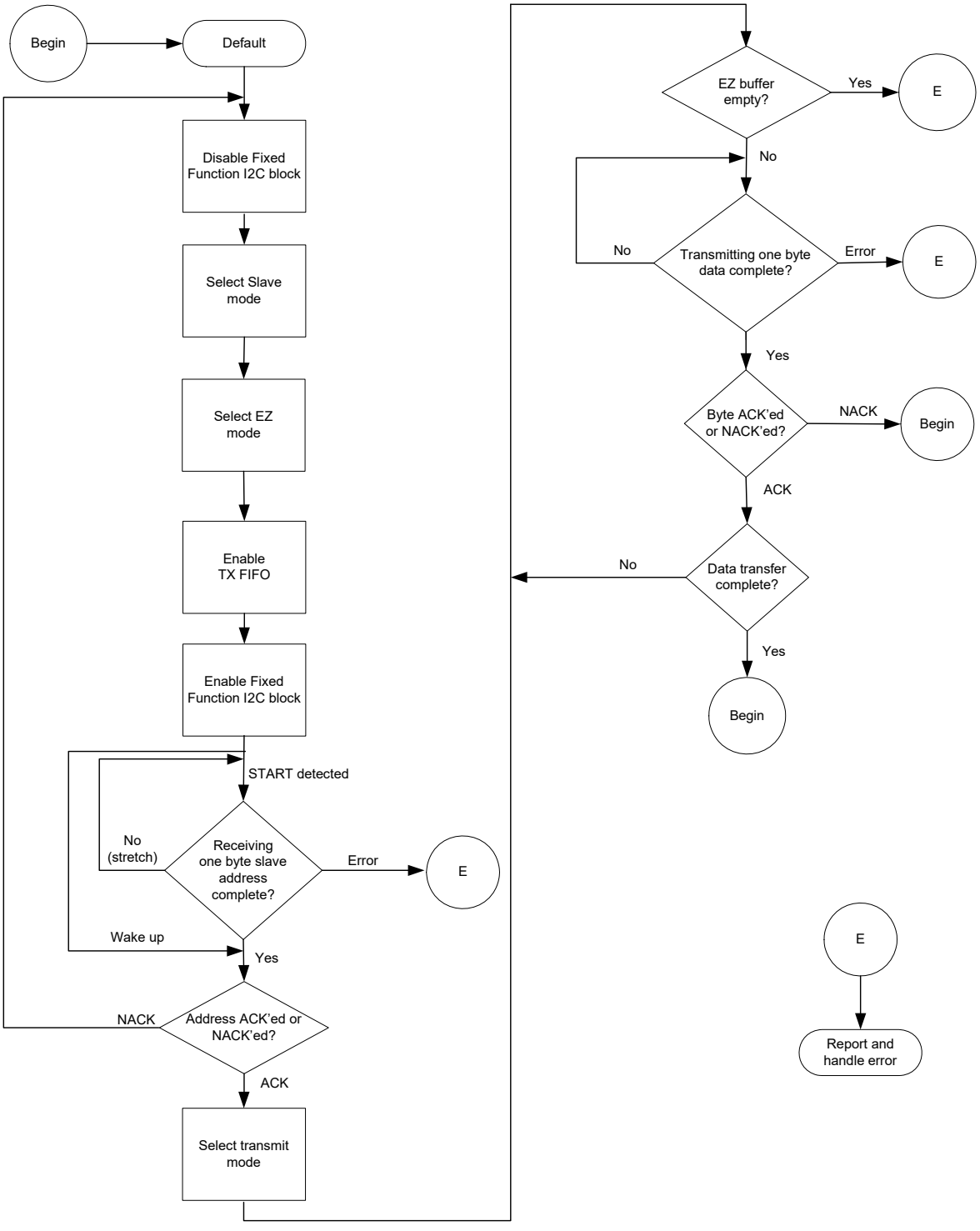


15.4.15 EZ Slave Mode Transfer Example

The EZ Slave mode transmits or receives data.

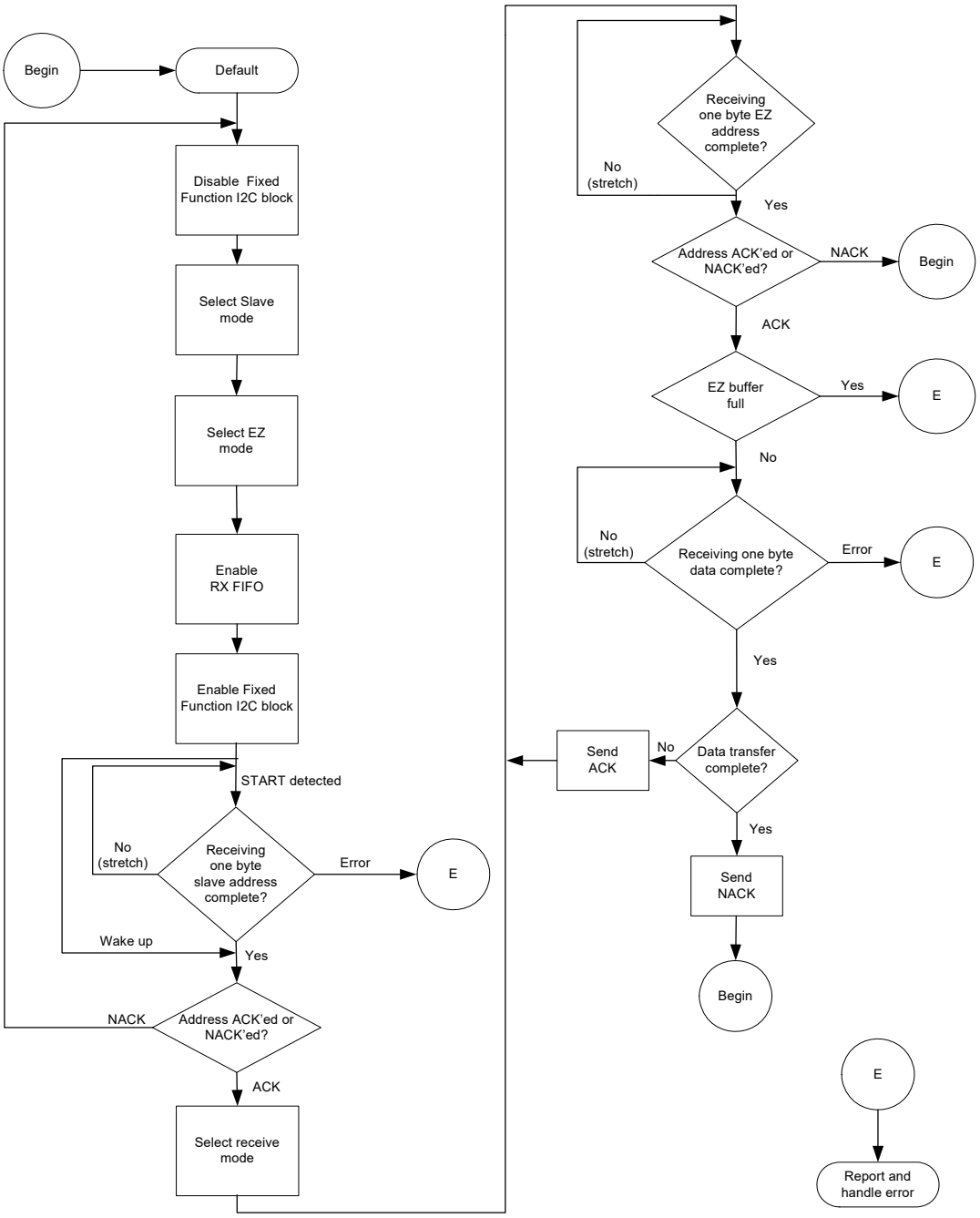
15.4.15.1 EZ Slave Transmit

Figure 15-26. EZI2C Slave Mode Write Operation Flow Chart



15.4.15.2 EZ Slave Receive

Figure 15-27. EZI2C Slave Mode Read Operation Flow Chart

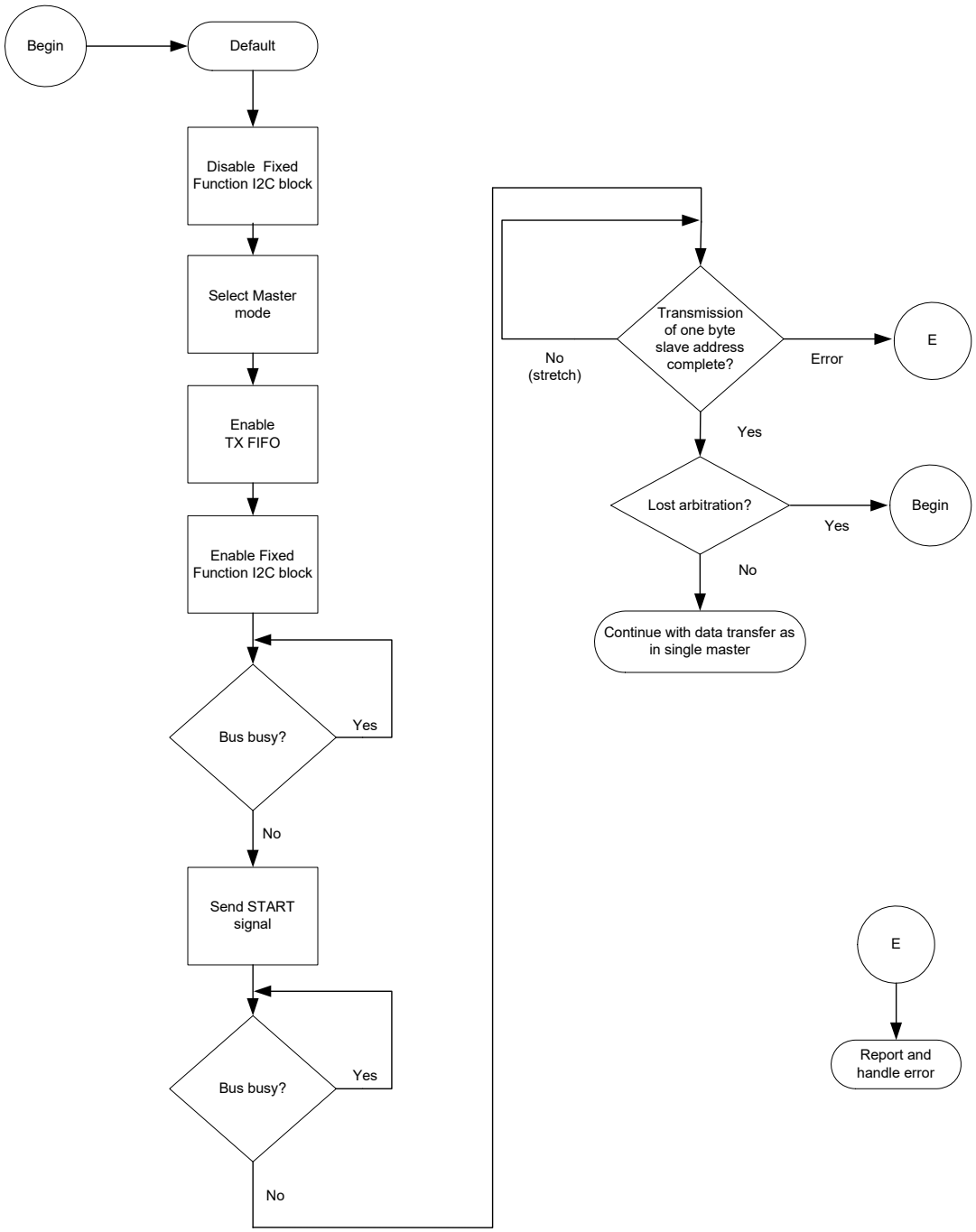


15.4.16 Multi-Master Mode Transfer Example

In multi-master mode, data can be transferred with the slave mode enabled or not enabled.

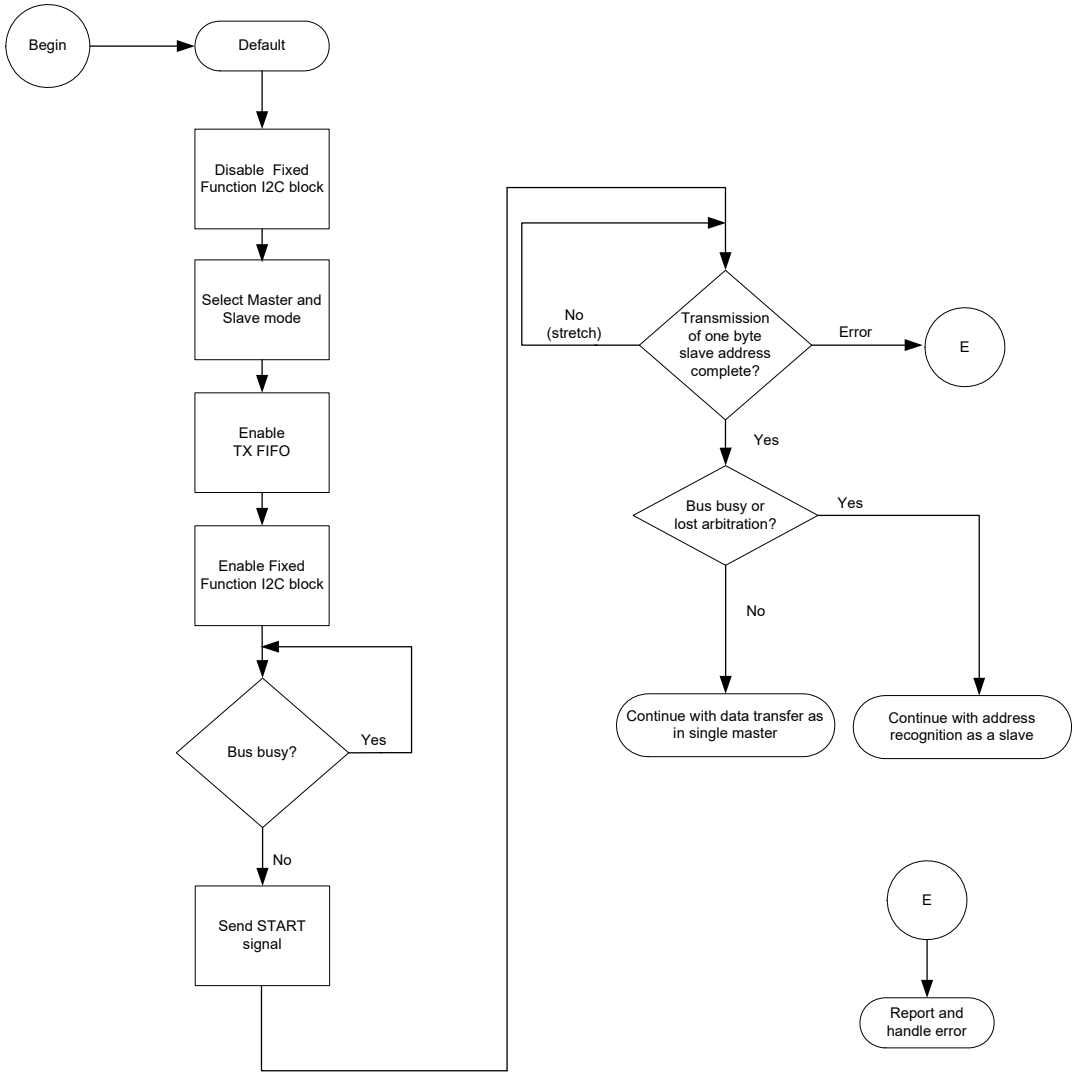
15.4.16.1 Multi-Master – Slave Not Enabled

Figure 15-28. Multi-Master, Slave Not Enabled Flow Chart



15.4.16.2 Multi-Master – Slave Enabled

Figure 15-29. Multi-Master, Slave Enabled Flow Chart



16. Cryptography (TR) Block

The Cryptography block in CCG7D implements a True Random Number Generator (TR) component which generates random numbers using up to six sets of ring oscillators.

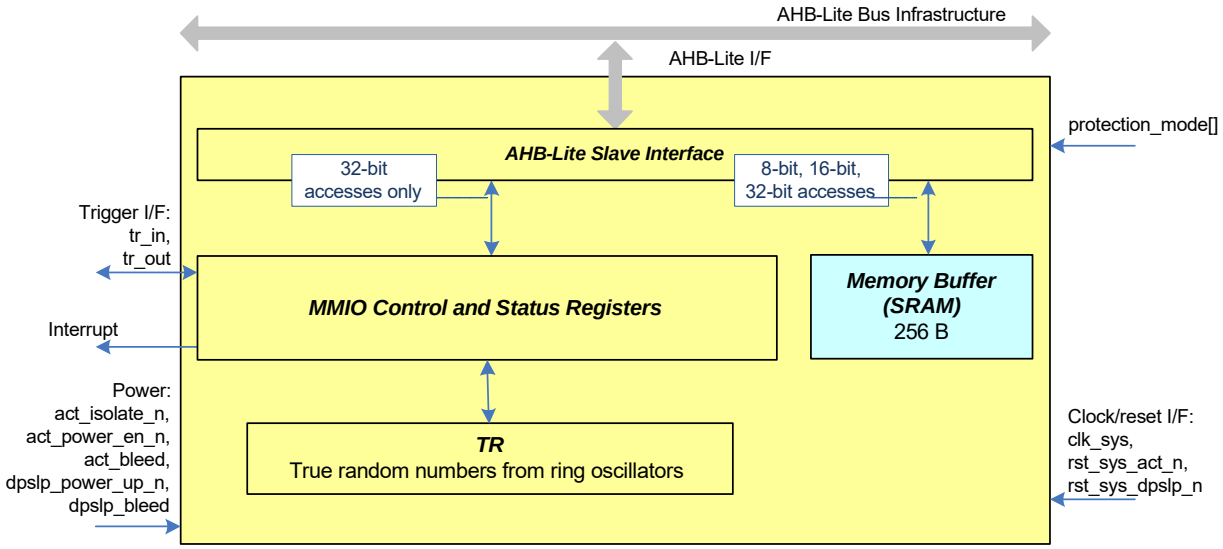
16.1 Features

This block supports the following features:

- True random number generator (TR)
- 256-byte SRAM memory buffer to store operands and results of operations
- Trigger interface to support HW initiation of an operation
- Interrupt notification on completion of operations

16.2 Block Diagram

Figure 16-1. Crypto Block Diagram



The Cryptography block in CCG7D supports the generation of True Random Numbers which can be used by the firmware for implementing security features such as USB-PD Authentication and encryption.

This block is connected to the bus infrastructure as an AHBLite slave: it can receive AHB-Lite transfers (from the CPU or another bus master) but it cannot initiate AHB-Lite transfers.

16.2.1 Block Application Overview

The TR component will be used to obtain random values, which can be used as initialization vector (IV) for AES encryption and for other purposes.

This block has the following interfaces:

- An AHB-Lite slave interface connects the block to the AHB-Lite infrastructure. This interface supports 8/16/32-bit AHB-Lite transfers. MMIO register accesses are 32-bit accesses only (8/16-bit accesses to MMIO registers results in an AHB-Lite bus error). Memory buffer access can be 8/16/32-bit accesses.
- Protection mode “protection_mode[]” signals (from the CPU subsystem) are provided to restrict the debug access port (DAP) accesses to the memory buffer.
- A trigger interface allows a block operation to be started by an external trigger (tr_in). When the operation completes, the block generates a trigger (tr_out).
- A single interrupt signal “interrupt” is used to signal the completion of an operation.
- A clock and reset signal interface connects to the System Resources subsystem (SRSS). The block operates of a gated version of clk_sys and uses both Active (rst_sys_act_n) and Deep Sleep (rst_sys_dpslp_n) reset signals.

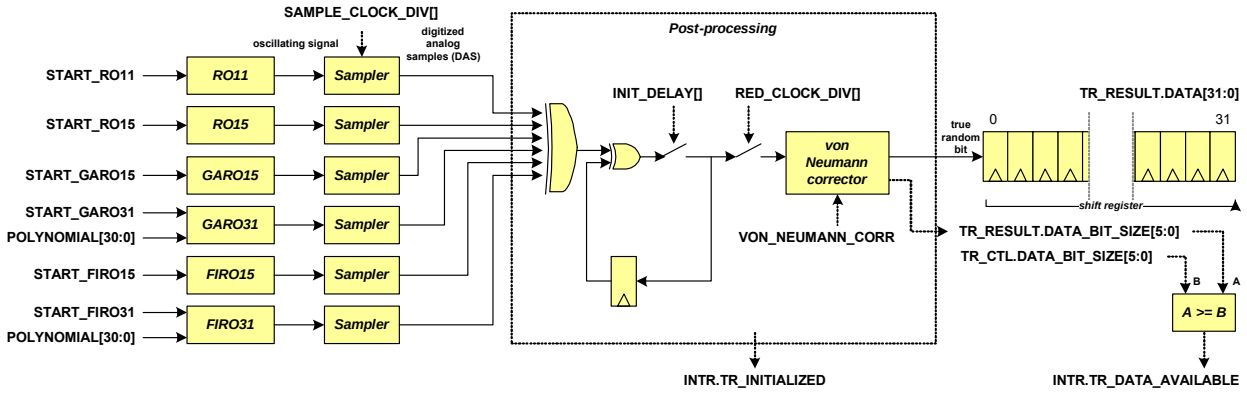
16.2.1.1 *True Random Number Generator (TRNG)*

The true random number generator component (TRNG) generates true random numbers. The bit size of these generated numbers is programmable in the range [0, 32].

The TRNG relies on up to six ring oscillators to provide physical noise sources. A ring oscillator consists of a series of inverters connected in a feedback loop to form a ring. Post-processing produces bit samples that are considered true random bit samples. The true random bit samples are shifted into a register, to provide random values of up to 32 bits.

Figure 16-2 shows an overview of the TRNG component.

Figure 16-2. TRNG Overview



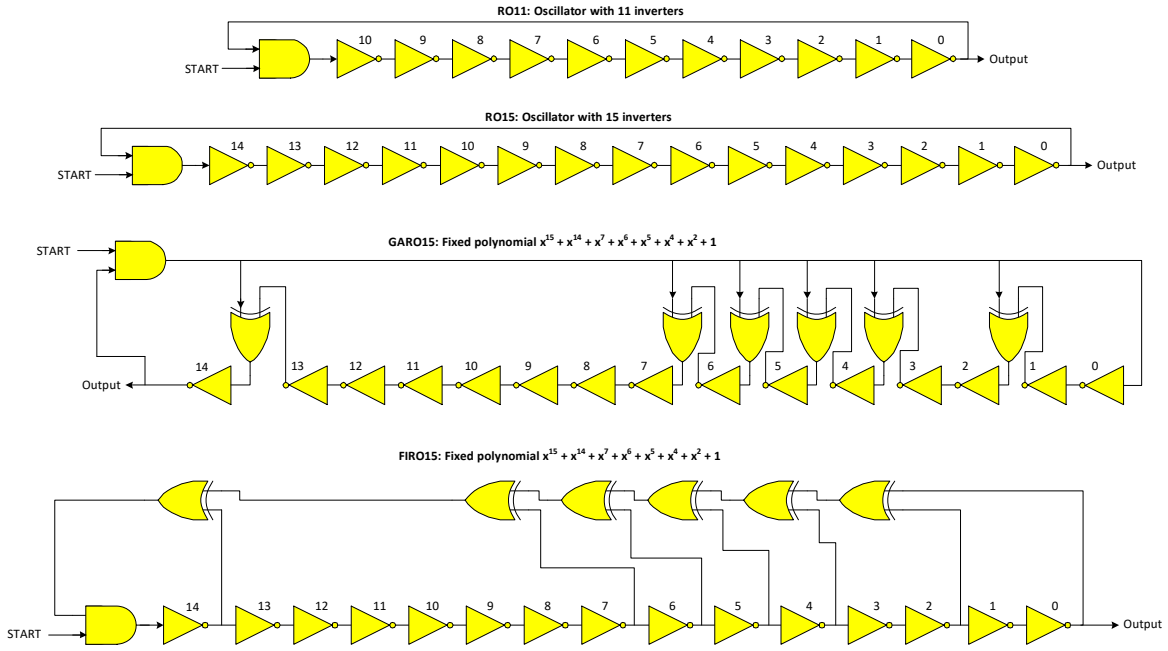
The following are the ring oscillators on which the TRNG relies:

- RO11: A fixed ring oscillator consisting of 11 inverters
- RO15: A fixed ring oscillator consisting of 15 inverters
- GARO15: A fixed Galois based ring oscillator of 15 inverters
- GARO31: A flexible Galois based ring oscillator of up to 31 inverters. A programmable polynomial of up to order 31 provides the flexibility in the oscillator feedback.
- FIRO15: A fixed Fibonacci based ring oscillator of 15 inverters.
- FIRO31: A flexible Fibonacci based ring oscillator of up to 31 inverters. A programmable polynomial of up to order 31 provides the flexibility in the oscillator feedback.

Each ring oscillator can be started or stopped. When stopped, the ring is “broken” to prevent switching.

Figure 16-3 illustrates the schematics of the fixed ring oscillators.

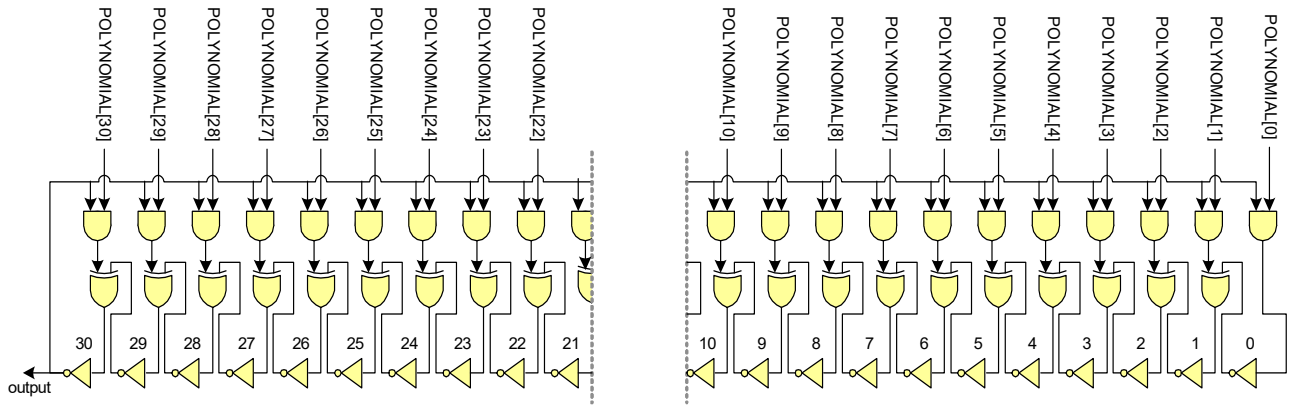
Figure 16-3. Fixed Ring Oscillators: RO11, RO15, GARO15, FARO15



The START signals originate from a MMIO register field. The flexible Galois and Fibonacci based ring oscillators rely on programmable polynomials to specify the oscillator feedback. This allows for rings of 1, 3, 5, …, 31 inverters (an odd number is required to generate an oscillating signal).

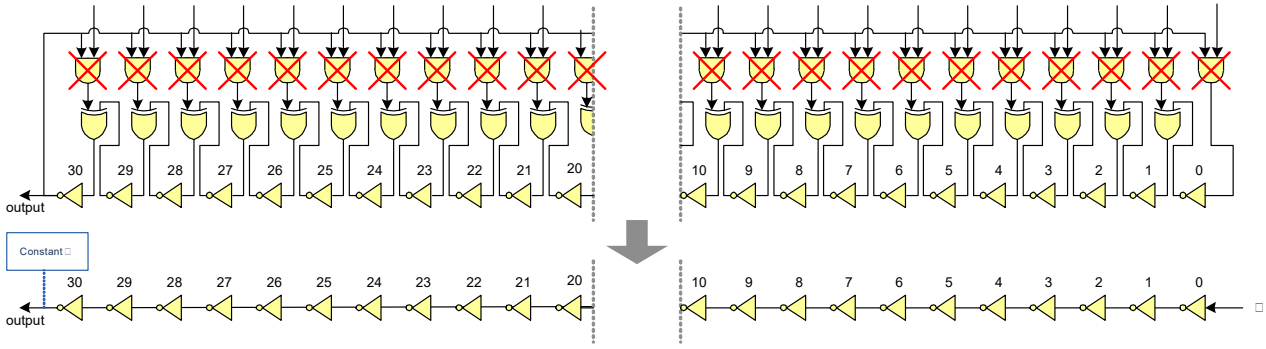
Figure 16-4 provides an overview of the Galois based flexible ring oscillator (GARO31).

Figure 16-4. Flexible Galois Ring Oscillator (GARO31)



When the ring oscillator is stopped, the polynomial is forced to “0” and the ring is broken as illustrated in Figure 16-5.

Figure 16-5. Flexible GARO31 when not running



The programmable polynomial defined through the TR_GARO_CTL register specifies the oscillator feedback. Figure 16-6 illustrates two example configurations for the GARO31 oscillator.

Figure 16-6. Flexible GARO31 Examples

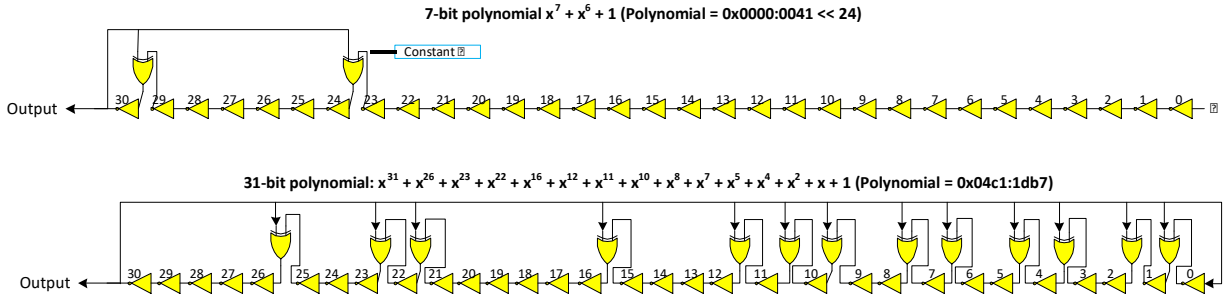
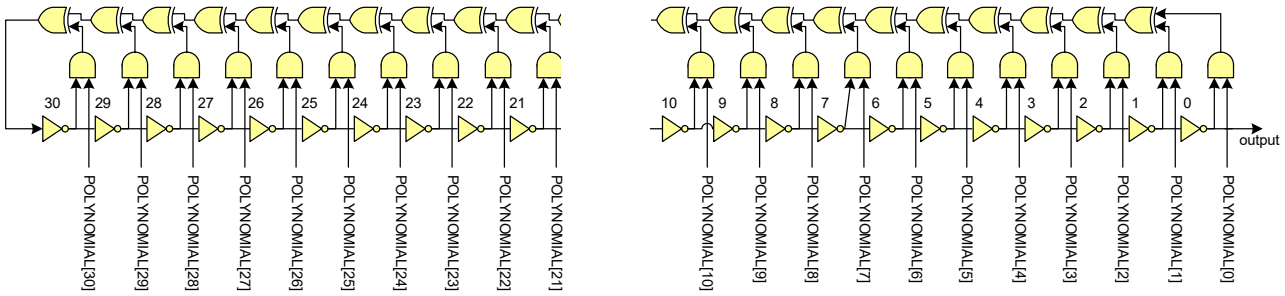


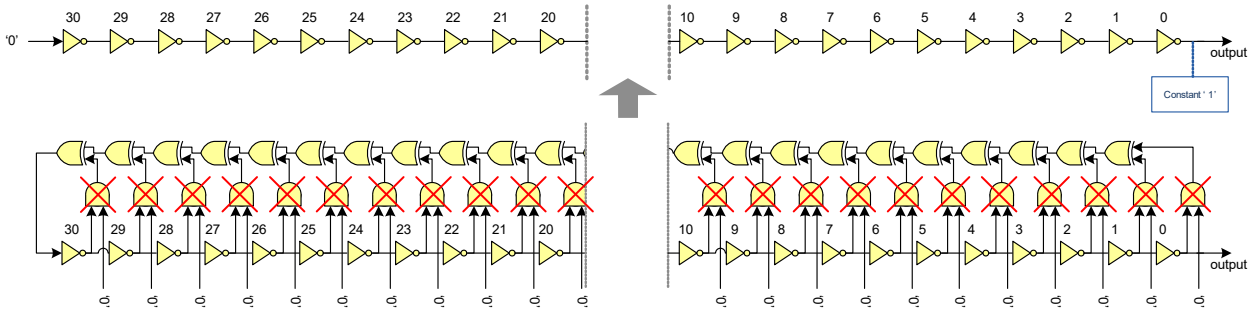
Figure 16-7 gives an overview of the Fibonacci based flexible ring oscillator (FIRO31).

Figure 16-7. Flexible Fibonacci Ring Oscillator FIRO31



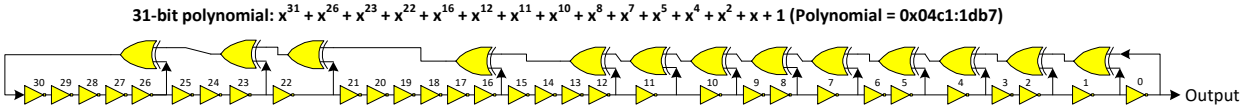
When the ring oscillator is stopped, the polynomial is forced to “0” and the ring is broken as illustrated in Figure 16-8.

Figure 16-8. Flexible FIRO31 when not running



The programmable polynomial defined through the TR_FIRO_CTL register specifies the oscillator feedback. Figure 16-9 illustrates an example usage for the FIRO31 ring oscillator.

Figure 16-9. Flexible FIRO31 Example



The TRNG has a build-in health monitor that performs tests on the digitized noise source to detect deviations from the intended behavior. For example, the health monitor detects “stuck at” faults in the digitized analog samples. The health monitor tests one out of three selected digitized bit streams:

- DAS bitstream. This is XOR of the digitized analog samples.
- RED bitstream. This is the bitstream of reduction bits. Note that each reduction bit may be calculated over multiple DAS bits.
- TR bitstream. This is the bitstream of true random bits (after the “von Neumann reduction” step).

The health monitor performs two different tests:

- The repetition count test. This test checks for the repetition of the same bit value ('0' or '1') in a bitstream. A detection indicates that a specific active bit value (specified by a status field BIT) has repeated for a preprogrammed number of bits (specified by a control field CUTOFF_COUNT[7:0]). The test uses a counter to maintain the number of repetitions of the active bit value (specified by a status field REP_COUNT[7:0]).

  If the test is started (specified by START_RC field) and a change in the bitstream value is observed, the active bit value BIT is set to the new bit value and the repetition counter REP_COUNT[] is set to “1”. If the bitstream value is unchanged, the repetition counter REP_COUNT[] is incremented by “1”.

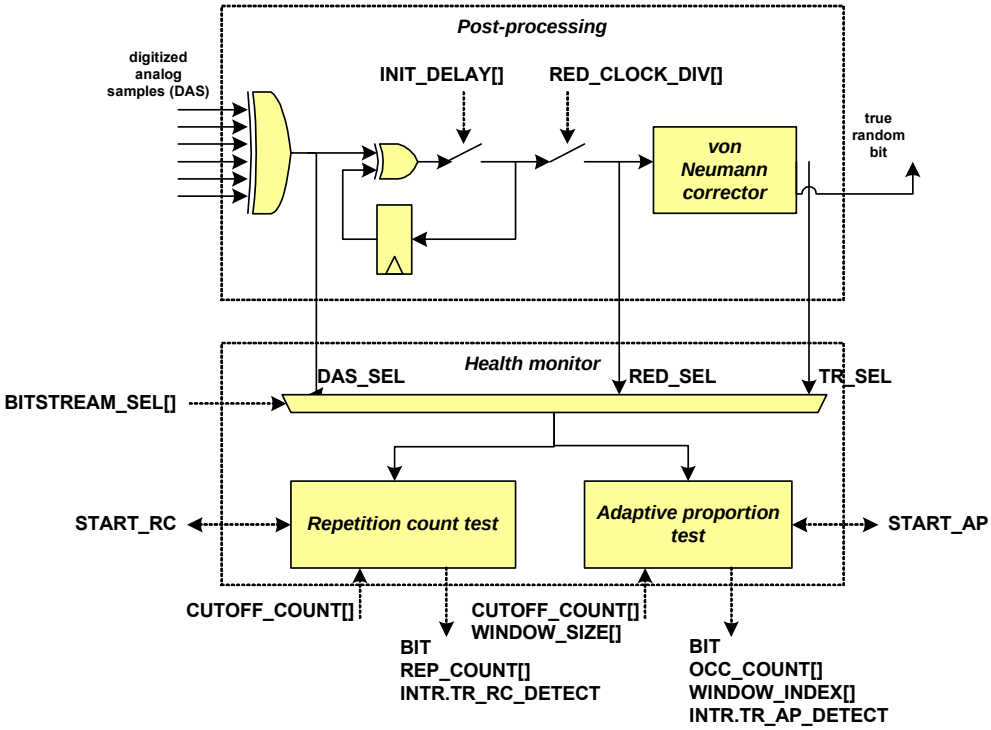
  A detection stops the test (the START_RC field is set to '0'), sets the associated interrupt status field to '1' and ensures that HW does NOT modify the status fields. When the test is stopped, REP_COUNT[] equals CUTOFF_COUNT[].

  A detection stops the TRNG functionality (all TR_CMD START fields are set to '0') if TR_CTL.STOP_ON_RC_DETECT is set to '1'.
- The adaptive proportion test. This test checks for a disproportionate occurrence of a specific bit value ('0' or '1') in a bit stream. A detection indicates that a specific active bit value (specified by a status field BIT) has occurred a preprogrammed number of times (specified by a control field CUTOFF_COUNT[15:0]) in a bit sequence of a specific bit window size (specified by a control field WINDOW_SIZE[15:0]). The test uses a counter to maintain an index in the current window (specified by WINDOW_INDEX[15:0]) and a counter to maintain the number of occurrences of the active bit value (specified by a status field OCC_COUNT[15:0]).
- If the test is started (specified by START_AP field), the bitstream is partitioned in bit sequences of a specific window size. At the first bit of a bit sequence, the active bit value BIT is set to the first bit value, the counter WINDOW_INDEX is set to “0” and the counter OCC_COUNT is set to “1”. For all other bits of a bit sequence, the counter WINDOW_INDEX is incremented by “1”. If the new bit value equals the active bit value BIT, the counter OCC_COUNT[15:0] is incremented by “1”. Note that the active bit value BIT is only set at the first bit of a bit sequence.

- A detection stops the test (the START_AP field is set to '0'), sets the associated interrupt status field to '1' and ensures that HW does NOT modify the status fields. When the test is stopped, OCC_COUNT[] equals CUTOFF_COUNT[] and the WINDOW_INDEX identifies the bit sequence index on which the detection occurred.
- A detection stops the TRNG functionality (all TR_CMD START fields are set to '0') if TR_CTL.STOP_ON_AP_DETECT is set to '1'.

Figure 16-10 illustrates the TRNG health monitor functionality.

