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

**How to Check the Ordering Part Number**

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

**For More Information**

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

**About Cypress**

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

AN204728

# F²MC-8FX Family MB95310/370 Series 8-Bit Microcontroller I²C Source Code API

**Associated Part Family: MB95310/370 Series**

This document introduces API for I²C code.

## Contents


1 Introduction....1
2 MB95F310 I²C Register....1
2.1 IBCR00 and IBCR10 (I²C Bus Control Register)....1
2.2 IBSR0 (I²C Bus Status Register)....3
2.3 IDDR0 (I²C Data Register)....4
2.4 IAAR0 (I²C Address Register)....4
2.5 ICCR0 (I²C Clock Control Register)....5
3 Steps of Realizing I²C in Software....6
4 I²C Sample Code....7
5 Usage Demo....10
5.1 Register Use Attention....10
6 Additional Information....10
Document History....11


## 1 Introduction

This document introduces API for I²C code.

In MB95F310 MCU there is an I²C module which can transfer data to other I²C device synchronously. In following chapters we should describes the MB95F310 registers and codes in C and the compilation.

## 2 MB95F310 I²C Register

This chapter describes MB95F310 I²C register

There are six registers in MB95F310. Those registers are bus control register IBCR00, bus control register IBCR10, bus status register IBSR0, I²C data register IDDR0, I²C address register IAAR0, and clock control register ICCR0.

In following sections, these registers are described in detail.

### 2.1 IBCR00 and IBCR10 (I²C Bus Control Register)

The two registers are mainly used to select the operating mode and to enable or disable interrupts, acknowledgment, general call acknowledgment, and MCU standby wakeup function.

Figure 1 describes register IBCR00

Figure 1. IBCR00 Register

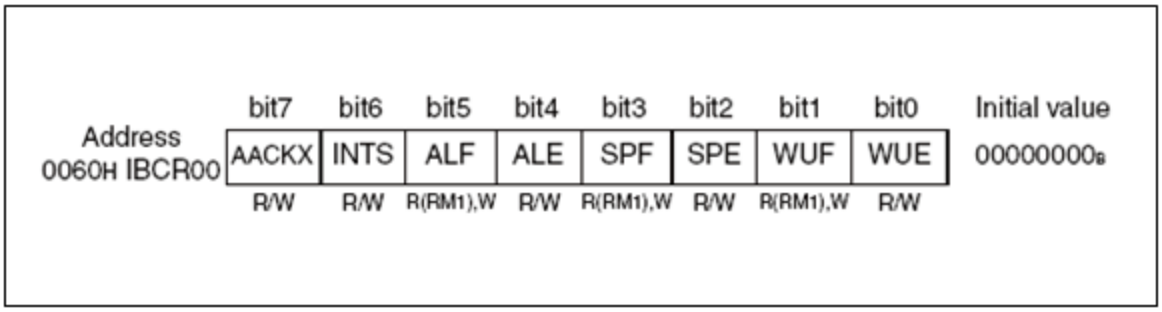



Table 1 describes every bit of IBCR00

Table 1. IBCR00 Register Description

| Register | Description | |
|---|---|---|
| AACKX | In slave mode, it sends “1” when received first data as acknowledge. | |
| INTS | Set transfer finish interrupt at ninth SCL cycle or eighth SCL cycle | 0: interrupt in ninth SCL cycle |
| | | 1: interrupt in eighth SCL cycle |
| ALF | When “1” is read, Arbitration lost is detected. Writing “0” will clear lost interrupt | |
| ALE | When “1” is written, it will enable Arbitration interrupt | |
| SPF | When “1” is read, stop condition is detected. Write “0” will clear stop interrupt | |
| SPE | When “1” is written, stop interrupt is enabled | |
| WUF | When “1” is read, start condition detected. Writing “0” will clear start interrupt. | |
| WUE | Writing “1” is enable and writing “0” is disable the function to wake up from standby mode | |

