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.

AN205072

F²MC-8FX Family MB95350L Series I2C Slave Module API

Associated Part Family: MB95350L Series

This Application Note introduces API for I²C slave module.

Contents

1 Introduction....1
2 Description of I2C Slave Protocol....1
2.1 I2C Slave Start....2
2.2 I2C Slave Address....2
2.3 I2C Slave Acknowledge....2
2.4 I2C Slave Data....3
2.5 I2C Slave Stop....3
3 MB95350L I2C Slave Registers....3
4 I2C Slave Library Functions List....4
5 I2C Slave Function Detail....5
5.1 SlaveI2C_PrepareForInter Function....5
5.2 INTER_I2CSlaveModeWakeUp Function....6
5.3 Write_StandBy Function....6
6 Usage Demo....7
6.1 Hardware Design....7
6.2 Steps for Add I2C Slave Library....8
6.3 I2C Slave Feed Back....10
7 Debug....11
8 I2C Slave Sample Code....13
9 Additional Information....15
Document History....16
Worldwide Sales and Design Support....17
Products....17
PSoC® Solutions....17
Cypress Developer Community....17
Technical Support....17

1 Introduction

This document introduces API for I²C slave module.

In MB95350L MCU there is an I²C slave module which can transfer data to/receive data from other I²C master device synchronously. In following chapters we should describes how to use the module and its' libraries.

2 Description of I²C Slave Protocol

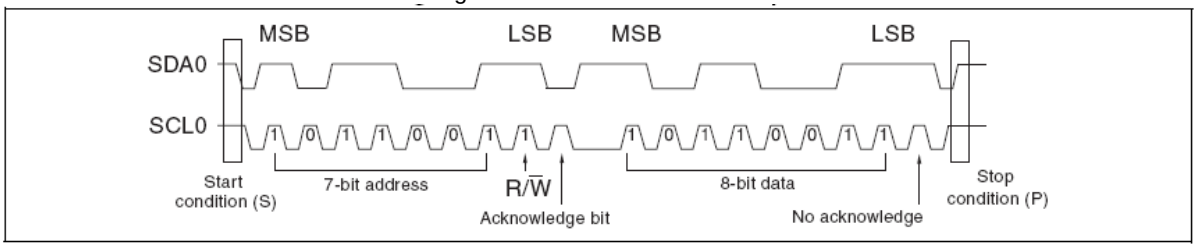
This chapter introduces protocol of I²C slave module.

I²C slave module is mainly used to receive data from I2C master module and feedback data to I²C master module. The I²C slave has SDA and SCL two lines.

I²C slave module use the same bus with I²C master. The slave SCL generates signal when I2C slave received I²C master call flag (slave address matches with master send address). The slave data register read data on SDA and give acknowledge when address matched or data received.

Following figure describes the I²C protocol.

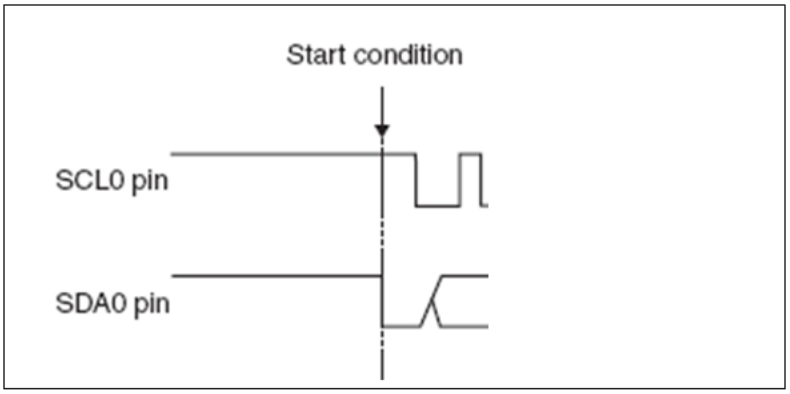
Figure 2-1. I²C Protocol



2.1 I²C Slave Start

When I²C slave detected start signal, I²C slave will begin to receive address from master. Following figure describes the start condition.