Figure 16-10. TRNG Health Monitor Overview



17. Timer and PWM

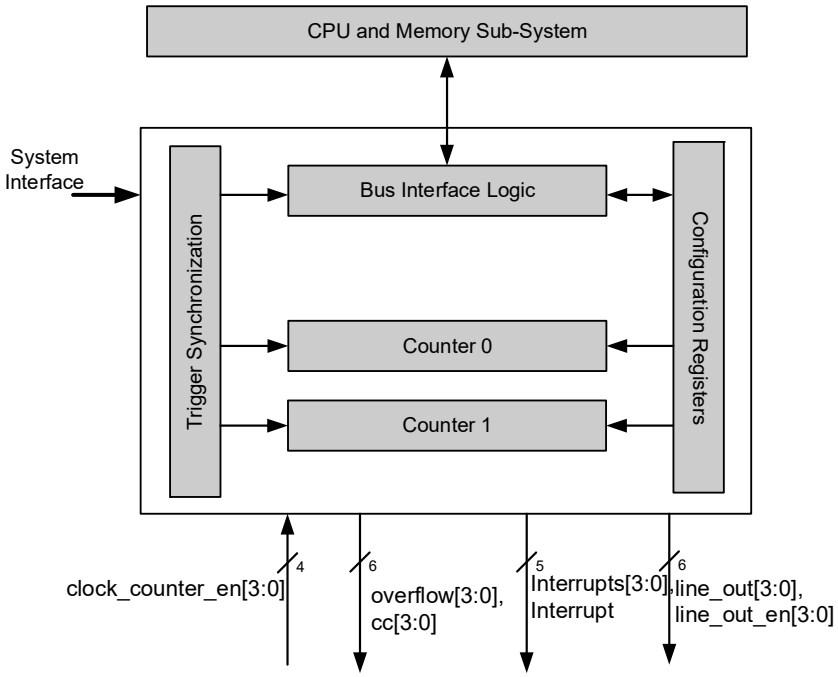
The Timer and Pulse Width Modulator block in CCG7D implements the 16-bit timer and pulse width modulator (PWM). The block can be used to count the input clock signal (timer) or generate PWM signals. This chapter explains the features, implementation, and operational modes of the block.

17.1 Features

- Four 16-bit timers or PWMs
- Supports the following operational modes:
  - Timer
  - Counter
  - Pulse width modulation
  - Pseudo-random PWM
- Multiple counting modes – up, down, and up/down
- Clock prescaling (division by 1, 2, 4, ...64, 128)
- Double buffering of compare/capture and period values
- Supports interrupt on:
  - Terminal Count (TC) – The final value in the counter register is reached
  - Compare (CC) – The counter value equals the compare value

17.2 Block Diagram

Figure 17-1. TCPWM Block Diagram



The block has these interfaces:

- Bus interface: Connects the block to the CPU subsystem.
- Interrupts: Provides interrupt request signals from each counter, based on terminal count (TC) or CC conditions, and a combined interrupt signal generated by the logical OR of all the interrupt request signals.
- System interface: Consists of control signals such as clock and reset from the system resources subsystem.

This TCPWM block can be configured by writing to the TCPWM registers. See TCPWM Registers on page 123 for more information on all registers required for this block.

The Trigger_in interfaces of the TCPWM blocks are connected to internal trigger outputs from the USBPD block. TCPWM on CCG7D does not support any external triggers for the counters.

17.2.1 Enabling and Disabling Counter in TCPWM Block

The counter can be enabled by setting the COUNTER_ENABLED field (bit 0) of the control register, TCPWM_CTRL.

Note The counter must be configured before enabling it. If the counter is enabled after being configured, registers are updated with the new configuration values. Disabling the counter retains the values in the registers until it is enabled again (or reconfigured).

17.2.2 Clocking

The TCPWM receives the HFCLK through the system interface to synchronize all events in the block. The counter enable signal (counter_en), which is generated when the counter is enabled, gates the HFCLK to provide a counter-specific clock (counter_clock). Output triggers (explained later in this chapter) are also synchronized with the HFCLK.

Clock Prescaling: counter_clock can be prescaled, with divider values of 1, 2, 4… 64, 128. This is done by modifying the GENERIC field of the counter control (TCPWM_CNTx_CTRL) register, as shown in Table 17-1.

Table 17-1. Bit-Field Setting to Prescale Counter Clock

GENERIC[10:8]	Description
0	Divide by 1
1	Divide by 2
2	Divide by 4
3	Divide by 8
4	Divide by 16
5	Divide by 32
6	Divide by 64
7	Divide by 128

Note Clock prescaling cannot be done in quadrature mode and pulse width modulation mode with dead time (PWM-DT).

17.2.3 Events Based on Trigger Inputs

These are the events triggered by hardware or software.

- Reload
- Start
- Stop
- Count
- Switch

Hardware triggers can be level signal, rising edge, falling edge, or both edges.

Figure 17-2. TCPWM Trigger Selection and Event Detection

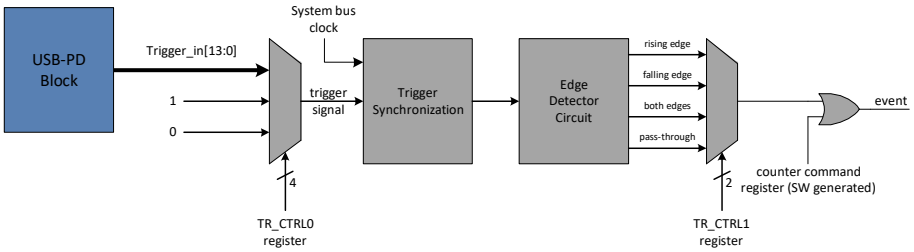


Figure 17-2 shows the trigger selection and event detection in the TCPWM block. The trigger control register 0 (TCPWM_CNTx_TR_CTRL0) selects one of the 16 trigger inputs as the event signal. Additionally, constant '0' and '1' signals are available to be used as the event signal.

As shown in the figure, the Trigger_in inputs on CCG7D/SF have been connected internally to trigger outputs generated by the USBPD block. External (GPIO) trigger inputs are not supported.

Any edge (rising, falling, or both) or level (high or low) can be selected for the occurrence of an event by configuring the trigger control register 1 (TCPWM_CNTx_TR_CTRL1). This edge/level configuration can be selected for each trigger event separately. Alternatively, firmware can generate an event by writing to the counter command register (TCPWM_CMD), as shown in Figure 17-2.

The events derived from these triggers can have different definitions in different modes of the TCPWM block.

- **Reload:** A reload event initializes and starts the counter.
  - In up counting mode, the count register (TCPWM_CNTx_COUNTER) is initialized with ‘0’.
  - In down counting mode, the counter is initialized with the period value stored in the TCPWM_CNTx_PERIOD register.
  - In up/down counting mode, the count register is initialized with ‘0’.
  - In quadrature mode, the reload event acts as a quadrature index event. An index/reload event indicates a completed rotation and can be used to synchronize quadrature decoding.
- **Start:** A start event is used to start counting; it can be used after a stop event or after re-initialization of the counter register to any value by software. Note that the count register is not initialized on this event.
- **Count:** A count event causes the counter to increment or decrement, depending on its configuration.
- **Stop:** A stop event stops the counter from incrementing or decrementing. A start event will start the counting again.
  - In the PWM modes, the stop event acts as a kill event. A kill event disables or floats all PWM output lines.

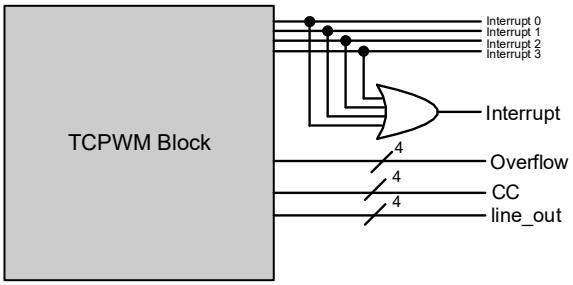
Notes

- All trigger inputs are synchronized to the HFCLK.
- When more than one event occurs in the same counter clock period, one or more events may be missed. This can happen for high-frequency events (frequencies close to the counter frequency) and a timer configuration in which a prescaled (divided) counter clock is used.

17.2.4 Output Signals

The TCPWM block generates several output signals, as shown in Figure 17-3.

Figure 17-3. TCPWM Output Signals



17.2.4.1 Signals upon Trigger Conditions

- Counter generates an internal overflow (OV) condition when counting up and the count register reaches the period value.
- Counter generates an internal underflow (UN) condition when counting down and the count register reaches zero.
- The compare (CC) condition is generated by the TCPWM when the counter is running and one of the following conditions occur:
  - The counter value equals the compare value.

Note These signals, when they occur, remain at logic high for one cycle of the HFCLK. For reliable operation, the condition that causes this trigger should occur in a frequency less than a quarter of the HFCLK. For example, if the HFCLK is running at 24 MHz, the condition causing the trigger should occur at a frequency less than 6 MHz.

17.2.4.2 Interrupts

The TCPWM block provides a dedicated interrupt output signal from the counter. An interrupt can be generated for a TC condition or a CC condition. The exact definition of these conditions is mode-specific. All interrupt output signals from the four TCPWMs are also OR'ed together to produce a single interrupt output signal.

Four registers are used for interrupt handling in this block, as shown in Table 17-2.

Table 17-2. Interrupt Register

Interrupt Registers	Bits	Name	Description
TCPWM_CNTx_INTR (Interrupt request register)	0	TC	This bit is set to '1', when a terminal count is detected. Write '1' to clear this bit.
	1	CC_MATCH	This bit is set to '1' when the counter value matches capture/compare register value. Write with '1' to clear this bit.
TCPWM_CNTx_INTR_SET (Interrupt set request register)	0	TC	Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.
	1	CC_MATCH	Write '1' to set the corresponding bit in the interrupt request register. When read, this register reflects the interrupt request register status.
TCPWM_CNTx_INTR_MASK (Interrupt mask register)	0	TC	Mask bit for the corresponding TC bit in the interrupt request register.
	1	CC_MATCH	Mask bit for the corresponding CC_MATCH bit in the interrupt request register.
TCPWM_CNTx_INTR_MASKED (Interrupt masked request register)	0	TC	Logical AND of the corresponding TC request and mask bits.
	1	CC_MATCH	Logical AND of the corresponding CC_MATCH request and mask bits.

17.2.4.3 Outputs

The TCPWM has only the line_out output; the line_compl_out (complementary of line_out) is not present. Note that the OV, UN, and CC conditions can be used to drive line_out if needed, by configuring the TCPWM_CNTx_TR_CTRL2 register (see Table 17-3).

Table 17-3. Configuring Output Line for OV, UN, and CC Conditions

Field	Bit	Value	Event	Description
CC_MATCH_MODE Default Value = 3	1:0	0	Set line_out to '1	Configures output line on a compare match (CC) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	

Table 17-3. Configuring Output Line for OV, UN, and CC Conditions

Field	Bit	Value	Event	Description
OVERFLOW_MODE Default Value = 3	3:2	0	Set line_out to '1	Configures output line on a overflow (OV) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	
UNDERFLOW_-MODE Default Value = 3	5:4	0	Set line_out to '1	Configures output line on a underflow (UN) event
		1	Clear line_out to '0	
		2	Invert line_out	
		3	No change	

17.2.5 Power Modes

The TCPWM block works in Active and Sleep modes. The TCPWM block is powered from VCCD. The configuration registers and other logic are powered in Deep Sleep mode to keep the states of configuration registers. See Table 17-4.

Table 17-4. Power Modes in TCPWM Block

Power Mode	Block Status
Active	This block is fully operational in this mode with clock running and power switched ON.
Sleep	All counter clocks are on, but bus interface cannot be accessed.
Deep-Sleep	In this mode, the power to this block is still on but no bus clock is provided; hence, the logic is not functional. All the configuration registers will keep their state.

17.3 Modes of Operation

The counter block can function in three operational modes, as shown in Table 17-5. The MODE [26:24] field of the counter control register (TCPWM_CNTx_CTRL) configures the counter in the specific operational mode.

Table 17-5. Operational Mode Configuration

Mode	MODE Field [26:24]	Description
Timer	000	Implements a timer or counter. The counter increments or decrements by '1' at every counter clock cycle in which a count event is detected.
PWM	100	Implements edge/center-aligned PWMs with an 8-bit clock prescaler and buffered compare/period registers.
PWM-PR	110	Implements a pseudo-random PWM using a 16-bit linear feedback shift register (LFSR).

The counter can be configured to count up, down, and up/down by setting the UP_DOWN_MODE[17:16] field in the TCPWM_CNTx_CTRL register, as shown in Table 17-6.

Table 17-6. Counting Mode Configuration

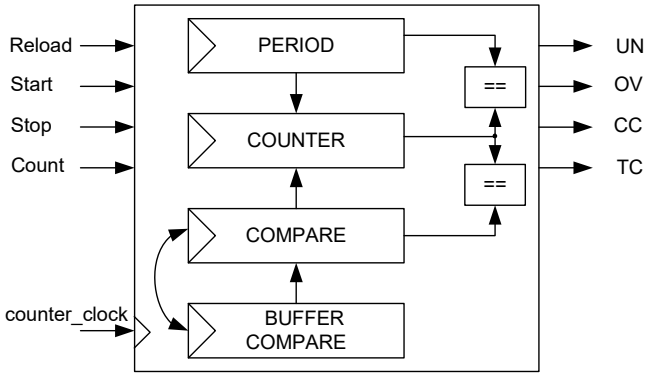
Counting Modes	UP_DOWN_MODE[17:16]	Description
UP Counting Mode	00	Increments the counter until the period value is reached. A Terminal Count (TC) condition is generated when counter reaches the period value.
DOWN Counting Mode	01	Decrements the counter from the period value until 0 is reached. A TC condition is generated when the counter reaches ‘0’.
UP/DOWN Counting Mode 0	10	Increments the counter until the period value is reached, and then decrements the counter until ‘0’ is reached. A TC condition is generated only when ‘0’ is reached.
UP/DOWN Counting Mode 1	11	Similar to up/down counting mode 0 but a TC condition is generated when the counter reaches ‘0’ and when the counter value reaches the period value.

17.3.1 Timer Mode

The timer mode is commonly used to measure time of occurrence of an event or to measure the time difference between two events. Here, reload, start, stop, and count signals are all software triggers. No wire bonding options are available to perform hardware trigger from an external GPIO.

17.3.1.1 Block Diagram

Figure 17-4. Timer Mode Block Diagram



17.3.1.2 How It Works

The timer can be configured to count in up, down, and up/down counting modes. It can also be configured to run in either continuous mode or one-shot mode.

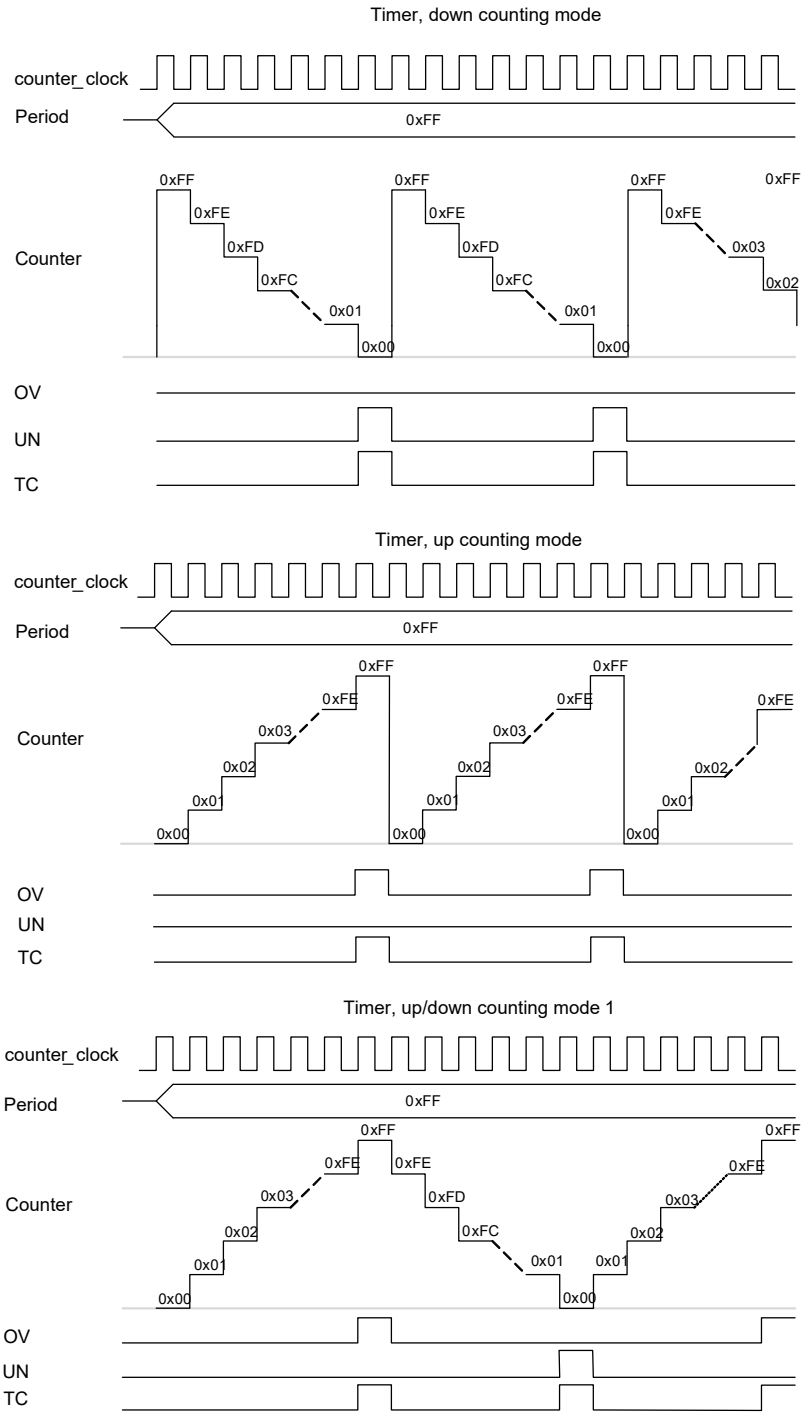
The following describes the working of the timer:

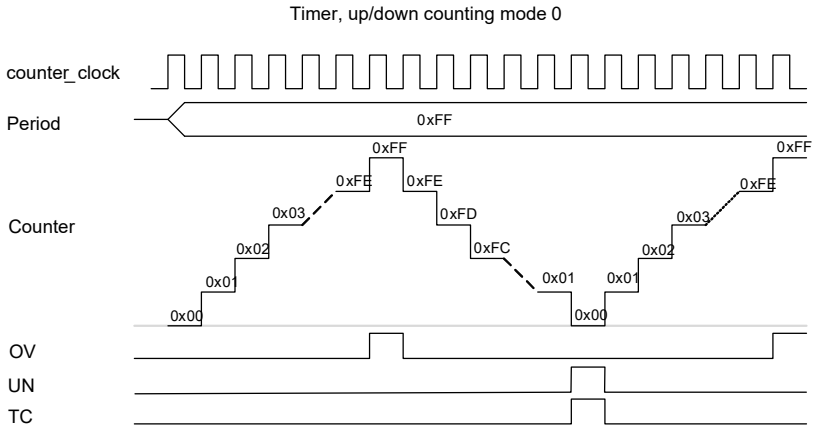
- The timer is an up, down, and up/down counter.
  - The current count value is stored in the count register (TCPWM_CNTx_COUNTER). **Note** It is not recommended to write values to this register while the counter is running.
  - The period value for the timer is stored in the period register.
- The counter is re-initialized in different counting modes as follows:
  - In the up counting mode, after the count reaches the period value, the count register is automatically reloaded with 0.
  - In the down counting mode, after the count register reaches zero, the count register is reloaded with the value in the period register.
  - In the up/down counting modes, the count register value is not updated upon reaching the terminal values. Instead the direction of counting changes when the count value reaches 0 or the period value.
- The CC condition is generated when the count register value equals the compare register value. Upon this condition, the compare register and buffer compare register switch their values if enabled by the AUTO_RELOAD_CC bit-field of the counter control (TCPWM_CNTx_CTRL) register. This condition can be used to generate an interrupt request.

Figure 17-5 shows the timer operational mode of the counter in four different counting modes. The period register contains the maximum counter value.

- In the up counting mode, a period value of A results in A+1 counter cycles (0 to A).
- In the down counting mode, a period value of A results in A+1 counter cycles (A to 0).
- In the two up/down counting modes (both modes 0 and 1 both), a period value of A results in 2*A counter cycles (0 to A and back to 0).

Figure 17-5. Timing Diagram for Timer in Multiple Counting Modes





Note The OV and UN signals remain at logic high for one cycle of the HFCLCK, as explained in Signals upon Trigger Conditions on page 112. The figures in this chapter assumes that HFCLK and counter clock are the same.

17.3.1.3 Configuring Counter for Timer Mode

The steps to configure the counter for Timer mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select Timer mode by writing '000' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register.
4. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register. Set AUTO_RELOAD_CC field of counter control register, if required to switch values at every CC condition.
5. Set clock prescaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-1.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-6.
7. The timer can be configured to run either in continuous mode or one-shot mode by writing 0 or 1, respectively to the ONE_SHOT[18] field of the TCPWM_CNTx_CTRL register.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Stop, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge of the trigger that causes the event (Reload, Start, Stop, and Count).
10. If required, set the interrupt upon TC or CC condition, as shown in Interrupts on page 112.
11. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

17.3.2 Pulse Width Modulation Mode

The PWM mode is also called the Digital Comparator mode. In this mode, the reload, start, stop, switch, and count signals are all software triggers. No wire bonding options are available to perform hardware trigger from an external GPIO. Also, line_out_compl (line out complimentary) and UN (underflow) signals are not bonded for external signal routing.

The comparison output is a PWM signal whose period depends on the period register value and duty cycle depends on the compare and period register values.

PWM period = (period value/counter clock frequency) in left- and right-aligned modes

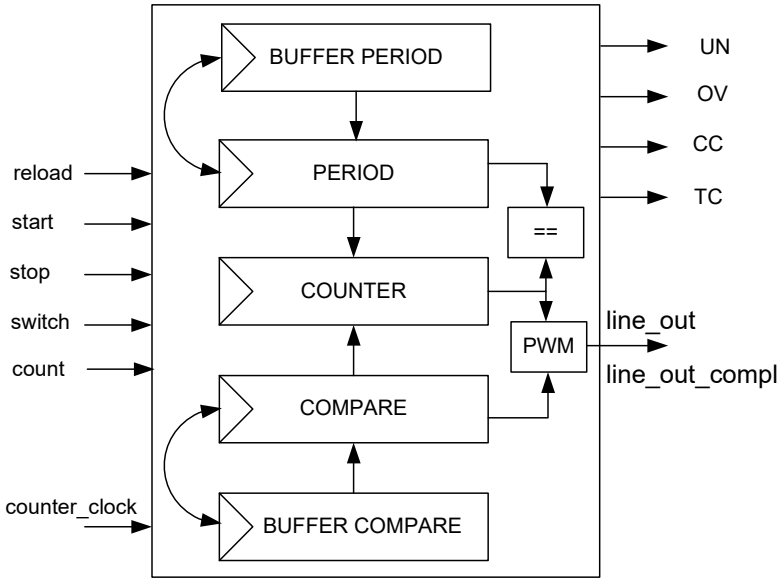
PWM period = (2 × (period value/counter clock frequency)) in center-aligned mode

Duty cycle = (compare value/period value) in left- and right-aligned modes

Duty cycle = ((period value-compare value)/period value) in center-aligned mode

17.3.2.1 Block Diagram

Figure 17-6. PWM Mode Block Diagram



17.3.2.2 How It Works

The PWM mode can output left, right, center, or asymmetrically aligned PWM signals. The desired output alignment is achieved by using the counter's up, down, and up/down counting modes selected using UP_DOWN_MODE [17:16] bits in the TCPWM_CNTx_CTRL register, as shown in Table 17-6.

This CC signal along with OV and UN signals control the PWM output line. The signals can toggle the output line or set it to a logic '0' or '1' by configuring the TCPWM_CNTx_TR_CTRL2 register. By configuring how the signals impact the output line, the desired PWM output alignment can be obtained.

The recommended way to modify the duty cycle is:

- The buffer period register and buffer compare register are updated with new values.
- On TC, the period and compare registers are automatically updated with the buffer period and buffer compare registers when there is an active switch event. The AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register are set to '1'. When a switch event is detected, it is remembered until the next TC event. Pass through signal (selected during event detection setting) cannot trigger a switch event.
- Updates to the buffer period register and buffer compare register should be completed before the next TC with an active switch event; otherwise, switching does not reflect the register update, as shown in Figure 17-8.

In the center-aligned mode, the output line is set to '0' at TC and toggled at the CC condition.

At the reload event, the count register is initialized and starts counting in the appropriate mode. At every count, the count register value is compared with compare register value to generate the CC signal on match.

Figure 17-7 illustrates center-aligned PWM with buffered period and compare registers (up/down counting mode 0).

Figure 17-7. Timing Diagram for Center Aligned PWM

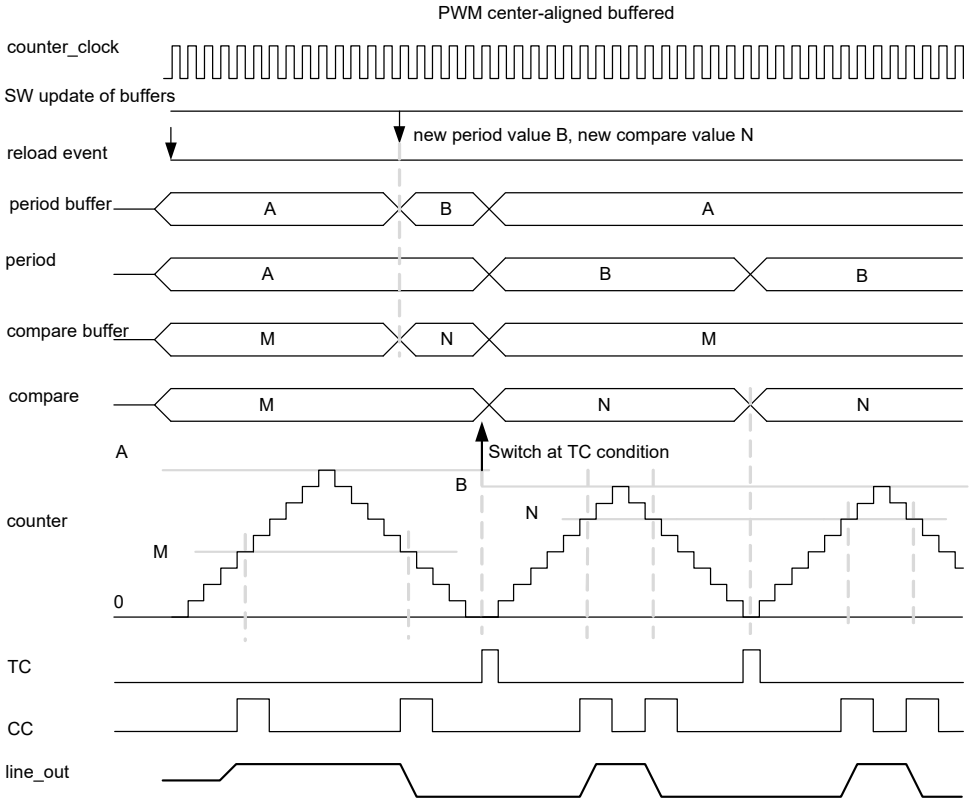
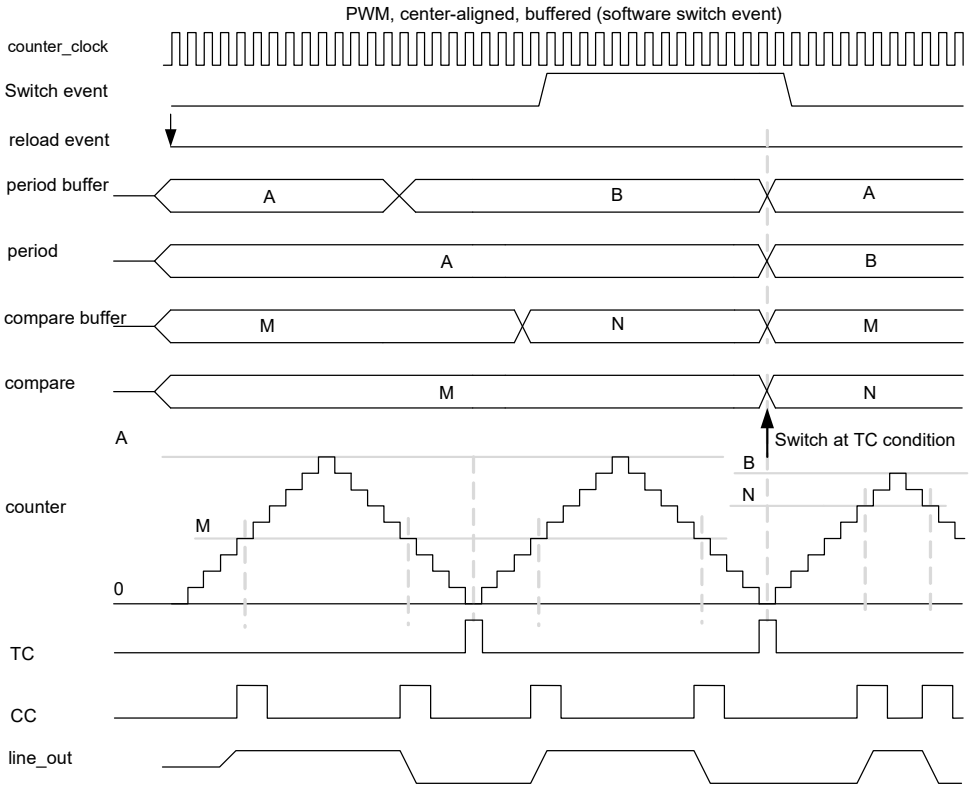


Figure 17-8 illustrates center-aligned PWM with software generated switch events:

- Software generates a switch event only after both the period buffer and compare buffer registers are updated.
- Because the updates of the second PWM pulse come late (after the terminal count), the first PWM pulse is repeated.
- Note that the switch event is automatically cleared by hardware at TC after the event takes effect.

Figure 17-8. Timing Diagram for Center Aligned PWM (software switch event)



17.3.2.3 Other Configurations

- For asymmetric PWM, the up/down counting mode 1 should be used. This causes a TC when the counter reaches either ‘0’ or the period value. To create an asymmetric PWM, the compare register is changed at every TC (when the counter reaches either ‘0’ or the period value), whereas the period register is only changed at every other TC (only when the counter reaches ‘0’).
- For left-aligned PWM, use the up counting mode; configure the OV condition to set output line to '1' and CC condition to reset the output line to '0'. See Table 17-3.
- For right-aligned PWM, use the down counting mode; configure UN condition to reset output line to '0' and CC condition to set the output line to '1'. See Table 17-3.

17.3.2.4 Kill Feature

Kill feature gives the ability to disable both output lines immediately. This event can be programmed to stop the counter by modifying the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the counter control register, as shown in Table 17-7.

Table 17-7. Field Setting for Stop on Kill Feature

PWM_STOP_ON_KILL Field	Comments
0	The kill trigger temporarily blocks the PWM output line but the counter is still running.
1	The kill trigger temporarily blocks the PWM output line and the counter is also stopped.

A kill event can be programmed to be asynchronous or synchronous, as shown in Table 17-8.

Table 17-8. Field Setting for Synchronous/Asynchronous Kill

PWM_SYNC_KILL Field	Comments
0	An asynchronous kill event lasts as long as it is present. This event requires pass through mode.
1	A synchronous kill event disables the output lines until the next TC event. This event requires rising edge mode.

In the synchronous kill, PWM cannot be started before the next TC. To restart the PWM immediately after the kill input is removed, the kill event should be asynchronous (see Table 17-8). The generated stop event disables both output lines. In this case, the reload event can use the same trigger input signal but should be used in falling edge detection mode.

17.3.2.5 Configuring Counter for PWM Mode

The steps to configure the counter for the PWM mode of operation and the affected register bits are as follows.

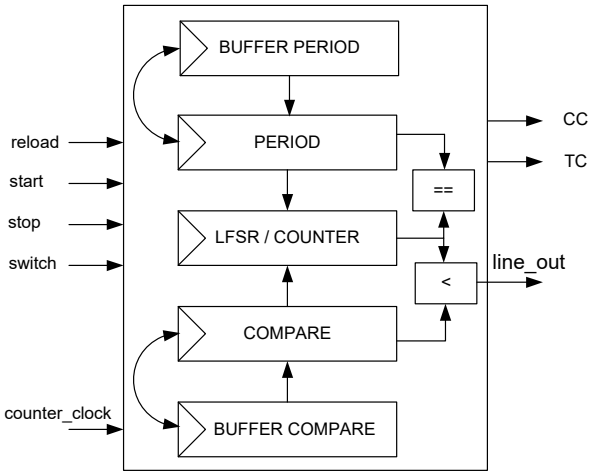
1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select PWM mode by writing '100' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set clock prescaling by writing to the GENERIC[10:8] field of the TCPWM_CNTx_CTRL register, as shown in Table 17-1.
4. Set the required 16-bit period in the TCPWM_CNTx_PERIOD register and the buffer period value in the TCPWM_CNTx_PERIOD_BUFF register to switch values, if required.
5. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register to switch values, if required.
6. Set the direction of counting by writing to the UP_DOWN_MODE[17:16] field of the TCPWM_CNTx_CTRL register to configure left-aligned, right-aligned, or center-aligned PWM, as shown in Table 17-6.
7. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNTx_CTRL register as required.
8. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, Switch, and Count).
9. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, Switch, and Count).
10. line_out and line _out_compl can be controlled by the TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
11. If required, set the interrupt upon TC or CC condition, as shown in Interrupts on page 112.
12. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register. A start trigger must be provided through firmware (TCPWM_CMD register) to start the counter if the hardware start signal is not enabled.

17.3.3 Pulse Width Modulation Pseudo-Random Mode

This mode uses the linear feedback shift register (LFSR), whose input bit is a linear function of its previous state. In this mode, the reload, start, stop, and switch signals are all software triggers. No wire bonding options are available to perform hardware trigger from an external GPIO.

17.3.3.1 Block Diagram

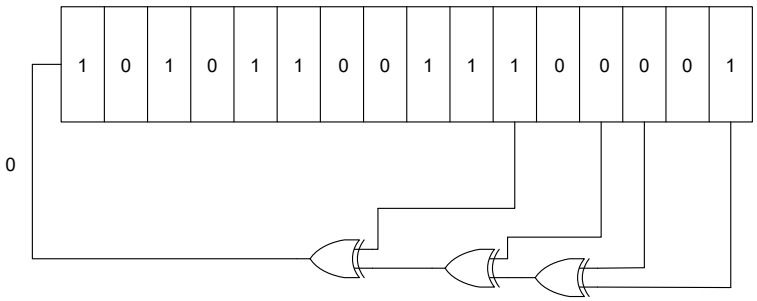
Figure 17-9. PWM-PR Mode Block Diagram



17.3.3.2 How It Works

The counter register is used to implement LFSR with the polynomial: $x^{16}+x^{14}+x^{13}+x^{11}+1$, as shown in Figure 17-10. It generates all the numbers in the range [1, 0xFFFF] in a pseudo-random sequence. Note that the counter register should be initialized with a non-zero value.

Figure 17-10. Pseudo-Random Sequence Generation using Counter Register

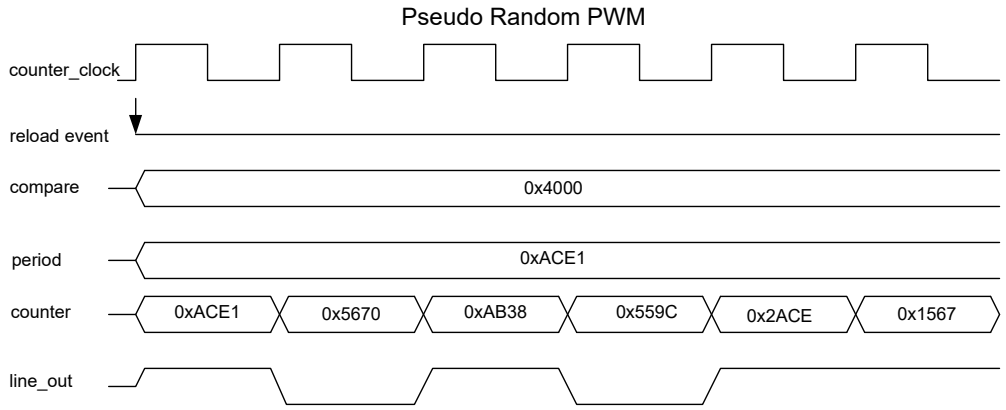


The following steps describe the process:

- The PWM output line ‘line_out’ is driven with '1' when the lower 15-bit value of the counter register is smaller than the value in the compare register (when counter[14:0] < compare[15:0]). A compare value of ‘0x8000’ or higher always results in a '1' on the PWM output line. A compare value of ‘0’ always results in a '0' on the PWM output line.
- A reload event behaves similar to a start event; however, it does not initialize the counter.
- Terminal count is generated when the counter value equals the period value. LFSR generates a predictable pattern of counter values for a certain initial value. This predictability can be used to calculate the counter value after a certain amount of LFSR iterations ‘n’. This calculated counter value can be used as a period value and the TC is generated after ‘n’ iterations.
- At TC, a switch/capture event conditionally switches the compare and period register pairs (based on the AUTO_RELOAD_CC and AUTO_RELOAD_PERIOD fields of the counter control register).
- A kill event can be programmed to stop the counter as described in previous sections.
- One shot mode can be configured by setting the ONE_SHOT field of the counter control register. At terminal count, the counter is stopped by hardware.
- In this mode, underflow, overflow, and trigger condition events do not occur.

- CC condition occurs when the counter is running and its value equals compare value. Figure 17-11 illustrates pseudo-random noise behavior.
- A compare value of 0x4000 results in 50 percent duty cycle (only the lower 15 bits of the 16- bit counter are used to compare with the compare register value).

Figure 17-11. Timing Diagram for Pseudo-Random PWM



A capture/switch input signal may switch the values between the compare and compare buffer registers and the period and period buffer registers. This functionality can be used to modulate between two different compare values using a trigger input signal to control the modulation.

Note Capture/switch input signal can only be triggered by an edge (rising, falling, or both). This input signal is remembered until the next terminal count.

17.3.3.3 Configuring Counter for Pseudo-Random PWM Mode

The steps to configure the counter for pseudo-random PWM mode of operation and the affected register bits are as follows.

1. Disable the counter by writing '0' to the COUNTER_ENABLED field of the TCPWM_CTRL register.
2. Select pseudo-random PWM mode by writing '110' to the MODE[26:24] field of the TCPWM_CNTx_CTRL register.
3. Set the required period (16 bit) in the TCPWM_CNTx_PERIOD register and buffer period value in the TCPWM_CNTx_PERIOD_BUFF register to switch values, if required.
4. Set the 16-bit compare value in the TCPWM_CNTx_CC register and the buffer compare value in the TCPWM_CNTx_CC_BUFF register to switch values.
5. Set the PWM_STOP_ON_KILL and PWM_SYNC_KILL fields of the TCPWM_CNTx_CTRL register as required.
6. Set the TCPWM_CNTx_TR_CTRL0 register to select the trigger that causes the event (Reload, Start, Kill, and Switch).
7. Set the TCPWM_CNTx_TR_CTRL1 register to select the edge that causes the event (Reload, Start, Kill, and Switch).
8. line_out and line_out_compl can be controlled by the TCPWM_CNTx_TR_CTRL2 register to set, reset, or invert upon CC, OV, and UN conditions.
9. If required, set the interrupt upon TC or CC condition, as shown in Interrupts on page 112.
10. Enable the counter by writing '1' to the COUNTER_ENABLED field of the TCPWM_CTRL register.