Figure 2 describes register IBCR10

Figure 2. IBCR10 Register

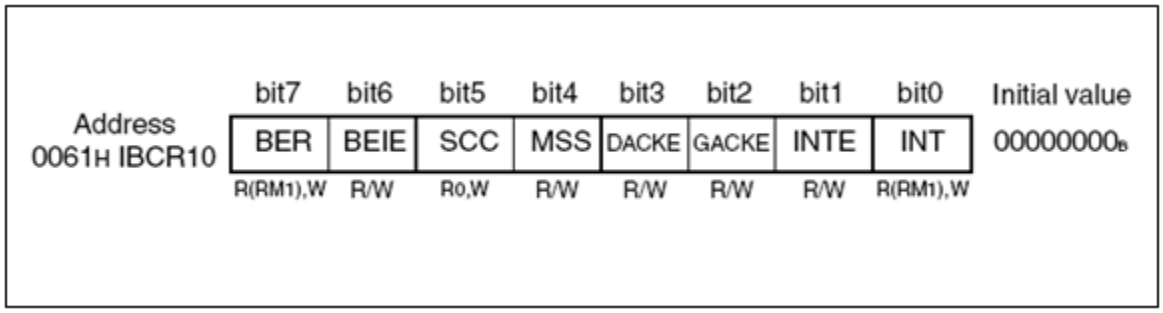


Table 2 describes every bit of IBCR10

Table 2. Description of IBCR10 Register

| Register | Description |
|---|---|
| BER | When “1” is read, bus error is detected. Writing “0” will clear flag |
| BEIE | When writing “1” to it, bus error interrupt is enabled |
| SCC | When “1” is written, start signal generated again |
| MSS | When “1” is written, becoming as master mode, generating start condition and starting address data transfer |
| | When “0” is written, generating stop condition and being transfer to slave mode |
| DACKE | When “1” is written, data acknowledge is enabled |
| GACKE | When “1” is written, call address acknowledge enabled |
| INTE | When “1” is written, data transfer completed interrupt is enabled |
| INT | Write “1” is read, 0ne-byte data transfer completed |

## 2.2 IBSR0 (I²C Bus Status Register)

This is a read only register. It is used to feedback the status of I²C bus.

Figure 3 describes register IBSR0

Figure 3. IBSR0 Register

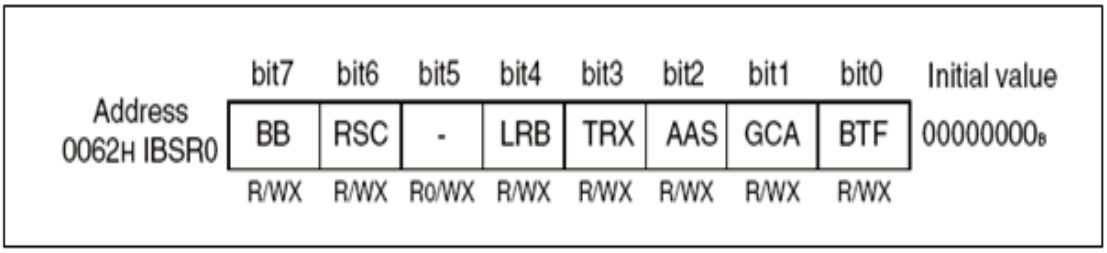


Table 3 describes every bit of IBSR0

Table 3. Description of IBSR0 Register

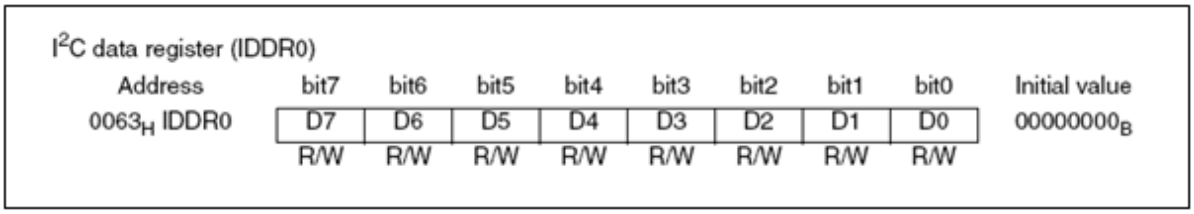
| Register | Description | |
|---|---|---|
| BB | When “1” is read, bus is busy or else bus is idle | |
| RSC | When “1” is read, repeated start condition detected | |
| - | Not used | |
| LRB | When “1” is read, there is no acknowledge from slave chip | |
| | When “0” is read, acknowledge is detected or a start or a stop condition is detected | |
| TRX | It is decided by the last bit of written address, and will reverse the value for read detect | When “0” is read, it is receive mode |
| | | When “1” is read, it is transmit mode |
| AAS | When “1” is read, MCU has addressed in slave mode | |
| | When “0” is read, a start or stop condition is detected | |
| GCA | When “1” is read, the general call address is received in slave mode | |
| FBT | When “1” is read, received data is the first address data | |

## 2.3 IDDR0 (I²C Data Register)

The IDDR0 register is used to set the data or address to send and to hold the data or address received.

Figure 4 describes register IDDR0.

