AN205281

F²MC-8FX Family, MB95200H/210H Series External Interrupts

This application note describes how to set and use the External Interrupt and also describes the function of the External Interrupts and gives some examples.

Contents

1 Introduction....1
2 External Interrupt....1
2.1 Key Features....1
2.2 Block Diagram....2
2.3 Connection Diagram....2
2.4 Registers....3
3 External Interrupt Timing....4
4 External Interrupt Examples....5
4.1 Basic Functions....5
4.2 External interrupts and Wake Up from Stop Mode....7
5 Precautions when Using External Interrupt Circuit....9
6 Additional Information....9
6.1 Sample Code....10
7 Document History....15

1 Introduction

This application note describes how to set and use the External Interrupt.

This application note describes the function of the External Interrupts and gives some examples.

2 External Interrupt

This chapter describes the basic features of the interrupt module

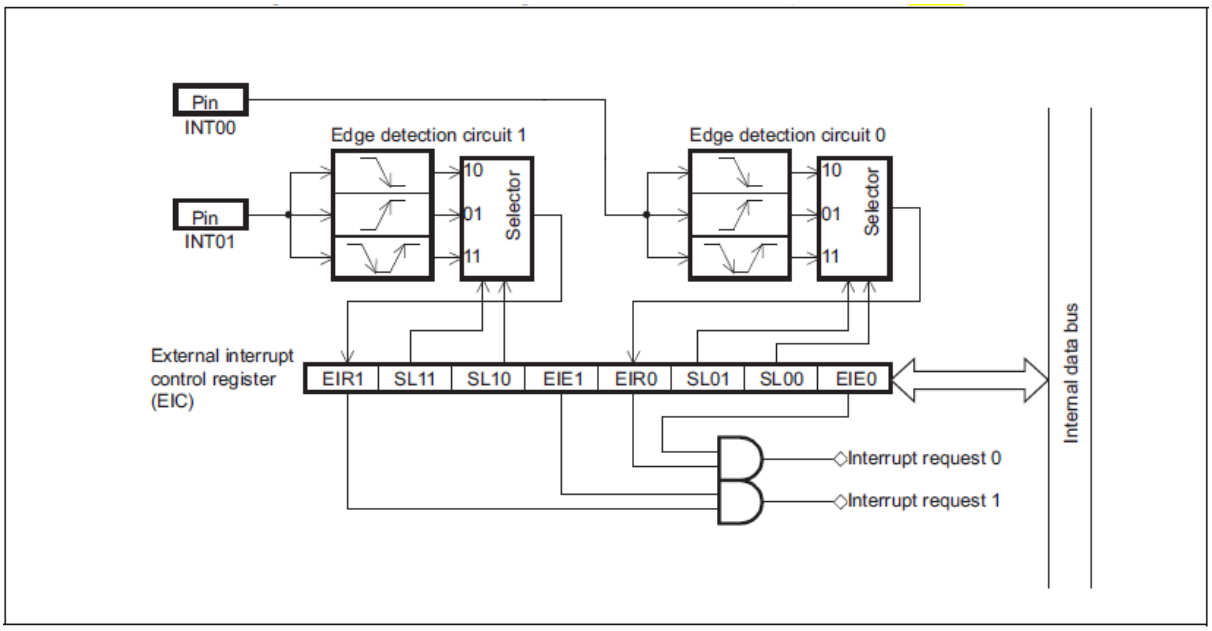
2.1 Key Features

- Edge detection circuit
- External Interrupt control register

2.2 Block Diagram

Figure 1 shows the internal block diagram of an External Interrupt channel.