17.4 TCPWM Registers

Table 17-9. List of TCPWM Registers

Register	Comment	Features
TCPWM_CTRL	TCPWM control register	Enables the counter block
TCPWM_CMD	TCPWM command register	Generates software events
TCPWM_INTR_CAUSE	TCPWM counter interrupt cause register	Determines the source of the combined interrupt signal
TCPWM_CNTx_CTRL	Counter control register	Configures counter mode, encoding modes, one shot mode, switching, kill feature, dead time, clock prescaling, and counting direction
TCPWM_CNTx_STATUS	Counter status register	Reads the direction of counting, dead time duration, and clock prescaling; checks if counter is running
TCPWM_CNTx_COUNTER	Count register	Contains the 16-bit counter value
TCPWM_CNTx_CC	Counter compare/capture register	Captures the counter value or compares the value with the counter value
TCPWM_CNTx_CC_BUFF	Counter buffered compare/capture register	Buffer register for counter CC register; switches compare value
TCPWM_CNTx_PERIOD	Counter period register	Contains upper value of the counter
TCPWM_CNTx_PERIOD_BUFF	Counter buffered period register	Buffer register for counter period register; switches period value
TCPWM_CNTx_TR_CTRL0	Counter trigger control register 0	Selects trigger for specific counter events
TCPWM_CNTx_TR_CTRL1	Counter trigger control register 1	Determines edge detection for specific counter input signals
TCPWM_CNTx_TR_CTRL2	Counter trigger control register 2	Controls counter output lines upon CC, OV, and UN conditions
TCPWM_CNTx_INTR	Interrupt request register	Sets the register bit when TC or CC condition is detected
TCPWM_CNTx_INTR_SET	Interrupt set request register	Sets the corresponding bits in the interrupt request register
TCPWM_CNTx_INTR_MASK	Interrupt mask register	Mask for interrupt request register
TCPWM_CNTx_INTR_MASKED	Interrupt masked request register	Bitwise AND of interrupt request and mask registers

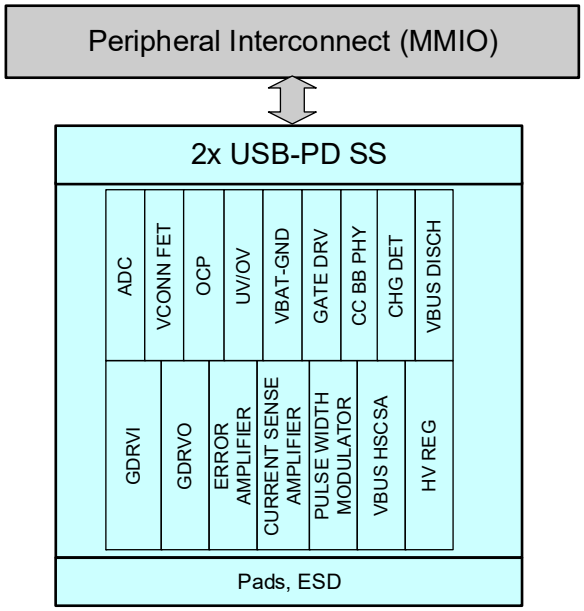
Section F: USB Power Delivery

This section encompasses the following chapter:

- USB Power Delivery chapter on page 125

Top Level Architecture

CCG7D USB Power Delivery Block Diagram



18. USB Power Delivery

CCG7D is highly-integrated dual port USB Type-C Power Delivery (PD) solution with integrated buck-boost controllers. CCG7D uses a built-in USB PD baseband PHY. CCG7D contains two instances of USB-PD port. This PHY supports all the features according to the latest USB Power Delivery 3.0 specification and USB Type-C Cable and Connector specification rev 1.3. It is targeted for automotive charger applications such as head-unit charger, rear seat chargers, and rear seat entertainment.

The USB PD block implements the PD communication using a dedicated Communication Channel (CC) using Bi-phase Mark Coding (BMC). The block contains a dedicated SRAM to store USB PD RX and TX header and messages coming as a part of USB PD negotiations. In addition, it includes all termination resistors required to implement Type-C roles such as DFP and UFP. This block has two 8-bit SAR ADCs, one per USB-PD port.

CCG7D has integrated gate drivers for VBUS NFET on the provider path. It also includes hardware-controlled protection features on the VBUS. CCG7D supports a wide input voltage range (4 V to 24 V with 40-V tolerance) and programmable switching frequency (150 kHz to 600 kHz) in an integrated PD solution.

CCG7D is the most programmable USB-PD solution with on-chip 32-bit Arm® Cortex®-M0 processor, 128KB flash, 16KB RAM, and 32KB ROM that leaves most flash available for user application use. It also includes various analog and digital peripherals such as ADC, PWMs, and timers. The inclusion of a fully-programmable MCU with analog and digital peripherals allows the implementation of custom system management functions such as power throttling, load sharing, temperature monitoring, and black box.

18.1 Features

The USB PD block has the following features:

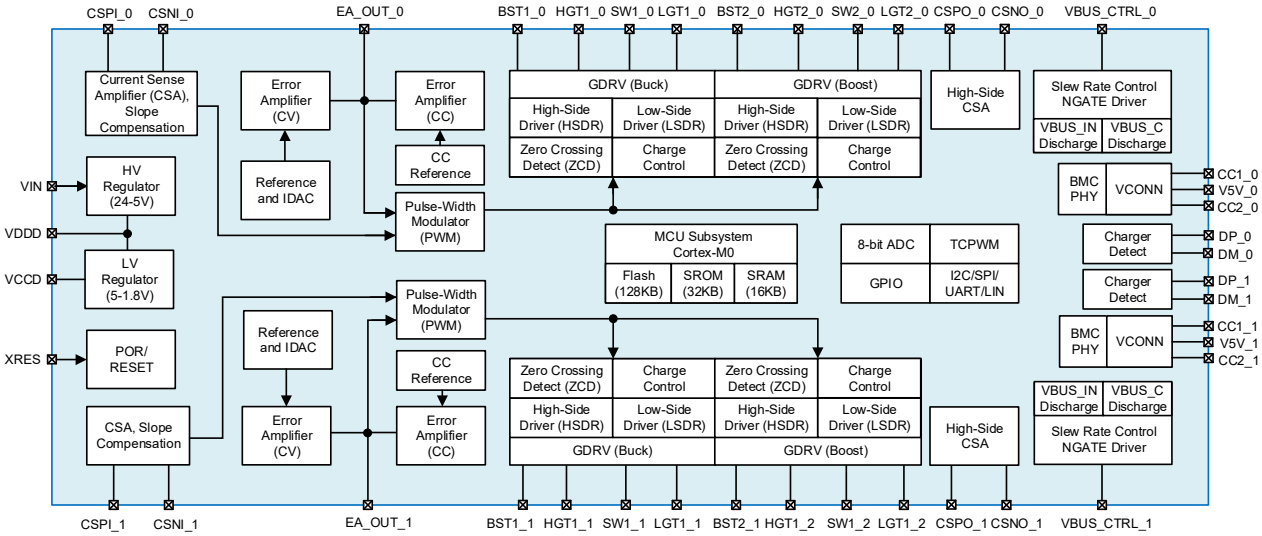
- USB PD PHY implementation according to the USB Power Delivery 3.0 specification
- Complete USB PD BMC transceiver
  - BMC encoder with 300 Kbps bit rate
  - Programmable digital implementation of BMC decoder
- Power delivery PHY and protocol layer
  - 4b/5b encoding/decoding
  - SOP/EOP detection and generation
  - 32-bit CRC generation and compare
  - 64 bytes TX and 64 bytes RX SRAM to hold USB PD header and message
  - Interrupts to report key events (CC valid data, hard reset, message received, message error, and plug insertion)
  - BIST support according to USB Power Delivery 3.0 specification
  - Automatic good CRC message transmission per data/control type
  - USBPD extended data messaging
  - USBPD data chunking
- Integrated DFP (Rp) and UFP (Rd) termination resistors
- Cable detection
  - Type-C connector identification
- AHB Lite interface
- Hot Plug Detect (HPD) functionality according to VESA DisplayPort Alt Mode on USB Type-C Standard

- Supports Deep Sleep mode when block is not transmitting or receiving packets
- Integrated USB Analog Mux
- Integrated OV (over voltage) protection on provider paths
- Integrated OCP protection on the VCONN
- Integrated RC (reverse current), SC (short circuit), and OC (over current) protection on VBUS provider path
- Over temperature thermal shutdown

18.2 Block Diagram

Figure 18-1 shows the USB PD system block diagram.

Figure 18-1. USB PD Block Diagram



The USB PD block implements complete BMC transceivers, the PHY layer, and an AHB Lite infrastructure, which enables the CPU to exchange PD messages with the controller. The block implements a 64-byte TX and RX SRAM to hold a USB PD header and message.

The USB PD block contains:

- A transceiver for communication over the Type-C connector CC1 or CC2 lines.
- Configurable pull-up and pull-down termination to communicate current capabilities and detach/attach events.
- Integrated VCONN FETs
- Integrate over-current (OC) and short-circuit (SC) protection
- NFET gate driver for provider path with slew-rate control feature during turn-on
- Comparators to sense the connected device termination, determine current source capabilities, and detect detach/attach events.
- A positive charge pump boosts the VCCD/VDDD supply input of 1.8 V / 3.3 V to an output of ~5.2 V. This pumped voltage level is used to control the gates of NFET devices, which control the transmission of USB-PD signals.
- An 8-bit DAC and comparator used to form an 8-bit SAR ADC with the appropriate SAR control logic. It is used to measure voltages on the analog multiplexer (AMUX) buses and measure the supply voltage level.
- The analog baseband PHY contains the USB PD transceiver for baseband signaling communication over the CC lines.
- Digital baseband PHY takes care of USB PD protocol handling in the data to and from the analog baseband PHY.
- Local reference generator circuits to enable detection of various voltage and current level inside the USB PD block.
- Integrated USB Analog MUX to route the system D± lines to the Type-C top or bottom pins based on the CC orientation

18.3 How It Works

The USB PD block of CCG7D is designed according to the USB Power Delivery 3.0 specification. This block is divided into:

- Power delivery PHY
- UVOV Block
- SAR Analog to Digital Converter Block
- REFGEN Block
- Provider Path NGDO Block
- VBUS_DISCHG Block
- DPSLP_Ref Block (only one instance and shared between all PD ports)
- VCONN FETs
- VBAT to GND protection
- Charger detection
- Gate drivers for input and output side
- Error amplifier
- High-side CSA for VIN
- Pulse-width modulator
- High-side CSA for VBUS
- VIN regulator

18.3.1 Power Delivery PHY

This PHY (consisting of analog and digital parts) takes care of all the USB PD protocols over the BMC protocol and USB PD transceiver for baseband signaling communication over the CC lines. This block, as shown in Figure 18-2, contains the USB PD transceiver for baseband signaling communication over the CC1/CC2 lines. The transmitter (TX) drives 1.125-V slew-rate controlled data onto the chosen CC line. The receiver (RX) converts received data on the chosen CC line to a full-rail signal for decoding. The comparator (COMP) block contains two deep-sleep comparators used to detect attach/detach and CC line activity events, and determine the voltage range on the chosen CC line. The idle voltage on the CC lines is generated by the voltage divider of the pull-up and pull-down termination, which indicates the advertised current capability of the DFP. The reference generator block generates the necessary voltages for the transmitter voltage, receiver threshold, and comparator thresholds. Each of these voltage references can be trimmed separately, as indicated by the v_*_trim<2:0> signal going to the Reference Trim block.

The IPU blocks generate current sources for the pull-up termination on the CC lines; they are configurable to indicate which current profile the DFP or DRP supports.

When used in a DFP application and no cable is attached, it is important to have the lowest possible deep-sleep power consumption. When waiting for attach, all blocks are disabled except for the Deep-Sleep (DS) Attach Detect block. This implements a pull-up resistor and senses the CC lines to determine if any pull-down termination is attached.

The Rd pull-down resistors are used for the UFP termination. When no power is present, the dead battery termination is presented by the Rd_db pull-down resistors. The impedance of these resistors will be presented if the voltage on the CC line rises above a certain threshold, even if the rest of the device is not powered. The Rd_db termination is brought out to separate rd1_term and rd2_term ports, so that chip-level connections can determine if dead battery termination is present on the CC lines.

18.3.1.1 Analog Baseband PHY

Figure 18-2. Analog Baseband PHY

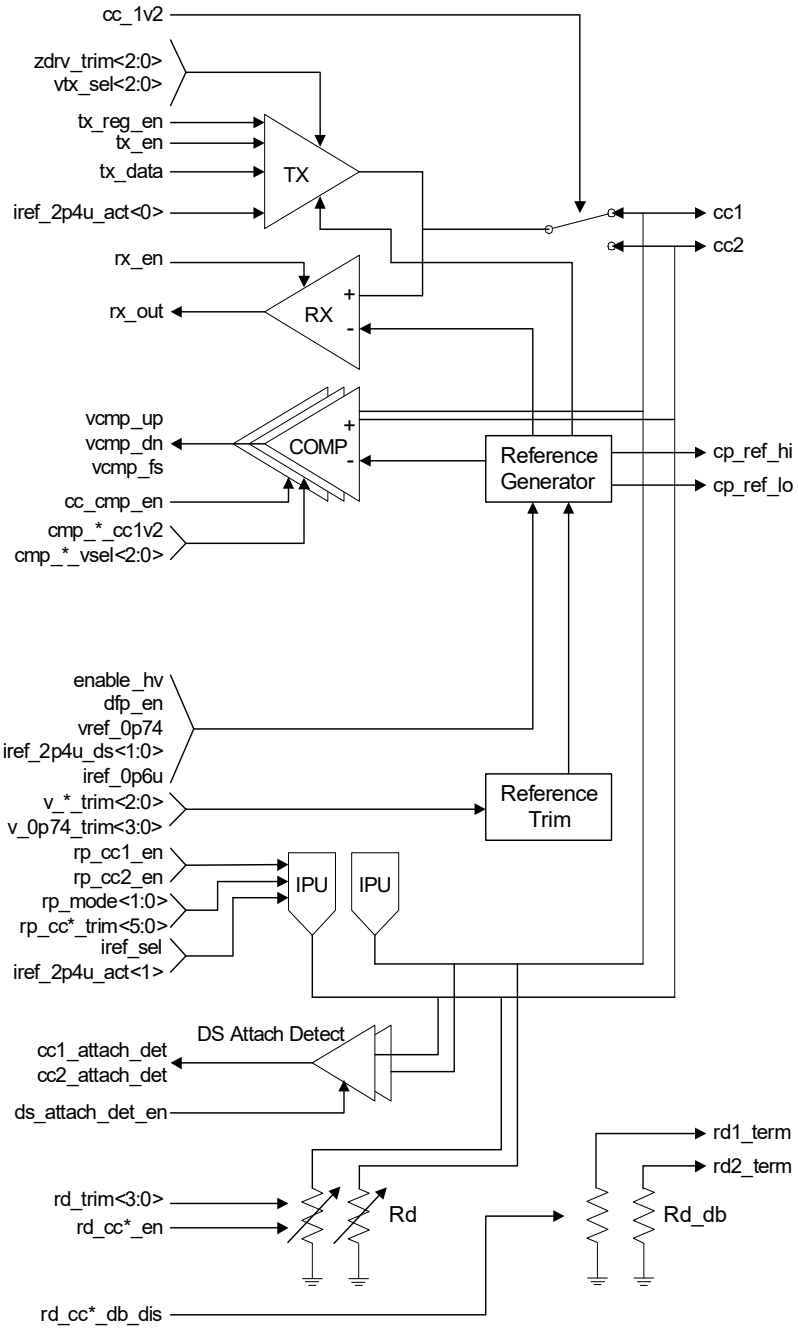


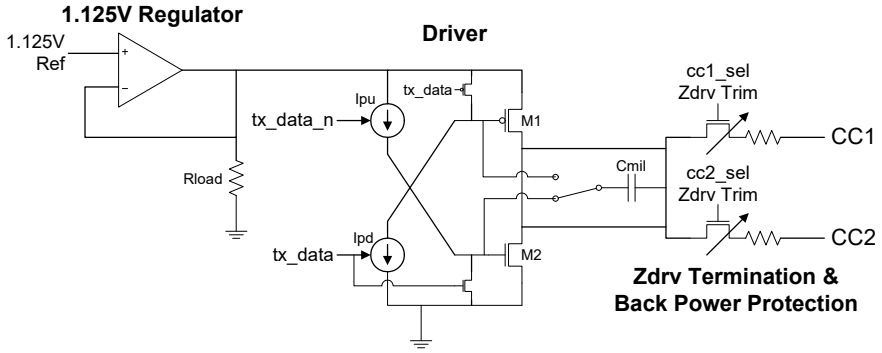
Table 18-1. Configuration Options for Type-C Applications

Configuration	Rp1	Rp2	Rd1	Rd2	Ra1	Ra2
DFP	CC1	CC2				
UFP			CC1	CC2		
DRP	CC1	CC2	CC1	CC2		

Table 18-1 shows the possible configuration options to support different Type-C applications. The DFP does not need any pull-down termination, so the chip-level CC1 and CC2 pins are connected to the block level CC1 and CC2 ports. The UFP, DRP, and Debug Accessory applications need to have Rd termination on the CC1 and CC2 pins, so they are additionally connected to the pull-down termination resistors. The same physical connections support all three applications.

Analog Baseband Transmitter PHY

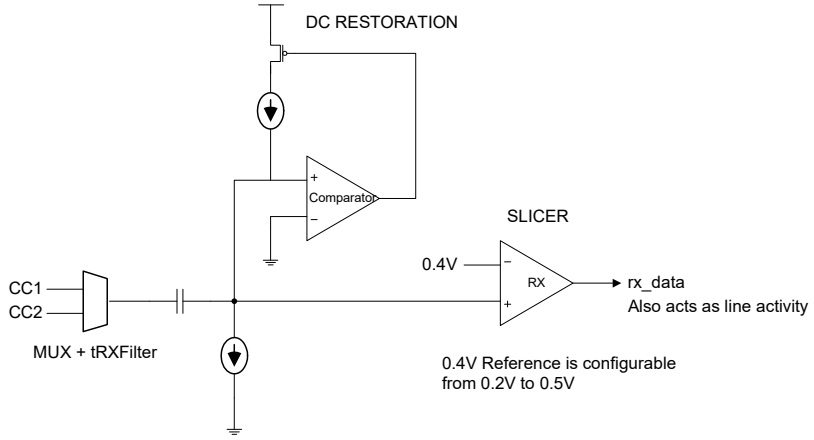
Figure 18-3. Analog Baseband Transmitter PHY



The BMC transmitter is composed of a regulator that generates a local 1.125-V supply, a driver circuit, and termination impedance. The driver circuit uses current sources to drive the gate control of the output drivers. The miller capacitor (Cmil in Figure 18-3) limits the slew rate on the output by keeping the gate voltage near the transistor Vt threshold. Diode-connected devices are also used to bring the gate voltage to the threshold quickly, which improves rise and fall time matching. The driver termination impedance is trimmed by controlling the number of NMOS and resistor paths that are turned on while transmitting. The NMOS gate acts as back power protection when the device is powered down.

Analog Baseband Receiver PHY

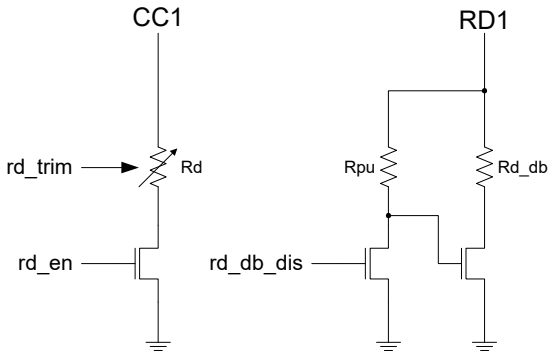
Figure 18-4. Analog Baseband Receiver PHY



One comparator acts as an opamp in a feedback loop to restore the DC level of the input signal. The second comparator is used to slice the data. The “MUX + tRXFilter” implements a mux that chooses between CC1 and CC2 and passes the signal through a 100-ns time constant RC filter. This signal is then AC-coupled into the feedback circuit. The comparator takes a voltage reference on its negative input and the DC restore node on its positive input. There is a weak pull-down current source on the DC restore point node. Whenever the voltage goes below the reference, the comparator turns on the pull-up current source to restore the voltage. When a transmit or noise signal is present on the CC line, it couples through the capacitor. If there is a DC shift, then the feedback loop will eventually shift the low point back to the reference, while the AC component is passed through. The difference is the signal magnitude slice value. If the signal swing is larger than the reference difference, then the signal is received by the comparator and output onto the rx_data output.

Dead Battery Detection Function

Figure 18-5. Dead Battery Detection



The USB PD block has two types of Rd termination for the CC1 and CC2 lines. The first is powered Rd, which has a specified maximum variation of ±10 percent. To meet this requirement, the trim is implemented to center the resistance value. The dead battery termination is implemented with a weak pull-up resistor from the Rd1 terminal to the gate controlling the Rd_db resistance. When the device is powered down, this node will be pulled high when Rd1 goes high, turning on the Rd_db resistance. This resistance is disabled when the device is powered by driving the Rd_CCx_DB_DIS field of the CC_CTRL_0 register high. The CC1 and Rd1 terminals are shorted at the device application level as necessary.

18.3.1.2 Digital Baseband PHY

The digital baseband PHY takes care of USB PD protocol handling in the data to and from the analog baseband PHY. The receive path logic does BMC decoding, detects SOP from incoming data bit stream from the analog baseband PHY, and performs the 5b/4b decoding before sending data for CRC32 check. The receive data is pushed into RX SRAM after removing SOP and EOP. RX SRAM is used to hold full RX USB PD header and message (64 bytes). Each location of the RX SRAM is mapped to an MMIO address for the CPU to access. The CPU can access the SRAM locations only in a 32-bit transfer.

Similarly, for the transmit path, the TX_data is taken from the TX SRAM (memory to hold full TX USB PD header and messages (64 bytes)). The digital baseband PHY logic fetches data from the TX SRAM and sends it to the analog baseband PHY serially after appending preamble, SOP, EOP, and CRC32, performing 4b/5b encoding and BMC encoding on the data. The transmit data to the analog baseband PHY is serialized using a 300-kHz clock. An interrupt is generated upon sending the last bit of data.

Refer to the digital baseband PHY in Figure 18-6.

18.3.1.3 Protocol Interface Description

Extended Data Messaging. This subsystem IP supports USB PD extended data messaging (according to USB Power Delivery Specification Revision 3.0) and data chunking. To support extended data messaging, the memory size used to store messages is increased from 32-byte to 64-byte in each direction. The extended data messaging is automatic in both directions and it is per packet as long as it is enabled. The USBPDx_HEADER_INFO register is used to enable or disable (EN_RX_EXTENDED_DATA, EN_TX_EXTENDED_DATA) this feature for each direction.

Transmit Path. In the transmit path, firmware must ensure that the data is written into TX_MEM_DATA to avoid under flow.

Follow these steps to process an extended data messaging packet in the transmit direction:

1. Firmware programs USBPDx_HEADER_INFO.
2. Firmware programs USBPDx_TX_HEADER.
3. Firmware writes data to USBPDx_TX_MEM_DATA.
4. Firmware requests hardware to start transmitting the packet by setting TX_GO in TX_CTRL register.
5. Hardware clears the TX_GO and start transmitting the packet.
6. Hardware generates USBPDx_INTR0.TX_SRAM_HALF_END informing firmware that it has transmitted the lower 32-bytes of TX memory. Firmware should write the next 32-byte packet in the lower 32-bytes of memory. This must be done before hardware finishes transmitting the upper 32-bytes of TX memory; otherwise, an under flow will occur and hardware will create USBPDx_INTR2.TX_SRAM_UNDER_FLOW. In the under flow condition, hardware will keep reading the TX_MEMORY for the total packet size and if the USBPDx_TX_CTRL.TX_CORRUPT_CRC_ON_UNDER_FLOW is set, hardware will corrupt TX CRC.
7. Hardware also generates USBPDx_INTR0.TX_SRAM_HALF_END after it transmits the upper 32-bytes of TX memory.
8. Firmware can use USBPDx_SRAM_PTR.TX_FUNC_Rd_PTR to see which location of the TX_MEMORY hardware is processing.

Receive Path. In the receive path, firmware must ensure that the data is read from USBPDx_RX_MEM_DATA to avoid over flow.

Follow these steps to process an extended data messaging packet in the receive direction:

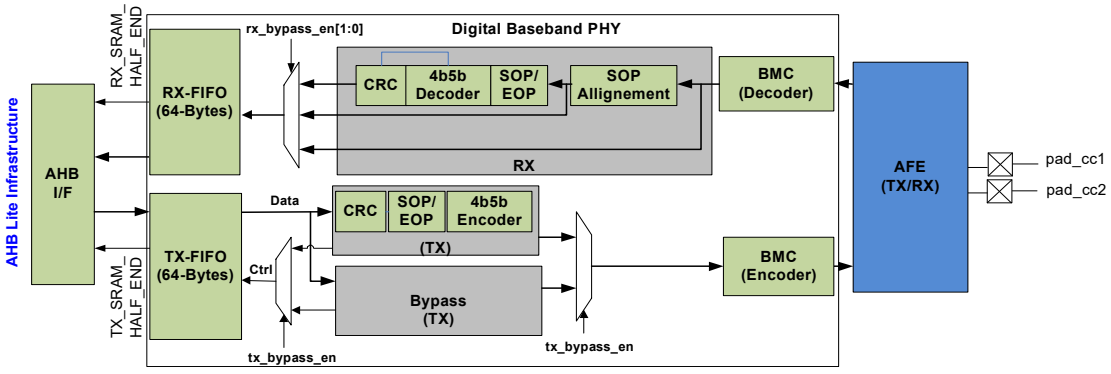
1. Firmware programs USBPDx_HEADER_INFO.
2. Hardware receives an extended data message and generates USBPDx_INTR2.EXTENDED_MSG_DET.
3. Hardware writes the header to USBPDx_RX_HEADER and data to USBPDx_RX_MEM_DATA.
4. Hardware generates USBPDx_INTR0.RX_SRAM_HALF_END informing firmware that hardware has stored 32-bytes of data in the lower 32-bytes of RX memory. Firmware must read the 32-byte before hardware storing the next 32-bytes in the upper 32-bytes of RX memory; otherwise, an over flow will occur and hardware will create USBPDx_INTR2.RX_SRAM_OVER_FLOW. In the over flow condition, hardware will keep writing the incoming packet to RX_MEMORY.
5. Hardware also generates USBPDx_INTR0.RX_SRAM_HALF_END after it stores packet in the upper 32-bytes of RX memory.
6. Firmware can use USBPDx_SRAM_PTR.RX_FUNC_Rd_PTR to see which location of the RX_MEMORY hardware is processing.

Bypass Mode. This subsystem IP is designed with a provision to allow firmware to do most of the physical layer processing, except the BMC encoding/decoding. To allow firmware to perform most functions of the physical layer, including when the firmware is running on a remote microcontroller, several functions inside the subsystem can be bypassed.

The bypass mode is used for both normal packet traffic and BIST operations.

The block diagram of the data path is shown in Figure 18-6. The top and bottom halves of the diagram show the RX and TX path respectively.

Figure 18-6. Data Path for Digital Baseband PHY



Transmit Path. As shown in Figure 18-6, there is one bypass option for the transmit path. When this option is enabled, the CRC generation, SOP/EOP insertion, 4b5b encoding steps are bypassed and the data is presented directly to the BMC encoder. This is achieved by setting the USBPDx_CTRL.TX_BYPASS_EN register. The data is read from the TX-FIFO and transferred to BMC encoder. In this mode, the firmware programs the length of the data bits to be transferred and the hardware asserts appropriate control signals to complete the transfer under firmware control.

Although the transmit buffer size (TX_FIFO) is only 64-bytes, when in bypass mode, the maximum data size is not limited to 64 bytes. The max data bit size (2047 bits) is defined by USBPDx_CRC_COUNTER.CRC_COUNTER [10:0] register.

In Transmit Bypass mode, the following steps take place:

1. Firmware programs the USBPDx_CRC_COUNTER .CRC_COUNTER[10:0] with the total number of bits to transmit. This is the size of the complete packet.
2. Firmware writes the data in the TX-FIFO. If the packet is more than 64 bytes, the USBPDx_INTR0.TX_SRAM_ HALF_END interrupt can be used to write the rest the data in the TX-FIFO during data transfer. The first USBPDx_INTR0.TX_SRAM_HALF_END interrupt indicates that the lower 32 bytes of TX-FIFO have been transmitted and are available for firmware to write the rest of the data. The second USBPDx_INTR0.TX_SRAM_HALF_END interrupt indicates that upper 32 bytes of TX-FIFO have been transmitted and it is available for firmware to write the rest of the data.
3. To start transmitting, write ‘1’ to USBPDx_TX_CTRL.TX_GO.
4. The USBPDx_TX_CTRL.TX_GO will be cleared by hardware when the transaction starts.
5. The USBPDx_INTR0.TX_PACKET_DONE will be set when all the data bits specified by USBPDx_CRC_COUNTER.CRC_COUNTER[10:0] is transferred.

Receive. As shown in Figure 18-6, there are two bypass receive functions. In both cases, the SOP/EOP, 4b5b, and CRC functions are bypassed.

- When USBPDx_CTRL.RX_BYPASS_EN[1:0] is ‘1’, the data written into RX-FIFO is aligned with SOP. All the valid data after SOP alignment is stored in the RX_FIFO.
- When USBPDx_CTRL.RX_BYPASS_EN[1:0] is ‘2’, the data written into RX_FIFO is not aligned with SOP and all the valid data received from BMC decoder is store in RX_FIFO as individual bits. Firmware must detect the SOP and data alignment from these assembled bits.

If the size of incoming data is more than 64 bytes, the write pointer rolls over. The first USBPDx_INTR0.RX_SRAM_ HALF_END interrupt indicates that the lower 32 bytes of data is available for firmware to be read. The second interrupt indicates the upper 32 bytes is also available to be read. USBPDx_INTR0. CC_NO_VALID_DATA_DETECTED indicates the end of data reception.

18.3.2 UVOV Block on VBUS_C

18.3.2.1 UVOV Block Architecture and Functional Description

Figure 18-7. UVOV Block Diagram (CCG7D)

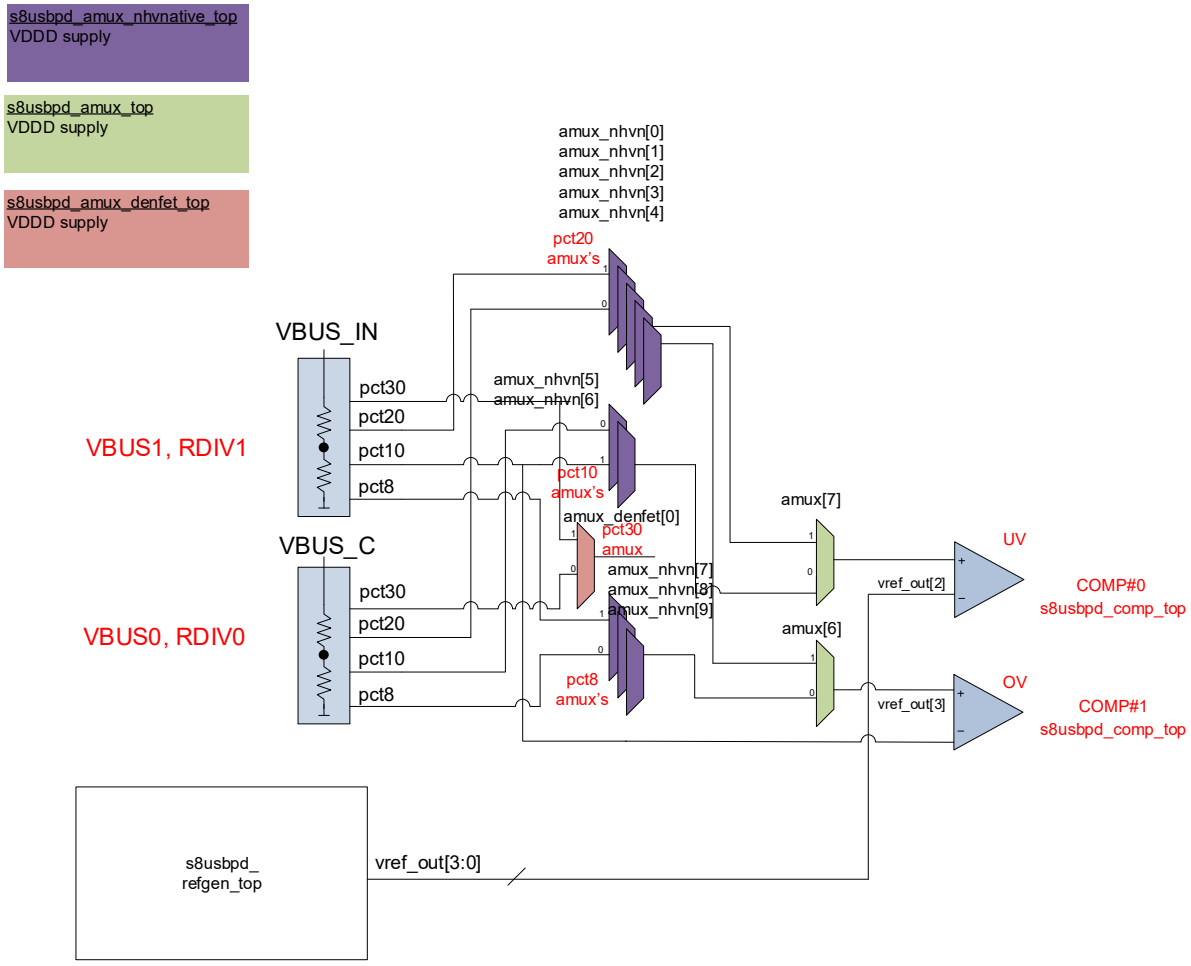


Figure 18-7 shows how the UVOV block is used to detect the over-voltage and under-voltage conditions on the VBUS input line. The threshold levels for UV and OV detection can be set independent of each other. Both UV and OV detectors have programmable thresholds and can be programmed in 10 mV increments within the range of 200 mV to 2.1 V.

A single resistor string from VBUS is used to drop the voltage from VBUS down to zero. 30%, 20%, 10% or 8% of the VBUS voltage is selected based on the AMUX_NHV_CONTROL register value and connected to the comparator. The comparator compares the switch output to a configurable reference voltage from the RefGen block.

18.3.2.2 CCG7D Filtering Comparator Output

The VBUS comparators on CCG7D are associated with an output filter which can be used to avoid false trips due to a momentary glitch or surge on the comparator input side. Table 18-2 shows the mapping of each comparator with the appropriate filter on the device.

As shown, the UV/OV comparator outputs can be filtered using a fast filter which uses an IMO derived clock. The filter can be bypassed completely, if so required.

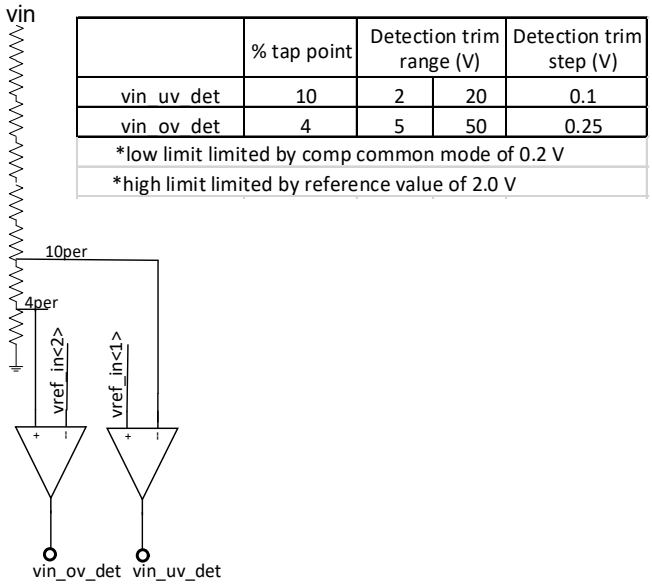
Table 18-2. CCG7D Comparator Configuration and Interrupt Mapping

Comparator Name	Comparator ID	Filter Clock	Filter Number	Filter Configuration Register	Interrupt Status
UV	#0	Can be CLK HF or CLK LF based on USBPD_CTRL.SEL_CLK_FILTER	#0	INTR5_FILTER_CFG[0]	INTR5.EDGE_CHANGED[0]
OV	#1		#1	INTR5_FILTER_CFG[1]	INTR5.EDGE_CHANGED[1]
CBL_1	#2		#2	INTR5_FILTER_CFG[2]	INTR5.EDGE_CHANGED[2]
DISCHG	#3		#3	INTR5_FILTER_CFG[3]	INTR5.EDGE_CHANGED[3]
CBL_0	#4		#4	INTR5_FILTER_CFG[4]	INTR5.EDGE_CHANGED[4]
VSRC_NEW_M	#5	CLK-LF	#0	INTR7_FILTER_CFG[0]	INTR7.EDGE_CHANGED[0]
VSRC_NEW_P	#6	CLK-LF	#1	INTR7_FILTER_CFG[1]	INTR7.EDGE_CHANGED[1]

18.3.3 UVOV Block on VIN

Figure 18-8 shows the implementation of VIN under-voltage/over-voltage detection. 10 percent tap points are taken for under-voltage detection and 4 percent tap points are taken for over-voltage detection. The figure mentions the range and step values.

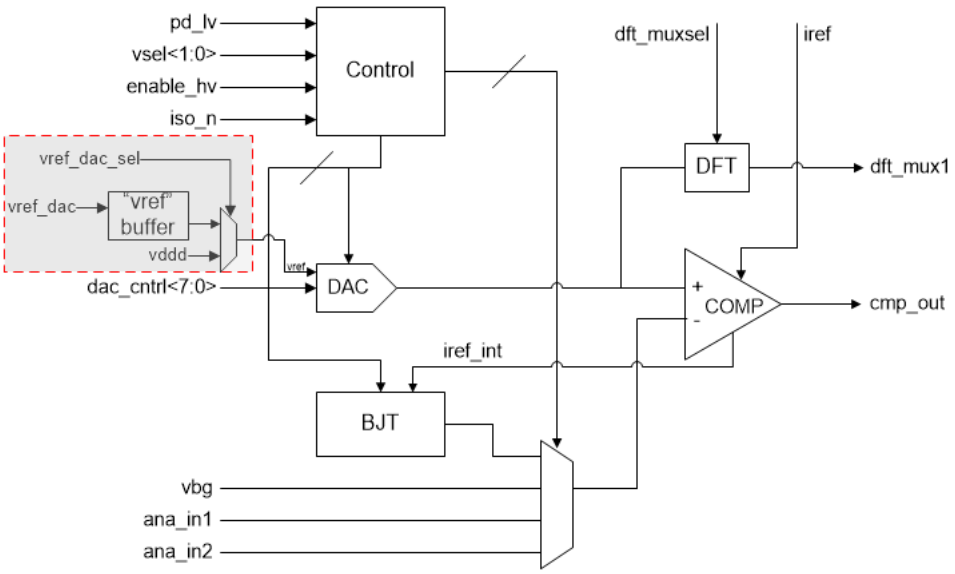
Figure 18-8. VIN UVOV Detection



	% tap point	Detection trim range (V)		Detection trim step (V)
vin_uv_det	10	2	20	0.1
vin_ov_det	4	5	50	0.25
*low limit limited by comp common mode of 0.2 V				
*high limit limited by reference value of 2.0 V				

18.3.4 SAR Analog to Digital Converter Block

Figure 18-9. SAR ADC Block



This block contains the analog circuitry necessary to implement an 8-bit Successive Approximation Register (SAR) ADC. The block contents are: control logic for level shifting and power down, 8-bit DAC, DFT switch, comparator (COMP), and a BJT NPN device. Figure 18-9 shows these blocks.

The DAC takes an 8-bit control signal and outputs a voltage referenced to the supply voltage. The comparator compares the DAC voltage versus one of the following input voltages: either the amux_a or amux_b line, the bandgap voltage (vbg), or the voltage generated from sourcing current into a diode-connected BJT NPN device. If the DAC voltage potential is greater than the multiplexed voltage, then the comparator digital output is high. The BJT NPN device may be used as a temperature sensor as well.