Figure 4. IDDR0 Register

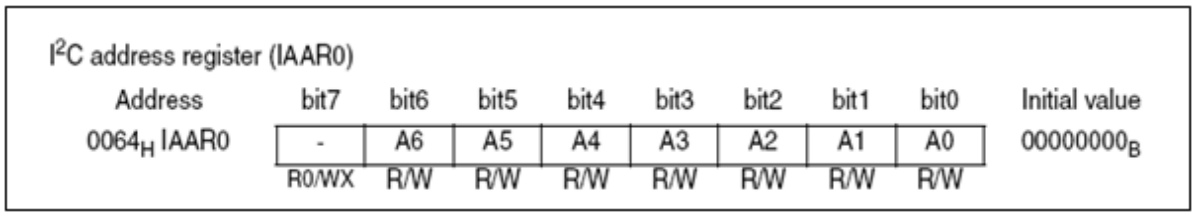



## 2.4 IAAR0 (I²C Address Register)

The IAAR0 register is used to set the slave address. The received address is compared with it when in slave mode.

Figure 5 describes register IAAR0

Figure 5. IAAR0 Register



## 2.5 ICCR0 (I²C Clock Control Register)

The ICCR0 register is used to enable I²C operation and select the shift clock frequency.

Figure 6 describes register ICCR0

Figure 6. ICCR0 Register

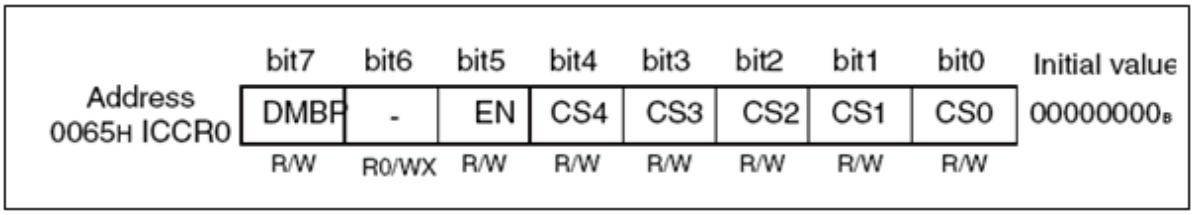



Table 4 describes each bit of ICCR0

Table 4. Description of ICCR0 Register

| Register | Description |
|---|---|
| DMBP | When “1” is written, divider m will be bypassed |
| EN | When “1” is written, I2C operation be enabled |
| CS4 | Divider m, reference Table 4 - 5 |
| CS3 | Divider m, reference Table 4 - 5 |
| CS2 | Divider n, reference Table 4 - 6 |
| CS1 | Divider n, reference Table 4 - 6 |
| CS0 | Divider n, reference Table 4 - 6 |

Table 5 describes divider m

Table 5. Description of Divider m

| CS4 | CS3 | Divider m |
|---|---|---|
| 0 | 0 | 5 |
| 0 | 1 | 6 |
| 1 | 0 | 7 |
| 1 | 1 | 8 |

Table 6 describes divider n

Table 6. Description of Divider n

| CS2 | CS1 | CS0 | Divider n |
|---|---|---|---|
| 0 | 0 | 0 | 4 |
| 0 | 0 | 1 | 8 |
| 0 | 1 | 0 | 22 |
| 0 | 1 | 1 | 38 |
| 1 | 0 | 0 | 98 |
| 1 | 0 | 1 | 128 |
| 1 | 1 | 0 | 256 |
| 1 | 1 | 1 | 512 |

# 3 Steps of Realizing I²C in Software

This chapter describes steps of how to realize I²C.

About I²C register they are divided to initialization register, status register and data register.

Following items will describe how to use these registers to realize I²C function.

- Step one is initializing I²C:
  IBCR00 register sets interrupt type
  IBCR10 register sets I²C mode and acknowledge generate condition
  ICCR0 register enables I²C and sets the shift frequency
- Step two is generating start signal
  IBSR0 register feeds back the I²C status
  IDDR0 register sends out the address of slave device, when set to master mode the data send out
- Step three is write data to slave device
  IBCR10 register sets condition to which I²C can set data
  IDDR0 register saves the data and sends out
- Step four is setting read condition to read data
  IBCR10 register feeds back the status which data is received
  IDDR0 register saves the data received
- Step five is setting condition to detect acknowledge
  IBSR0 register detects whether bus has received acknowledge
- Step fix is generating stop signal
  IBCR10 register sets the mode to stop
  IBSR0 register detects whether bus is released

# 4 I²C Sample Code

This chapter describes MB95F310 I²C C code.

Figure 7 describes the initial function.

Figure 7. Initial Code

```
void I2C_Init( void )
{
  ICCR0_EN  = 0;                  // clear I2C interface
  ICCR0_CS4 = 1;                  // set clock divider 'm' => 7
  ICCR0_CS3 = 0;
  ICCR0_CS2 = 1;                  // set clock divider 'n' => 98
  ICCR0_CS1 = 0;
  ICCR0_CS0 = 0;                  // Fsck = MCLK / (m * n +2) => 16MHz/(7*98 +2) = 16MHz/688 = 23kHz
  ICCR0_EN  = 1;                  // enable I2C interface
  IDDR0     = 000;          // clear data register
  IBCR00    = 0x04;         // enable address acknowledge bit,
  IBCR10    = 0x00;         // set to slave mode first, disable data acknowledge bit,
}
```

Figure 8 lists the start code.