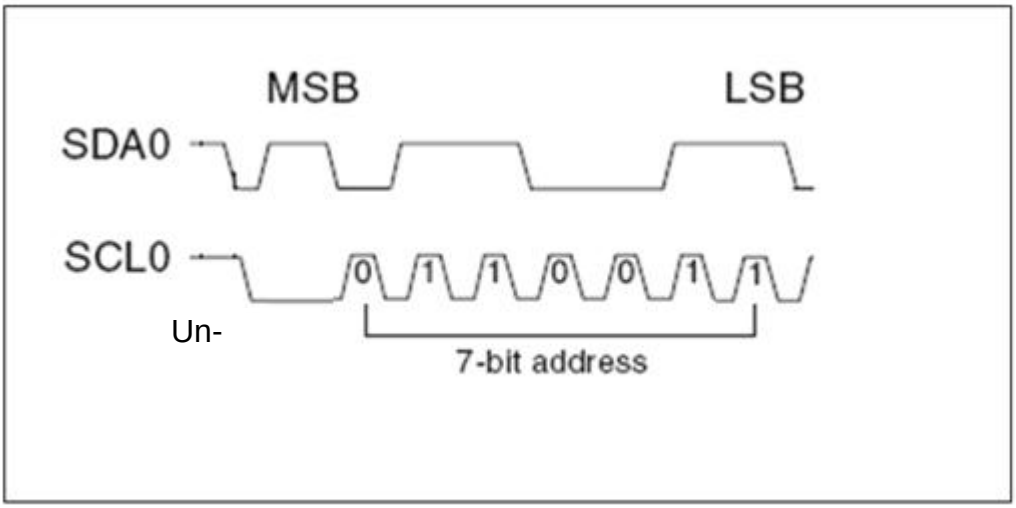
Figure 2-2. I²C Start Condition



2.2 I2C Slave Address

MB95350L has a register IAAR0 (IAAR1) which saved the MCU address. User can define the address by write data to this register. Following figure describes the address form.

Figure 2-3. I²C Slave Address

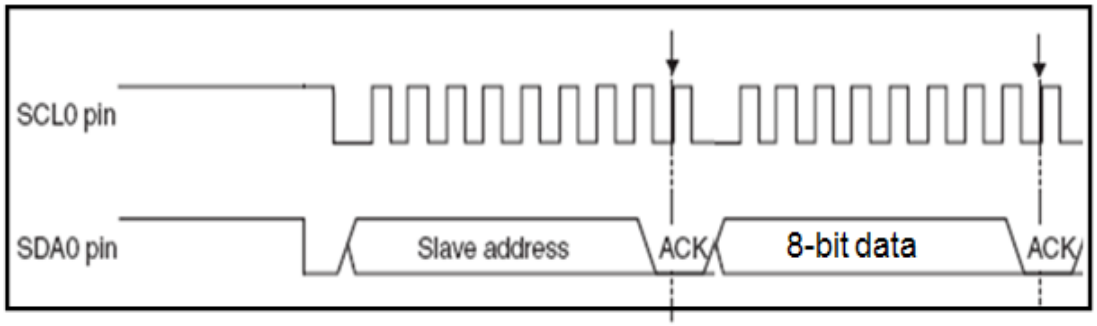


Note: Writing address to address register IAAR0 (IAAR1) only needs write the 7bit address and the bit7 will be ignored.

2.3 I²C Slave Acknowledge

There are two conditions which the I²C slave will generate acknowledge. One is the slave received the address which matches with the MCU address. Another is the salve finished 8-bit data receiving. Following figure describes the acknowledge form.

Figure 2-4. I²C Slave Acknowledge



2.4 I²C Slave Data

In MB95350L MCU the register IDDR0 (IDDR1) is used to save the data. The data is transmitted by SDA and when master transmit bit7 first, then the slave received the first data is bit7.

2.5 I²C Slave Stop

The I²C slave will stop the bus when detected stop signal which sent by master. So the slave device does not need generate stop, it just need detect the master stop signal.

3 MB95350L I²C Slave Registers

This chapter describes MB95350 I²C slave register.

There are two I²C channels in MB95350L and each channel can be operated singly. Each channel has sixth registers. Following table describes these registers.