18.3.5 REFGEN Block

18.3.5.1 REFGEN Functional Description

There are two reference generator blocks, one per port to provide an additional layer of reference signal derivation by providing the following:

- Common selection of the source Vref signal to be either the bandgap (1.2 V) or beta-multiplier (0.74 V) reference.
- Voltage, current, and clock reference signals necessary to support the following blocks:
  - UVOV Block
  - PWM Block
  - High-side 20CSA Block
  - High-side 40CSA Block
  - ADC Block
  - EA Block
  - 40Vreg
  - CC Phy

Table 18-3. Mapping between Refgen Signals and Blocks

Refgen Out	Reference Used by Block	REFGEN_X_CTRL Register Field	Description
vref_out[0]	PWM, vsrc_new_p[0]	SEL0	Reference for the PWM block and vsrc_new_p comparator
vref_out[1]	PWM, vsrc_new_m[0]	SEL1	Reference to PWM block and vsrc_new_m comparator
vref_out[2] / vref_uv	UV	SEL2	Reference for Type-C VBUS under-voltage detect comparator.
vref_out[3] / vref_ov	OV	SEL3	Reference for Type-C VBUS over-voltage detect comparator.
vref_out[4]	20CSA (SCP)	SEL4	Reference to high side 20CSA block to detect Short Current Protection (OCP).
vref_out[5]	20CSA (OCP)	SEL5	Reference to high side 20CSA block to detect Over Current Protection (OCP).
vref_out[6]	Reference for CBL_1 Comparator	SEL6	Reference to CBL_1 (PFC) Comparator
vref_out[7]	Reference for ILIM	SEL7	Reference to high side 40 CSA block to implement current limit
vref_out[8]	Reference for CBL_0 Comparator	SEL8	Reference to CBL_0 (SR) comparator
vref_out[9]	Reference for PWM comparator	SEL9	Reference to PWM skip comparator
vref_out[10]	Reference for EA	SEL10	Reference to EA CC block
vref_out[11]	Reference for discharge Comparator	SEL11	Reference to discharge enable comparator
vref_out[12]	Reference for 40Vreg	SEL12	0.74V Reference to VDDD regulator
vref_out[13] / vref_adc	Reference for ADC	SEL13	Reference to 8-bit ADC.
vref_out[14]	Reference for CC Phy (1.1 V)	SEL14	Reference to CC PHY used by transmitter and receiver.

18.3.6 Provider Path NGDO Block

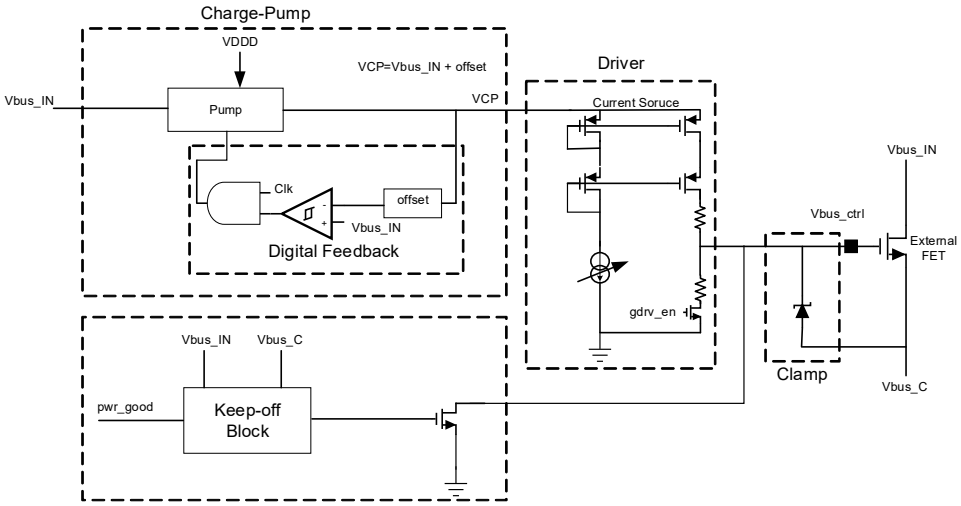
18.3.6.1 Provider Path NGDO Block Architecture and Functional Overview

Figure 18-10 shows the block diagram of the NMOS power FET gate driver block. The charge-pump block is used to boost the voltage to provide higher gate voltage for the external power-NFET. A comparator circuit is added at the output of charge pump which compares the charge pump output to VBUS + 8 V and turns off the clock whenever charge pump output goes higher than VBUS by 8 V. This is to ensure that Vbus_ctrl never goes higher than VBUS + 10 V.

The external NFET gate is charged to the charge pump output voltage using a slew rate control block which contains programmable current source. This is done to avoid sudden turn-on of the external NFET.

The NGDO also contains SHV clamp block which makes sure that Vgs of the external NFET is maintained < 12 V once it has turned on. During turn-off, NGDO gate could be at 0 V with 20 V on VBUS_IN or VBUS_C, i.e., to have -20 V VGS/VGD on external NFET.

Figure 18-10. Provider Path NGDO Block Diagram



18.3.7 VBUS_DISCHG Block

18.3.7.1 VBUS_DISCHG Block Architecture and Functional Overview

Figure 18-11 shows the block diagram of the VBUS_DISCHG block, which has a 21.5-V drain extended NFET connected from VBUS_IN supply to ground. An additional resistor ladder and a comparator tracks the value of VBUS_IN. VBUS_IN can be discharged to the value set by dischg_ds[4:0].

Figure 18-11. VBUS_DISCHG Block Diagram for VBUS_IN

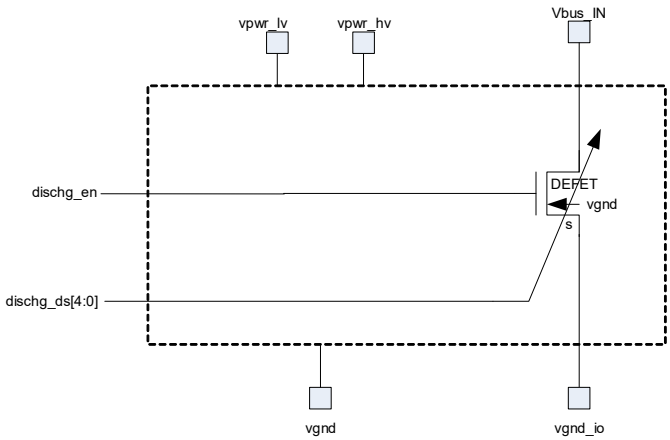
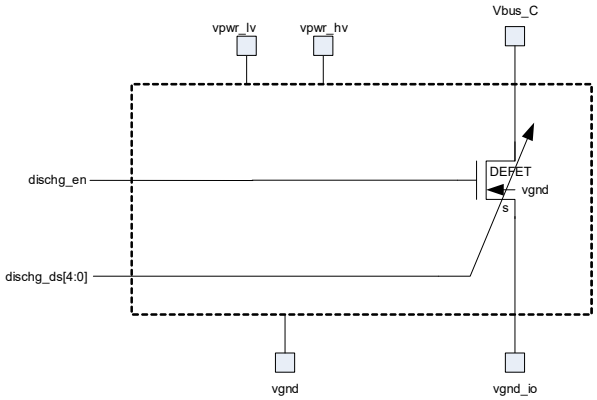


Figure 18-12 shows the block diagram of the VBUS_DISCHG block, which has a 21.5-V drain extended NFET connected from VBUS_C supply to ground. An additional resistor ladder and a comparator track the VBUS_C value. VBUS_C can be discharged to the value set by dischg_ds[4:0]

Figure 18-12. VBUS_DISCHG Block Diagram for VBUS_C



18.3.8 DPSLP_Ref Block

18.3.8.1 Architecture and Function Description

Figure 18-13. DPSLP_Ref Block Diagram

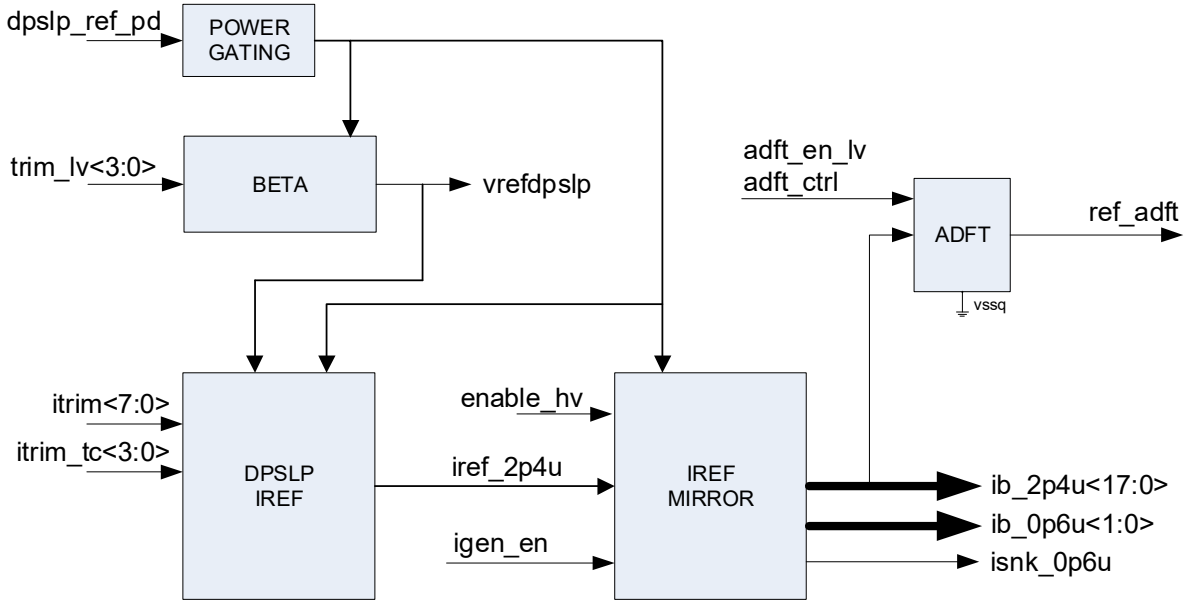


Figure 18-13 shows the DPSLP_Ref block diagram. This block is common to both Port0 and Port1. It consists of three sub-blocks – Beta-Multiplier, Deep-Sleep IREF, and IREF mirror.

The beta multiplier block generates the reference voltage for the System Resources Subsystem block in the Deep Sleep mode. The output of this block (vrefdpslp) is based on the difference between vth of the nhv device and the nhvnative device; its default value is 740 mV.

The Deep-Sleep current reference generator creates a voltage and temperature compensated 2.4-µA reference current. The current reference mirror block takes in one 2.4-µA input and creates 20 copies of the same current by using a precision current mirror. It also creates a 0.6-µA reference. CCG7D has only one instance of this block and it is shared across both USBPD ports.

18.3.9 VCONN FETs

18.3.9.1 Architecture and Function Description

Figure 18-14. VCON Block Diagram

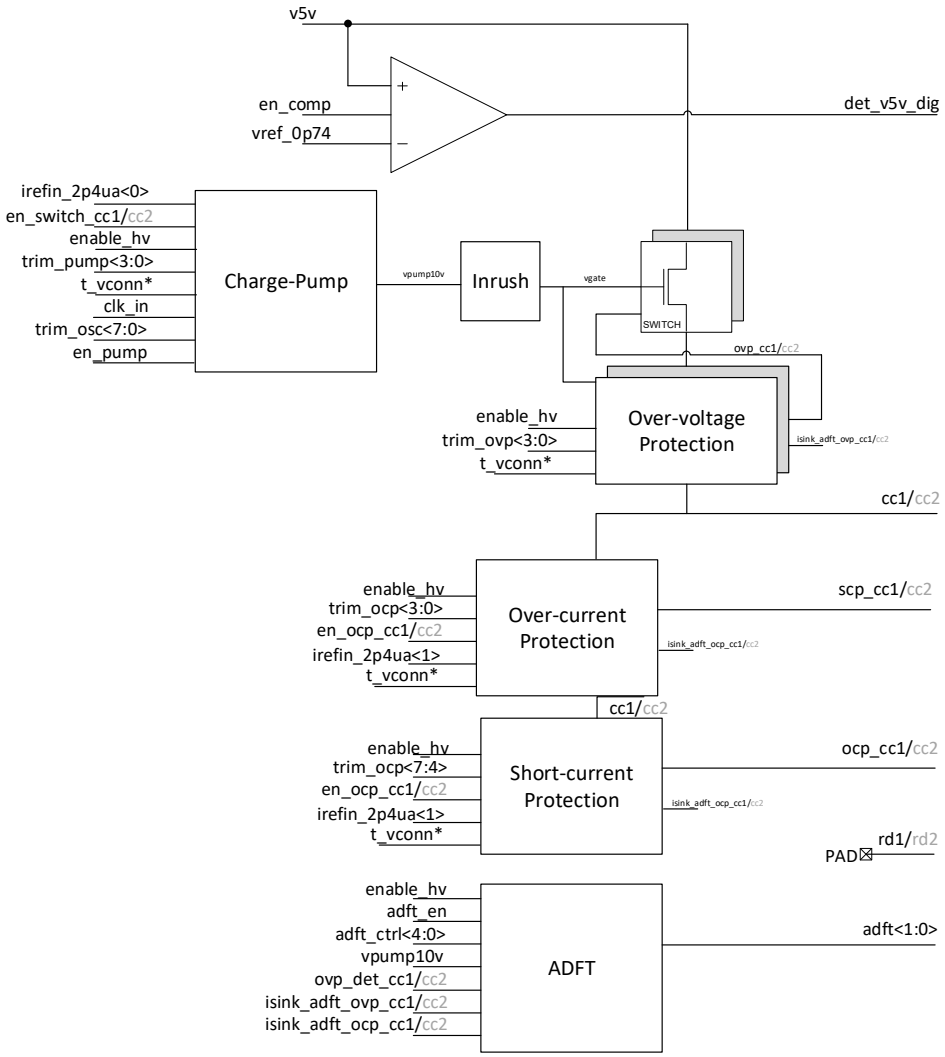


Figure 18-14 shows the block diagram of VCONN block. The main purpose of this block is to provide 100 mW power to CC1/CC2 lines from the V5V input. It is a 5.5-V tolerant block with integrated 20-V over-voltage protection (OVP) on CC1/CC2 pins to protect internal circuits, and V5V against accidental shorts to the VBUS pin on the Type-C connector. However, it has a crude over-voltage protection to avoid reverse current when CC1/CC2 goes higher than V5V when the switch is on. It also has an over-current detection/protection in case of higher current demand from CC1/CC2. A control circuitry is added to limit in-rush current through the switch during power-up. In CCG7D, V5V of the VCONN block is connected to an internally-regulated VDDD.

The VCONN block consists of:

- A comparator block, which is a crude V5V detect and gives out a digital signal to define if V5V is high enough to allow transistors to work.
- A charge pump to give vpump10v (V5V+5.2V (typ)) gate input to enable switching between V5V and CC1/CC2 pins.
- A switch that comprises of a NMOS to enable power (up to 100 mW) output on CC1/CC2 pins from V5V.
- A crude 5 V over-voltage circuit, which trips when CC1/CC2 goes higher than V5V to avoid huge reverse current.

- An over-current protection circuit that trips beyond a predefined threshold of current and sends a signal to firmware to take action (mainly to disable switch).
- An in-rush control circuit to limit the in-rush current during power-up to 100 mA.
- An ADFT block that brings out internal nodes for monitoring during testing.

The switch block is NMOS-based with its control defined at vpump10v (V5V+5.2V) to allow NFET to have high drive to support 100 mW without increasing NMOS size to high.

18.3.10 VBAT to Ground Protection

18.3.10.1 Architecture and Function Description

Figure 18-15. VBAT to Ground Block Diagram

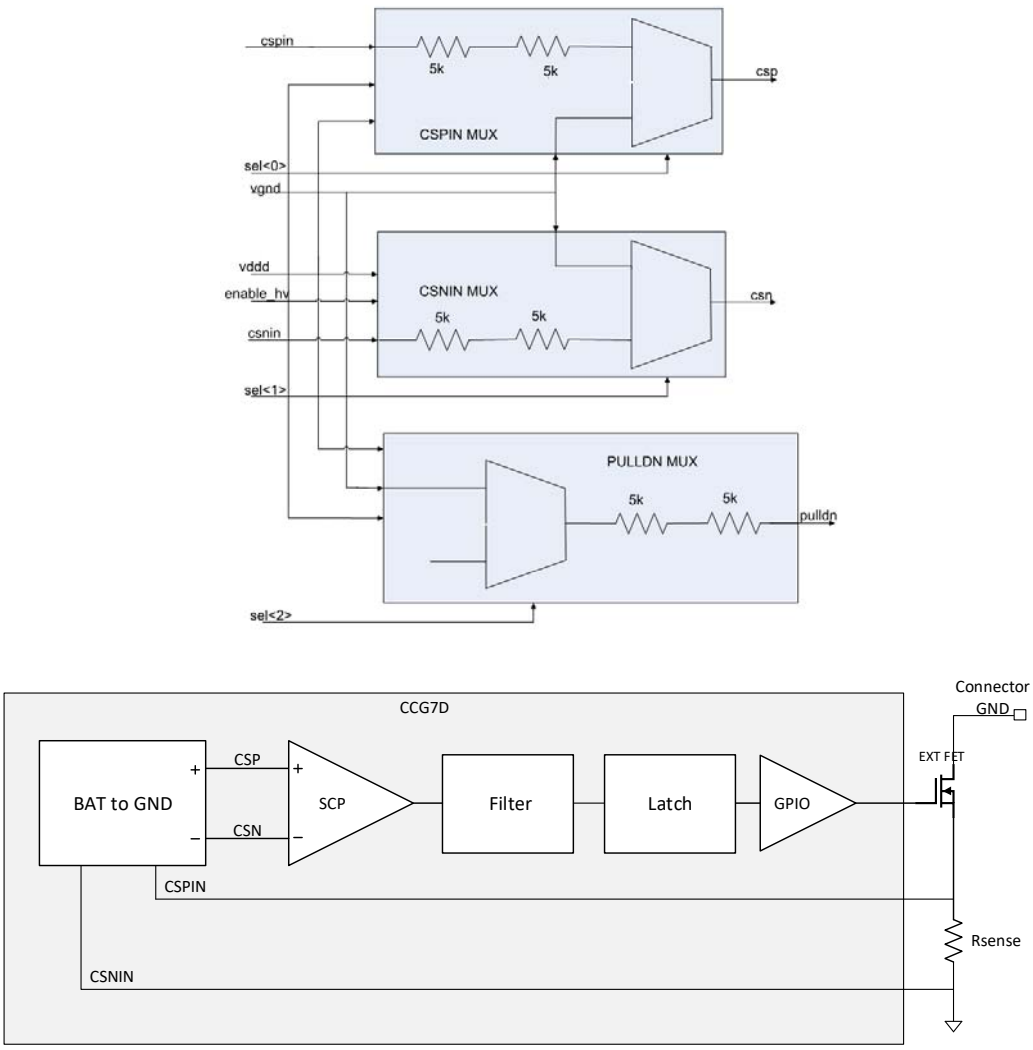


Figure 18-15 shows the block diagram of the VBAT to ground block. This block has the following functionalities:

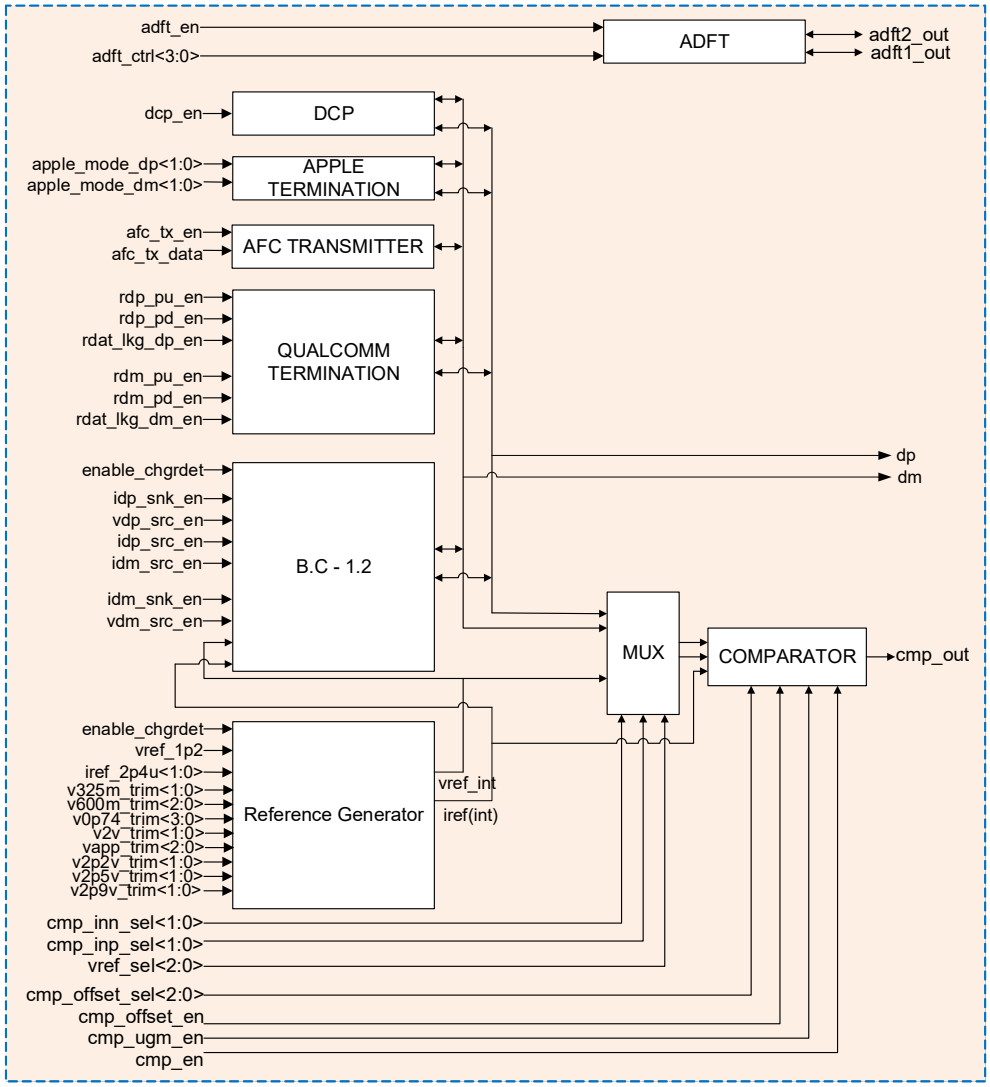
- Allows the SCP inputs to be connected to low-side CSPIN/CSNIN or vgnd.
- Allows low-side CSNIN connection to be routed to GPIO while connecting the SCP to vgnd.
- Discharge path to vgnd on pulldn node.
- Allows pulldn node discharge through GPIO.

18.3.11 Charger Detection

18.3.11.1 Architecture and Function Description

Figure 18-16 shows the CHGDET (charger detect) block's top level architecture diagram. This block implements a charger detection circuit that is compliant with the BC 1.2 specification.

Figure 18-16. CHGDET Block Diagram



18.3.12 Gate Drivers for Input and Output Side of Buck-Boost Switches

18.3.12.1 Architecture and Function Description for Input

Figure 18-17. Gate Driver for Input Block Diagram

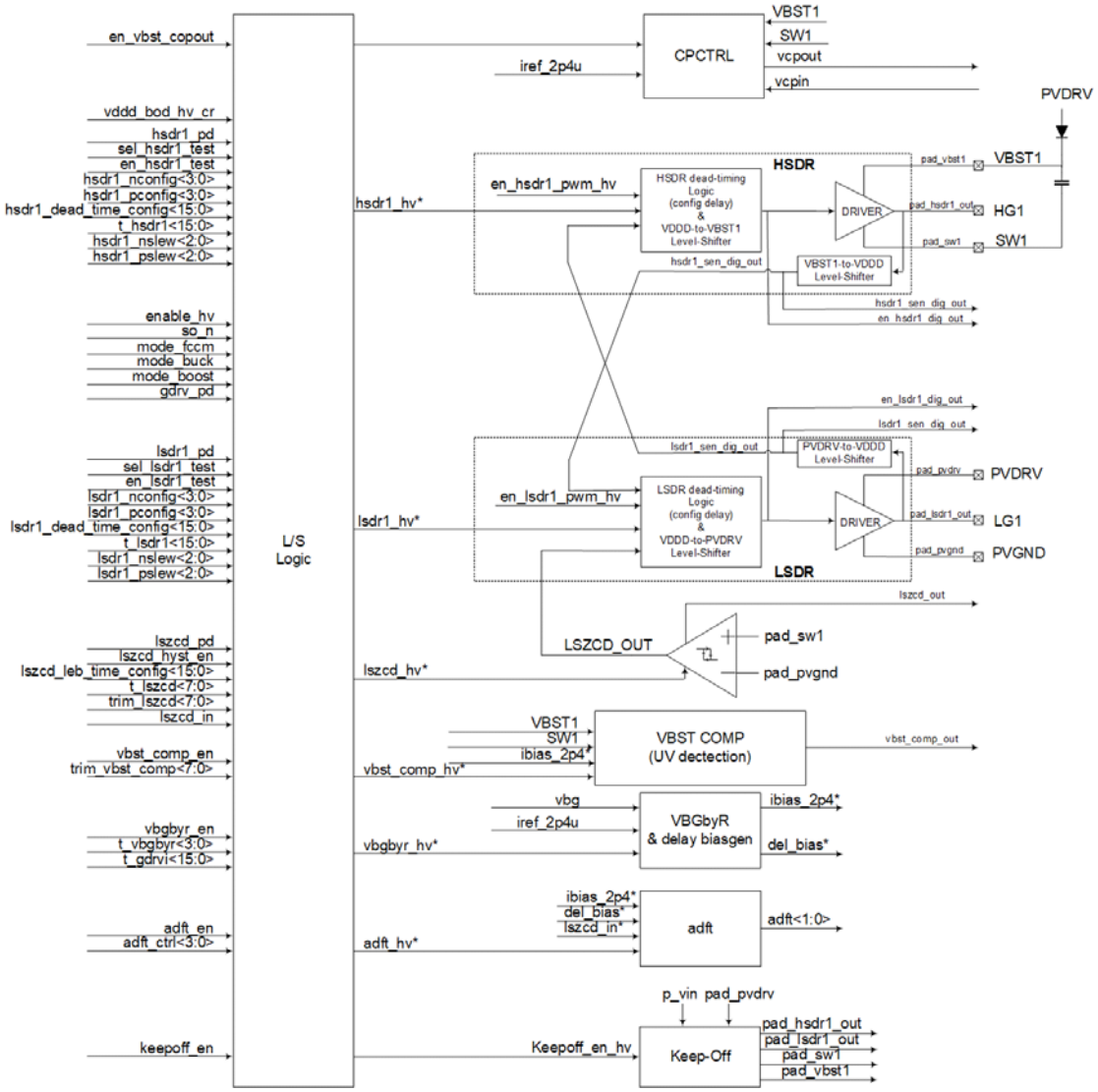


Figure 18-17 shows the top-level block diagram of input side gate driver. The main function of this block is to provide gate control signal for external NFET switches (HS1 and LS1).

The gate drive for input side consists of following sub-blocks:

- **VBGbyR:** A Vbg/R current generation circuit that buffers vref_1p2 (Vbg from SRSS) and generates multiple internal current references (ibias*) used by sub-blocks on this public cell.
- **Keep-off:** Keeps external NFETs (HS1 and LS1) off until VIN and PVDRV attains proper power level.
- **LSDR:** It is a high-speed driver to drive the gate of lower-side external N-channel MOSFET (LS1) and operates cooperatively with high-side driver (HSDR1) to form a synchronous converter.

  LSDR includes switch control logic circuitry, level-shifters, and gate driver. All control input and output signals to and from LSDR are referenced to VDDD. (PVDRV-PVGND) sets the driver output voltage swing. All logic operates in VDDD and the final driver control signal is level-shifted from VDDD-to-PVDRV and Driver operates in PVDRV.

The driver is controlled by PWM. It has a programmable dead timer, which provides enough blanking time to ensure high-side MOSFET is turned off before turning on low-side MOSFET when LSDR is used as a gate driver in synchronous with a high-side MOSFET. A status output signal (lsdr1_sen_dig_out) provides the state of LSDR Out/Gate voltage of low-side external MOSFET(LG1) to protect against cross-conduction when synchronously driving external MOSFETs.

The gate driver strength is programmable and has independent configuration control bits for the pull-up and pull-down on-resistances.

- **LSZCD:** It is a low-side high-speed comparator to detect inductor reverse current from VOUT to PVGND through external MOSFET(LS1) in DCM. It has programmable leading-edge blanking timer to avoid the false LSZCD output.
- **HSDR:** It is a high-speed driver to drive the gate of high-side external N-channel MOSFET(HS1) and operate co-operatively with a low-side driver (LSDR1) to form a synchronous converter.

  HSDR includes switch control circuitry, level-shifters and floating gate high-side drivers. All control input and output signals to and from HSDR are referenced to VDDD. (VBST1-SW1) sets the driver output voltage swing. All logic operates in VDDD and the final driver control signal is level-shifted from VDDD-to-VBST1 and driver operates in (VBST1-SW1).

  The driver is controlled by PWM. It has a programmable dead timer, which provides enough blanking time to ensure low-side MOSFET is turned off before turning on high-side MOSFET when HSDR is used as a gate driver in synchronous with a low-side MOSFET. A status output signal (hsdr1_sen_dig_out) provides the state of HSDR Out/Gate voltage of high-side external MOSFET(HG1) to protect against cross-conduction when synchronously driving external MOSFETs.

  The gate driver strength is programmable and has independent configuration control bits for the pull-up and pull-down on-resistances
- **VBST_COMP:** it is VBST under-voltage detection comparator.
- **VBST_CPCTRL:** The VBST charge pump controller transfers the charge from VBST1 to floating supply VBST2 in buck mode when under voltage detects on VBST2 or receives charge on VBST1 from VBST2 in boost mode when the under-voltage is detected on VBST1.

18.3.12.2 Architecture and Function Description for Output

Figure 18-18. Gate Driver for Output Block Diagram

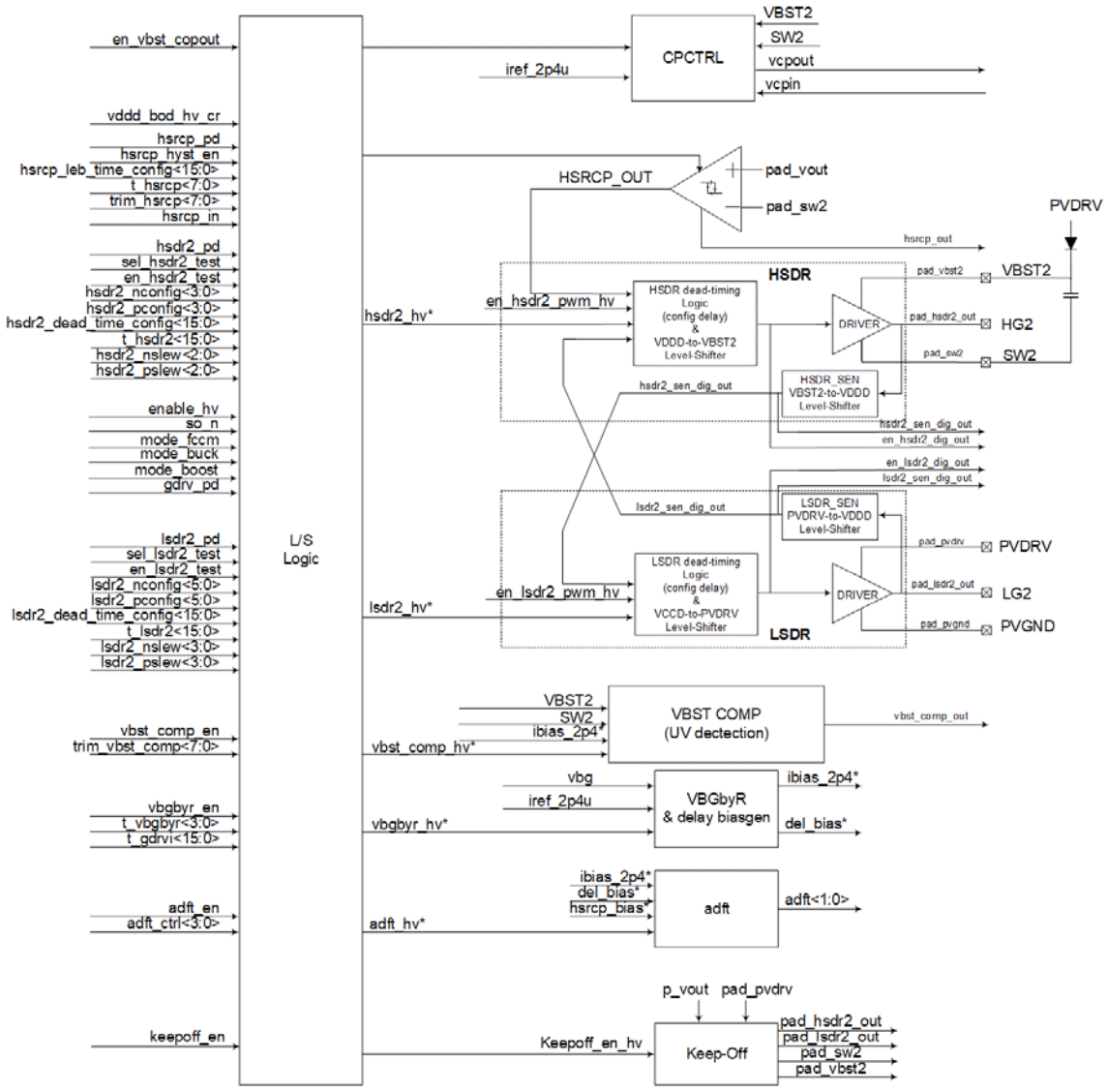


Figure 18-18 shows the top-level block diagram. The main function of this block is to provide gate control signal for external NFET switches (HS2 and LS2).

The gate driver for output side consists of following sub-blocks:

- **VBGbyR:** This sub-block is used to generate multiple internal current references (ibias*) used by sub-blocks on this public cell.
- **Keep-off:** It keeps external NFETs (HS2 and LS2) off until PVDRV attains a proper power level to prevent back current from VOUT to VIN.
- **LSDR:** This sub-block is used to drive the gate of lower-side external N-channel MOSFET(LS2) and operates co-operatively with high-side driver (HSDR2) to form a synchronous converter.
- **HSRCP:** It is a high-side high-speed comparator to detect inductor reverse current from VOUT to PVGND or VIN through external MOSFET HS2 in DCM. It has programmable leading-edge blanking timer to avoid the false HSRCP output.

- **HSDR:** This sub-block is used to drive the gate of high-side external N-channel MOSFET(HS2) and operate co-operatively with a low-side driver (LSDR2) to form a synchronous converter.
- **VBST_COMP:** It is VBST under-voltage detection comparator.
- **VBST_CPCTRL:** The VBST charge pump controller transfers the charge from VBST2 to floating supply VBST1 in boost mode when under-voltage detects on VBST1 or receives charge from VBST1 in the buck mode.

18.3.13 Error Amplifier

18.3.13.1 Architecture and Function Description

Figure 18-19. Error Amplifier Block Diagram

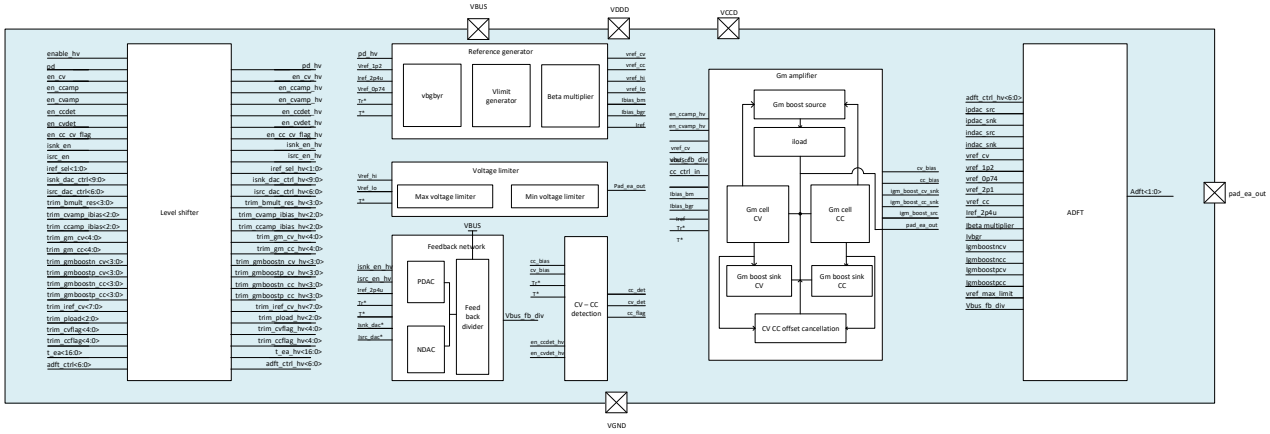
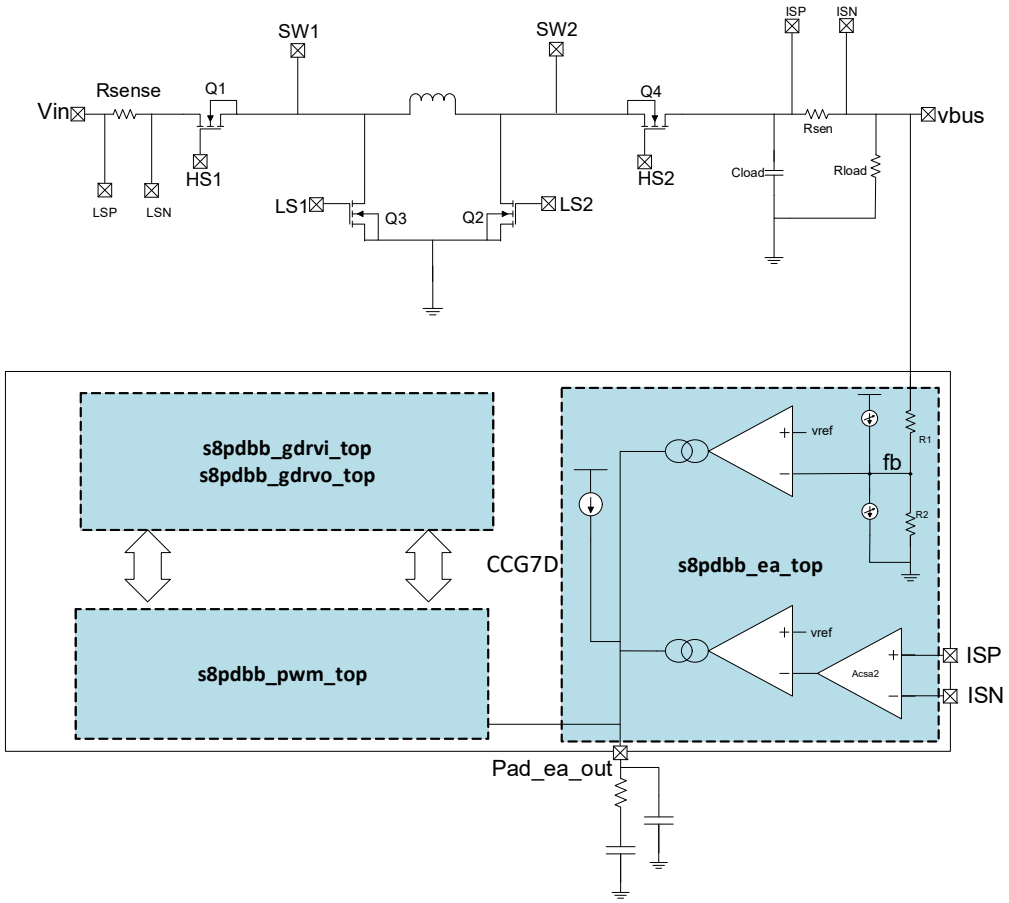


Figure 18-19 shows the top-level block diagram. This block makes sure that the buck boost controller will act as a constant voltage or constant current source by controlling the voltage on pad_ea_out.