Figure 8. Start Code

```
void I2C_Start( unsigned char slave_address )
{
 do
 {
  IBCR10_BER = 0;             // clear bus error interrupt flag
  IDDR0     = slave_address;   // slave_address is sent out with start condition
  IBCR10_MSS = 1;             // set to master mode now and send start condition
  IBCR10_INT = 0;            // clear transfer end interrupt flag
  while(IBCR10_INT == 0);       // look if transfer is in process
 }
 while (IBCR10_BER == 1 | IBCR00_ALF == 1);       // retry if Bus-Error detected or arbitration lost
 while(IBSR0_LRB == 1)          // no acknowledge means device not ready
 {                        // maybe last write cycle not ended yet
  IBCR10_SCC = 1;             // try restart (= continue)
  while (IBCR10_INT == 0);      // wait that transfer is finished
 }
}
```

Figure 9 lists the write code.

Figure 9. Write Code

```
void I2C_Write( unsigned char value )
{
 IDDR0  = value;              // load data or address in to register
 IBCR10_INT = 0;              // clear transfer end intrerupt flag
 while (IBCR10_INT == 0);     // look if transfer is in process
 I2C_Acknowlegde();           // wait for Acknowledge
}
```

Figure 10 lists the read code.

Figure 10. Read Code

```
unsigned char I2C_Read( void )
{
 IBCR10_DACKE = 1;            // acknowledge has to be send
 IBCR10_INT = 0;              // clear transfer end interrupt flag
 while (IBCR10_INT == 0);     // wait that transfer is finished
 return(IDDR0);               // read received data out
}
```

Figure 11 lists the acknowledge code.

Figure 11. Acknowledge Code

```
void I2C_Acknowlegde( void )
{
  while(IBSR0_LRB == 1);          // no anwser from slave, program stucks,
                    // here a timeout mechanism should be implemented
}
```

Figure 12 lists the stop code.

Figure 12. Stop Code

```
void I2C_Stop( void )
{
 while (IBCR10_INT == 0);     // wait that transfer is finished
 IBCR10_MSS = 0;              // change to slave and release stop condition
 IBCR10_INT = 0;              // clear transfer end interrupt flag
 while (IBSR0_BB);            // wait till bus free
}
```

Figure 13 describes the write function.

Figure 13. Write Function

```
Void Write_I2C_Proc(unsigned char MainAddr, unsigned char SubAddr, unsigned char I2Cdata)
{
        I2C_Init();
        I2C_Start(MainAddr);
        I2C_Acknowlegde();
        I2C_Write(SubAddr);
        I2C_Acknowlegde();
        I2C_Write(SubAddr);
        I2C_Acknowlegde();
        I2C_Stop();
}
```

Figure 14 describes the read function.

Figure 14. Read Function

```
unsigned char RD_I2C( unsigned char Main_Addr, unsigned char Sub_Addr )
{
        unsigned char Temp;

        I2C_Start( (Main_Addr ) | I2C_WRITE );
        I2C_Write( Sub_Addr );
        I2C_Continue( (Main_Addr ) | I2C_READ );
       Temp = I2C_Read();
        I2C_Stop();
        return (Temp);
}
```

# 5 Usage Demo

This chapter describes something we must pay attention to when we use register to realize $I^2C$.

## 5.1 Register Use Attention

- When beginning the $I^2C$, the $I^2C$ bus must be free,
- If the master register IBSR0_LRB is “1” the slave device did not receive the address or data.
- The read address is 1 bigger than write address.

# 6 Additional Information

For more information about how to use MB95310 EV-board, BGM Adaptor and SOFTUNE, please refer to EV-Board MB2146-450-E User Manual, or visit website:

http://www.cypress.com/documentation/application-notes/mb95310370-mb2146-450-e-lcd-evb-user-manual

## Document History

Document Title: AN204728 – F²MC-8FX Family MB95310/370 Series 8-Bit Microcontroller I²C Source Code API

Document Number: 002-04728

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | - | HUAL | 11/13/2009 | Original version |
| *A | 5235154 | HUAL | 04/26/2016 | Migrated Spansion Application Note MCU-AN- 500051-E-10 to Cypress format. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Lighting & Power Control | cypress.com/powerpsoc |
| Memory | cypress.com/memory |
| PSoC | cypress.com/psoc |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless/RF | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

## Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corporation. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2009-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.
TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.
Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.