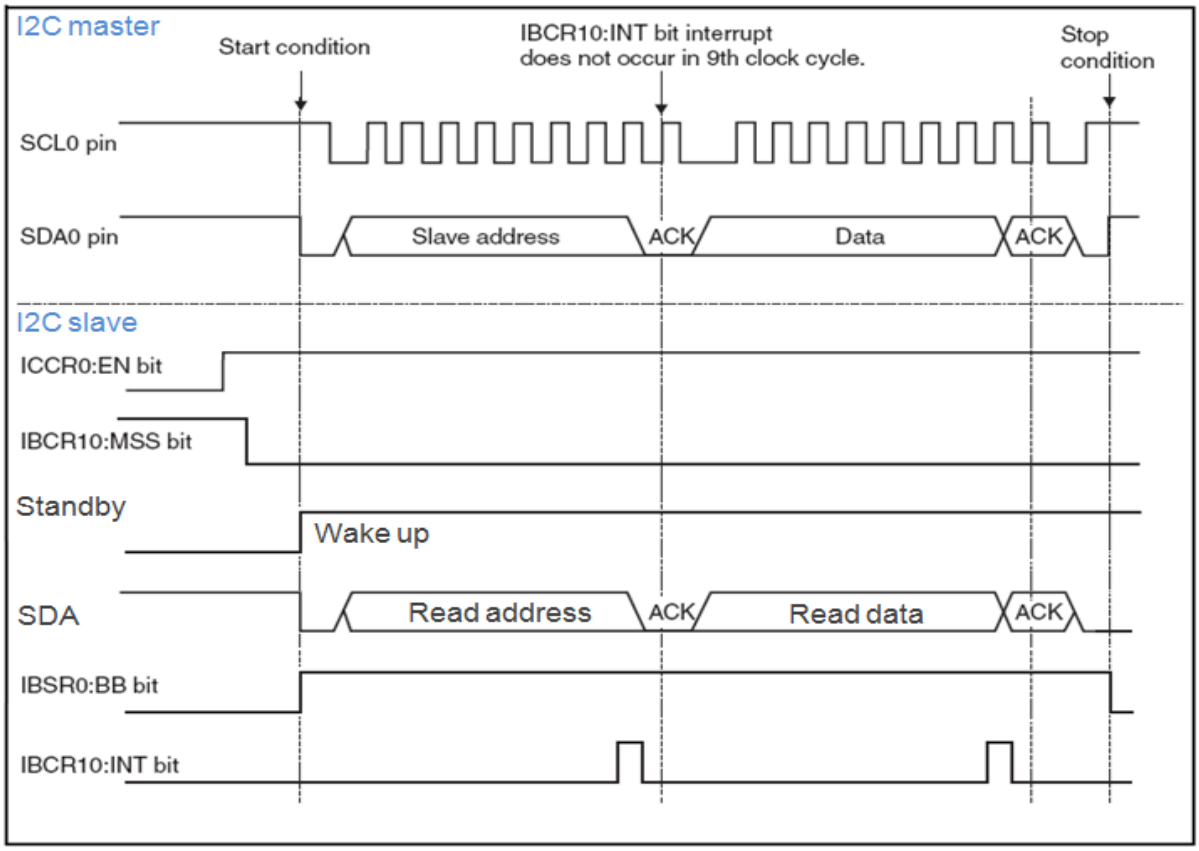
Table 3-1. I²C Registers

Register		Description
I²C Channel 0	IBCR00	Bus control register 0
	IBCR10	Bus control register 1
	IBSR0	Bus status register
	IDDR0	I2C data register
	IAAR0	I2C address register
	ICCR0	Clock control register
I²C Channel 1	IBCR01	Bus control register 0
	IBCR11	Bus control register 1
	IBSR1	Bus status register
	IDDR1	I2C data register
	IAAR1	I2C address register
	ICCR1	Clock control register

Note: the operation method of channel0 register is same as channel1

Following figure describes the work condition of I²C slave.

Figure 3-1. I²C Slave Work Condition



4 I²C Slave Library Functions List

This chapter introduces all functions in I²C slave library in project *I2C Slave.prj* which MCU is MB95350L.

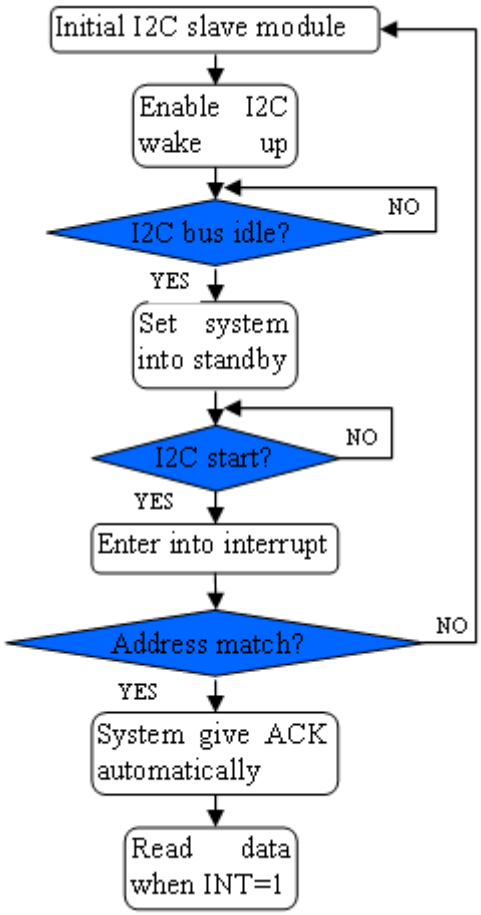
Table 4-1 lists the I²C slave functions.