Following are the applications of error amplifier:

- Voltage control in buck-boost controllers - Used to increase/decrease the VBUS voltage in a DC-DC converter. VBUS value can be adjusted to a value in the range 3.3 V to 21.5 V, in steps of 20 mV.
- Constant voltage control loop to regulate VBUS voltage.
- Constant current control loop to regulate VBUS current along with 20 CSA and Refgen blocks; the EA block is used to provide the constant current to the battery of the mobile phones.
- Detect signal (cc_flag) to indicate that loop is in CC mode.
- Supports Type-2 compensation to complete power system, to compensate for the pole of the power system.

Figure 18-20. Error Amplifier Typical Application Diagram



Voltage control. In this mode, voltage on VBUS is changed by changing current source/sink at feedback (fb) node of VBUS resistor divider. The power plant senses the voltage on the fb node and modifies the VBUS voltage accordingly. The voltage can be modified by sourcing or sinking the current from fb node. A 10-bit current sinking and 7-bit current sourcing DAC is designed to source/sink the current to fb node. When sinking the current “I” the fb voltage is:

$$VFB = \frac{(VBUS - I \times R1) \times R2}{(R1 + R2)}$$

Here, R1 is the resistance between VBUS and fb and R2 is resistance between fb and vgnd. The power plant senses the change in feedback voltage and tries to bring the VFB voltage to the fixed reference voltage by modifying the VBUS voltage. The modified VBUS voltage will be:

$$VBUS = VBUS + I \times R1$$

Similarly, when sourcing the current “I” the fb voltage is:

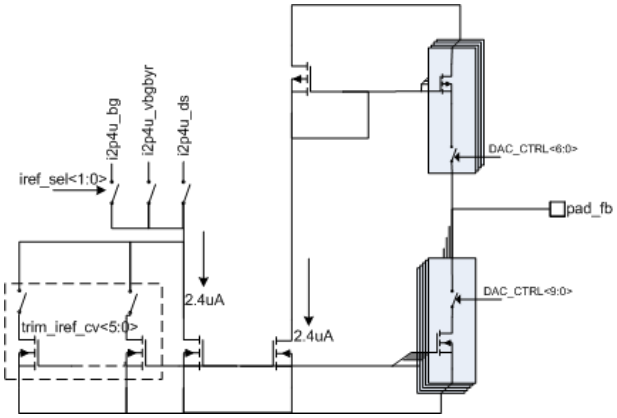
$$VFB = \frac{(VBUS + I \times R1) \times R2}{(R1 + R2)}$$

The power plant senses the change in feedback voltage and tries to bring the VFB voltage to the fixed reference voltage by modifying the VBUS voltage. The modified VBUS voltage will be

$$VBUS = VBUS - I \times R1$$

According to the latest USBPD ECR, the step size required on VBUS should be 20 mV and accuracy of the newly requested VBUS voltage is ±5%.

Figure 18-21. Current Steering DAC Block Diagram



The typical input current required for the DAC is 2.4 µA. The input current can be selected from three current source inputs.

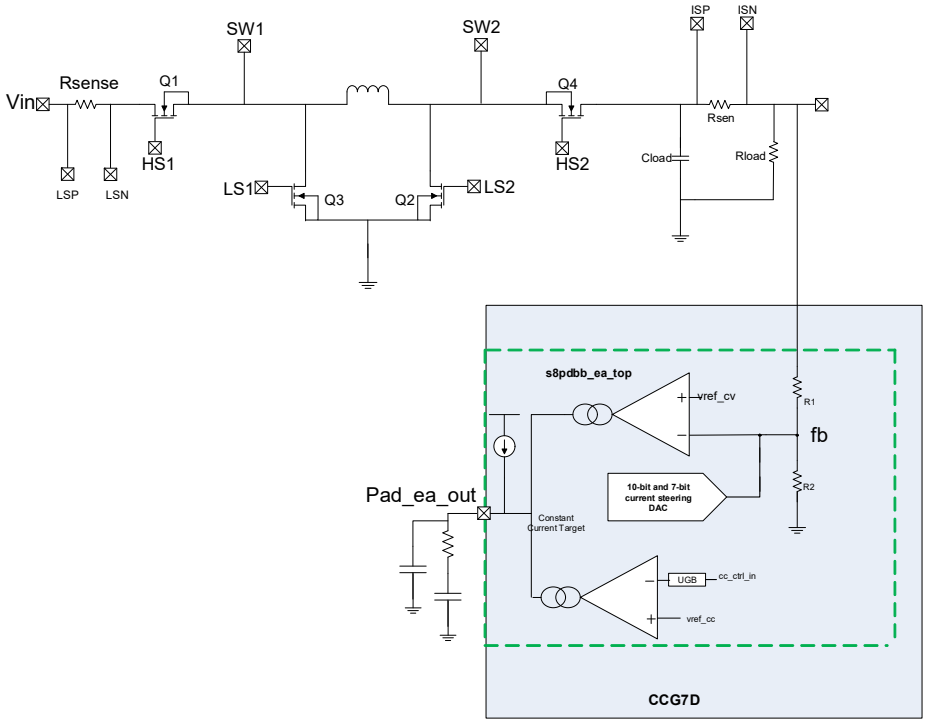
- Bandgap current (±5%)
- Deep-sleep reference current (from s8usbpd_dpslp_ref_top)
- Internal I=Vbg/R reference

For precision, it relies on a highly accurate input band gap reference voltage at vbg, a matched resistor divider, and a comparator with very low input offset.

Constant voltage control

In this mode, the block senses the voltage on VBUS through the resistor divider and modulates the current going to external compensation network on pad_ea_out.

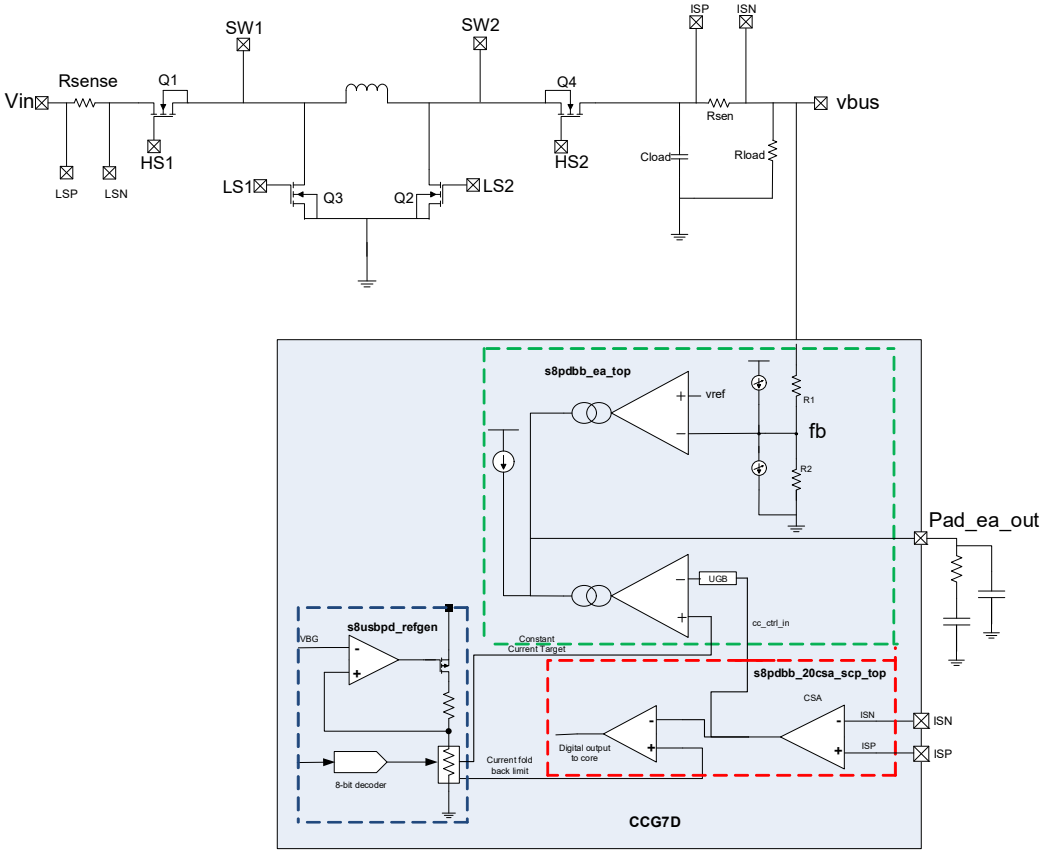
Figure 18-22. CV Loop Application Diagram



The pad_ea_out signal output is given to the PWM block, which modulates the duty cycle of the buck-boost controller switches. This change in pulse width will regulate the VBUS value. The default voltage on VBUS is dictated by the reference (0.74 V) from VBG buffer and resistor divider (R1 and R2). The reference (0.74 V) and resistor divider (R1 = 200 kΩ and R2 = 34 kΩ) is chosen such that the default voltage on VBUS is 5 V. To modify the voltage on VBUS the 10-bit/7-bit current steering DAC is used. The input current to the DAC is generated from V/R gen block. This approach improves the VBUS accuracy compared to the bandgap current input.

Constant current control. In this mode the current through the power plant is controlled using 20CSA high-side current sense amplifier refgen block and error amplifier block.

Figure 18-23. Constant Current Control Loop Application Diagram



The CSA block senses the voltage across the external sense resistor and produce amplified output (cc_ctrl_in). The required current target is set by the gain of 20CSA block and refgen block (vref_cc). CC loop takes 'cc_ctrl_in' and 'vref_cc' as inputs to the error amplifier and modulates the current going to the external capacitor connected to pad_ea_out pad, which will modulate the duty cycle of buck boost controller.

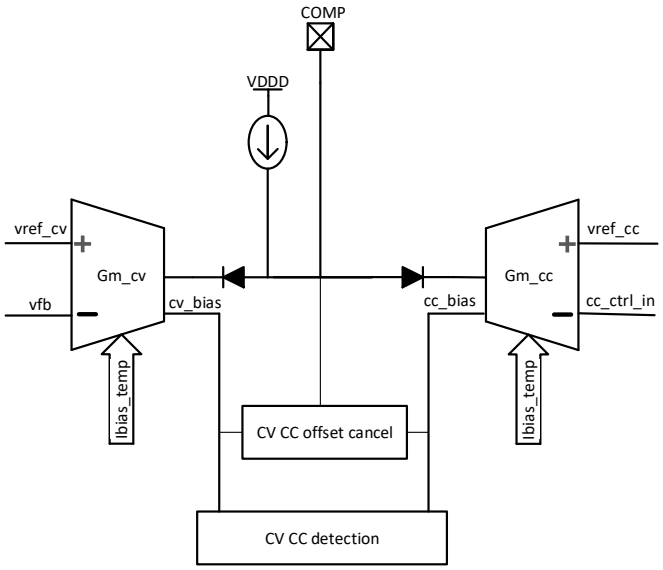
cc_flag, cv_det and cc_det . This circuit block is used to indicate whether the loop is in constant current mode. Implementation is such that the current in CV loop and CC loop are measured using trimmable current mirrors. For example, if cv_det trim is set to 1.2 µA, then cv_det will go high if current in CV loop is more than 3*1.2 µA. Similarly, if cc_det trim is set to 2.4 µA, then cc_det will go high if current in CC loop is more than 3*2.4 µA.

There are separate trims to measure current in the CV loop and CC loop. Whenever current in CV loop is more than trim current, then cv_det goes high. Similarly, when current in CC loop is more than trim current, then cc_det goes high. cc_flag is generated with cv_detb && cc_det logic. For proper operation, trim settings for current mirrors for cv_det and cc_det must be such that the current from pad_ea_out is greater than the current for which cv_det and cc_det is set.

Following are the important blocks in this cell:

- **Level shifter:** This block is used to convert the control/trim/test signals coming from LV domain to HV.
- **ADFT:** This block is used to test the internal block signals
- **Reference generator:** This block is used to generate the reference voltages and currents based on the references from vbg and iref from SRSS. This block also contains a temperature compensation circuitry to minimize the Gm variation of the Gm amplifier.
- **Feedback network:** This derives the feedback voltage from VBUS supply, which is used to control the loop. By modifying this feedback voltage using a PDAC and NDAC, we can control the VBUS voltage; see Voltage control for more information.
- **Voltage limiter:** This block limits the maximum and minimum excursion of the voltage on pad_ea_out pad. This will help in transient response as there is no need to charge the pad_ea_out from supply rails to operating range.
- **Gm boosting:** This block boosts the Gm of amplifier whenever the input difference exceeds a limit. This will help improve the load transient response of the amplifier and the overall system.
- **Gm amplifier block:** This block can be divided into two sections – CV and CC. The CV section consists of a Gm amplifier whose output current is equal to gm_cv * (vref_cv – vfb). This block controls the loop when the system is in CV mode. Similarly, the CC section has a CC Gm cell. The CC amplifier gives an output current equal to gm_cc * (vref_cc – cc_ctrl_in). To remove the interference of CV and CC loops at the boundary, a CV/CC offset cancellation block is present, which delivers the current equivalent to minimum of CV/CC amplifier output current. Both outputs are connected to a load current as shown in Figure 18-24.

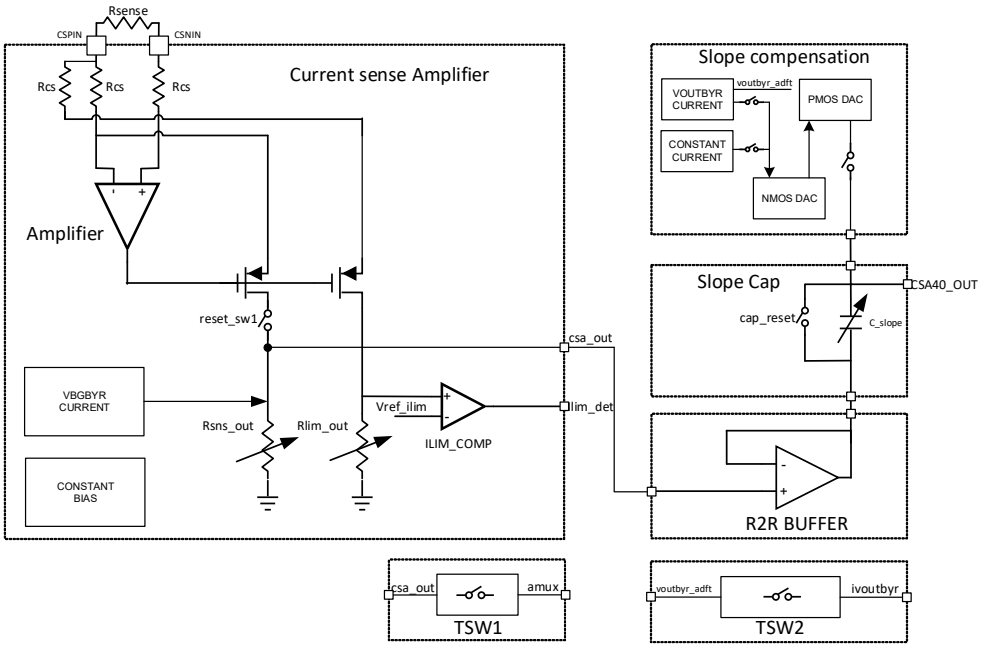
Figure 18-24. Gm Amplifier Block for CV and CC



18.3.14 High-Side Current Sense Amplifier for VIN

18.3.14.1 Architecture and Function Description

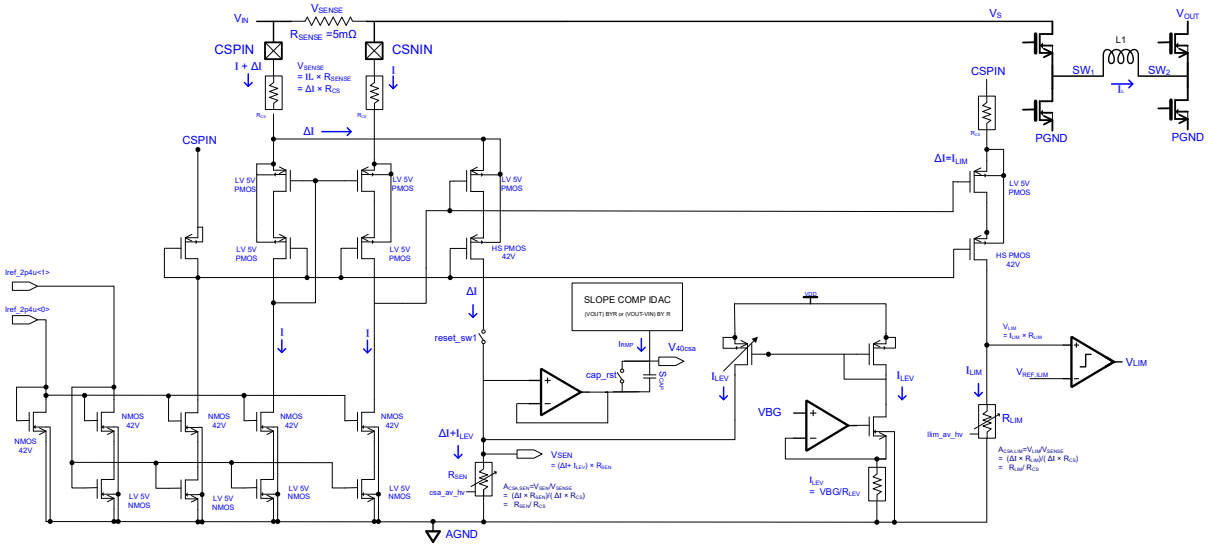
Figure 18-25. High-side Current Sense Amplifier for VIN



The functions of the various sub-blocks are as follows:

- Level shifter: This block takes control signals from VCCD domain and converts them into the VDDD domain.
- CSA - CSA 1st STAGE: This block is the CSA amplifier first stage, which senses the difference between the CSPIN and CSNIN to determine the current through the inductor.
- CONSTANT BIAS - Reference generation block: This block generates bias currents based on constant reference current of 2.4 µA, which will be used by other blocks inside 40CSA.
- VBGBYR - Reference generation block: This block generates bias currents dependent on resistor variation, which will be used by other blocks inside 40CSA.
- ILIM_COMP - Current limit comparator block: This block compares the current limit signal of the CSA first stage block with the external current limit voltage reference.
- SCOMP - Slope Compensation block: This block generates the VOUTBYR and CONSANT current dependent slope currents for the slope compensation.
- R2R BUFFER - Voltage buffer block: This block drives the 1st stage CSA output voltage on the bottom plate of the slope cap.
- SLOPE_CAP - Capacitor block: This block is used to add both the CSA first stage output and the slope compensation output to get the final 40CSA output.
- ADFT - ADFT block: To bring internal signals out on amuxbusa/b.
- TSW1 - Transmission gate switch: To route CSA first stage output to amux in the chip.
- TSW2 - Transmission gate switch: To bypass voutbyr_adft to ivoutbyr output pin.

Figure 18-26. Current Sense Amplifier Circuit for PWM Control and Current Limit



As shown in Figure 18-26, the CSA measures the voltages, V_{SENSE}, converts to current, ΔI, flowing through R_{SEN} and R_{LIM} to generate V_{SEN} and V_{LIM} for current mode PWM control and cycle by cycle current limit.

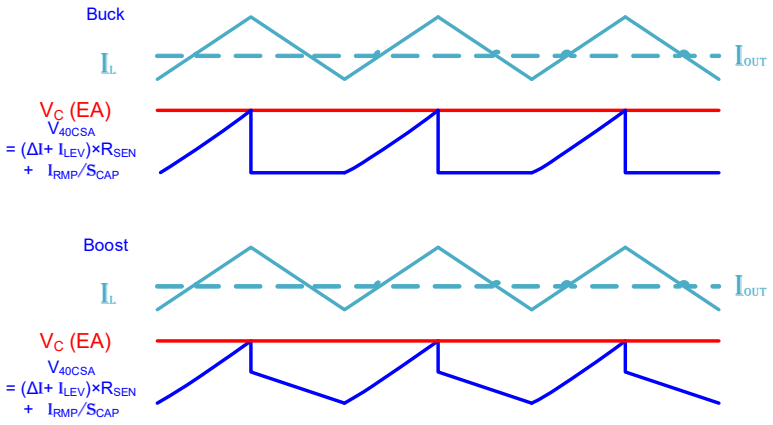
$V_{SENSE} = I_L \times R_{SENSE} = \Delta I \times R_{CS}$

The gain for current mode current sense, $A_{CSA,SEN} = V_{SEN}/V_{SENSE} = (\Delta I \times R_{SEN})/(\Delta I \times R_{CS}) = R_{SEN}/ R_{CS}$

The gain for cycle by cycle current limit, $A_{CSA,LIM} = V_{LIM}/V_{SENSE} = (\Delta I \times R_{LIM})/(\Delta I \times R_{CS}) = R_{LIM}/ R_{CS}$

V_{SEN} is summated by compensation slope ramp, I_{RMP} on the top plate of the slope capacitor and common offset level, V_{LEV} on the bottom plate. Therefore, the final output is obtained as $V_{40CSA} = (?I+I_{LEV})\times R_{SEN} + I_{RMP}/S_{CAP}$. R_{SEN} is selectable to configure different CSA current sense gain. I_{LEV} is programmable to set the desired output level. I_{RMP} is programmable to obtain various voltage ramps.

Figure 18-27. Current Sense Signal for Buck and Boost Mode



After duty cycle is set to ON, the rising V_{40CSA} signal is compared with the error amplifier output VC to trigger reset signal of duty cycle for peak current mode PWM control (see Figure 18-27).

V_{LEV} is the bottom level of current sense and compensation ramp. $V_{LIM} = \Delta I \times R_{LIM}$. R_{LIM} is selectable to configure different CSA current limit gain. VREF, ILIM is a programmable voltage coming from REFGEN block.

Figure 18-28. Cycle by Cycle Current Limit

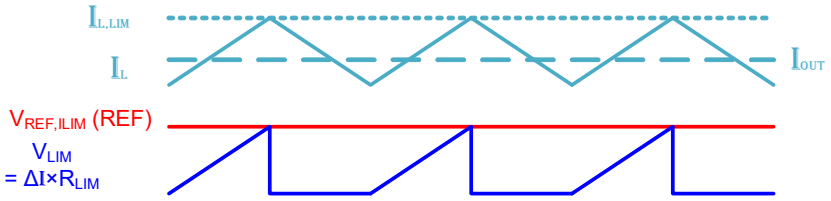
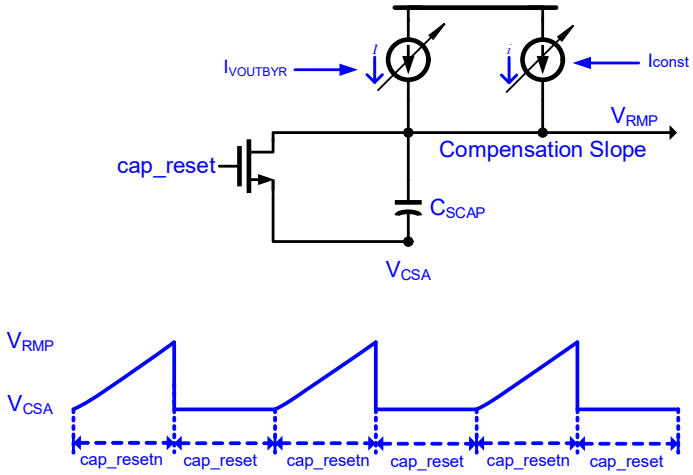


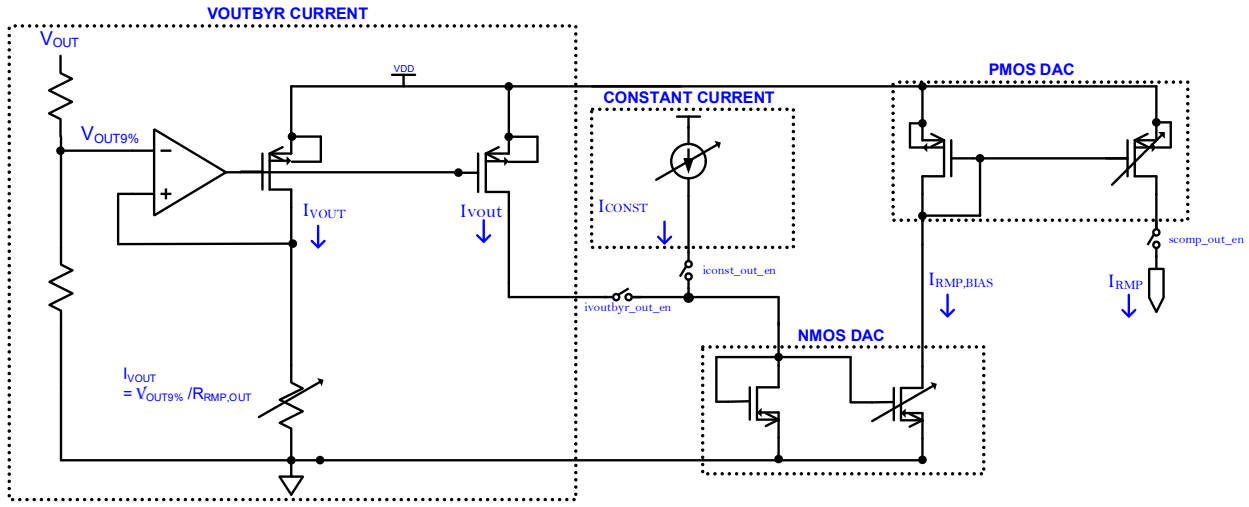
Figure 18-28 show the rising V_{LIM} is cycle by cycle comparing with current limit threshold to shut off duty cycle and limit the peak current.

Figure 18-29. Slope Compensation Architecture and Waveform



V_{RMP} (= $m_{RMP} \times t$) in Figure 18-29 is the compensation slope voltage; m_{RMP} is its slope. The compensation slope provides stability for current mode PWM control. The programmable slope, m_{RMP} is summated by certain ratio of down slope, $m_2 = I_{VOUT}/C_{SCAP}$ (I_{VOUT} is proportional to V_{OUT}), or certain ratio of a constant slope $m_I = I_{CONST}/C_{SCAP}$ to keep enough compensation slope for stability and minimum compensation slope for current mode characteristic. Figure 18-30 shows the slope compensation circuit.

Figure 18-30. Slope Compensation Circuit



18.3.15 Pulse-Width Modulator

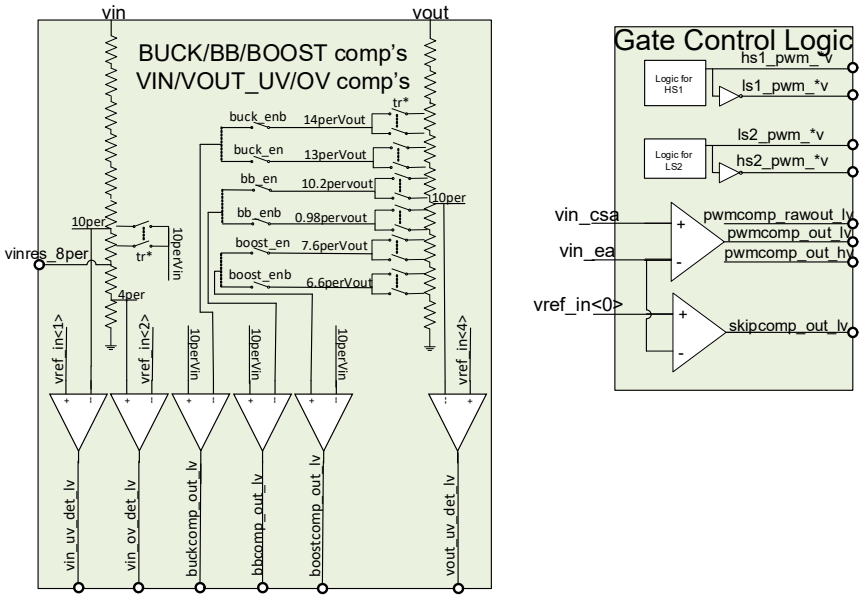
18.3.15.1 Architecture and Function Description

This block has the following functions:

- Takes inputs from RTL/CSA/EA and generates control signals for all four external NFETs of four-switch buck-boost controller. These control signals are given as input to GDRVI/O to drive external NFETs.
- Senses VIN/VOUT voltage values and gives indication on buckcomp_out_lv, bbcomp_out_lv, and boostcomp_out_lv outputs.
- Provides VIN under/over voltage detections.

Figure 18-31 shows the top-level block diagram of this block.

Figure 18-31. Pulse Width Modulator Block Diagram



Buck-boost mode selection: This block has resistor dividers on VIN and VOUT. On VIN, a 10% tap point is taken for comparison and on VOUT, different tap points can be selected based on requirement of mode transition threshold. Comparator outputs are given to RTL, which will decide the mode of operation of the converter.

18.3.16 High-Side Current Sense Amplifier for VBUS

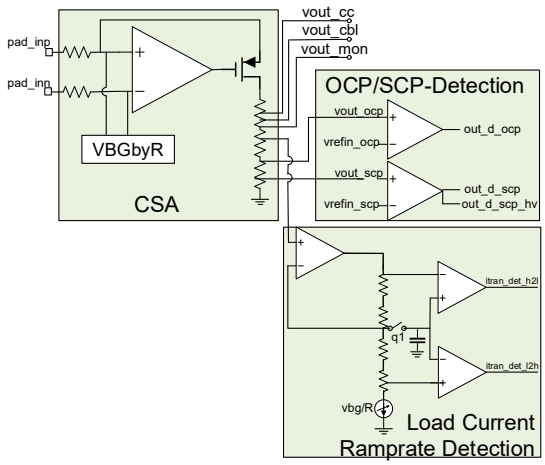
18.3.16.1 Architecture and Function Description

This block has the following functions:

- CC_LOOP regulation: Senses current (from 1 A to 5 A) in VBUS and gives voltage input (with AV = 40, 50, 60, 70, 80, 90, 100, 110) to error amplifier for CC loop regulation.
- OCP detection: Detection of over current (1 A, 2 A, 3 A, 4 A, 5 A) in VBUS.
- SCP detection: Detection of short-circuit current (6 A, 10 A) in VBUS.
- VBUS load current ramp rate detection: Detects VBUS load current ramp rate whenever it crosses programmed value.
- VOUT_CBL: This output is for cable compensation with gain options AV = 40/70/150/200.
- VOUT_MON: this output is for VBUS current live monitoring with gain options Av=40, 70, 150, 200

Figure 18-32 shows the top-level block diagram.

Figure 18-32. High-Side Current Sense Amplifier Block Diagram



18.3.17 OCP on VBUS

OCP on VBUS is part of the functionality of high-side current sense amplifier for VBUS. See High-Side Current Sense Amplifier for VBUS on page 153 for details.

18.3.18 VIN Regulator

18.3.18.1 Architecture and Function Description

Figure 18-33. VIN Regulator Block Diagram

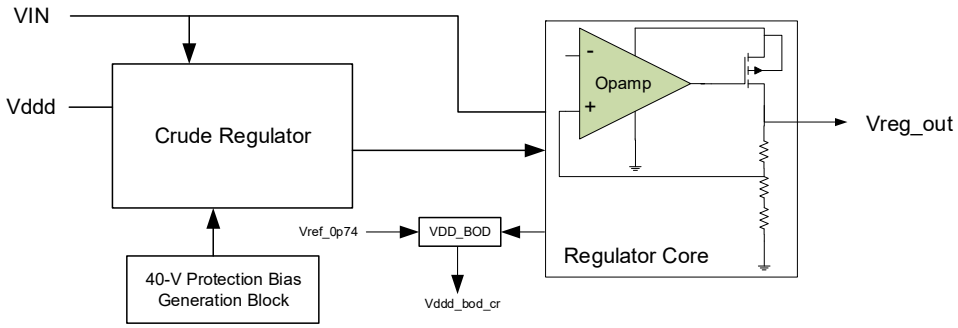


Figure 18-33 shows the block diagram of the 40V regulator, which works from VIN voltage in the range 4 V to 40 V. The regulator system has the following functionalities.

- Crude regulator - This block gives a crude output voltage of minimum 2 V whenever VIN is up and ≥ 3V. Hence, it is guaranteed that the downstream blocks will receive power and there is no start-up issue.
- VIN detect circuit - This block gives out vin_good_vcrude signal indicating whether VIN is present irrespective of VDDD. This detect also gives a signal during powerup/ brownout, even when VDDD is present.
- 40-V protection bias generation - This block is used to generate high-voltage protection bias voltages for the regulator core.
- Regulator core - Core is low dropout based architecture with a 40-V tolerant PMOS pass element. Typical regulator output is 5 V. The load compensation scheme is used to attain stability using an external capacitor.

  Regulator core features the in-rush current detection and control functions. Typically, In-rush event is detected when load current exceeds the inrush limit. Upon detection of in-rush event, gate of the PMOS pass element is clamped to limit the maximum supplied current.
- VDDD_BOD - This block generates an indication for VDDD brownout during power down or brownout events.

18.3.19 Low Frequency Counter (LF_CNTR)

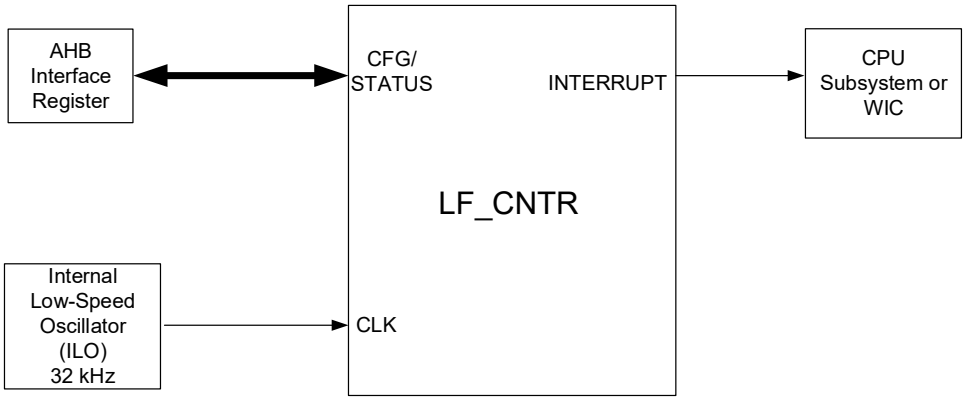
The LF_CNTR is a free-running counter on the ILO that triggers an interrupt at a firmware programmed terminal count. Every time the counter reaches a value set in the LF_CNTR_MATCH.MATCH register field, an interrupt is generated. Firmware may also modify the value of LF_CNTR_MATCH.MATCH field dynamically to control the time until the next interrupt is generated. Firmware may read the current timer (LF_CNTR.COUNTER) value, increment it with the desired interval, and write this value into the match register field(LF_CNTR_MATCH.MATCH).

18.3.19.1 Features

The LF_CNTR has these features:

- Configurable timer period
- Can generate an interrupt in Sleep and Deep-Sleep power modes to wake up the device
- Can generate an interrupt in Active mode after a specified interval

Figure 18-34. LF_CNTR Block Diagram



18.3.19.2 How It Works

The LF_CNTR asserts interrupts at the end of each, interval, that is programmable of up to a maximum of 2048 ms, unless it is periodically serviced in firmware. The LF_CNTR is a 16-bit free-running wraparound up-counter.

The LF_CNTR register provides the count value of the LF_CNTR. The LF_CNTR generates an interrupt when the count value in LF_CNTR equals the match value stored in the LF_CNTR_MATCH register, but it does not reset the count to '0'. Instead, the LF_CNTR keeps counting until it overflows and rolls back to 0. When the count value again reaches the match value, another interrupt is generated.

The LF_CNTR_MATCH bit in the MXUSBPD_INTR1 register is set whenever the LF_CNTR interrupt occurs. For details, see the USBPD Register Map section in the *CCG7D Registers TRM*.

Follow these steps to use LF_CNTR as a periodic interrupt generator:

1. Write the desired match value to the LF_CNTR_MATCH register.
2. Clear the LF_CNTR_MATCH bit in MXUSBPD_INTR1 to clear any pending LF_CNTR interrupt.
3. Enable the LF_CNTR interrupt by setting the LF_CNTR_MATCH bit in MXUSBPD_INTR1 _MASK.
4. In the ISR, clear the LF_CNTR interrupt and add the desired match value to the existing match value. By doing so, another periodic interrupt will be generated when the counter reaches the new match value.
5. The IGNORE_BITS field in the LF_CNTR_MATCH register can be used to reduce the entire LF_CNTR period. The ignore bits can specify the number of MSbs that need to be discarded. For example, if the IGNORE_BITS value is 3, then LF_CNTR counter becomes a 13-bit counter. The LF_CNTR is a free-running counter that cannot be disabled unless the USB-PD block itself is disabled.

18.3.19.3 LF_CNTR Interrupts and Low-Power Modes

The LF_CNTR counter sends the interrupt requests to the CPU in Active power mode and to the Wake-up Interrupt Controller (WIC) in Sleep and Deep-Sleep power modes. It works as follows:

- Active Mode: In Active power mode, the LF_CNTR sends the interrupt to the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.
- Sleep or Deep Sleep Mode: In this mode, the CPU subsystem is powered down. Therefore, the interrupt request from the LF_CNTR is directly sent to the WIC, which will then wake up the CPU. The CPU acknowledges the interrupt request and executes the ISR. The interrupt must be cleared after entering the ISR in firmware.

18.4 PD Block Register Usage

This section details the usage of the PD block in initialization, packet sending and receiving, VBUS, and CC line state measuring procedures. It describes how the CCG7D firmware uses key PD block registers and handles PD interrupts.

The USB-PD Registers section of *CCG7D Registers TRM* contains the complete list of PD block registers. Refer to this TRM while going through the sections that follow.

The steps to execute the various functions that can be handled by the PD block are as follows:

- Initialization, enabling, and disabling the PD block
- Deep-Sleep entry and exit
- Changing termination on CC lines
- Measuring state of CC line
- Sending a PD packet
- Receiving a PD packet
- Initialization and usage of ADC
- Initialization and usage of UP/DOWN and ADC comparator output filters

18.4.1 PD Block Registers and Interrupt Handling Flow

18.4.1.1 PD Block Clocking

The PD block needs to be configured for the following operational frequencies.

Table 18-4. Selecting Peripheral Clocks

Clock	Clock Select Register	Recommended Frequency
USBPD (RX)	PERI_PCLK_CTL1	12 MHz
USBPD (TX)	PERI_PCLK_CTL2	600 kHz
USBPD (SAR)	PERI_PCLK_CTL3	100 kHz
USBPD (Filter 1)	PERI_PCLK_CTL4	500 kHz

18.4.1.2 PD Block Key Registers

PD block is configured and controlled through the following key registers.

Table 18-5. USB PD Block Key Registers

Register Name	Description
CTRL	Generic control register. Enables PD block.
HEADER_INFO	Initially configures positions of bits in an ext. message header. Switches transceiver between Regular or Extended data messaging modes.
TX_HEADER	Forms message header of transmitted packet.
TX_MEM_DATA[0..15]	64-byte long TX data buffer.Only word-aligned access is possible. Data is stored in the little endian format.
RX_HEADER	Contains message header of received packet.
RX_MEM_DATA[0..15]	64-byte long RX data buffer. Only word-aligned access is possible. Data is stored in the little endian format.
SRAM_PTR	FW uses these RX/TX pointers to read/write more data after the TX/RX_SRAM_HALF_END interrupts
STATUS	Generic status register. FW checks only TX/RX_BUSY, CC_DATA_VALID, and SENDING_GOOD_CRC_MSG status bits.
RX_SOP_GOOD_CRC_EN_CTRL	Sends GoodCrcMsg automatically. FW turns off this possibility during initialization.