Figure 1. External Interrupts Block Diagram



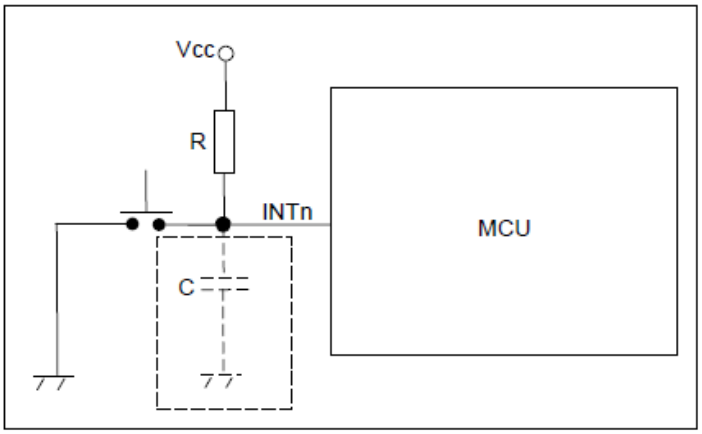
2.3 Connection Diagram

Figure 2 shows the connection diagram depicting how the External Interrupt pin INTn is interfaced to the external circuitry.

The pull-up resistor R is connected to limit the current when the key is pressed. The capacitor (hardware debounce) circuit shown in dotted line is optional, and is used to eliminate the bouncing of the switch. The RC time constant needs to be chosen according to the bouncing time or debounce delay of the switch. If such arrangement is not used, then the bouncing needs to be taken care in the software and vice versa.

The INTn needs to be configured to detect a falling edge.

Figure 2. External Interrupt Connection Diagram (Application of switch function)



Please note that this application is only used for switch or keyboard functions.

2.4 Registers

Please refer to MB95200H/210H Series Hardware Manual Chapter 15 for detailed register setting.

2.4.1 External Interrupt Control Register (EICxy)

The external interrupt control register is used to control interrupt and select the edge polarity for the external interrupt input and control interrupts.

Table 1. External Interrupt Control Register

	bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Corresponding Ext. Int. No.	
EIC10	EIR1	SL11	SL10	EIE1	EIR0	SL01	SL00	EIE0	INT03	INT02
EIC20	EIR1	SL11	SL10	EIE1	EIR0	SL01	SL00	EIE0	INT05	INT04
EIC30	EIR1	SL11	SL10	EIE1	EIR0	SL01	SL00	EIE0	INT07	INT06

2.4.2 Interrupt Level Setting Registers (ILR0 to ILR5)

Table 2. Configuration of External Interrupt Level Setting Registers

ILR0	L03	[1:0]	L02	[1:0]	L01	[1:0]	L00	[1:0]
ILR1	L07	[1:0]	L06	[1:0]	L05	[1:0]	L04	[1:0]
ILR2	L11	[1:0]	L10	[1:0]	L09	[1:0]	L08	[1:0]
ILR3	L15	[1:0]	L14	[1:0]	L13	[1:0]	L012	[1:0]
ILR4	L19	[1:0]	L18	[1:0]	L17	[1:0]	L016	[1:0]
ILR5	L23	[1:0]	L22	[1:0]	L21	[1:0]	L020	[1:0]

One of the interrupt level setting registers (ILR0) contains 4 pairs of bits assigned to interrupt requests from an External Interrupt.

Table 3. Interrupt Source

Interrupt Request No.	Bit Name in Interrupt Level Setting Register	Interrupt Source
IRQ0	L00[1:0]	External Interrupt ch.4
IRQ1	L01[1:0]	External Interrupt ch.5
IRQ2	L02[1:0]	External Interrupt ch.2/ ch.6
IRQ3	L03[1:0]	External Interrupt ch.3/ ch.7

Each pair of bits (interrupt level setting bits as two-bit data) sets an interrupt level.

Table 4. Relationships between Interrupt Level Setting Bits and Interrupts Levels

LXX[1:0]	Interrupt Level	Priority
00	0	Highest
01	1	
10	2	
11	3	Lowest (No interrupts accepted)

XX: represents the corresponding interrupt number 00 to 03

During execution of a main program, the interrupt level bits in the condition code register (CCR: IL1, IL0) usually indicates “11_B”.