Table 4-1. I²C Slave Functions

Function name	Description
void SlaveI2C_PrepareForInter(void)	Initializes I²C slave module and sets MCU address
__interrupt void INTER_I2CSlaveModeWakeUp(void)	Waking up interrupt to read the I²C SDA data
void Write_StandBy(void)	Sets system enter into standby

Following flow chart roughly describes the FW design.

Figure 4-1. FW Flow Chart



5 I²C Slave Function Detail

This chapter introduces the detail of I²C slave functions.

5.1 SlaveI²C_PrepareForInter Function

Table 5- describes Slave I2C_PrepareForInter function.

Table 5-1. SlaveI²C_PrepareForInter Function

Function name	SlaveI2C_PrepareForInter
Function prototype	Void SlaveI2C_PrepareForInter(void)
Behavior description	Initializes I2C slave module and start slave mode
Input parameter	None
Return value	None
Example	The library function sets I2C slave clock to 115K, MCU address to 0x60, enable I²C slave pin, enable transfer complete interrupt and enable acknowledge: SlaveI2C_PrepareForInter ();

5.2 INTER_I2CSlaveModeWakeUp Function

Table 5- describes INTER_I2CSlaveModeWakeUp function.

Table 5-2. INTER_I2CSlaveModeWakeUp Function

Function name	INTER_I2CSlaveModeWakeUp
Function prototype	__interrupt void INTER_I2CSlaveModeWakeUp(void)
Behavior description	Interrupt when start signal or 8-bit data transfer completely
Input parameter	None
Return value	None
Interrupt level	IRQ10: 8/16-bit PPG ch1 (upper) | I2C ch1 or IRQ16: I2C ch0
Example	Used in vectors file: __interrupt void INTER_I2CSlaveModeWakeUp(void);

5.3 Write_StandBy Function

Table 5- describes Write_StandBy function.

Table 5-3. Write_StandBy Function

Function name	Write_StandBy
Function prototype	void Write_StandBy(void)
Behavior description	Makes system enter into stop mode
Input parameter	None
Return value	None
Example	Used in main function: Write_StandBy();

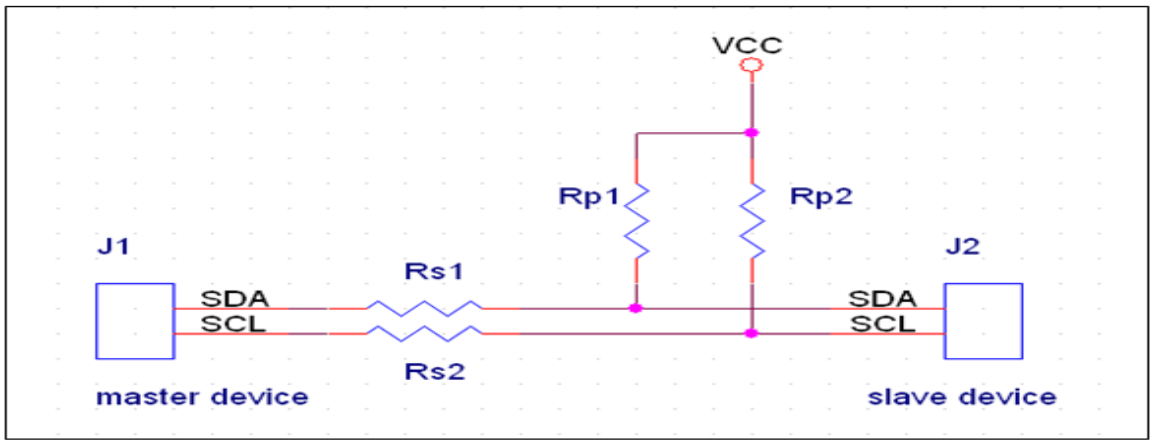
6 Usage Demo

This chapter describes something we must pay attention to when we use.