Table 18-5. USB PD Block Key Registers

Register Name	Description
RX_EXPECT_GOODCRC_MSG	FW configures the expected packet SOP type of expected GoodCRC message before committing a new packet to be transferred from TX_MEM.
RX_GOODCRC_MSG	The two-byte header of the received GoodCRC message is stored here.
RX_CC_0_CFG	RX data sample point and RX state machine timeout adjustment. FW configures them once at initialization time.
RX_CC_1_CFG	Receiver tuning register. Default value is left here.
RX_ORDER_SET_CTRL	FW enables detection of different types of RX packets at runtime by choosing the appropriate SOP/RST type mask.
TX_GOODCRC_MSG_ORDER_SET	Configures transmit GoodCRC message header to SOP' and SOP'' during initialization.
TX_CTRL	Configures transmit GoodCRC message header to SOP During BIST test: Enables TX regulator and starts BIST CM2 Enables retry of TX packet Begins transmission
TX_SOP_ORDER_SET	Configures transmit message header (except GoodCRC message) during transmission.
TX_HARD_CABLE_ORDER_SET	Configures transmit hard reset message header during transmission.
CRC_COUNTER	Configures expiration period of CRC timer once during initialization.
DEBUG_CTRL	FW uses only RESET_RX/TX bits to stop any ongoing transmission.
DEBUG_CC_2	During initialization, FW configures bit-wise mask (for comparison) for expected RX GOOD-CRC_MSG.
LF_CNTR	Provides actual counter value for LF counter.
LF_CNTR_MATCH	FW loads here a hardware period for LF counter interrupt.
ADC_SAR_CTRL	FW starts ADC measurement here and gets the measurement result.
CC_UFP_NORD_0_CTRL	Enables receiver part of PD transceiver block. Enables UP/DN compactors to detect CC-line voltage changes. Configures UP/DN comparator thresholds. FW configures these thresholds differently in UNATTACHED and ATTACHED Type-C states.
DPSLP_REF_CTRL	FW initially enables PHY part of PD block by writing to this register.
PUMP_CTRL	FW initially enables PD block voltage pump by writing to this register.
ADC_CTRL	FW configures ADC input multiplexer, enables ADC, and sets DAC output voltage level for SAR ADC. ADC comparator output state is checked by reading the ADC_CTRL_CMP_OUT bit state. ADC is used only for VBUS voltage measurement.

18.5 USB PD Interrupts

The USB PD block has two types of interrupts:

Fixed-Function Interrupt – This block has one fixed-function interrupt: interrupt_usbpd pin. The interrupts specified in the USBPDx_INTR0 and USBPDx_INTR2 registers are combined (logically OR'd) to form a single USB PD fixed-function interrupt. Table 18-6 lists the various fixed-function interrupts supported by each PD block. The USBPDx_INTR0 and USBPDx_INTR2 registers should be read to check which of these interrupts are active. IRQ14 and IRQ15 (synchronous USBPD Port0 and Port1 interrupt) refers to these fixed-function interrupts.

Table 18-6. USB PD Fixed-Function Interrupts

Interrupt Status Bit	Description
INTR0.RCV_GOOD_PACKET_COMPLETE	PD PHY finished receiving a good (well formed) packet and stored it into the RX_FIFO.
INTR0.RCV_BAD_PACKET_COMPLETE	PD PHY finished receiving a malformed packet.
INTR0.RX_SOP	PD PHY received a SOP
INTR0.RCV_GOODCRC_MSG_COMPLETE	PD PHY received a GoodCRC message.
INTR0.RCV_EXPT_GOODCRC_MSG_COMPLETE	PD PHY received the expected GoodCRC message in response to previously transmitted message.
INTR0.RX_OVER_RUN	RX_FIFO overflowed
INTR0.TX_PACKET_DONE	PD PHY finished sending message from TX_FIFO
INTR0.TX_HARD_RESET_DONE	PD PHY finished sending Hard Reset
INTR0.RCV_RST	Hard Reset received
INTR0.TX_GOODCRC_MSG_DONE	Finished sending GoodCRC response to received message
INTR0.CRC_RX_TIMER_EXP	GoodCRC receiver timer expired after sending message.
INTR0.CC_VALID_DATA_DETECTED	Valid data detected on the CC line
INTR0.CC_NO_VALID_DATA_DETECTED	Valid data got de-asserted on the CC line
INTR0.TX_SRAM_HALF_END	Hardware finished reading data from one half of TX_FIFO
INTR0.RX_SRAM_HALF_END	Hardware filled data into one half of RX_FIFO
INTR0.SAR_DONE	SAR ADC conversion complete
INTR2.HPD_IRQ	HPD receiver detected IRQ
INTR2.HPD_PLUGED	HPD receiver detected plug event
INTR2.HPD_UNPLUGED	HPD receiver detected unplug event
INTR2.HPD_QUEUE	An entry to HPD queue is made
INTR2.HPDT_COMMAND_DONE	HPD transmitter finished sending requested command.
INTR2.EXTENDED_MSG_DET	Extended message detected by PD receiver
INTR2.RX_SRAM_OVER_FLOW	Hardware has passed writing the data to Half or End of the RX SRAM Memory Location but CPU has not read the Data
INTR2.TX_SRAM_UNDER_FLOW	Hardware has passed reading the data to Half or End of the TX SRAM Memory Location but CPU has not wrote the Data
INTR2.SWAP_RCVD	Fast Role Swap receiver detected request on CC line
INTR2.SWAP_COMMAND_DONE	Fast Role Swap transmitter finished sending swap request pulse.

Wakeup Interrupts – The USB-PD block has multiple sources of wakeup interrupts listed under the USBPDx_INTR1, USBPDx_INTR3, USBPDx_INTR5, USBPDx_INTR7, and USBPDx_INTR13 registers. Table 18-7 provides the description of these interrupts. IRQ6 and IRQ7 as ganged (wakeup) USBPD Port0 and Port1 interrupts refer to these wakeup interrupts.

Table 18-7. USB PD Wakeup Interrupts

Pin Interrupt	Description
INTR1.CC1_CHANGED	CC1 line voltage change detected
INTR1.CC2_CHANGED	CC2 line voltage change detected
INTR1.VCMP_LA_CHANGED	CC line comparator detected a change; used to detect CC voltage drop during fast role swap
INTR1.VCMP_UP_CHANGED	Up comparator on CC line detected a change
INTR1.VCMP_DN_CHANGED	Down comparator on CC line detected a change
INTR1.V5V_CHANGED	V5V state changed
INTR1.CCx_OCP_CHANGED	VCONN Over-Current condition detected on CC line with VCONN switch turned ON
INTR1.CCx_OVP_CHANGED	Over-Voltage (possible VBUS short) detected on CC line
INTR1.VSWAP_VBUS_LESS_5_DONE	Fast Role Swap signal received and VBUS voltage ≤ 5 V
INTR1.DRP_ATTACHED_DETECTED	Attach detect when RPRD toggling
INTR1.LF_CNTR_MATCH	LF Interrupt request
INTR3.CMP_OUT_CHANGED	ADC output crossed programmed threshold
INTR3.SBU1_SBU2_OVP_CHANGED	Over-voltage detected on SBU1 or SBU2 pins
INTR5.EDGE_CHANGED	Interrupt from OV and UV comparators
INTR7.CLK_LF_EDGE_CHANGED	Interrupt from VSYS detect comparators
INTR13.CSA_OCP_CHANGED	CSA (Over-current) comparator output changed
INTR15.CC_FLAG_CHANGED	CC flag output from error amplifier changed
INTR15.PDS_SCP_CHANGED	PDS SCP (short-circuit protection) output changed
INTR15.PWM_OUT_CHANGED	PWM output from buck-boost PWM block changed
INTR15.SKIP_OUT_CHANGED	Skip output from buck-boost PWM block changed
INTR17.PDBB_40VREG_VIN_DET	Voltage regulator input detection
INTR17.PDBB_40VREG_IN_RUSH_DET	Voltage regulator in-rush detection
INTR17.PDBB_20CSA_ITRAN_DET_L2H	Low-side CSA current transition detection low to high
INTR17.PDBB_20CSA_ITRAN_DET_H2L	Low-side CSA current transition detection high to low
INTR17.PDBB_40CSA_ILIM_DIG_OUT	High-side CSA limiting current detection
INTR17.PDBB_PWM_OUT_BLK	Buck-boost PWM block output
INTR17.PDBB_PWM_VIN_UV_DET	Buck-boost PWM input under voltage detection
INTR17.PDBB_PWM_VIN_OV_DET	Buck-boost PWM input over voltage detection
INTR17.PDBB_PWM_BUCK_COMP	Buck PWM comparator output
INTR17.PDBB_PWM_BOOST_COMP	Boost PWM comparator output
INTR17.PDBB_PWM_BB_COMP	Buck-boost PWM comparator output
INTR17.PDBB_GDRVO_VBST_COMP	High-side gate driver VBST comparator output
INTR17.PDBB_GDRVO_HSRCP	High-side gate driver reverse current protection output
INTR17.PDBB_GDRVI_VBST_COMP	Low-side gate driver VBST comparator output
INTR17.PDBB_GDRVI_LSZCD	Low-side gate driver zero cross detection output
INTR17.PDBB_EA_CC_DET	Error amplifier CC flag output
INTR17.PDBB_EA_CV_DET	Error amplifier CV flag output
INTR17.PDBB_CC1_SCP	Low-side amplifier CC1 short circuit protection
INTR17.PDBB_CC2_SCP	Low-side amplifier CC2 short circuit protection
INTR17.PDBB_VREG_VDDD_BOD	Voltage regulator VDDD brownout detection (BOD) output

Note: It is recommended that the SCP and RCP blocks be programmed to automatically turn-off the provider power path when the fault is detected. The corresponding interrupts can be used to trigger the recovery actions.

18.5.1 Basic Flow for PD Interrupt Handling

Figure 18-35. Interrupt 0 Handler Flow Chart

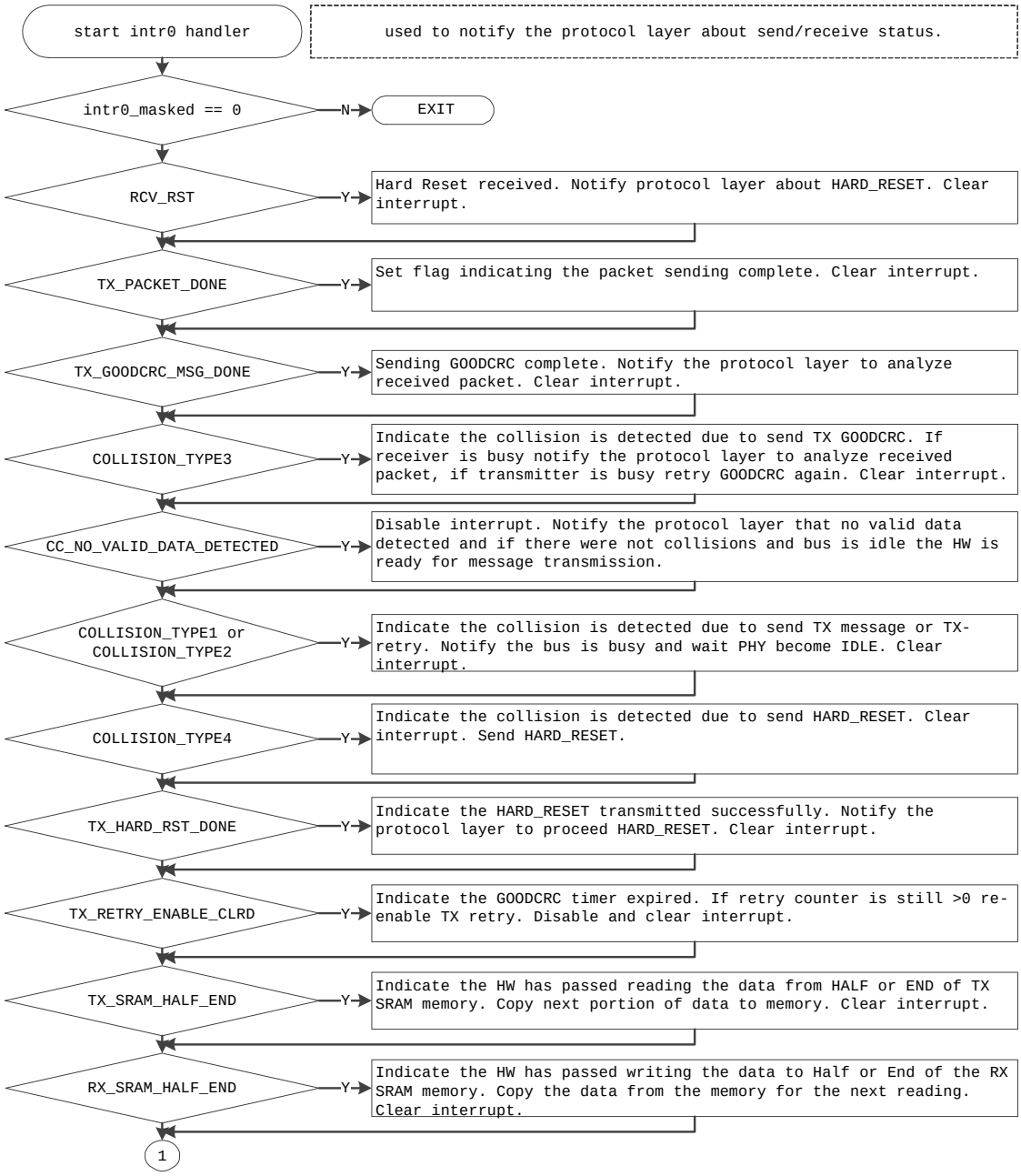


Figure 18-36. *Interrupt 0 Handler Flow Chart (continued)*

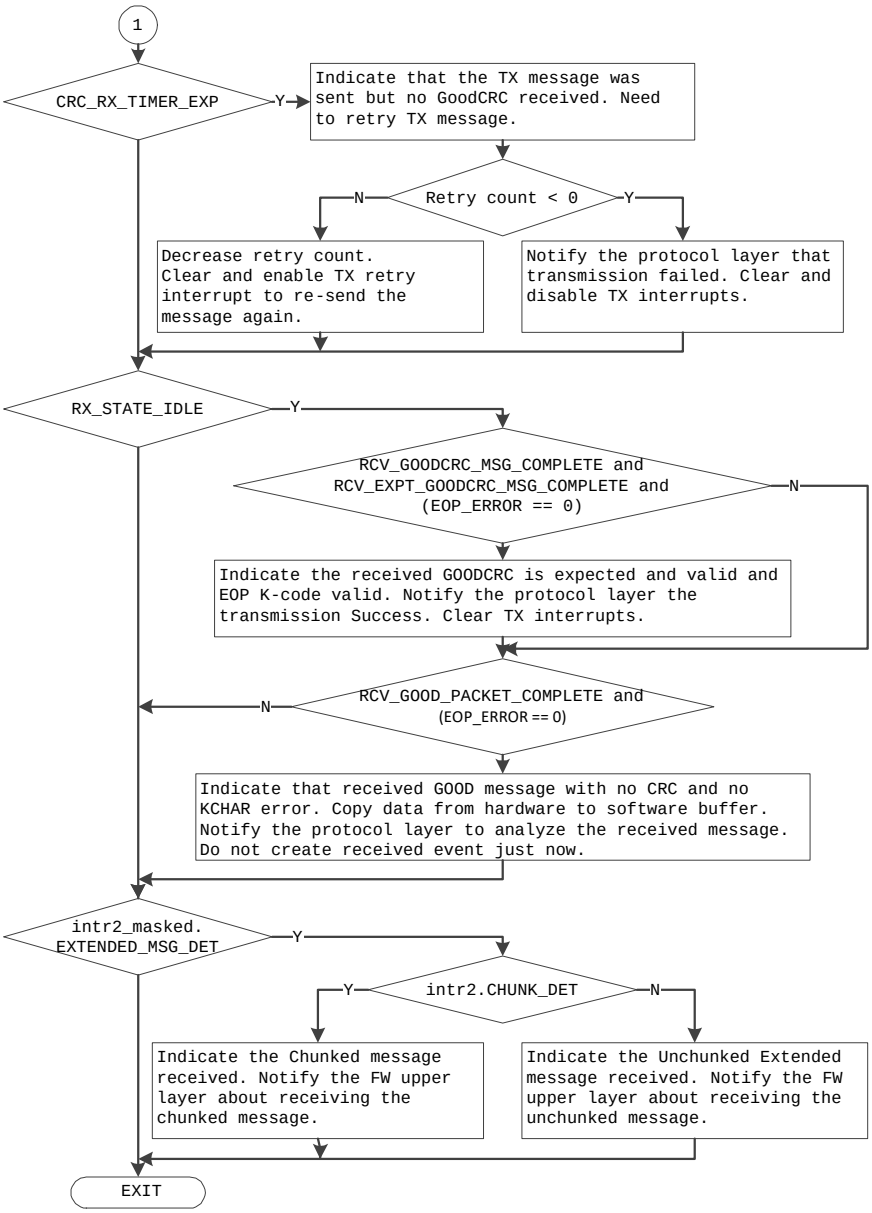
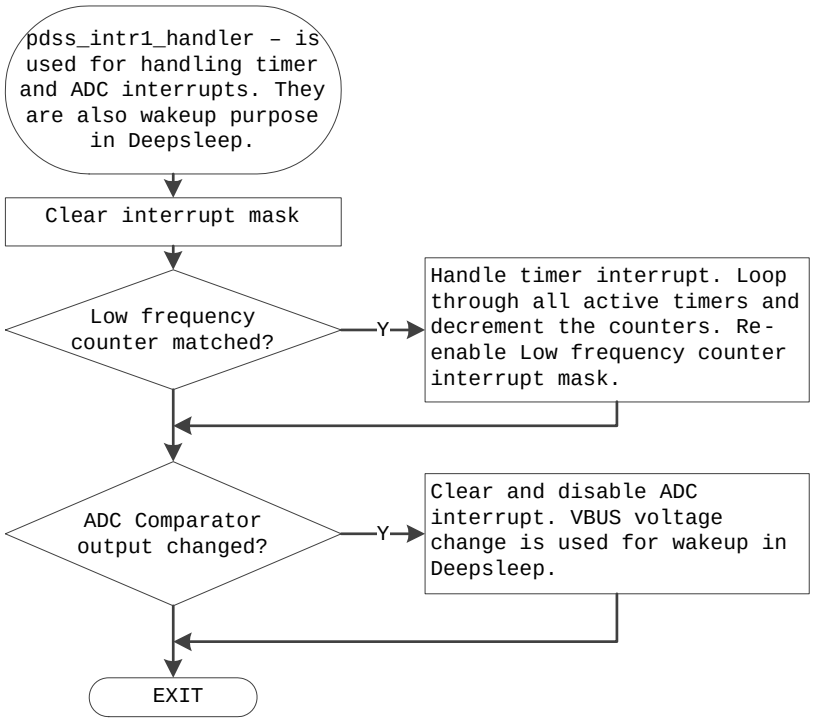


Figure 18-37. Interrupt 1 Handler Flow Chart



18.6 USB PD Block Power Modes

The USB PD block supports all three power modes implemented in the CCG7D controller. This block remains completely functional in Active and Sleep power modes. In Deep-Sleep power mode, the block's clock and the dedicated TX and RX RAM logic is powered OFF to save power. However, the block's internal logic is kept ON and it can generate interrupts to exit the device from the Deep Sleep mode. For more details on interrupts, see USB PD Interrupts on page 158. For more details on power modes, see the Power Modes chapter on page 55.

18.7 USB PD Functional Modes

The two main functional operating modes are Disable mode and Enable mode.

- Disable mode: In this mode, the block is disabled by the CPU by setting the USBPDx_CTRL.IP_ENABLED to '0'.
- Enable mode: In this mode, the block is enabled by the CPU by setting the USBPDx_CTRL.IP_ENABLED to '1'. In this mode, the block can transmit and receive USB PD packets.

18.8 Procedures Handled by PD Block - Type-C PD Initialization and Operation

18.8.1 Initialization, Enabling, and Disabling the PD Block

Enabling the PD block

1. Enable the PD block by setting PDSS_CTRL_IP_ENABLED bit in the CTRL register.
2. Turn on the CC Transceiver/Detect block by clearing PD_DPSLP bit in the DPSLP_REF_CTRL register.
3. Enable Deep-Sleep current reference outputs by setting IGEN_EN bit in the DPSLP_REF_CTRL register; wait 100 uS to settle outputs.
4. If high-frequency filtering is needed on the ADC comparator output, then enable HF clock as a source for the HS filter by setting SEL_CLK_FILTER bit in the CTRL register.

5. Initialize ADC (see Initialization and Usage of ADC on page 165).

Initializing the PD block Type-C registers

1. Initially disable filters on UP/DN comparator outputs by clearing PDSS_INTR1_CFG_VCMP_UP_DOWN_LS_VCMP_DN_FILT_EN and PDSS_INTR1_CFG_VCMP_UP_DOWN_LS_VCMP_UP_FILT_EN bits in the INTR1_CFG_VCMP_UP_DOWN_LS register.
2. Disable filters on ADC comparator output by clearing PDSS_INTR3_CFG_ADC_HS_FILT_EN bit and setting PDSS_INTR3_CFG_ADC_HS_FILT_BYPASS bit in the INTR3_CFG_ADC_HS register.
3. Enable the pump by clearing PDSS_PUMP_CTRL_PD_PUMP and PDSS_PUMP_CTRL_BYPASS_LV bits in the PUMP_C- TRL register.

Enabling PD transceiver and CC line comparators

1. Enable PD transceiver and CC line comparators by setting UFP_RX_EN and UFP_CMP_EN bits in the CC_UFP_NORD_0_CTRL register.
2. Configure Line Activity comparator to both edge detection for wake-up by setting VCMP_LA_CFG bits in the INTR1_CFG register.

Initialize PHY block of PD transceiver

1. Configure RX_SOP_GOOD_CRC_EN_CTRL.
2. Configure EXTENDED HEADER INFO register.
3. Configure RX_CC register.
4. Configure RX_ORDER_SET_CTRL.
5. Configure CRC_COUNTER reg.
6. Configure INTER_PACKET_COUNTER reg.
7. Disable all PD interrupts.
8. Configure DEBUG_CC_2 register to disable CC monitoring during idle gap before transmitting GoodCRC and set expected GoodCRC message header mask.
9. Configure SOP_PRIME and SOP_DPRIME Auto GoodCRC header.
10. Configure RX_SOP_GOOD_CRC_EN_CTRL.
11. Enable SOP and HARD_RESET packet receiving by setting the appropriate bits of the SOP_RST_EN bit-field in the RX_ORDER_SET_CTRL register.
12. Disable SOP_PRIME and SOP_DOUBLE_PRIME packet receiving by clearing the appropriate bits of the SOP_RST_EN bit-field in the RX_ORDER_SET_CTRL register.
13. Configure data role, power role, and the supported PD revision for Transmit GoodCRC Message Header (set GOODCRC_MSG_BITS bit-field in the TX_CTRL reg.).

Disabling of PD block

1. Disable PD_PORT0_INTR0 interrupt.
2. Disable PD_PORT0_INTR1 interrupt.
3. Disable the PD block by clearing PDSS_CTRL_IP_ENABLED bit in the CTRL register.

18.8.2 Deep-Sleep Entry and Exit Procedures

Deep-Sleep Entry

If the device is in Type-C ATTACHED state:

1. Clear pending Line Activity comparator interrupt and enable it by setting the PDSS_INTR1_VCMP_LA_CHANGED bit in the INTR1 and INTR1_MASK.
2. Configure UP/DN comparators to trigger on both rising and falling edges of CC line voltage and set appropriate comparator thresholds.
3. Measure CC-line state (see Measuring CC-line State on page 164).
4. If result differs from previous CC-line state, then set PDSS_INTR1_VCMP_UP_CHANGED bit in INRT1_SET to fire VCMP_UP_CHANGED interrupt immediately.

5. Return from Deep-Sleep entry procedure.
6. Calibrate ADC (see Initialization and Usage of ADC on page 165).
7. Configure the ADC for comparator functionality with the requested threshold (detach voltage level) and interrupt generation on negative voltage level transition (see Initialization and Usage of ADC on page 165).

   Note: Comparator output will trigger opposite to the voltage transition direction, so interrupt on RISING_EDGE should be configured.
8. Optionally enable a de-glitch filter on ADC comparator outputs (see Initialization and Usage of UP/DOWN and ADC Comparator Output Filters on page 166)

If the device is in Type-C UNATTACHED state:

1. Disable UP/DN comparators and UP/DN comparator output filters.
2. Configure UP comparator to trigger on rising of CC line voltage and set 0.2 V as comparator threshold.
3. Optionally, enable filter on UP comparator output (see Initialization and Usage of UP/DOWN and ADC Comparator Output Filters on page 166).
4. Clear pending Line Activity, UP and DOWN comparator INTR1 interrupts, and enable UP comparator interrupt by setting the PDSS_INTR1_UP_CHANGED bit in the INTR1_MASK.
5. Check the INTR1_STATUS register; if UP comparator has already triggered then:
   a. Immediately rise interrupt by writing of PDSS_INTR1_VCMP_UP_CHANGED bit to INRT1_SET.

Deep-Sleep Exit

1. Disable Deep-Sleep interrupts (from Line Activity, UP and DOWN comparators).

   Note: ADC comparator output change interrupt is cleared and disabled directly in interrupt handler.
2. Clears pending Deep-Sleep interrupts (from Line Activity, UP and DOWN comparators).
3. If device is in Type-C ATTACHED state:

   Disable the detach detection (disable ADC comparator, see Initialization and Usage of ADC on page 165).
4. If device is in Type-C UNATTACHED state:

   Disable filters on UP/DN comparator outputs (if they were enabled on Deep-Sleep entry).

18.8.3 Changing Termination on CC-lines

CCG7D never changes termination on CC line as it operates only in UFP/SINK role with externally placed CC-line pull-down resistor.

18.8.4 Measuring CC-line State

1. If Type-C state is ATTACHED and transceiver is busy (TX_BUSY OR CC_DATA_VALID are set in STATUS register) then return to procedure.
2. Set UP/DOWN thresholds to UFP_CMP_UP_SEL and UFP_CMP_DN_SEL bits of CC_UFP_NORD_0_CTRL.
3. Wait 50 µs while UP/DN comparator outputs settle.
4. Return CC line state as state of VCMP_UP_STATUS and VCMP_DN_STATUS bits in INTR1_STATUS.

18.8.5 Sending a PD Packet

1. Configure Transmit Message Header data role, power role, supported PD revision, message type, ID, and count fields in TX_HEADER.
2. If an extended un-chunked packet should be transmitted then put Extended Header in upper 16 bits of TX_HEADER.
3. Set EN_TX_EXTENDED_DATA bit in HEADER_INFO according to the type of packet (regular/extended).
4. Enable SOP packet transmitting by setting appropriate bits in TX_SOP_ORDER_SET_CTRL.
5. Configure the expected SOP type of expected GoodCRC response in RX_EXPECT_GOOD_CRC_MSG. The expected Sop* header type in GoodCRC Response should match the header type sent in transmit packet.
6. Disable VALID_DATA_DETECTED interrupt by setting CC_VALID_DATA_DETECTED_MASK bit in the INTR0_MASK register.

7. Clear pending TX interrupts by setting RCV_GOODCRC_MSG_COMPLETE.
8. Set CRC_RX_TIMER_EXP, COLLISION_TYPE1, TX_PACKET_DONE, COLLISION_TYPE2, RCV_EXPT_GOOD-CRC_MS- G_COMPLETE, TX_RETRY_ENABLE_CLRD, CC_NO_VALID_DATA_DETECTED, and TX_S-RAM_HALF_END bits in INTR0.
9. If retransmission counter is negative then create CRC_RX_TIMER_EXP interrupt in INTR0_SET and exit from the sending procedure.
10. If CC line is busy (RX_BUSY OR SENDING_GOOD_CRC_MSG bits are set in the STATUS register) then enable NO_VALID_DATA_DETECTED interrupt in the INTR0_MASK register and exit from the sending procedure.
11. Write data to TX_MEM_DATA. TX_FUNC_RD_PTR bit field of SRAM_PTR points to cell of TX_MEM_DATA where the actual part of transmitted packet should be copied.
12. If there are more packet data to be loaded in TX_MEM_DATA then enable TX_SRAM_HALF_END interrupt in INTR0_MASK.
13. Enable TX interrupts by setting RCV_GOODCRC_MSG_COMPLETE.
14. Set CRC_RX_TIMER_EXP, COLLISION_TYPE1, TX_PACKET_DONE, COLLISION_TYPE2 bits in INRT0_MASK.
15. Begin transmission by setting TX_GO and TX_RETRY_ENABLE (if automatic hardware retry is need) bits in TX_CTRL.

18.8.6 Receiving a PD Packet

See Figure 18-37 for details.

- The CCG7D firmware programs HEADER_INFO.
- The CCG7D hardware receives a data message and generates INTR0.RCV_GOOD_PACKET_COMPLETE or (for extended messages) INTR2.EXTENDED_MSG_DET.
- CCG7D hardware writes the header to RX_HEADER and data to RX_MEM_DATA.
- CCG7D hardware generates INTR0.RX_SRAM_HALF_END informing firmware that hardware has stored 32-bytes of data in the lower 32-bytes of RX memory. Firmware must read the 32-byte before hardware stores the next 32-bytes in the upper 32-bytes of RX memory. Otherwise, an overflow will occur and hardware will create INTR2.RX_SRAM_OVER_-FLOW. In the over flow condition, hardware will keep writing the incoming packet to RX_MEMORY.
- CCG7D hardware also generates INTR0.RX_SRAM_HALF_END after it stores packet in the upper 32-bytes of RX memory.
- CCG7D firmware can use SRAM_PTR.RX_FUNC_RD_PTR to see which location of the RX_MEMORY HW is processing.

18.8.7 Initialization and Usage of ADC

The USB-PD block provides a SAR ADC block. The block can operate as a comparator or a SAR ADC.

Steps for ADC Initialization

1. Enable ADC and power it down by setting ADC_ISO_N and PD_LV bits in the ADC_CTRL register.
2. Calibrate ADC (as described below).

Steps for ADC Calibration

1. Measure 1.2-V bandgap reference voltage (by pd_adc_sample()).
2. Use measured value to calculate actual VDD voltage (in mV).
3. Store actual VDD voltage (which is the reference full-scale voltage for ADC).

Steps for SAR ADC Measurement

1. Configure ADC MUX to appropriate input (to measure pin/VBUS divider/bandgap voltage) by setting VSEL bits in ADC_C-TRL.
2. Power up ADC by setting PD_LV bits in the ADC_CTRL register.
3. Wait 30 microseconds while ADC MUX settles.
4. Clear SAR_DONE status in INTR0.
5. Set SAR_EN in ADC_SAR_CTRL to run sampling.
6. Wait for SAR_DONE INTR0 interrupt or poll this bit.

7. Read SAR_OUT bits in ADC_SAR_CTRL to get sampling result.
8. Clear SAR_DONE status in INTR0.

Steps to Use ADC Block as Voltage Comparator (polling method)

1. Configure ADC MUX to the appropriate input (to set comparator input to pin or VBUS divider) by setting VSEL bits in ADC_CTRL.
2. Set comparator reference level in the DAC_CNTRL bit field of ADC_CTRL.
3. Power up ADC block by setting PD_LV bits in ADC_CTRL.
4. Wait 30 microseconds while ADC MUX settles.
5. Read state of CMP_OUT bit in ADC_CTRL.

Note: This bit is HIGH when input voltage is less than DAC voltage.

Steps to Configure ADC Block as Voltage Comparator for Wakeup (Interrupt method)

- When entering Deep Sleep mode:

1. Disable interrupts.
2. Set awaited rising/falling voltage edge in the FILT_CFG bit of INTR3_CFG_ADC_HS.
3. Configure input ADC MUX (to set comparator input to pin/VBUS divider) by setting VSEL bits in ADC_CTRL.
4. Set comparator requested threshold (detach voltage level) in the DAC_CNTRL bit of ADC_CTRL.
5. Power up ADC by setting PD_LV bits in ADC_CTRL.
6. Wait 30 microseconds while ADC MUX settles.
7. Clear pending ADC interrupt (INTR3_CMP_OUT_CHANGED bit in INTR3).
8. Optionally, configure and enable output de-glitch filter (see Initialization and Usage of UP/DOWN and ADC Comparator Output Filters on page 166).
9. Enable ADC interrupt (set INTR3_CMP_OUT_CHANGED bit in INTR3_MASK).
10. If CMP_OUT state was changed in the awaited direction then rise an interrupt immediately by setting INTR3_CMP_OUT_CHANGED bit in INTR3.
11. Enable interrupts.

- On wake-up interrupt:

1. Set PD_LV bit in ADC_CTRL to power down ADC.
2. Disable ADC interrupt (clear INTR3_CMP_OUT_CHANGED bit in INTR3_MASK).
3. Disable and bypass ADC de-glitch filter (see Initialization and Usage of UP/DOWN and ADC Comparator Output Filters on page 166) if it was enabled before.

18.8.8 Initialization and Usage of UP/DOWN and ADC Comparator Output Filters

The PD block provides digital filters on comparator outputs to de-bounce the CC-line signal. These filters can be hooked up to HF or LF clock. See Table 18-8 for available clock options. The appropriate clock option should be chosen by setting the SEL_CLK_FILTER bit in the CTRL register.

Table 18-8. CCG7D Wake-up Interrupt Mapping

Function	Clock Option	Interrupt	Additional Filter
VCMP_LA	Clk_lf(no filtering)	INTR1[4]	
VCMP_UP	Clk_lf	INTR1[5]	
VCMP_DN	Clk_lf	INTR1[6]	
ADC_CMP_OUT	Clk_lf/Clk_hf	INTR3[1]	Clk_hf, INTR11
VCMP_VBUS	Clk_lf/Clk_hf	INTR13[9]	

The filter logic is completely bypassed when not enabled.

When enabled and FILTER_SEL is programmed to zero, it acts as a synchronizer. When FILTER_SEL is non-zero, 1 to 31 cycle programming will give between 4 to 35 cycles of filtering. All the flops in the filter can be reset to either 0 or 1 to set the

initial state of the filter. To reset the filter, set this bit to the appropriate value and toggle FILTER_EN. Firmware can set values such as reset and configuration values when the FILTER_EN is 0; in the next cycle it can take the filter out of reset by writing '1' to FILTER_EN. FILTER_EN = 0 acts as an asynchronous reset.

Internally when the filter is enabled, the deassertion of reset is synchronized and it takes three to four cycles. So firmware should wait at least five cycles before dynamically changing filter values and applying reset again.

During deep sleep, based on the filter configuration the following options are available for wake-up:

- Bypass Filters: This can be used to instantaneously wake up or enable the shutoff logic. This has a risk that any glitch will cause a trigger.
- Program CLK_LF as clock source to the filter before entering deep sleep: This will make sure the filter is operational in Deep Sleep mode, but has greater delays. Minimum wake-up delay is 100 µs.
- Use AUTO DPSLP_MODE mode: An additional feature is provided where the firmware can program the filter to keep using the peripheral clock; however, it can reduce the latency of fault detection from a minimum 100 s to approximately 35 µs + 2 cycles of peripheral clock. If this mode is enabled by CPU, then as soon as hardware enters Deep Sleep mode, it puts the filter into bypass mode so that any fault can wake up the CPU/chip. It also turns off the corresponding auto shut-off logic. As soon as clocks are available on wake-up (~ 35 µs later), filter bypassing is removed and the original programming takes over.

18.8.9 Regulation Sequence in Firmware

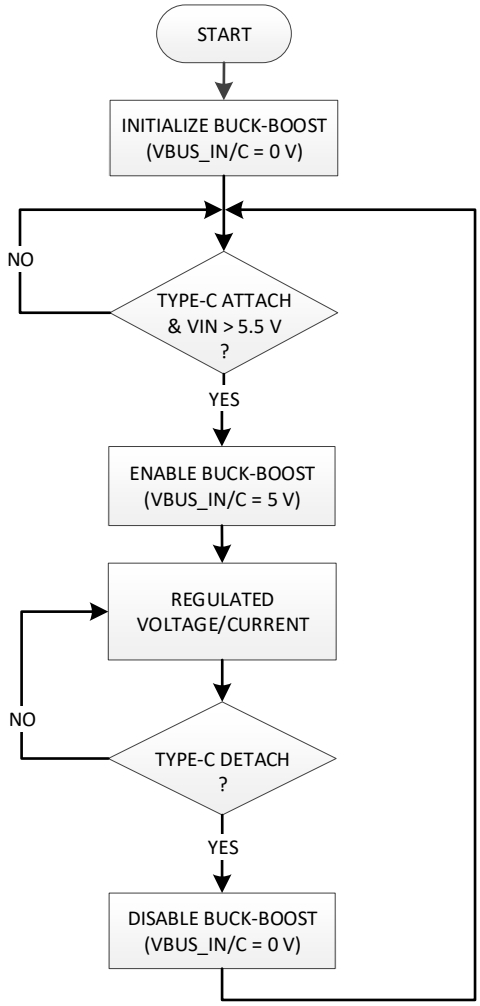
The external regulator is enabled/disabled when Type-C is attached/detached respectively to save power consumption.

The sequence is as follows (Figure 18-38),

1. During PD HAL initialization, buck-boost module is configured for the necessary settings and kept disabled by default. VBUS_IN voltage stays at 0 V.
2. When the Type-C attach is detected, buck-boost module is enabled (Buck-Boost Enable sequence) to regulate VBUS_IN voltage at 5 V.
3. Buck-boost enable is dependent on VIN voltage. If VIN voltage is less than 5.5 V, then buck-boost is not enabled. Since the minimum VIN voltage required is 5.5 V, buck-boost soft start needs to operate only in Deep Buck mode.
4. Once in contract, buck-boost regulation is hardware controlled based on the EA level. The EA level is determined by IDAC and REFGEN setting configured by the firmware for the selected PDO voltage and current.

After Type-C detach is detected, buck-boost module is disabled. Since regulation is disabled, VBUS_IN stays at 0 V.

Figure 18-38. Buck-Boost Regulation Sequence in Firmware



18.8.9.1 Buck-Boost Enable Sequence

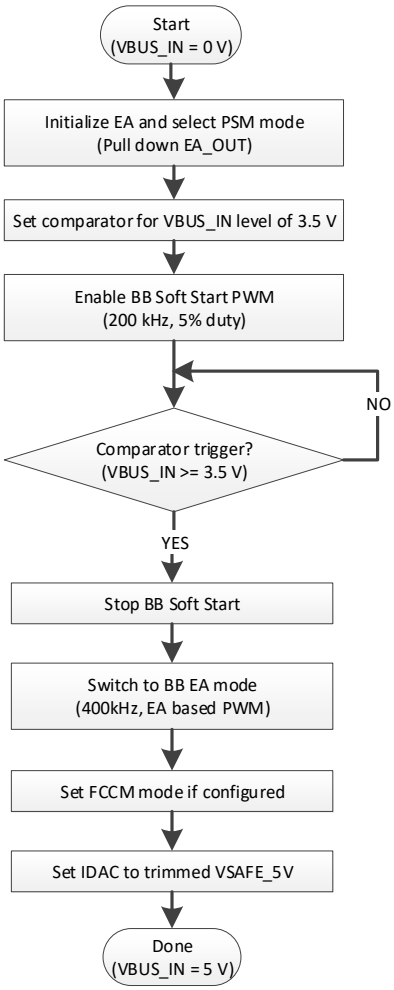
Input voltage (VIN) is 12-V battery in CCG7D-based automotive applications. Buck-boost requires a soft start in buck mode to build initial VBUS_IN voltage of 3.5 V and then transitions to EA-controlled mode or normal mode to maintain the regulation. The following are details of the enable sequence (Figure 18-39):

1. Check if VIN is at least 5.5 V; otherwise, do not proceed for buck-boost enable.
2. CCG7D supports two PWM modes:
   a. Pulse Skip Mode or Discontinuous Conduction Mode (PSM/DCM)
   b. Forced Continuous Conduction Mode (FCCM)

   Configure Buck-Boost PWM mode to PSM/DCM by default to make sure there are no spikes in VBUS_IN during soft start to EA mode transition. However, keep buck-boost disabled.
3. Enable regulation feedback by enabling EA, but keep EA_OUT low by enabling the pull-down option. EA_OUT needs to stay low until soft start is done to build up the initial 3.5 V. Also, EA pull-down helps in faster EA movement from 0 V to 1.2 V (VSAFE_5V equivalent on VBUS_IN), which is faster compared to EA movement from default pull up voltage.
4. Set a comparator (reuse OVP) on VBUS_IN for 3.5 V (soft start complete threshold) detection.
5. Keep 40Vreg inrush detection disabled during buck-boost enable. This is because there will be a sudden load on 40Vreg during buck-boost enable, which triggers unnecessary 40vreg inrush interrupts.
6. Clear all buck-boost fault status and reset state machine to ensure a clean soft start.