3 External Interrupt Timing

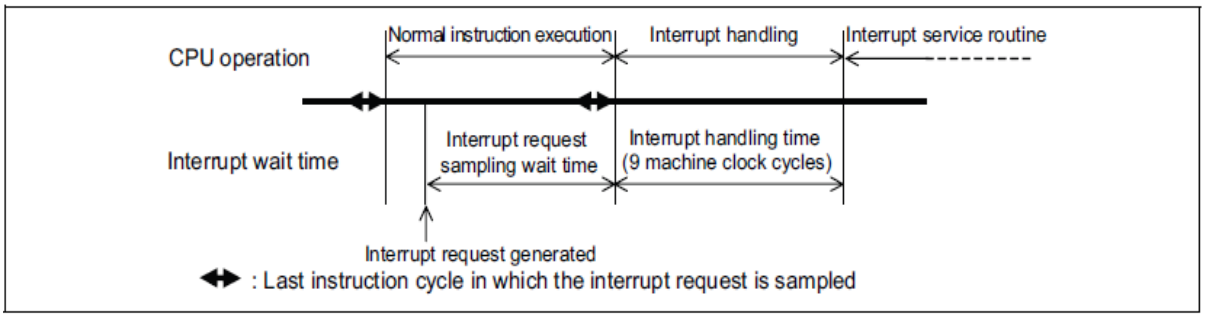
This chapter describes timing of external interrupts

The following figure shows events for External Interrupts timing.

After receiving an interrupt, the CPU requires 9 machine clock cycles to perform the following interrupt processing setup:

- Save the program counter (PC) and program status (PS) value
- Set the PC to the start address (interrupt vector) of interrupt service routine
- Update the interrupt level bits (PS:CCR:IL1,IL0) in the program status (PS) register

Figure 3. Interrupt Processing Time



4 External Interrupt Examples

This chapter describes examples on external interrupts

4.1 Basic Functions

The following example shows how to set up the External Interrupt Channel 2 and Channel 6 for operation.

Program will enter ISR while generate an Ext Int on channel 2 or channel 6, and clear flag and wait next external interrupt signal.

```
/* initial interrupt setting */
void InitExtInt0206 (void)
{
       AIDRL |= 0x04;       // disable AN02 input
       DDR0 &= 0xBB;        // P02 P06 input

       EIC10 = 0x03;        // External Interrupt 2 rising edge
                            // & Enable external interrupt output
       EIC30 = 0x03;        // External Interrupt 6 rising edge
                            // & Enable external interrupt output
}


/* external ISR */
__interrupt void ExInt0206 (void)
{
       if (EIC10_EIR0)            // Ext Int 02 ?
       {
              EIC10_EIR0 = 0;     // Clear interrupt request flag

              ...                 // interrupt server routine

       }
       if (EIC30_EIR0)            // Ext Int 06 ?
       {
              EIC30_EIR0 = 0;     // Clear interrupt request flag

              ...                 // interrupt server routine

       }
}

/* main routine */
void main (void)
{
       ...
       InitExtInt0206();
       ...
}
```

Please note that the corresponding interrupt vector and level should be defined in the vector.c module of Cypress's standard template project.

```
      ...
      ILR0 = 0xCF;      //IRQ2: external interrupt ch2 | ch6
      ...
__interrupt void ExInt0206 (void);  // Prototype
      ...
#pragma intvect ExInt0206     2     // IRQ2: external interrupt ch2 | ch6
```

Refer to Appendix Sample Code for project "EI_Basic".

4.2 External interrupts and Wake Up from Stop Mode

The following example shows that External Interrupts can be used to request stop mode and wake up the MCU from this mode.

External Interrupt 2 is used to wake up the MCU while External Interrupt 6 is used to request stop mode.