6.1 Hardware Design

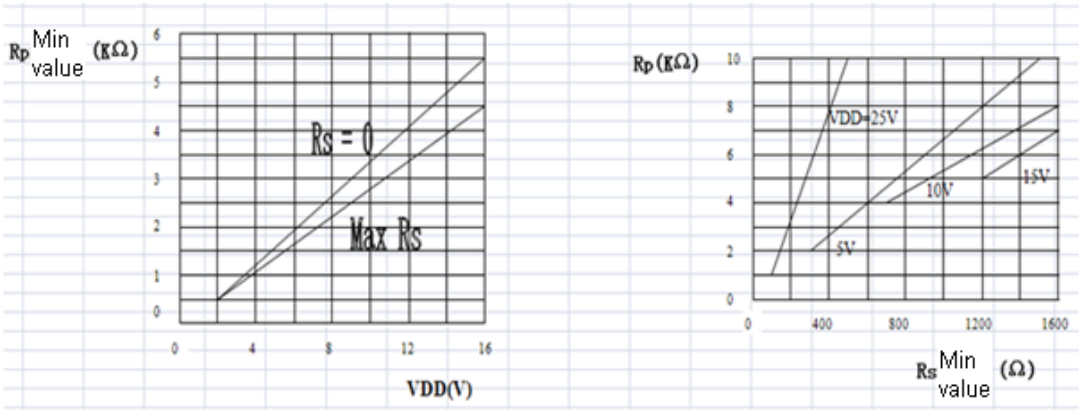
For hardware design the following figure may be referenced.

Figure 6-1. I²C Hardware Design



Following figure describes the R_S and R_P resistance definition

Figure 6-2. R_S and R_P Definition

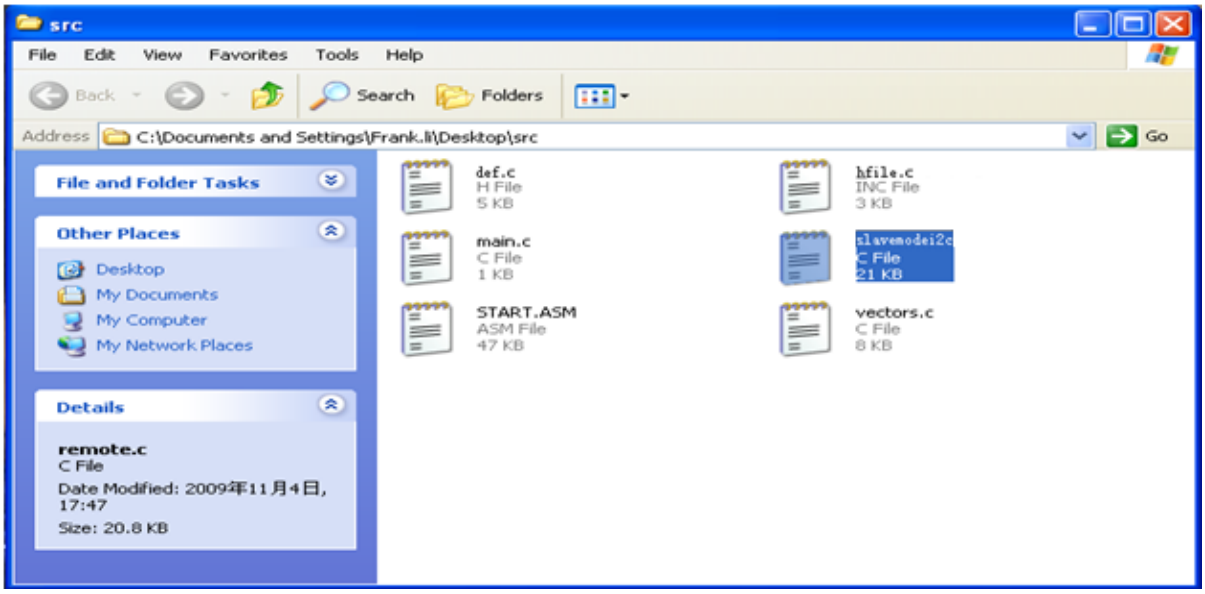


6.2 Steps for Add I²C Slave Library

When use this project please refer to following steps.