7. Start soft start PWM of 200 kHz with 5% in buck-mode duty to build up VBUS_IN voltage from 0 V to 5 V under no load conditions.
8. When VBUS_IN reaches 3.5 V, the configured OVP interrupt is triggered. Stop soft start and keep buck-boost in IDLE mode.
9. Clear all buck-boost fault status and reset state machine to ensure a clean EA mode.
10. Switch buck-boost to EA-based regulation or normal mode. At this point, buck-boost operates at 400 kHz or the configured switching frequency with pulse width as required by the regulation.
11. After a delay of 2 mS, if FCCM mode is configured, set Buck-Boost PWM mode to FCCM. Otherwise, stay in PSM/DCM mode.
12. Set EA IDAC to trimmed VSAFE_5V.
13. Enable auto cut-off of buck-boost module to the required faults.
14. Re-enable 40Vreg inrush fault detection, which was disabled during soft start. Also, enable the inductor current limit detection fault.
15. Now, buck-boost is operating in EA mode. After the PD contract, change IDAC using VBTR to set different regulated voltage.

Figure 18-39. Buck-Boost Enable Sequence



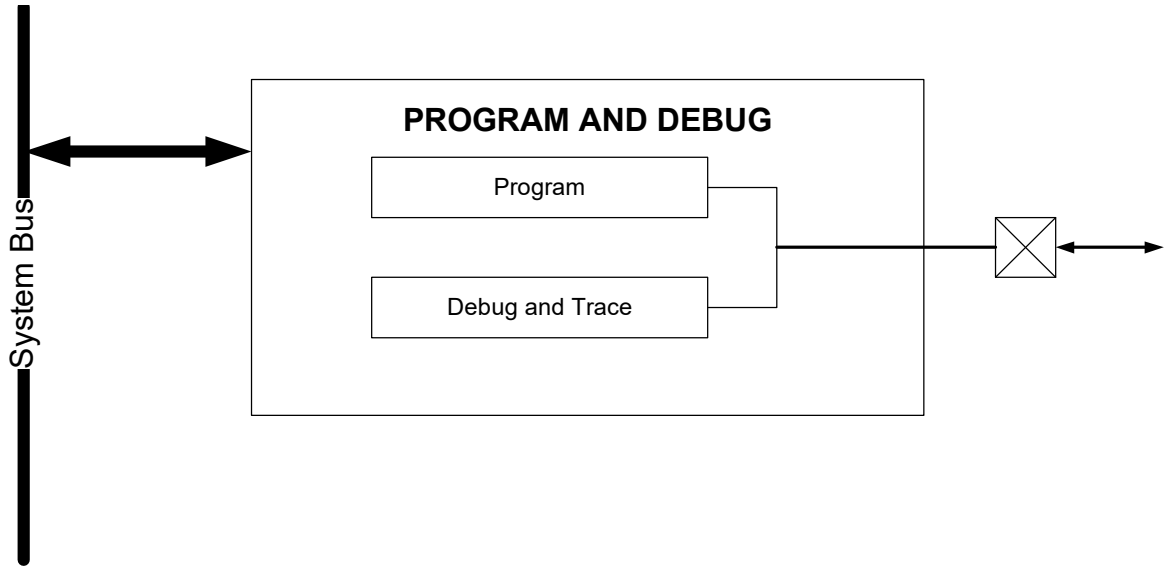
Section G: Program and Debug

This section encompasses the following chapters:

- Program and Debug Interface chapter on page 171
- Nonvolatile Memory Programming chapter on page 178

Top Level Architecture

Program and Debug Block Diagram



19. Program and Debug Interface

The CCG7D Program and Debug interface provides a communication gateway for an external device to perform programming or debugging. The external device can be a Cypress-supplied programmer and debugger, or a third-party device that supports CCG7D programming and debugging. The serial wire debug (SWD) interface is used as the communication protocol between the external device and CCG7D.

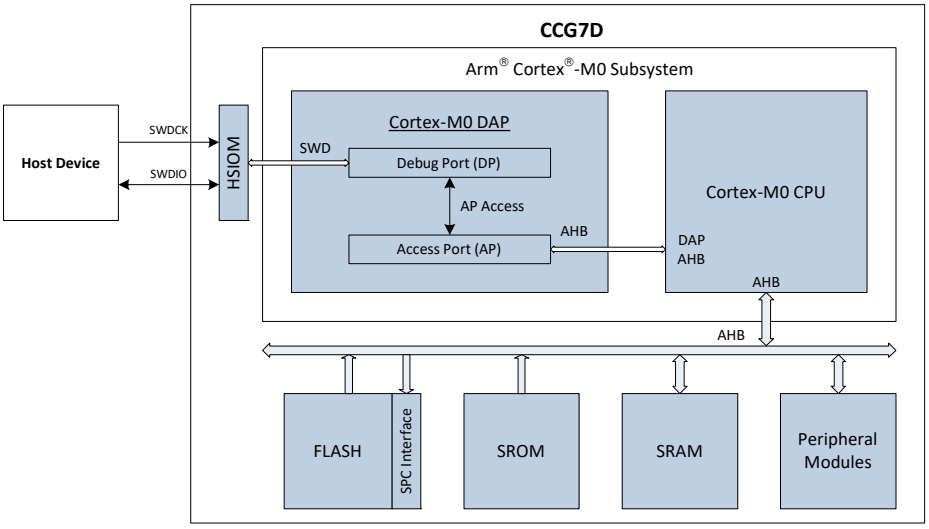
19.1 Features

- Programming and debugging through the SWD interface
- Four hardware breakpoints and two hardware watchpoints while debugging
- Read and write access to all memory and registers in the system while debugging, including the Cortex-M0 register bank when the core is running or halted

19.2 Functional Description

Figure 19-1 shows the block diagram of the program and debug interface in CCG7D. The Cortex-M0 debug and access port (DAP) acts as the program and debug interface. The external programmer or debugger, also known as the “host”, communicates with the DAP of the CCG7D “target” using the two pins of the SWD interface – the bidirectional data pin (SWDIO) and the host-driven clock pin (SWDCK). The SWD physical port pins (SWDIO and SWDCK) communicate with the DAP through the high-speed I/O matrix (HSIOM). See the I/O System chapter on page 40 for details on HSIOM.

Figure 19-1. Program and Debug Interface



The DAP communicates with the Cortex-M0 CPU using the Arm-specified advanced high-performance bus (AHB) interface. AHB is the systems interconnect protocol used inside CCG7D, which facilitates memory and peripheral register access by the AHB master. CCG7D has two AHB masters – Arm CM0 CPU core and DAP. The external device can effectively take control of the entire device through the DAP to perform programming and debugging operations.

19.3 Serial Wire Debug (SWD) Interface

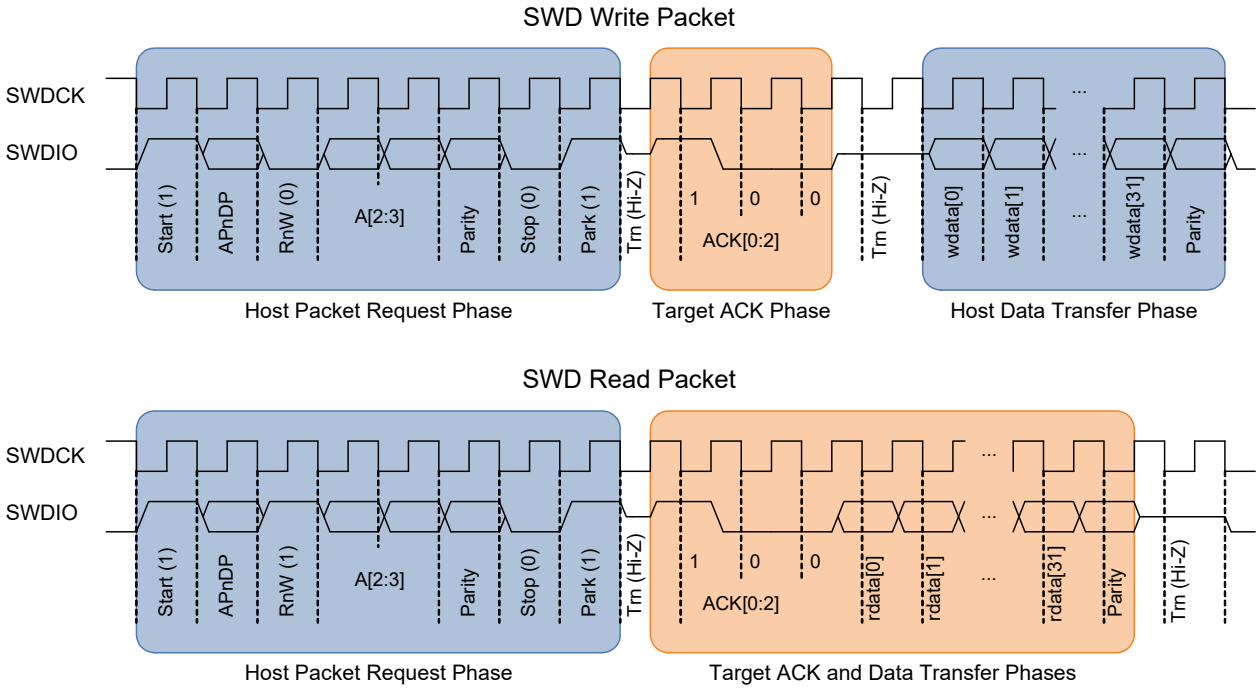
CCG7D's Cortex-M0 supports programming and debugging through the SWD interface. The SWD protocol is a packet-based serial transaction protocol. At the pin level, it uses a single bidirectional data signal (SWDIO) and a unidirectional clock signal (SWDCK). The host programmer always drives the clock line, whereas either the host or the target drives the data line. A complete data transfer (one SWD packet) requires 46 clocks and consists of three phases:

- **Host Packet Request Phase** – The host issues a request to the CCG7D target.
- **Target Acknowledge Response Phase** – The CCG7D target sends an acknowledgement to the host.
- **Data Transfer Phase** – The host or target writes data to the bus, depending on the direction of the transfer.

When control of the SWDIO line passes from the host to the target, or vice versa, there is a turnaround period (Trn) where neither device drives the line and it floats in a high-impedance (Hi-Z) state. This period is either one-half or one and a half clock cycles, depending on the transition.

Figure 19-2 shows the timing diagrams of read and write SWD packets.

Figure 19-2. SWD Write and Read Packet Timing Diagrams



The sequence to transmit SWD read and write packets are as follows:

1. Host Packet Request Phase: SWDIO driven by the host
   a. The start bit initiates a transfer; it is always logic 1.
   b. The "AP not DP" (APnDP) bit determines whether the transfer is an AP access – 1b1 or a DP access – 1b0.
   c. The "Read not Write" bit (RnW) controls which direction the data transfer is in. 1b1 represents a 'read from' the target, or 1b0 for a 'write to' the target.
   d. The Address bits (A[3:2]) are register select bits for AP or DP, depending on the APnDP bit value. See Table 19-3 and Table 19-4 for definitions.
      Note Address bits are transmitted with the LSB first.
   e. The parity bit contains the parity of APnDP, RnW, and ADDR bits. It is an even parity bit; this means, when XORed with the other bits, the result will be 0.
      If the parity bit is not correct, the header is ignored by CCG7D; there is no ACK response (ACK = 3b111). The programming operation should be aborted and retried again by following a device reset.
   f. The stop bit is always logic 0.
   g. The park bit is always logic 1.
2. Target Acknowledge Response Phase: SWDIO driven by the target
   a. The ACK[2:0] bits represent the target to host response, indicating failure or success, among other results. See Table 19-2 for definitions.
      Note ACK bits are transmitted with the LSB first.

3. Data Transfer Phase: SWDIO driven by either target or host depending on direction
   a. The data for read or write is written to the bus, LSB first.
   b. The data parity bit indicates the parity of the data read or written. It is an even parity; this means when XORed with the data bits, the result will be 0.

      If the parity bit indicates a data error, corrective action should be taken. For a read packet, if the host detects a parity error, it must abort the programming operation and restart. For a write packet, if the target detects a parity error, it generates a FAULT ACK response in the next packet.

According to the SWD protocol, the host can generate any number of SWDCK clock cycles between two packets with SWDIO low. It is recommended to generate three or more dummy clock cycles between two SWD packets if the clock is not free-running or to make the clock free-running in IDLE mode.

The SWD interface can be reset by clocking the SWDCK line for 50 or more cycles with SWDIO high. To return to the idle state, clock the SWDIO low once.

19.3.1 SWD Timing Details

The SWDIO line is written to and read at different times depending on the direction of communication. The host drives the SWDIO line during the Host Packet Request Phase and, if the host is writing data to the target, during the Data Transfer phase as well. When the host is driving the SWDIO line, each new bit is written by the host on falling SWDCK edges, and read by the target on rising SWDCK edges. The target drives the SWDIO line during the Target Acknowledge Response Phase and, if the target is reading out data, during the Data Transfer Phase as well. When the target is driving the SWDIO line, each new bit is written by the target on rising SWDCK edges, and read by the host on falling SWDCK edges.

Table 19-1 and Figure 19-2 illustrate the timing of SWDIO bit writes and reads.

Table 19-1. SWDIO Bit Write and Read Timing

SWD Packet Phase	SWDIO Edge	
	Falling	Rising
Host Packet Request	Host Write	Target Read
Host Data Transfer		
Target Ack Response	Host Read	Target Write
Target Data Transfer		

19.3.2 ACK Details

The acknowledge (ACK) bit-field is used to communicate the status of the previous transfer. OK ACK means that previous packet was successful. A WAIT response requires a data phase. For a FAULT status, the programming operation should be aborted immediately. Table 19-2 shows the ACK bit-field decoding details.

Table 19-2. SWD Transfer ACK Response Decoding

Response	ACK[2:0]
OK	3b001
WAIT	3b010
FAULT	3b100
NO ACK	3b111

Details on WAIT and FAULT response behaviors are as follows:

- For a WAIT response, if the transaction is a read, the host should ignore the data read in the data phase. The target does not drive the line and the host must not check the parity bit as well.
- For a WAIT response, if the transaction is a write, the data phase is ignored by the CCG7D. But, the host must still send the data to be written to complete the packet. The parity bit corresponding to the data should also be sent by the host.
- For a WAIT response, it means that the CCG7D is processing the previous transaction. The host can try for a maximum of four continuous WAIT responses to see if an OK response is received. If it fails, then the programming operation should be aborted and retried again.
- For a FAULT response, the programming operation should be aborted and retried again by doing a device reset.

19.3.3 Turnaround (Trn) Period Details

There is a turnaround period between the packet request and the ACK phases, as well as between the ACK and the data phases for host write transfers, as shown in Figure 19-2. According to the SWD protocol, the Trn period is used by both the host and target to change the drive modes on their respective SWDIO lines. During the first Trn period after the packet request, the target starts driving the ACK data on the SWDIO line on the rising edge of SWDCK. This ensures that the host can read the ACK data on the next falling edge. Thus, the first Trn period lasts only one-half cycle. The second Trn period of the SWD packet is one and a half cycles. Neither the host nor CCG7D should drive the SWDIO line during the Trn period.

19.4 Cortex-M0 Debug and Access Port (DAP)

The Cortex-M0 program and debug interface includes a Debug Port (DP) and an Access Port (AP), which combine to form the DAP. The debug port implements the state machine for the SWD interface protocol that enables communication with the host device. It also includes registers for the configuration of access port, DAP identification code, and so on. The access port contains registers that enable the external device to access the Cortex-M0 DAP-AHB interface. Typically, the DP registers are used for a one time configuration or for error detection purposes, and the AP registers are used to perform the programming and debugging operations. Complete architecture details of the DAP is available in the *Arm® Debug Interface v5 Architecture Specification*.

19.4.1 Debug Port (DP) Registers

Table 19-3 shows the Cortex-M0 DP registers used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always zero for DP register accesses. Two address bits (A[3:2]) are used for selecting among the different DP registers. Note that for the same address bits, different DP registers can be accessed depending on whether it is a read or a write operation. See the *Arm® Debug Interface v5 Architecture Specification* for details on all of the DP registers.

Table 19-3. Main Debug Port (DP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
ABORT	0 (DP)	2b00	0 (W)	AP Abort Register	This register is used to force a DAP abort and to clear the error and sticky flag conditions.
IDCODE	0 (DP)	2b00	1 (R)	Identification Code Register	This register holds the SWD ID of the Cortex-M0 CPU, which is 0x0BB11477.
CTRL/STAT	0 (DP)	2b01	X (R/W)	Control and Status Register	This register allows control of the DP and contains status information about the DP.
SELECT	0 (DP)	2b10	0 (W)	AP Select Register	This register is used to select the current AP. In CCG7D, there is only one AP, which interfaces with the DAP AHB.
RDBUFF	0 (DP)	2b11	1 (R)	Read Buffer Register	This register holds the result of the last AP read operation.

19.4.2 Access Port (AP) Registers

Table 19-4 lists the main Cortex-M0 AP registers that are used for programming and debugging, along with the corresponding SWD address bit selections. The APnDP bit is always one for AP register accesses. Two address bits (A[3:2]) are used for selecting the different AP registers.

Table 19-4. Main Access Port (AP) Registers

Register	APnDP	Address A[3:2]	RnW	Full Name	Register Functionality
CSW	1 (AP)	2b00	X (R/W)	Control and Status Word Register (CSW)	This register configures and controls accesses through the memory access port to a connected memory system (which is the CCG7D Memory map)
TAR	1 (AP)	2b01	X (R/W)	Transfer Address Register	This register is used to specify the 32-bit memory address to be read from or written to
DRW	1 (AP)	2b11	X (R/W)	Data Read and Write Register	This register holds the 32-bit data read from or to be written to the address specified in the TAR register

19.5 Programming the CCG7D Device

CCG7D is programmed using the following sequence. Refer to the CYPDXXXX Programming Specifications for complete details on the programming algorithm, timing specifications, and hardware configuration required for programming.

1. Acquire the SWD port in CCG7D.
2. Enter the programming mode.
3. Execute the device programming routines such as Silicon ID Check, Flash Programming, Flash Verification, and Checksum Verification.

19.5.1 SWD Port Acquisition

19.5.1.1 Primary and Secondary SWD Pin Pairs

The first step in device programming is to acquire the SWD port in CCG7D. CCG7D devices have two different pairs of pins that can be used as SWDCK and SWDIO. The primary pair is active by de-fault, but the selection can be changed to the secondary pins by firmware. Refer to the device datasheet for information on SWD pins.

19.5.1.2 SWD Port Acquire Sequence

The first step in device programming is for the host to acquire the target's SWD port. The host first performs a device reset by asserting the external reset (XRES) pin. After removing the XRES signal, the host must send an SWD connect sequence for the device within the acquire window to connect to the SWD interface in the DAP. The pseudo code for the sequence is given here.

Code 1. SWD Port Acquire Pseudo Code

```
ToggleXRES(); // Toggle XRES pin to reset device

//Execute Arm's connection sequence to acquire SWD-port
do
{
       SWD_LineReset(); //perform a line reset (50+ SWDCK clocks with SWDIO high)
       ack = Read_DAP ( IDCODE, out ID); //Read the IDCODE DP register

}while ((ack != OK) && time_elapsed < 5 ms); // retry connection until OK ACK or timeout

if (time_elapsed >= 5 ms) return FAIL; //check for acquire time out

if (ID != CM0_ID) return FAIL; //confirm SWD ID of Cortex-M0 CPU. (0x0BB11477)
```

In this pseudo code, SWD_LineReset() is the standard Arm command to reset the debug access port. It consists of more than 49 SWDCK clock cycles with SWDIO high. The transaction must be completed by sending at least one SWDCK clock cycle with SWDIO asserted LOW. This sequence synchronizes the programmer and the chip. Read_DAP() refers to the read of the IDCODE register in the debug port. The sequence of line reset and IDCODE read should be repeated until an OK ACK is received for the IDCODE read or a timeout (5 ms) occurs. The SWD port is said to be in the acquired state if an OK ACK is received within the time window and the IDCODE read matches with that of the Cortex-M0 DAP.

19.5.2 SWD Programming Mode Entry

After the SWD port is acquired, the host must enter the device programming mode within a specific time window. This is done by setting the TEST_MODE bit (bit 31) in the test mode control register (MODE register). The debug port should also be configured before entering the device programming mode. Timing specifications and pseudo code for entering the programming mode are detailed in the CYPDXXXX Programming Specifications document.

19.5.3 SWD Programming Routines Executions

When the device is in programming mode, the external programmer can start sending the SWD packet sequence for performing programming operations such as flash erase, flash program, checksum verification, and so on. The programming routines are explained in Nonvolatile Memory Programming chapter on page 178. The exact sequence of calling the programming routines is given in the CYPDXXXX Programming Specifications document.

19.6 CCG7D SWD Debug Interface

Cortex-M0 DAP debugging features are classified into two types: invasive debugging and noninvasive debugging. Invasive debugging includes program halting and stepping, breakpoints, and data watchpoints. Noninvasive debugging includes instruction address profiling and device memory access, which includes the flash memory, SRAM, and other peripheral registers.

The DAP has three major debug subsystems:

- Debug Control and Configuration registers
- Breakpoint Unit (BPU) – provides breakpoint support
- Debug Watchpoint (DWT) – provides watchpoint support. Trace is not supported in Cortex-M0 Debug.

See the *Armv6-M Architecture Reference Manual* for complete details on the debug architecture.

19.6.1 Debug Control and Configuration Registers

The debug control and configuration registers are used to execute firmware debugging. The registers and their key functions are as follows. See the *Armv6-M Architecture Reference Manual* for complete bit level definitions of these registers.

- Debug Halting Control and Status Register (CM0_DHCSR) – This register contains the control bits to enable debug, halt the CPU, and perform a single-step operation. It also includes status bits for the debug state of the processor.
- Debug Fault Status Register (CM0_DFSR) – This register describes the reason a debug event has occurred. This includes debug events, which are caused by a CPU halt, breakpoint event, or watchpoint event.
- Debug Core Register Selector Register (CM0_DCRSR) – This register is used to select the general-purpose register in the Cortex-M0 CPU to which a read or write operation must be performed by the external debugger.
- Debug Core Register Data Register (CM0_DCRDR) – This register is used to store the data to write to or read from the register selected in the CM0_DCRSR register.
- Debug Exception and Monitor Control Register (CM0_DEMCR) – This register contains the enable bits for global debug watchpoint (DWT) block enable, reset vector catch, and hard fault exception catch.

19.6.2 Breakpoint Unit (BPU)

The BPU provides breakpoint functionality on instruction fetches. The Cortex-M0 DAP in CCG7D supports up to four hardware breakpoints. Along with the hardware breakpoints, any number of software breakpoints can be created by using the BKPT instruction in the Cortex-M0. The BPU has two types of registers.

- The breakpoint control register (CM0_BP_CTRL) is used to enable the BPU and store the number of hardware breakpoints supported by the debug system (four for CM0 DAP in CCG7D).
- Each hardware breakpoint has a Breakpoint Compare Register (CM0_BP_COMPx). It contains the enable bit for the breakpoint, the compare address value, and the match condition that will trigger a breakpoint debug event. The typical use case is that when an instruction fetch address matches the compare address of a breakpoint, a breakpoint event is generated and the processor is halted.

19.6.3 Data Watchpoint (DWT)

The DWT provides watchpoint support on a data address access or a program counter (PC) instruction address. Trace is not supported by the Cortex-M0 in CCG7D. The DWT supports two watchpoints. It also provides external program counter sampling using a PC sample register, which can be used for noninvasive coarse profiling of the program counter. The most important registers in the DWT are as follows.

- The watchpoint compare (CM0_DWT_COMPx) registers store the compare values that are used by the watchpoint comparator for the generation of watchpoint events. Each watchpoint has an associated DWT_COMPx register.
- The watchpoint mask (CM0_DWT_MASKx) registers store the ignore masks applied to the address range matching in the associated watchpoints.
- The watchpoint function (CM0_DWT_FUNCTIONx) registers store the conditions that trigger the watchpoint events. They may be program counter watchpoint event or data address read/write access watchpoint events. A status bit is also set when the associated watchpoint event has occurred.
- The watchpoint comparator PC sample register (CM0_DWT_PCSR) stores the current value of the program counter. This register is used for coarse, noninvasive profiling of the program counter register.

19.6.4 Debugging the CCG7D Device

The host debugs the target CCG7D device by accessing the debug control and configuration registers, registers in the BPU, and registers in the DWT. All registers are accessed through the SWD interface; the SWD debug port (SW-DP) in the Cortex-M0 DAP converts the SWD packets to appropriate register access through the DAP-AHB interface.

The first step in debugging the target CCG7D device is to acquire the SWD port. The acquire sequence consists of an SWD line reset sequence and read of the DAP SWDID through the SWD interface. The SWD port is acquired when the correct CM0 DAP SWDID is read from the target device. For the debug transactions to occur on the SWD interface, the corresponding pins should not be used for any other purpose. See the I/O System chapter on page 40 to understand how to configure the SWD port pins, allowing them to be used only for SWD interface or for other functions such as GPIO. If debugging is required, the SWD port pins should not be used for other purposes. If only programming support is needed, the SWD pins can be used for other purposes.

When the SWD port is acquired, the external debugger sets the C_DEBUGEN bit in the DHCSR register to enable debugging. Then, the different debugging operations such as stepping, halting, breakpoint configuration, and watchpoint configuration are carried out by writing to the appropriate registers in the debug system.

Debugging the target device is also affected by the overall device protection setting, which is explained in the Device Security chapter on page 64. Only the OPEN protected mode supports device debugging. The external debugger and the target device connection is not lost for a device transition from Active mode to either Sleep or Deep Sleep modes. When the device enters the Active mode from either Deep-Sleep or Sleep modes, the debugger can resume its actions without initiating a connect sequence again.

19.7 Registers

Table 19-5. List of Registers

Register Name	Description
CM0_DHCSR	Debug Halting Control and Status Register
CM0_DFSR	Debug Fault Status Register
CM0_DCRSR	Debug Core Register Selector Register
CM0_DCRDR	Debug Core Register Data Register
CM0_DEMCR	Debug Exception and Monitor Control Register
CM0_BP_CTRL	Breakpoint control register
CM0_BP_COMPx	Breakpoint Compare Register
CM0_DWT_COMPx	Watchpoint Compare Register
CM0_DWT_MASKx	Watchpoint Mask Register
CM0_DWT_FUNCTIONx	Watchpoint Function Register
CM0_DWT_PCSR	Watchpoint Comparator PC Sample Register

20. Nonvolatile Memory

Programming

Nonvolatile memory programming refers to the programming of flash memory in the CCG7D device. This chapter explains the different functions that are part of device programming, such as erase, write, program, and checksum calculation. Cypress-supplied programmers and other third-party programmers can use these functions to program the CCG7D device with the data in an application hex file. They can also be used to perform bootload operations where the CPU will update a portion of the flash memory.

20.1 Features

- Supports programming through the debug and access port (DAP) and Cortex-M0 CPU
- Supports both blocking and non-blocking flash program and erase operations from the Cortex-M0 CPU

20.2 Functional Description

Flash programming operations are implemented as system calls. System calls are executed out of SROM in the privileged mode of operation. The user has no access to read or modify the SROM code. The DAP or the CM0 CPU requests the system call by writing the function opcode and parameters to the System Performance Controller Interface (SPCIF) input registers, and then requesting the SROM to execute the function. Based on the function opcode, the System Performance Controller (SPC) executes the corresponding system call from SROM and updates the SPCIF status register. The DAP or the CPU should read this status register for the pass/fail result of the function execution. As part of function execution, the code in SROM interacts with the SPCIF to do the actual flash programming operations.

CCG7D flash is programmed using a Program Erase Program (PEP) sequence. The flash cells are all programmed to a known state, erased, and then the selected bits are programmed. This increases the life of the flash by balancing the stored charge. When writing to flash the data is first copied to a page latch buffer. The flash write functions are then used to transfer this data to flash. External programmers program the flash memory in CCG7D using the SWD protocol by sending the commands to the DAP. The programming sequence for the CCG7D device with an external programmer is given in the CYPDXXXX Programming Specifications. Flash memory can also be programmed by the CM0 CPU by accessing the relevant registers through the AHB interface. This type of programming is typically used to update a portion of the flash memory as part of a bootload operation, or other application requirements, such as updating a lookup table stored in the flash memory. All write operations to flash memory, whether from the DAP or from the CPU, are done through the SPCIF.

Note It can take as much as 20 milliseconds to write to flash. During this time, the device should not be reset, or unexpected changes may be made to portions of the flash. Reset sources (see the Reset System chapter on page 62) include XRES pin, software reset, and watchdog; make sure that these are not inadvertently activated. In addition, the low-voltage detect circuits should be configured to generate an interrupt instead of a reset.

20.3 System Call Implementation

A system call consists of the following items:

- Opcode: A unique 8-bit opcode
- Parameters: Two 8-bit parameters are mandatory for all system calls. These parameters are referred to as key1 and key2, and are defined as follows:

  key1 = 0xB6

  key2 = 0xD3 + Opcode

  The two keys are passed to ensure that the user system call is not initiated by mistake. If the key1 and key2 parameters are not correct, the SROM does not execute the function, and returns an error code. Apart from these two parameters, additional parameters may be required depending on the specific function being called.
- Return Values: Some system calls also return a value on completion of their execution, such as the silicon ID or a checksum.
- Completion Status: Each system call returns a 32-bit status that the CPU or DAP can read to verify success or determine the reason for failure.

20.4 Blocking and Non-Blocking System Calls

System call functions can be categorized as blocking or non-blocking based on the nature of their execution. Blocking system calls are those where the CPU cannot execute any other task in parallel other than the execution of the system call. When a blocking system call is called from a process, the CPU jumps to the code corresponding in SROM. When the execution is complete, the original thread execution resumes. Non-blocking system calls allow the CPU to execute some other code in parallel and communicate the completion of interim system call tasks to the CPU through an interrupt.

Non-blocking system calls are only used when the CPU initiates the system call. The DAP will only use system calls during the programming mode and the CPU is halted during this process. The three non-blocking system calls are Non-Blocking Write Row, Non-Blocking Program Row, and Resume Non-Blocking, respectively. All other system calls are blocking.

Because the CPU cannot execute code from flash while doing an erase or program operation on the flash, the non-blocking system calls can only be called from a code executing out of SRAM. If the non-blocking functions are called from flash memory, the result is undefined and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

The System Performance Controller (SPC) is the block that generates the properly sequenced high-voltage pulses required for erase and program operations of the flash memory. When a non-blocking function is called from SRAM, the SPC timer triggers its interrupt when each of the sub-operations in a write or program operation is complete. Call the Resume Non-Blocking function from the SPC interrupt service routine (ISR) to ensure that the subsequent steps in the system call are completed. Because the CPU can execute code only from the SRAM when a non-blocking write or program operation is being done, the SPC ISR should also be located in the SRAM. The SPC interrupt is triggered once in the case of a non-blocking program function or thrice in a non-blocking write operation. The Resume Non-Blocking function call done in the SPC ISR is called once in a non-blocking program operation and thrice in a non-blocking write operation.

The pseudo code for using a non-blocking write system call and executing user code out of SRAM is given later in this chapter.

20.4.1 Performing a System Call

The steps to initiate a system call are as follows:

1. Set up the function parameters: The two possible methods for preparing the function parameters (key1, key2, additional parameters) are:
   a. Write the function parameters to the CPUSS_SYSARG register: This method is used for functions that retrieve their parameters from the CPUSS_SYSARG register. The 32-bit CPUSS_SYSARG register must be written with the parameters in the sequence specified in the respective system call table.
   b. Write the function parameters to SRAM: This method is used for functions that retrieve their parameters from SRAM. The parameters should first be written in the specified sequence to consecutive SRAM locations. Then, the starting address of the SRAM, which is the address of the first parameter, should be written to the CPUSS_SYSARG register. This starting address should always be a word-aligned (32-bit) address. The system call uses this address to fetch the parameters.
2. Specify the system call using its opcode and initiating the system call: The 8-bit opcode should be written to the SYSCALL_COMMAND bits ([15:0]) in the CPUSS_SYSREQ register. The opcode is placed in the lower eight bits [7:0] and 0x00 be written to the upper eight bits [15:8]. To initiate the system call, set the SYSCALL_REQ bit (31) in the CPUSS_SYSREG register. Setting this bit triggers a non-maskable interrupt that jumps the CPU to the SROM code referenced by the opcode parameter.
3. Wait for the system call to finish executing: When the system call begins execution, it sets the PRIVILEGED bit in the CPUSS_SYSREQ register. This bit can be set only by the system call, not by the CPU or DAP. The DAP should poll the PRIVILEGED and SYSCALL_REQ bits in the CPUSS_SYSREG register continuously to check whether the system call is completed. Both these bits are cleared on completion of the system call. The maximum execution time is one second. If these two bits are not cleared after one second, the operation should be considered a failure and aborted without executing the following steps. Note that unlike the DAP, the CPU application code cannot poll these bits during system call execution. This is because the CPU executes code

out of the SROM during the system call. The application code can check only the final function pass/fail status after the execution returns from SROM.

4. Check the completion status: After the PRIVILEGED and SYSCALL_REQ bits are cleared to indicate completion of the system call, the CPUSS_SYSARG register should be read to check for the status of the system call. If the 32-bit value read from the CPUSS_SYSARG register is 0xAXXXXXXX (where 'X' denotes don't care hex values), the system call was successfully executed. For a failed system call, the status code is 0xF00000YY where YY indicates the reason for failure. See Table 20-2 for the complete list of status codes and their description.
5. Retrieve the return values: For system calls that return values such as silicon ID and checksum, the CPU or DAP should read the CPUSS_SYSREG and CPUSS_SYSARG registers to fetch the values returned.

20.5 System Calls

Table 20-1 lists all the system calls supported in CCG7D along with the function description and availability in device protection modes. See the Device Security chapter on page 64 for more information on the device protection settings. Note that some system calls cannot be called by the CPU as given in the table. Detailed information on each of the system calls follows the table.

Table 20-1. List of System Calls

System Call	Description	DAP Access			CPU Access
		Open	Protected	Kill	
Silicon ID	Returns the device Silicon ID, Family ID, and Revision ID	✔	✔	–	✔
Load Flash Bytes	Loads data to the page latch buffer to be programmed later into the flash row, in 1 byte granularity, for a row size of 256 bytes	✔	–	–	✔
Write Row	Erases and then programs a row of flash with data in the page latch buffer	✔	–	–	✔
Program Row	Programs a row of flash with data in the page latch buffer	✔	–	–	✔
Erase All	Erases all user code in the flash array; the flash row-level protection data in the supervisory flash area	✔	–	–	
Checksum	Calculates the checksum over the entire flash memory (user and supervisory area) or checksums a single row of flash	✔	✔	–	✔
Write Protection	This programs both flash row-level protection settings and chip-level protection settings into the supervisory flash (row 0)	✔	✔	–	
Non-Blocking Write Row	Erases and then programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Non-Blocking Program Row	Programs a row of flash with data in the page latch buffer. During program/erase pulses, the user may execute code from SRAM. This function is meant only for CPU access	–	–	–	✔
Resume Non-Blocking	Resumes a non-blocking write row or non-blocking program row. This function is meant only for CPU access	–	–	–	✔

20.5.1 Silicon ID

This function returns a 12-bit family ID, 16-bit silicon ID, and an 8-bit revision ID, and the current device protection mode. These values are returned to the CPUSS_SYSARG and CPUSS_SYSREQ registers. Parameters are passed through the CPUSS_SYSARG and CPUSS_SYSREQ registers.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG Register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD3	Key2
Bits [31:16]	0x0000	Not used
CPUSS_SYSREQ register		
Bits [15:0]	0x0000	Silicon ID opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [7:0]	Silicon ID Lo	See the CYPDxxxx Programming Specifications for Silicon ID values for different part numbers
Bits [15:8]	Silicon ID Hi	
Bits [19:16]	Minor Revision Id	See the CYPDxxxx Programming Specifications for these values
Bits [23:20]	Major Revision Id	
Bits [27:24]	0xXX	Not used (don't care)
Bits [31:28]	0xA	Success status code
CPUSS_SYSREQ register		
Bits [11:0]	Family ID	Family ID is 0xC0 for CCG7D
Bits [15:12]	Chip Protection	See the Device Security chapter on page 64
Bits [31:16]	0xXXXX	Not used

20.5.2 Configure Clock

This function initializes the clock necessary for flash programming and erasing operations. This API is used to ensure that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz prior to calling the flash write and flash erase APIs. The flash write and erase APIs will exit without acting on the flash and return the "Invalid Pump Clock Frequency" status if the IMO is the source of the charge pump clock and is not 48 MHz.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE8	Key2
Bits [31:16]	0xXXXX	Don't care
CPUSS_SYSREQ register		
Bits [15:0]	0x0015	Configure clock opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

20.5.3 Load Flash Bytes

This function loads the page latch buffer with data to be programmed into a row of flash. The load size can range from 1-byte to the maximum number of bytes in a flash row, which is 128 bytes. Data is loaded into the page latch buffer starting at the location specified by the "Byte Addr" input parameter. Data loaded into the page latch buffer remains until a program operation is performed, which clears the page latch contents. The parameters for this function, including the data to be loaded into the page latch, are written to the SRAM; the starting address of the SRAM data is written to the CPUSS_SYSARG register. Note that the starting parameter address should be a word-aligned address.

Parameters

Address	Value to be Written	Description
SRAM Address – 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD7	Key2
Bits [23:16]	Byte Addr	Start address of page latch buffer to write data 0x00 – Byte 0 of latch buffer 0x7F – Byte 127 of latch buffer
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0
SRAM Address- 32'hYY + 0x04		
Bits [7:0]	Load Size	Number of bytes to be written to the page latch buffer. 0x00 – 1 byte 0x7F – 128 bytes
Bits [15:8]	0xXX	Don't care parameter
Bits [23:16]	0xXX	Don't care parameter
Bits [31:24]	0xXX	Don't care parameter
SRAM Address- From (32'hYY + 0x08) to (32'hYY + 0x08 + Load Size)		
Byte 0	Data Byte [0]	First data byte to be loaded
.	.	.
.	.	.
Byte (Load size –1)	Data Byte [Load size –1]	Last data byte to be loaded
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0004	Load Flash Bytes opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

20.5.4 Write Row

This function erases and then programs the addressed row of flash with the data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The parameters for this function are stored in SRAM. The start address of the stored parameters is written to the CPUSS_SYSARG register. This function clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. This function can do a write operation only if the corresponding flash row is not write protected.