At the beginning, program will run in normal mode, after an external interrupt (channel 6), set flag from “RUNMODE” to “STOPREQUEST” in the ISR, then MCU will enter stop mode, and also the external interrupt 2 will wake up the MCU. Therefore, the MCU mode can switch via external interrupt 2 and 6.

```
// initialise external interrupt 2 & 6
void InitExtInt0206 (void)
{
        AIDRL |= 0x04;          // disable AN02 input
        DDR0 &= 0xBB;           // P02 P06 input
        EIC10 = 0x03;           // External Interrupt 2 rising edge
                                // & Enable external interrupt output
        EIC30 = 0x03;           // External Interrupt 6 rising edge
                                // & Enable external interrupt output
}

/* external ISR */
__interrupt void ExInt0206 (void)
{
        if (EIC10_EIR0)                 // Ext Int 02 ?
        {
                status = RUNMODE;
                EIC10_EIE0 = 0;         // disable external interrupt 2
                EIC10_EIR0 = 0;         // clear interrupt request flag
                EIC10_EIE0 = 1;         // enable interrupt request
        }
        if (EIC30_EIR0)                 // Ext Int 06 ?
        {
                status = STOPREQUEST;
                EIC30_EIE0 = 0;         // disable external interrupt 6
                EIC30_EIR0 = 0;         // clear interrupt request flag
                EIC30_EIE0 = 1;         // enable interrupt request
        }
}

/* main routine */
void main (void)
{
        ...

        InitExtInt0206();
        ...
```

```
        status = RUNMODE;
        while (1)
        {
                if (status == STOPREQUEST) // stop mode request


                {
                        PDR0_P00 = 1;
                        STBC_STP = 1;        // enter stop mode
                }
                else          // run mode
                {
                        for (i = 0; i < 50000; i ++)      //wait loop
                        {
                                __asm("\tNOP");
                                __asm("\tNOP");
                        }
                        PDR0_P00 = ~PDR0_P00;      // show that MCU is running
                }
        }
}
```

Please note that the corresponding interrupt vector and level should be defined in the vector.c module of Cypress's standard template project.

```
/* Interrupt level setting */
void InitIrqLevels (void)
{
        ...

        ILR0 = 0xDF;         // IRQ2: external interrupt ch2 | cha6
        ...
}

__interrupt void ExInt0206 (void);       // Prototype

        ...

#pragma intvect ExInt0206  2      // IRQ2: external interrupt ch2 | ch6

        ...
```

Refer to Appendix Sample Code for project "EI_ModeSwitch".

5 Precautions when Using External Interrupt Circuit

Precautions

- Set the interrupt request enable bit (EIE) to “0” (disabling interrupt requests) in the process of setting the edge polarity select bit (SL). Moreover, clear the external interrupt request flag bit (EIR) to “0” after the setting of the edge polarity.
- The operation cannot recover from the interrupt processing routine if the external interrupt request flag bit is “1” and the interrupt request enable bit is enabled. Always clear the external interrupt request flag bit in the interrupt processing routine.

6 Additional Information

For more Information on MB95200 Products, visit the following website:

http://www.cypress.com/8fx-mb95200

6.1 Sample Code

6.1.1 Project Name: EI_Basic

Basic external ISR

main.c

```
#include "mb95200.h"
/*-------------------------------------------------------------------------
   name: InitExtInt0206 ();
   function: initial external interrupt
-------------------------------------------------------------------------*/
void InitExtInt0206 (void)
{
       AIDRL |= 0x04;       // disable AN02 input
       DDR0 &= 0xBB;        // P02 P06 input
       EIC10 = 0x03;        // External Interrupt 2 rising edge
                            // & Enable interrupt output
       EIC30 = 0x03;         // External Interrupt 6 rising edge
                            // & Enable external interrupt output
}
/*-------------------------------------------------------------------------
   name: ExInt0206 ();
   function: enter while generate an Ext Int
-------------------------------------------------------------------------*/
__interrupt void ExInt0206 (void)
{
       if (EIC10_EIR0)
       {
              EIC10_EIR0 = 0;            // Clear interrupt request flag
                                         // interrupt server routine
       }
       if (EIC30_EIR0)
       {
              EIC30_EIR0 = 0;             // Clear interrupt request flag
                                         // interrupt server routine
       }
}
```

```
/*-----------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------*/
void main(void)
{
       InitExtInt0206();
       InitIrqLevels();
       __EI();
       while(1);
}

vector.c
#include "mb95200.h"
/*-----------------------------------------------------------------------
   name: InitIrqLevels ();
   function: Interrupt level (priority) setting
-----------------------------------------------------------------------*/
void InitIrqLevels(void)
{
       ILR0 = 0xCF;        // IRQ0:  external interrupt ch4
                           // IRQ1:  external interrupt ch5
                           // IRQ2:  external interrupt ch2 | ch6
                           // IRQ3:  external interrupt ch3 | ch7
}
/*-----------------------------------------------------------------------
   Prototypes
-----------------------------------------------------------------------*/
__interrupt void ExInt0206 (void);

/*-----------------------------------------------------------------------
   Vector definition
-----------------------------------------------------------------------*/

#pragma intvect ExInt0206  2   // IRQ2:  external interrupt ch2 | ch6
```

