AN205496

F²MC-16FX, All Series, External Interrupts

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

Contents

1 Introduction....1
1.1 Key Features....1
2 External Interrupts....2
2.1 Block Diagram....2
2.2 Connection Diagram....2
2.3 Register....3
3 External Interrupt Timing....4
4 External Interrupt Examples....5
4.1 Basic Functionality....5
4.2 Disturbed Signals....6
4.3 External Interrupts and Wake Up from Stop Mode....10
5 Additional Information....13
Document History....14

1 Introduction

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

1.1 Key Features

The external interrupts have the following features:

- Edge sensitivity - rising and falling selectable
- Level sensitivity - low and high selectable
- ISR execution is started in 200 ns + minimum 10 CPU cycles after first rising / falling edge at external interrupt pin

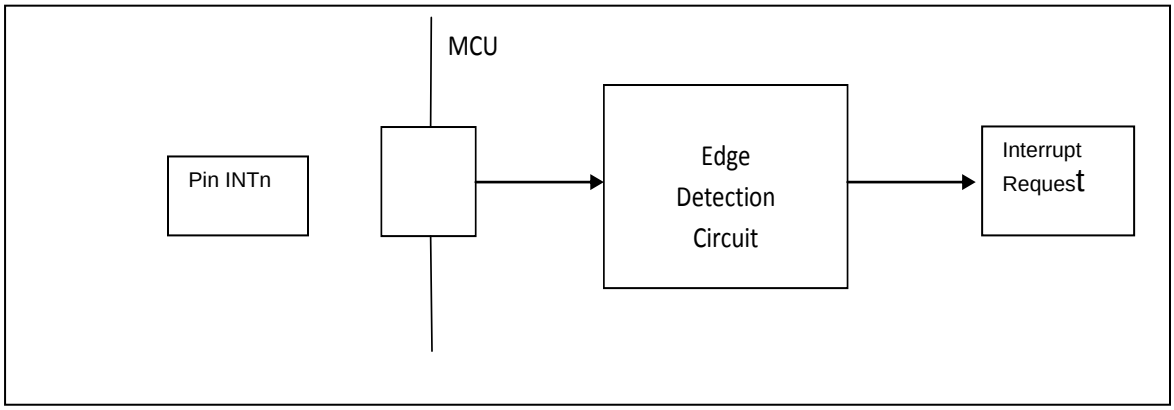
2 External Interrupts

The basic functionality of the external interrupt module

2.1 Block Diagram

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

Figure 1. External Interrupts Block Diagram



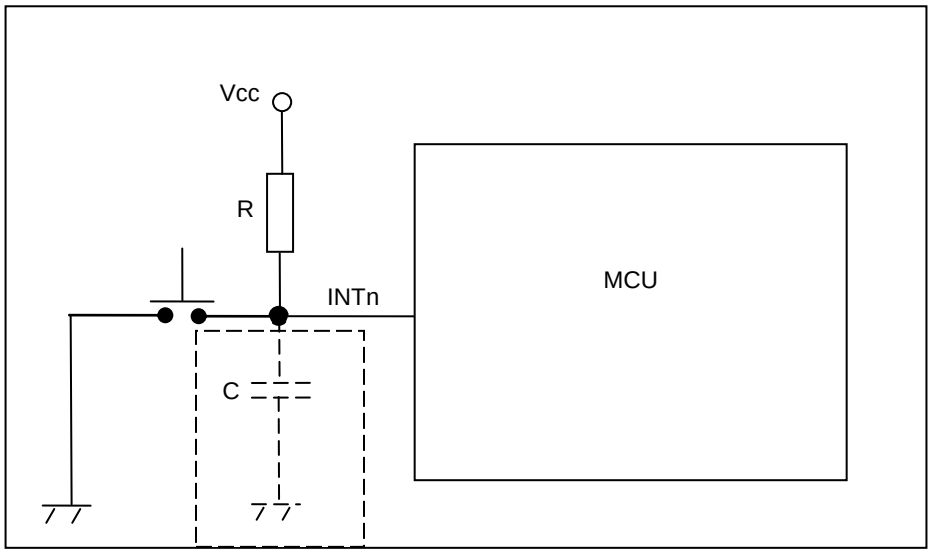
2.2 Connection Diagram

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 with the 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 also vice a versa.