Note that the SROM does not modify, enable, or disable any clock during any flash operation. Refer to the CLK_IMO_CONFIG register in the CCG7D Family EZ-PD CCG7D Registers TRM for more information.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD8	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0 0x01FF - Row 511
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0005	Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

20.5.5 Program Row

This function programs the addressed row of the flash, with data in the page latch buffer. If all data in the page latch buffer is 0, then the program is skipped. The row must be in an erased state before calling this function. This clears the page latch buffer contents after the row is programmed.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function. The row must be in an erased state before calling this function. This function can do a program operation only if the corresponding flash row is not write-protected.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xD9	Key2
Bits [31:16]	Row ID	Row number to program 0x0000 – Row 0 0x01FF - Row 511

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0006	Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

20.5.6 Erase All

This function erases all the user code in the flash main arrays and the row-level protection data in supervisory flash row 0.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. This API can be called only from the DAP in the programming mode and only if the chip protection mode is OPEN. If the chip protection mode is PROTECTED, then the Write Protection API must be used by the DAP to change the protection settings to OPEN. Changing the protection setting from PROTECTED to OPEN automatically does an erase all operation.

Parameters

Address	Value to be Written	Description
SRAM Address: 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDD	Key2
Bits [31:16]	0xXXXX	Don't care
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x000A	Erase All opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:0]	0xXXXXXXX	Not used (don't care)

20.5.7 Checksum

This function reads either the whole flash memory or a row of flash and returns the 24-bit sum of each byte read in that flash region. When performing a checksum on the whole flash, the user code and supervisory flash regions are included. When performing a checksum only on one row of flash, the flash row number is passed as a parameter. Bytes 2 and 3 of the parameters select whether the checksum is performed on the whole flash memory or a row of user code flash.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDE	Key2
Bits [31:16]	Row ID	Selects the flash row number on which the checksum operation is done Row number – 16-bit flash row number or 0x8000 – Checksum is performed on entire flash memory
CPUSS_SYSREQ register		
Bits [15:0]	0x000B	Checksum opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	Checksum	24-bit checksum value of the selected flash region

20.5.8 Write Protection

This function programs both the flash row-level protection settings and the device protection settings in the supervisory flash row. The flash row-level protection settings are programmed separately for each flash macro in the device. Each row has a single protection bit. The total number of protection bytes is the number of flash rows divided by eight. The chip-level protection settings (1-byte) are stored in flash macro zero in the last byte location in row zero of the supervisory flash. The size of the supervisory flash row is the same as the user code flash row size.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. The Load Flash Bytes function is used to load the flash protection bytes of a flash macro into the page latch buffer corresponding to the macro. The starting address parameter for the load function should be zero. The flash macro number should be one that needs to be programmed; the number of bytes to load is the number of flash protection bytes in that macro.

Then, the Write Protection function is called, which programs the flash protection bytes from the page latch to be the corresponding flash macro's supervisory row. In flash macro zero, which also stores the device protection settings, the device level protection setting is passed as a parameter in the CPUSS_SYSARG register.

Parameters

Address	Value to be Written	Description
CPUSS_SYSARG register		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xE0	Key2
Bits [23:16]	Device Protection Byte	Parameter applicable only for Flash Macro 0 0x01 – OPEN mode 0x02 – PROTECTED mode 0x04 – KILL mode
Bits [31:24]	Flash Macro Select	0x00 – Flash Macro 0
CPUSS_SYSREQ register		
Bits [15:0]	0x000D	Write Protection opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code
Bits [27:24]	0xX	Not used (don't care)
Bits [23:0]	0x000000	

20.5.9 Non-Blocking Write Row

This function is used when a flash row needs to be written by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from SRAM while the write operation is being done. Blocking and Non-Blocking System Calls on page 179 describes non-blocking system calls in detail.

The non-blocking write row system call has three phases: Preprogram, Erase, Program. Preprogram is the step in which all of the bits in the flash row are written a '1' in preparation for an erase operation. The erase operation clears all of the bits in the row, and the program operation writes the new data to the row.

While each phase is being executed, the CPU can execute code from SRAM. When the non-blocking write row system call is initiated, the user cannot call any system call function other than the Resume Non-Blocking function, which is required for completion of the non-blocking write operation. After the completion of each phase, the SPC triggers its interrupt. In this interrupt, call the Resume Non-Blocking system call.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz.

Note The device firmware must not attempt to put the device to sleep during a non-blocking write row. This will reset the page latch buffer and the flash will be written with all zeroes.

Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking write row function can be called only from the SRAM. This is because the CM0 CPU cannot execute code from flash while doing the flash erase program operations. If this function is called from the flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDA	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0 0x01FF - Row 511
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0007	Non-Blocking Write Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits[31:0]	32'hYY	SRAM addressed programmed by user in case of success. Will return status code in case of error.

20.5.10 Non-Blocking Program Row

This function is used when a flash row needs to be programmed by the CM0 CPU in a non-blocking manner, so that the CPU can execute code from the SRAM when the program operation is being done. Blocking and Non-Blocking System Calls on page 179 describes non-blocking system calls in detail. While the program operation is being done, the CPU can execute code from the SRAM. When the non-blocking program row system call is called, the user cannot call any other system call function other than the Resume Non-Blocking function, which is required for the completion of the non-blocking write operation.

Unlike the Non-Blocking Write Row system call, the Program system call only has a single phase. Therefore, the Resume Non-Blocking function only needs to be called once from the SPC interrupt when using the Non-Blocking Program Row system call.

Usage Requirements: Call the Configure Clock API before calling this function. The Configure Clock API ensures that the charge pump clock (clk_pump) and the HF clock (clk_hf) are set to IMO at 48 MHz. Call the Load Flash Bytes function before calling this function to load the data bytes that will be used for programming the row. In addition, the non-blocking program row function can be called only from SRAM. This is because the CM0 CPU cannot execute code from flash while doing flash program operations. If this function is called from flash memory, the result is undefined, and may return a bus error and trigger a hard fault when the flash fetch operation is being done.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDB	Key2
Bits [31:16]	Row ID	Row number to write 0x0000 – Row 0 0x01FF - Row 511
CPUSS_SYSARG register		

Address	Value to be Written	Description
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0008	Non-Blocking Program Row opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits[31:0]	32'hYY	SRAM addressed programmed by user in case of success. Will return status code in case of error.

20.5.11 Resume Non-Blocking

This function completes the additional phases of erase and program that were started using the non-blocking write row and non-blocking program row system calls. This function must be called thrice following a call to Non-Blocking Write Row or once following a call to Non-Blocking Program Row from the SPC ISR. No other system calls can execute until all phases of the program or erase operation are complete. More details on the procedure of using the non-blocking functions are available in Blocking and Non-Blocking System Calls on page 179.

Parameters

Address	Value to be Written	Description
SRAM Address 32'hYY (32-bit wide, word-aligned SRAM address)		
Bits [7:0]	0xB6	Key1
Bits [15:8]	0xDC	Key2
Bits [31:16]	0xXXXX	Don't care. Not used by SROM
CPUSS_SYSARG register		
Bits [31:0]	32'hYY	32-bit word-aligned address of the SRAM that stores the first function parameter (key1)
CPUSS_SYSREQ register		
Bits [15:0]	0x0009	Resume Non-Blocking opcode
Bits [31:16]	0x8000	Set SYSCALL_REQ bit

Return

Address	Return Value	Description
CPUSS_SYSARG register		
Bits [31:28]	0xA	Success status code in the case of the last resume call which completes the program or write operation.
Bits [31:0]	32'hYY	SRAM addressed programmed by user in case of intermediate resume operations.

20.6 System Call Status

At the end of every system call, a status code is written over the arguments in the CPUSS_SYSARG register. A success status is 0xAXXXXXXX, where X indicates don't care values or return data in the case of the system calls that return a value. A failure status is indicated by 0xF00000XX, where XX is the failure code.

Table 20-2. System Call Status Codes

Status Code (32-bit value in CPUSS_SYSARG Register)	Description
AXXXXXXXh	Success – The "X" denotes a don't care value, which has a value of '0' returned by the SROM, unless the API returns parameters directly to the CPUSS_SYSARG register.
F0000001h	Invalid Chip Protection Mode – This API is not available during the current chip protection mode.
F0000003h	Invalid Page Latch Address – The address within the page latch buffer is either out of bounds or the size provided is too large for the page address.
F0000004h	Invalid Address – The row ID or byte address provided is outside of the available memory.
F0000005h	Row Protected – The row ID provided is a protected row.
F0000007h	Resume Completed – All non-blocking APIs have completed. The resume API cannot be called until the next non-blocking API.
F0000008h	Pending Resume – A non-blocking API was initiated and must be completed by calling the resume API, before any other API's may be called.
F0000009h	System Call Still In Progress – A resume or non-blocking is still in progress. The SPC ISR must fire before attempting the next resume.
F000000Ah	Checksum Zero Failed – The calculated checksum was not zero.
F000000Bh	Invalid Opcode – The opcode is not a valid API opcode.
F000000Ch	Key Opcode Mismatch – The opcode provided does not match key1 and key2.
F000000Eh	Invalid Start Address – The start address is greater than the end address provided.
F0000012h	Invalid Pump Clock Frequency – IMO must be set to 48 MHz and HF clock source to the IMO clock source before flash write/erase operations.

20.7 Non-Blocking System Call Pseudo Code

This section contains pseudo code to demonstrate how to set up a non-blocking system call and execute code out of SRAM during the flash programming operations.

```
#define REG(addr)(*((volatile uint32 *) (addr)))
#define CM0_ISER_REG REG( 0xE000E100 )
#define CPUSS_CONFIG_REG REG( 0x40100000 )
#define CPUSS_SYSREQ_REG REG( 0x40100004 )
#define CPUSS_SYSARG_REG REG( 0x40100008 )
#define ROW_SIZE              128

//Variable to keep track of how many times SPC ISR is triggered
__ram int iStatusInt = 0x00;

__flash int main(void)
{
   DoUserStuff();

   //CM0 interrupt enable bit for spc interrupt enable
   CM0_ISER_REG |= 0x00004000;

   //Set CPUSS_CONFIG.VECT_IN_RAM because SPC ISR should be in SRAM
   CPUSS_CONFIG_REG |= 0x00000001;
   //Call non-blocking write row API

   NonBlockingWriteRow();

   //End Program
   while(1);
}

__sram void SpcIntHandler(void)
{
   /* Call Resume API */

   // Write key1, key2 parameters to SRAM
   REG( 0x20002000 ) = 0x0000DCB6;

   //Write the address of key1 to the CPUSS_SYSARG reg
   CPUSS_SYSARG_REG = 0x20002000;

   //Write the API opcode = 0x09 to the CPUSS_SYSREQ.COMMAND
   //register and assert the sysreq bit
   CPUSS_SYSREQ_REG = 0x80000009;

   iStatusInt ++; // Number of times the ISR has triggered
}
 __sram void NonBlockingWriteRow(void)
 {
     int iter;

     /*Load the Flash page latch with data to write*/

     //Write key1, key2, byte address,
     //and macro sel parameters to SRAM
     REG( 0x20002000 ) = 0x0000D7B6;

     //Write load size param (128 bytes) to SRAM
     REG( 0x20002004 ) = 0x0000007F;

     for(i = 0; i < ROW_SIZE/4; i += 1);
     {
         REG( 0x20002008 + i*4 ) = 0xDADADADA;
     }

     //Write the address of the key1 param to CPUSS_SYSARG reg
     CPUSS_SYSARG_REG = 0x20002000;
```

```
        //Write the API opcode = 0x04 to CPUSS_SYSREQ.COMMAND
        //register and assert the sysreq bit
        CPUSS_SYSREQ_REG = 0x80000004;

        /*Perform Non-Blocking Write Row on Row 200 as an example */

        //Write key1, key2, row id to SRAM
        //row id = 0xC8 -> which is row 200
        REG( 0x20002000 ) = 0x00C8DAB6;

        //Write the address of the key1 param to CPUSS_SYSARG reg
        CPUSS_SYSARG_REG = 0x20002000;

        //Write the API opcode = 0x07 to CPUSS_SYSREQ.COMMAND
        //register and assert the sysreq bit
        CPUSS_SYSREQ_REG = 0x80000007;

        //Execute user code until iStatusInt equals 3 to signify
        //3 SPC interrupts have happened. This should be 1 in case
        // of non-blocking program System Call
        while( iStatusInt != 0x03 )
        {
            DoOtherUserStuff();
        }

        //Get the success or failure status of System Call
        syscall_status = CPUSS_SYSARG_REG;
    }
```

In the code, the CM0 exception table is configured to be in SRAM by writing 0x01 to the CPUSS_CONFIG register. The SRAM exception table should have the vector address of the SPC interrupt as the address of the *SpcIntHandler()* function, which is also defined to be in SRAM. See the Interrupts chapter on page 28 for details on configuring the CM0 exception table to be in SRAM. The pseudo code for a non-blocking program system call is also similar, except that the function opcode and parameters will differ and the iStatusInt variable should be polled for 1 instead of 3. This is because the SPC ISR will be triggered only once for a non-blocking program system call.

Glossary

The Glossary section explains the terminology used in this technical reference manual. Glossary terms are characterized in ***bold, italic font*** throughout the text of this manual.

A

accumulator	In a CPU, a register in which intermediate results are stored. Without an accumulator, it is necessary to write the result of each calculation (addition, subtraction, shift, and so on.) to main memory and read them back. Access to main memory is slower than access to the accumulator, which usually has direct paths to and from the arithmetic and logic unit (ALU).
active high	1. A logic signal having its asserted state as the logic 1 state. 2. A logic signal having the logic 1 state as the higher voltage of the two states.
active low	1. A logic signal having its asserted state as the logic 0 state. 2. A logic signal having its logic 1 state as the lower voltage of the two states: inverted logic.
address	The label or number identifying the memory location (RAM, ROM, or register) where a unit of information is stored.
algorithm	A procedure for solving a mathematical problem in a finite number of steps that frequently involve repetition of an operation.
ambient temperature	The temperature of the air in a designated area, particularly the area surrounding the device.
analog	See ***analog signals***.
analog blocks	The basic programmable opamp circuits. These are SC (switched capacitor) and CT (continuous time) blocks. These blocks can be interconnected to provide ADCs, DACs, multi-pole filters, gain stages, and much more.
analog output	An output that is capable of driving any voltage between the supply rails, instead of just a logic 1 or logic 0.
analog signals	A signal represented in a continuous form with respect to continuous times, as contrasted with a digital signal represented in a discrete (discontinuous) form in a sequence of time.
analog-to-digital (ADC)	A device that changes an analog signal to a digital signal of corresponding magnitude. Typically, an ADC converts a voltage to a digital number. The *digital-to-analog (DAC)* converter performs the reverse operation.
AND	See *Boolean Algebra*.

API (Application Programming Interface) A series of software routines that comprise an interface between a computer application and lower-level services and functions (for example, user modules and libraries). APIs serve as building blocks for programmers that create software applications.

array An array, also known as a vector or list, is one of the simplest data structures in computer programming. Arrays hold a fixed number of equally-sized data elements, generally of the same data type. Individual elements are accessed by index using a consecutive range of integers, as opposed to an associative array. Most high-level programming languages have arrays as a built-in data type. Some arrays are multi-dimensional, meaning they are indexed by a fixed number of integers; for example, by a group of two integers. One- and two-dimensional arrays are the most common. Also, an array can be a group of capacitors or resistors connected in some common form.

assembly A symbolic representation of the machine language of a specific processor. Assembly language is converted to machine code by an assembler. Usually, each line of assembly code produces one machine instruction, though the use of macros is common. Assembly languages are considered low-level languages; where as C is considered a high-level language.

asynchronous A signal whose data is acknowledged or acted upon immediately, irrespective of any clock signal.

attenuation The decrease in intensity of a signal as a result of absorption of energy and of scattering out of the path to the detector, but not including the reduction due to geometric spreading. Attenuation is usually expressed in dB.

B

bandgap reference A stable voltage reference design that matches the positive temperature coefficient of V_T with the negative temperature coefficient of V_{BE}, to produce a zero temperature coefficient (ideally) reference.

bandwidth

1. The frequency range of a message or information processing system measured in hertz.
2. The width of the spectral region over which an amplifier (or absorber) has substantial gain (or loss); it is sometimes represented more specifically as, for example, full width at half maximum.

bias

1. A systematic deviation of a value from a reference value.
2. The amount by which the average of a set of values departs from a reference value.
3. The electrical, mechanical, magnetic, or other force (field) applied to a device to establish a reference level to operate the device.

bias current The constant low-level DC current that is used to produce a stable operation in amplifiers. This current can sometimes be changed to alter the bandwidth of an amplifier.

binary The name for the base 2 numbering system. The most common numbering system is the base 10 numbering system. The base of a numbering system indicates the number of values that may exist for a particular positioning within a number for that system. For example, in base 2, binary, each position may have one of two values (0 or 1). In the base 10, decimal, numbering system, each position may have one of ten values (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).

bit A single digit of a binary number. Therefore, a bit may only have a value of ‘0’ or ‘1’. A group of 8 bits is called a byte. Because the CCG7D's M8CP is an 8-bit microcontroller, the CCG7D devices's native data chunk size is a byte.

bit rate (BR) The number of bits occurring per unit of time in a bit stream, usually expressed in bits per second (bps).

block
1. A functional unit that performs a single function, such as an oscillator.
2. A functional unit that may be configured to perform one of several functions, such as a digital CCG7D block or an analog CCG7D block.

Boolean Algebra In mathematics and computer science, Boolean algebras or Boolean lattices, are algebraic structures which "capture the essence" of the logical operations AND, OR and NOT as well as the set theoretic operations union, intersection, and complement. Boolean algebra also defines a set of theorems that describe how Boolean equations can be manipulated. For example, these theorems are used to simplify Boolean equations, which will reduce the number of logic elements needed to implement the equation.

The operators of Boolean algebra may be represented in various ways. Often they are simply written as AND, OR, and NOT. In describing circuits, NAND (NOT AND), NOR (NOT OR), XNOR (exclusive NOT OR), and XOR (exclusive OR) may also be used. Mathematicians often use + (for example, A+B) for OR and • for AND (for example, A*B) (in some ways those operations are analogous to addition and multiplication in other algebraic structures) and represent NOT by a line drawn above the expression being negated (for example, ~A, A_, !A).

break-before-make The elements involved go through a disconnected state entering (‘break”) before the new connected state (“make”).

broadcast net A signal that is routed throughout the microcontroller and is accessible by many blocks or systems.

buffer
1. A storage area for data that is used to compensate for a speed difference, when transferring data from one device to another. Usually refers to an area reserved for I/O operations, into which data is read, or from which data is written.
2. A portion of memory set aside to store data, often before it is sent to an external device or as it is received from an external device.
3. An amplifier used to lower the output impedance of a system.

bus
1. A named connection of nets. Bundling nets together in a bus makes it easier to route nets with similar routing patterns.
2. A set of signals performing a common function and carrying similar data. Typically represented using vector notation; for example, address[7:0].
3. One or more conductors that serve as a common connection for a group of related devices.

byte A digital storage unit consisting of 8 bits.

C

C A high-level programming language.

capacitance	A measure of the ability of two adjacent conductors, separated by an insulator, to hold a charge when a voltage differential is applied between them. Capacitance is measured in units of Farads.
capture	To extract information automatically through the use of software or hardware, as opposed to hand-entering of data into a computer file.
chaining	Connecting two or more 8-bit digital blocks to form 16-, 24-, and even 32-bit functions. Chaining allows certain signals such as Compare, Carry, Enable, Capture, and Gate to be produced from one block to another.
checksum	The checksum of a set of data is generated by adding the value of each data word to a sum. The actual checksum can simply be the result sum or a value that must be added to the sum to generate a predetermined value.
clear	To force a bit/register to a value of logic ‘0’.
clock	The device that generates a periodic signal with a fixed frequency and duty cycle. A clock is sometimes used to synchronize different logic blocks.
clock generator	A circuit that is used to generate a clock signal.
CMOS	The logic gates constructed using *MOS* transistors connected in a complementary manner. CMOS is an acronym for complementary metal-oxide semiconductor.
comparator	An electronic circuit that produces an output voltage or current whenever two input levels simultaneously satisfy predetermined amplitude requirements.
compiler	A program that translates a high-level language, such as C, into machine language.
configuration	In a computer system, an arrangement of functional units according to their nature, number, and chief characteristics. Configuration pertains to hardware, software, firmware, and documentation. The configuration will affect system performance.
configuration space	In CCG7D devices, the register space accessed when the XIO bit, in the CPU_F register, is set to ‘1’.
crowbar	A type of over-voltage protection that rapidly places a low-resistance shunt (typically an SCR) from the signal to one of the power supply rails, when the output voltage exceeds a predetermined value.
CPUSS	CPU subsystem
crystal oscillator	An oscillator in which the frequency is controlled by a piezoelectric crystal. Typically a piezoelectric crystal is less sensitive to ambient temperature than other circuit components.
cyclic redundancy check (CRC)	A calculation used to detect errors in data communications, typically performed using a linear feedback shift register. Similar calculations may be used for a variety of other purposes such as data compression.

D

data bus	A bi-directional set of signals used by a computer to convey information from a memory location to the central processing unit and vice versa. More generally, a set of signals used to convey data between digital functions.
data stream	A sequence of digitally encoded signals used to represent information in transmission.
data transmission	Sending data from one place to another by means of signals over a channel.
debugger	A hardware and software system that allows the user to analyze the operation of the system under development. A debugger usually allows the developer to step through the firmware one step at a time, set break points, and analyze memory.
dead band	A period of time when neither of two or more signals are in their active state or in transition.
decimal	A base-10 numbering system, which uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 (called digits) together with the decimal point and the sign symbols + (plus) and – (minus) to represent numbers.
default value	Pertaining to the predefined initial, original, or specific setting, condition, value, or action a system will assume, use, or take in the absence of instructions from the user.
device	The device referred to in this manual is the CCG7D device, unless otherwise specified.
die	An non-packaged integrated circuit (IC), normally cut from a wafer.
digital	A signal or function, the amplitude of which is characterized by one of two discrete values: ‘0’ or ‘1’.
digital blocks	The 8-bit logic blocks that can act as a counter, timer, serial receiver, serial transmitter, CRC generator, pseudo-random number generator, or SPI.
digital logic	A methodology for dealing with expressions containing two-state variables that describe the behavior of a circuit or system.
digital-to-analog (DAC)	A device that changes a digital signal to an analog signal of corresponding magnitude. The *analog-to-digital (ADC)* converter performs the reverse operation.
direct access	The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative positions by means of addresses that indicate the physical location of the data.
duty cycle	The relationship of a clock period *high time* to its *low time*, expressed as a percent.

E

External Reset (XRES_N) An active high signal that is driven into the CCG7D device. It causes all operation of the CPU and blocks to stop and return to a predefined state.

F

falling edge A transition from a logic 1 to a logic 0. Also known as a negative edge.

feedback The return of a portion of the output, or processed portion of the output, of a (usually active) device to the input.

filter A device or process by which certain frequency components of a signal are attenuated.

firmware The software that is embedded in a hardware device and executed by the CPU. The software may be executed by the end user, but it may not be modified.

flag Any of various types of indicators used for identification of a condition or event (for example, a character that signals the termination of a transmission).

Flash An electrically programmable and erasable, *volatile* technology that provides users with the programmability and data storage of EPROMs, plus in-system erasability. Nonvolatile means that the data is retained when power is off.

Flash bank A group of flash ROM blocks where flash block numbers always begin with ‘0’ in an individual flash bank. A flash bank also has its own block level protection information.

Flash block The smallest amount of flash ROM space that may be programmed at one time and the smallest amount of flash space that may be protected. A flash block holds 256 bytes. A flash block is also called as a flash row.

flip-flop A device having two stable states and two input terminals (or types of input signals) each of which corresponds with one of the two states. The circuit remains in either state until it is made to change to the other state by application of the corresponding signal.

frequency The number of cycles or events per unit of time, for a periodic function.

G

gain The ratio of output current, voltage, or power to input current, voltage, or power, respectively. Gain is usually expressed in dB.

gate

1. A device having one output channel and one or more input channels, such that the output channel state is completely determined by the input channel states, except during switching transients.
2. One of many types of combinational logic elements having at least two inputs (for example, AND, OR, NAND, and NOR (also see *Boolean Algebra*)).

ground

1. The electrical neutral line having the same potential as the surrounding earth.
2. The negative side of DC power supply.
3. The reference point for an electrical system.
4. The conducting paths between an electric circuit or equipment and the earth, or some conducting body serving in place of the earth.

H

hardware

A comprehensive term for all of the physical parts of a computer or embedded system, as distinguished from the data it contains or operates on, and the software that provides instructions for the hardware to accomplish tasks.

hardware reset

A reset that is caused by a circuit, such as a POR, watchdog reset, or external reset. A hardware reset restores the state of the device as it was when it was first powered up. Therefore, all registers are set to the POR value as indicated in register tables throughout this document.

hexadecimal

A base 16 numeral system (often abbreviated and called hex), usually written using the symbols 0-9 and A-F. It is a useful system in computers because there is an easy mapping from four bits to a single hex digit. Thus, one can represent every byte as two consecutive hexadecimal digits. Compare the binary, hex, and decimal representations:

bin = hex = dec

0000b = 0x0 = 0

0001b = 0x1 = 1

0010b = 0x2 = 2

...

1001b = 0x9 = 9

1010b = 0xA = 10

1011b = 0xB = 11

...

1111b = 0xF = 15

So the decimal numeral 79 whose binary representation is 0100 1111b can be written as 4Fh in hexadecimal (0x4F).

high time

The amount of time the signal has a value of ‘1’ in one period, for a periodic digital signal.

I

I²C	A two-wire serial computer bus by Phillips Semiconductors (now NXP Semiconductors). I²C is an Inter-Integrated Circuit. It is used to connect low-speed peripherals in an embedded system. The original system was created in the early 1980s as a battery control interface, but it was later used as a simple internal bus system for building control electronics. I²C uses only two bidirectional pins, clock and data, both running at +5 V and pulled high with resistors. The bus operates at 100 Kbps in standard mode and 400 Kbps in fast mode.
idle state	A condition that exists whenever user messages are not being transmitted, but the service is immediately available for use.
impedance	1. The resistance to the flow of current caused by resistive, capacitive, or inductive devices in a circuit. 2. The total passive opposition offered to the flow of electric current. Note the impedance is determined by the particular combination of resistance, inductive reactance, and capacitive reactance in a given circuit.
input	A point that accepts data, in a device, process, or channel.
input/output (I/O)	A device that introduces data into or extracts data from a system.
instruction	An expression that specifies one operation and identifies its operands, if any, in a programming language such as C or assembly.
instruction mnemonics	A set of acronyms that represent the opcodes for each of the assembly-language instructions, for example, ADD, SUBB, MOV.
integrated circuit (IC)	A device in which components such as resistors, capacitors, diodes, and *transistors* are formed on the surface of a single piece of semiconductor.
interface	The means by which two systems or devices are connected and interact with each other.
interrupt	A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.
interrupt service routine (ISR)	A block of code that normal code execution is diverted to when the M8CP receives a hardware interrupt. Many interrupt sources may each exist with its own priority and individual ISR code block. Each ISR code block ends with the RETI instruction, returning the device to the point in the program where it left normal program execution.

J

jitter	1. A misplacement of the timing of a transition from its ideal position. A typical form of corruption that occurs on serial data streams. 2. The abrupt and unwanted variations of one or more signal characteristics, such as the interval between successive pulses, the amplitude of successive cycles, or the frequency or phase of successive cycles.

L

latency	The time or delay that it takes for a signal to pass through a given circuit or network.
least significant bit (LSb)	The binary digit, or bit, in a binary number that represents the least significant value (typically the right-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in LSb.
least significant byte (LSB)	The byte in a multi-byte word that represents the least significant values (typically the right-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in LSB.
Linear Feedback Shift Register (LFSR)	A shift register whose data input is generated as an *XOR* of two or more elements in the register chain.
load	The electrical demand of a process expressed as power (watts), current (amps), or resistance (ohms).
logic function	A mathematical function that performs a digital operation on digital data and returns a digital value.
lookup table (LUT)	A logic block that implements several logic functions. The logic function is selected by means of select lines and is applied to the inputs of the block. For example: A 2 input LUT with 4 select lines can be used to perform any one of 16 logic functions on the two inputs resulting in a single logic output. The LUT is a combinational device; therefore, the input/output relationship is continuous, that is, not sampled.
low time	The amount of time the signal has a value of ‘0’ in one period, for a periodic digital signal.
low-voltage detect (LVD)	A circuit that senses VDDD and provides an interrupt to the system when VDDD falls below a selected threshold.

M

M8CP	An 8-bit Harvard Architecture microprocessor. The microprocessor coordinates all activity inside a CCG7D device by interfacing to the flash, SRAM, and register space.
macro	A programming language macro is an abstraction, whereby a certain textual pattern is replaced according to a defined set of rules. The interpreter or compiler automatically replaces the macro instance with the macro contents when an instance of the macro is encountered. Therefore, if a macro is used five times and the macro definition required 10 bytes of code space, 50 bytes of code space will be needed in total.
mask	1. To obscure, hide, or otherwise prevent information from being derived from a signal. It is usually the result of interaction with another signal, such as noise, static, jamming, or other forms of interference. 2. A pattern of bits that can be used to retain or suppress segments of another pattern of bits, in computing and data processing systems.

master device A device that controls the timing for data exchanges between two devices. Or when devices are cascaded in width, the master device is the one that controls the timing for data exchanges between the cascaded devices and an external interface. The controlled device is called the *slave device*.

microcontroller An integrated circuit device that is designed primarily for control systems and products. In addition to a CPU, a microcontroller typically includes memory, timing circuits, and I/O circuitry. The reason for this is to permit the realization of a controller with a minimal quantity of devices, thus achieving maximal possible miniaturization. This in turn, will reduce the volume and the cost of the controller. The microcontroller is normally not used for general-purpose computation as is a microprocessor.

mnemonic A tool intended to assist the memory. Mnemonics rely on not only repetition to remember facts, but also on creating associations between easy-to-remember constructs and lists of data. A two to four character string representing a microprocessor instruction.

mode A distinct method of operation for software or hardware. For example, the Digital CCG7D block may be in either counter mode or timer mode.

modulation A range of techniques for encoding information on a carrier signal, typically a sine-wave signal. A device that performs modulation is known as a modulator.

Modulator A device that imposes a signal on a carrier.

MOS An acronym for metal-oxide semiconductor.

most significant bit (MSb) The binary digit, or bit, in a binary number that represents the most significant value (typically the left-hand bit). The bit versus byte distinction is made by using a lower case “b” for bit in MSb.

most significant byte (MSB) The byte in a multi-byte word that represents the most significant values (typically the left-hand byte). The byte versus bit distinction is made by using an upper case “B” for byte in MSB.

multiplexer (mux)

1. A logic function that uses a binary value, or address, to select between a number of inputs and conveys the data from the selected input to the output.
2. A technique which allows different input (or output) signals to use the same lines at different times, controlled by an external signal. Multiplexing is used to save on wiring and I/O ports.

N

NAND See *Boolean Algebra*.

negative edge A transition from a logic 1 to a logic 0. Also known as a falling edge.

net The routing between devices.

nibble A group of four bits, which is one-half of a byte.

noise

1. A disturbance that affects a signal and that may distort the information carried by the signal.
2. The random variations of one or more characteristics of any entity such as voltage, current, or data.

NOR	See *Boolean Algebra*.
NOT	See *Boolean Algebra*.

O

OR	See *Boolean Algebra*.
oscillator	A circuit that may be crystal controlled and is used to generate a clock frequency.
output	The electrical signal or signals which are produced by an analog or digital block.

P

parallel	The means of communication in which digital data is sent multiple bits at a time, with each simultaneous bit being sent over a separate line.
parameter	Characteristics for a given block that have either been characterized or may be defined by the designer.
parameter block	A location in memory where parameters for the SSC instruction are placed prior to execution.
parity	A technique for testing transmitting data. Typically, a binary digit is added to the data to make the sum of all the digits of the binary data either always even (even parity) or always odd (odd parity).
path	1. The logical sequence of instructions executed by a computer. 2. The flow of an electrical signal through a circuit.
pending interrupts	An interrupt that is triggered but not serviced, either because the processor is busy servicing another interrupt or global interrupts are disabled.
phase	The relationship between two signals, usually the same frequency, that determines the delay between them. This delay between signals is either measured by time or angle (degrees).
pin	A terminal on a hardware component. Also called lead.
pinouts	The pin number assignment: the relation between the logical inputs and outputs of the CCG7D device and their physical counterparts in the printed circuit board (PCB) package. Pinouts will involve pin numbers as a link between schematic and PCB design (both being computer generated files) and may also involve pin names.
port	A group of pins, usually eight.
positive edge	A transition from a logic 0 to a logic 1. Also known as a rising edge.
posted interrupts	An interrupt that is detected by the hardware but may or may not be enabled by its mask bit. Posted interrupts that are not masked become pending interrupts.

Power On Reset (POR)	A circuit that forces the CCG7D device to reset when the voltage is below a preset level. This is one type of *hardware reset*.
program counter	The instruction pointer (also called the program counter) is a register in a computer processor that indicates where in memory the CPU is executing instructions. Depending on the details of the particular machine, it holds either the address of the instruction being executed, or the address of the next instruction to be executed.
protocol	A set of rules. Particularly the rules that govern networked communications.
pulse	A rapid change in some characteristic of a signal (for example, phase or frequency), from a base-line value to a higher or lower value, followed by a rapid return to the baseline value.
pulse width modulator (PWM)	An output in the form of duty cycle which varies as a function of the applied measure.

R

RAM	An acronym for random access memory. A data-storage device from which data can be read out and new data can be written in.
register	A storage device with a specific capacity, such as a bit or byte.
reset	A means of bringing a system back to a know state. See *hardware reset* and *software reset*.
resistance	The resistance to the flow of electric current measured in ohms for a conductor.
revision ID	A unique identifier of the CCG7D device.
ripple divider	An asynchronous ripple counter constructed of flip-flops. The clock is fed to the first stage of the counter. An n-bit binary counter consisting of n flip-flops that can count in binary from 0 to $2^n - 1$.
rising edge	*See positive edge*.
ROM	An acronym for read only memory. A data-storage device from which data can be read out, but new data cannot be written in.
routine	A block of code, called by another block of code, that may have some general or frequent use.
routing	Physically connecting objects in a design according to design rules set in the reference library.
runt pulses	In digital circuits, narrow pulses that, due to non-zero rise and fall times of the signal, do not reach a valid high or low level. For example, a runt pulse may occur when switching between asynchronous clocks or as the result of a race condition in which a signal takes two separate paths through a circuit. These race conditions may have different delays and are then recombined to form a glitch or when the output of a flip-flop becomes metastable.

S

sampling	The process of converting an analog signal into a series of digital values or reversed.
schematic	A diagram, drawing, or sketch that details the elements of a system, such as the elements of an electrical circuit or the elements of a logic diagram for a computer.
seed value	An initial value loaded into a linear feedback shift register or random number generator.
serial	1. Pertaining to a process in which all events occur one after the other. 2. Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel.
set	To force a bit/register to a value of logic 1.
settling time	The time it takes for an output signal or value to stabilize after the input has changed from one value to another.
shift	The movement of each bit in a word one position to either the left or right. For example, if the hex value 0x24 is shifted one place to the left, it becomes 0x48. If the hex value 0x24 is shifted one place to the right, it becomes 0x12.
shift register	A memory storage device that sequentially shifts a word either left or right to output a stream of serial data.
sign bit	The most significant binary digit, or bit, of a signed binary number. If set to a logic 1, this bit represents a negative quantity.
signal	A detectable transmitted energy that can be used to carry information. As applied to electronics, any transmitted electrical impulse.
silicon ID	A unique identifier of the CCG7D silicon.
skew	The difference in arrival time of bits transmitted at the same time, in parallel transmission.
slave device	A device that allows another device to control the timing for data exchanges between two devices. Or when devices are cascaded in width, the slave device is the one that allows another device to control the timing of data exchanges between the cascaded devices and an external interface. The controlling device is called the master device.
software	A set of computer programs, procedures, and associated documentation about the operation of a data processing system (for example, compilers, library routines, manuals, and circuit diagrams). Software is often written first as source code, and then converted to a binary format that is specific to the device on which the code will be executed.
software reset	A partial reset executed by software to bring part of the system back to a known state. A software reset will restore the M8CP to a know state but not CCG7D blocks, systems, peripherals, or registers. For a software reset, the CPU registers (CPU_A, CPU_F, CPU_PC, CPU_SP, and CPU_X) are set to 0x00. Therefore, code execution will begin at flash address 0x0000.

SRAM An acronym for static random access memory. A memory device allowing users to store and retrieve data at a high rate of speed. The term static is used because, when a value is loaded into an SRAM cell, it will remain unchanged until it is explicitly altered or until power is removed from the device.

SROM An acronym for supervisory read only memory. The SROM holds code that is used to boot the device, calibrate circuitry, and perform flash operations. The functions of the SROM may be accessed in normal user code, operating from flash.

stack A stack is a data structure that works on the principle of Last In First Out (LIFO). This means that the last item put on the stack is the first item that can be taken off.

stack pointer A stack may be represented in a computer's inside blocks of memory cells, with the bottom at a fixed location and a variable stack pointer to the current top cell.

state machine The actual implementation (in hardware or software) of a function that can be considered to consist of a set of states through which it sequences.

sticky A bit in a register that maintains its value past the time of the event that caused its transition, has passed.

stop bit A signal following a character or block that prepares the receiving device to receive the next character or block.

switching The controlling or routing of signals in circuits to execute logical or arithmetic operations, or to transmit data between specific points in a network.

switch phasing The clock that controls a given switch, PHI1 or PHI2, in respect to the switch capacitor (SC) blocks. The CCG7D SC blocks have two groups of switches. One group of these switches is normally closed during PHI1 and open during PHI2. The other group is open during PHI1 and closed during PHI2. These switches can be controlled in the normal operation, or in reverse mode if the PHI1 and PHI2 clocks are reversed.

synchronous

1. A signal whose data is not acknowledged or acted upon until the next active edge of a clock signal.
2. A system whose operation is synchronized by a clock signal.

T

tap The connection between two blocks of a device created by connecting several blocks/components in a series, such as a shift register or resistive voltage divider.

terminal count The state at which a counter is counted down to zero.

threshold The minimum value of a signal that can be detected by the system or sensor under consideration.

Thumb-2 The Thumb-2 instruction set is a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size, and performance. The Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions.

transistors The transistor is a solid-state semiconductor device used for amplification and switching, and has three terminals: a small current or voltage applied to one terminal controls the current through the other two. It is the key component in all modern electronics. In digital circuits, transistors are used as very fast electrical switches, and arrangements of transistors can function as logic gates, RAM-type memory, and other devices. In analog circuits, transistors are essentially used as amplifiers.

tristate A function whose output can adopt three states: 0, 1, and Z (high impedance). The function does not drive any value in the Z state and, in many respects, may be considered to be disconnected from the rest of the circuit, allowing another output to drive the same *net*.

U

UART A UART or universal asynchronous receiver-transmitter translates between parallel bits of data and serial bits.

user The person using the CCG7D device and reading this manual.

user modules Prebuild, pretested hardware/firmware peripheral functions that take care of managing and configuring the lower level Analog and Digital CCG7D Blocks. User Modules also provide high level *API (Application Programming Interface)* for the peripheral function.

user space The bank 0 space of the register map. The registers in this bank are more likely to be modified during normal program execution and not just during initialization. Registers in bank 1 are most likely to be modified only during the initialization phase of the program.

V

VDDD A name for a power net meaning “voltage drain.” The most positive power supply signal. Usually 5 or 3.3 volts.

volatile Not guaranteed to stay the same value or level when not in scope.

W

watchdog timer	A timer that must be serviced periodically. If it is not serviced, the CPU will reset after a specified period of time.
waveform	The representation of a signal as a plot of amplitude versus time.

X

XOR	See *Boolean Algebra*.