6.1.2 Project Name: EI_ModeSwitch

The MCU wakes up from stop mode via an external interrupt

main.c

```
#include "mb95200.h"
#define RUNMODE                  1
#define STOPREQUEST              0
unsigned char status;

/*-----------------------------------------------------------------------------
   name: InitExtInt0206 ();
   function: initial external interrupt
-----------------------------------------------------------------------------*/
void InitExtInt0206 (void)
{
       AIDRL |= 0x04;            // disable AN02 input
       DDR0 &= 0xBB;             // P02 P06 input
       EIC10 = 0x03;             // External Interrupt 2 rising edge
                                 // & Enable interrupt output
       EIC30 = 0x03;             // External Interrupt 6 rising edge
                                 // & Enable external interrupt output
}

/*-----------------------------------------------------------------------------
   name: ExInt0206 ();
   function: mode switch
-----------------------------------------------------------------------------*/
__interrupt void ExInt0206 (void)
{
       if (EIC10_EIR0)
       {
              status = RUNMODE;
              EIC10_EIE0 = 0;     // disable external interrupt 2
              EIC10_EIR0 = 0;     // clear interrupt request flag
              EIC10_EIE0 = 1;     // enable interrupt request
}
if (EIC30_EIR0)
{
              status = STOPREQUEST;
```

```
            EIC30_EIE0 = 0;     // disable external interrupt 6
            EIC30_EIR0 = 0;     // clear interrupt request flag
            EIC30_EIE0 = 1;     // enable interrupt request
        }
}

/*-----------------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------------*/
void main (void)
{
        unsigned int i;
        InitExtInt0206();
        DDR0_P05 = 1;
        InitIrqLevels();
        __EI();
        status = RUNMODE;
        while (1)
        {
        if (status == STOPREQUEST) // stop mode request
        {
                PDR0_P00 = 1;
                STBC_STP = 1;       // go to stop mode
        }
        else          // run mode
        {
                for (i = 0; i < 5000; i ++)       //wait loop
                {
                __asm("\tNOP");
                __asm("\tNOP");
                }
                PDR0_P05 = ~PDR0_P05;      // show, that MCU is running
                }
        }
}
```

```
vector.c
#include "mb95200.h"
/*-----------------------------------------------------------------------
   name: InitIrqLevels ();
   function: Interrupt level (priority) setting
-----------------------------------------------------------------------*/
void InitIrqLevels(void)
{
       ILR0 = 0xCF;        // IRQ0:  external interrupt ch4
                           // IRQ1:  external interrupt ch5
                           // IRQ2:  external interrupt ch2 | ch6
                           // IRQ3:  external interrupt ch3 | ch7
}
/*-----------------------------------------------------------------------
   Prototypes
-----------------------------------------------------------------------*/
__interrupt void ExInt0206 (void);

/*-----------------------------------------------------------------------
   Vector definition
-----------------------------------------------------------------------*/

#pragma intvect ExInt0206  2   // IRQ2:  external interrupt ch2 | ch6
```

7 Document History

Document Title: AN205281 - F²MC-8FX Family, MB95200H/210H Series External Interrupts

Document Number: 002-05281

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	HUAL	03/20/2008	Initial release
			07/18/2008	Added URL in chapter 6 Additional Information; modified Some sample code.
*A	5264352	HUAL	06/03/2016	Migrated Spansion Application Note MCU-AN-500007-E-11 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, 2008-2016. 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.