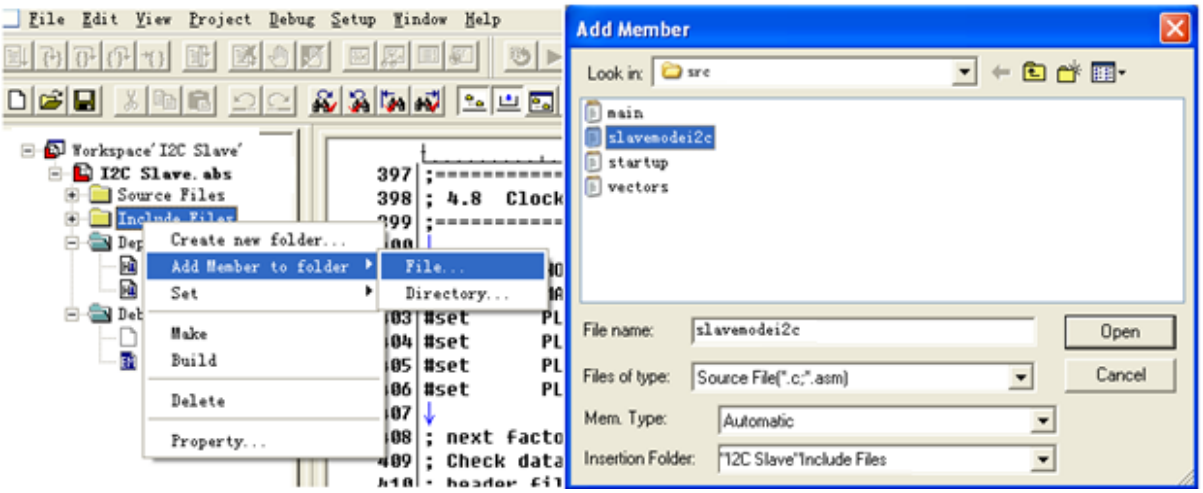
1. **First step is adding file to folder, Figure 6-3 describes this step.**

Figure 6-3. Library Use First Step



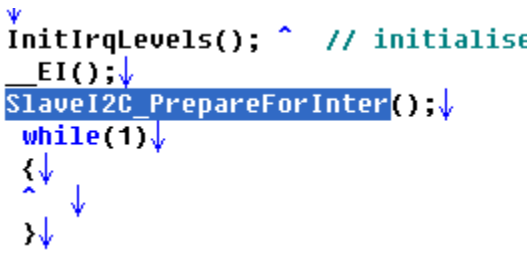
2. **Second step is adding function to project, Figure 6-4 describes this step**

Figure 6-4. Library Use Second Step



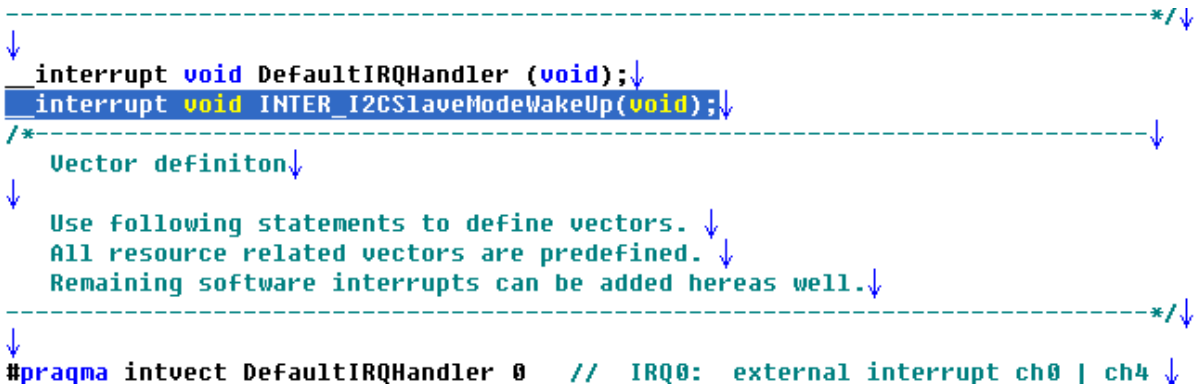
3. **Third step is adding initial function to main.c, Figure 6-5 describes this step.**

Figure 6-5. Library Use Third Step



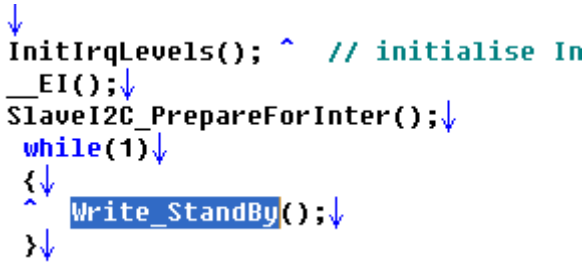
4. **Fourth step is adding interrupt function to vector.c, Figure 6-6 describes this step.**

Figure 6-6. Library Use Fourth Step



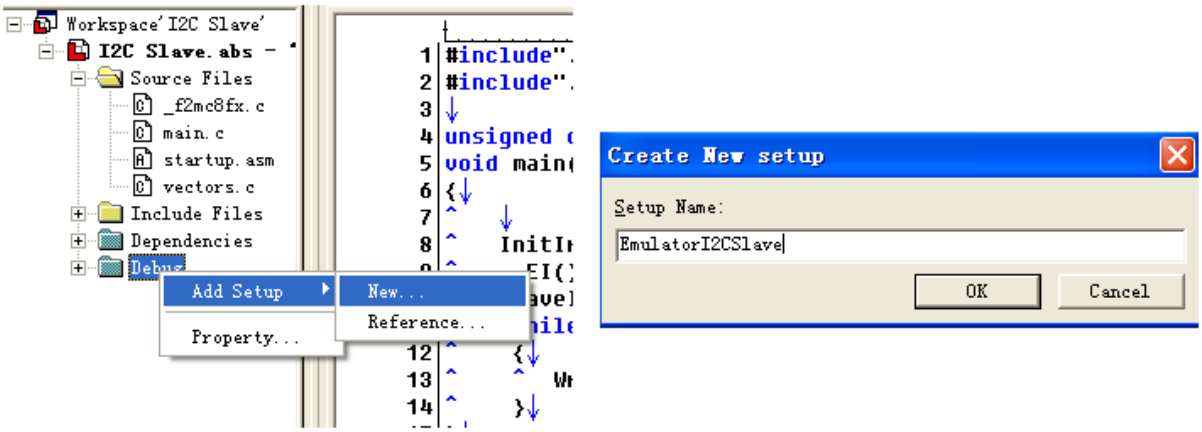
5. **Fifth step is adding Write_StandBy function to main.c, Figure 6-7 describes this step.**

Figure 6-7. Library Use Fifth Step



6. **The sixth step is debugging, before debugging set a environment is first, Figure 6-8 describes this step.**

Figure 6-8. Library Use Sixth Step



The emulator use MB2146-08, for more condition please refers to chapter 7.

6.3 I²C Slave Feed Back

When master device read I2C slave, the MCU will feed back 0x01 to master for indicate the slave feedback is work successfully.

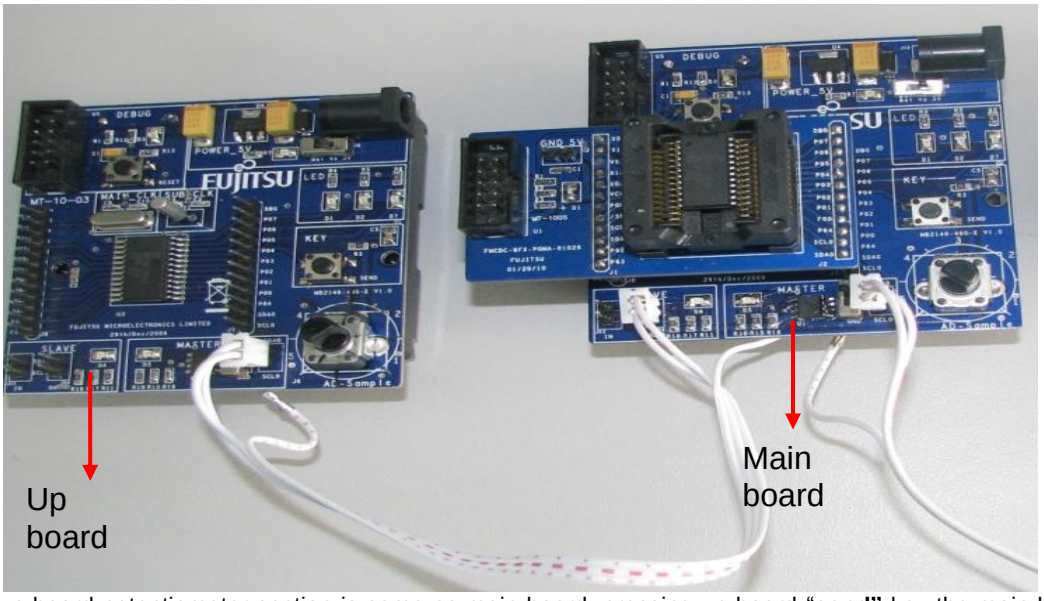
7 Debug

This chapter describes how to debug the sample code on EV-board and what will happen when the code is running.

There is a simple project *I²C Slave.prj* to be debugging. This project is based on our EV-board MB2146-460-E and the target MCU is MB95350L.

When debugging, the hardware connects please reference to Figure 7-1. For test board connects following figure maybe referred.

Figure 7-1. Test Board Connect



During up board potentiometer section is same as main board, pressing up board "**send"** key the main board LED7 will light on which indicates the I²C slave function work successfully. And in project the global parameter DatSlavI2CRead record the I²C slave received data.

Figure 7-2. Debugging Description



8 I2C Slave Sample Code

This chapter describes MB95350L I^2C slave C code.

Figure 8-1 describes the SlaveI2C_PrepareForInter function.

Figure 8-1. SlaveI2C_PrepareForInter Code

```
      SYSC = 0x05;                    //enable SDA1 and SCL1
      ICCR1_EN = 0;
       ILSR_P16 = 1;                  //CMOS level
       ILSR_P17 = 1;
       DDR1_P16 = 0;
       DDR1_P17 = 0;

      ICCR1_EN = 0;             // clear I2C interface
      ICCR1_CS4 = 0;            // set clock divider 'm' => 6
      ICCR1_CS3 = 1;
      ICCR1_CS2 = 0;            // set clock divider 'n' => 4
      ICCR1_CS1 = 0;
      ICCR1_CS0 = 0;            // F_SCK = MCLK / (m * n +2) => 3MHz/(6*4
+2)
                                //= 3MHz/26 = 115 kHz
      ICCR1_EN = 1;             // enable I2C interface
      IDDR1 = 0x00;             // clear data register
      IBCR01 = 0x05;            // enable address acknowledge bit,
                                // transfer completion interrupt after
nine cycles,
                                // enable stop detection interrupt
      IBCR11 = 0x4e;            // set slave mode, enable data acknowledge
bit,
                                // enable bus error and transfer complete
interrupt
      IAAR1 = 0x30;                   // slave address 0X60
```

Figure 8-2 describes the INTER_I2CSlaveModeWakeUp function.

Figure 8-2. INTER_I2CSlaveModeWakeUp Code

```
      IBCR01_WUE = 0;               //disable i2c wake up interrupt
      if (IBCR01_SPF == 0x01)             // stop condition detected
            IBCR01_SPF = 0x00;      // clear stop condition

      if (IBCR11_INT == 0x01)             // transfer completed
      {
                  ReaDat = IDDR1;
                  Tim ++;
                  if(Tim == 1)
                  {
                        ReaDat &= 0x01;
                        if(ReaDat == 0x01)
                        IDDR1 = 0X01;     //IDDR1 =
PraGlo.DatSlavI2CRead;
                  }
                  else if(Tim == 2)
                  {
                        Tim = 0;
                  DatSlavI2CRead = ReaDat;            //read I2C SDA
data
                        DDR0_P05 = 1;
                  PDR0_P05 = 0;
                  }
                  IBCR11_INT = 0x00;            // clear bit
      }

      if(IBCR11_BER == 0x01)        // bus error detected
            IBCR11_BER = 0x00;      // bit cleared
```

Figure 8-3 describes the Write_StandBy function.

Figure 8-3. Write_StandBy Code

```
      SlaveI2C_PrepareForInter();
      IBCR01_WUE = 1;               //enable i2c wake up//
      while(IBSR1_BB);
      STBC = 0x80;                        //Stop ENTERED
```

Figure 8-4 describes the main function.

Figure 8-4. Main Function Code

```
      InitIrqLevels();  // initialise Interrupt level register and IRQ
vector table
      __EI();
      SlaveI2C_PrepareForInter();
      while(1)
       {
            Write_StandBy();
       }
```

9 Additional Information

For more information about how to use MB95350L EV-board, BGM Adaptor and SOFTUNE, please refer to EV-Board MB2146-460-E User Manual (MCU-AN-500083-E-10).

Please visit following website for more information on MB2146-460-E EV Board:

http://www.cypress.com/documentation/development-kitsboards/mb2146-460-e

Please contact your local support team for any technical question.

Document History

Document Title: AN205072 - F²MC-8FX Family MB95350L Series I2C Slave Module API

Document Number: 002-05072

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	—	HUAL	04/27/2009	Initial release
*A	5260173	HUAL	05/27/2016	Migrated Spansion Application Note “MCU-AN- 500095-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.