Here the INTn needs to be configured to detect a falling edge.

Figure 2. External Interrupts Connection Diagram



2.3 Register

2.3.1 External Interrupt Enable Register (ENIR)

This 16 bit register contains the interrupt enable bits for each external interrupt channel. Writing ‘1’ to these bits enables the corresponding external interrupt requests.

2.3.2 External Interrupt Request Register (EIRR)

This 16 bit register contains the flag which is set if the specified interrupt event occurs at the corresponding external interrupt pin. Writing “0” to it clears the request. The interrupt request bit must be cleared by the ISR. These flags can also be used for polling if the corresponding Interrupt Enable bit is disabled.

2.3.3 Request Level Setting Register (ELVR)

This register contains a pair of control bits LBx:LAx for each channel for the detection kind. The following table shows the possible settings:

Table 1. ELVR

LBx	LAx	Functionality
0	0	“L” Level Input
0	1	“H” Level Input
1	0	Rising Edge Pin Input
1	1	Falling Edge Pin Input

Where x = 0 to 7 for ELVR0 and 8 to 15 for ELVR1

2.3.4 Port Input Enable Register

External Interrupts need a port pin that is set to digital input mode. Please enable the corresponding input pin before enabling the external interrupt. See pin assignment in datasheet to determine which port pin must be enabled.

Example:

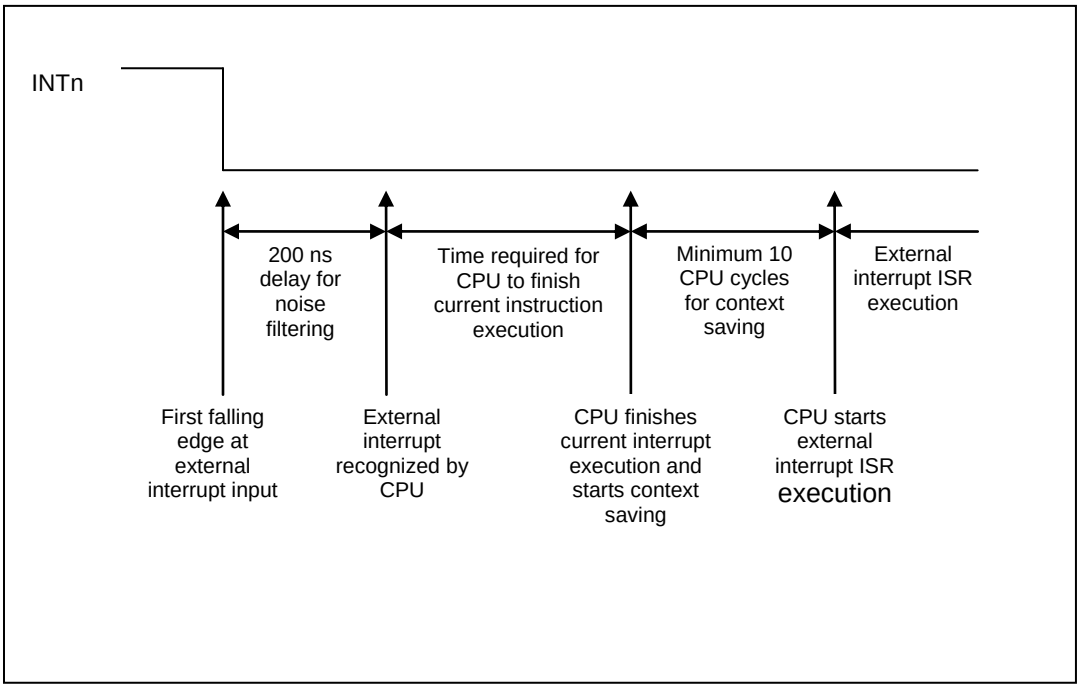
External Interrupt 0 shall be used. This is shown as INT0 in pin assignment. On MB96340 series, this function shares pin P07_0. Hence, this pin must be enabled for input:

PIER07_IE0 = 1;

3 External Interrupt Timing

The following figure shows timing of events for external interrupt. Here it is considered that external interrupt pin INTn is configured to detect falling edge.

Figure 3. External Interrupt Timing



Time required for CPU to finish the current instruction execution is dependent on type of instruction being executed. If it is one of the interrupt deferring instructions / prefix codes then it is definitely more than all other instructions. For further details please refer section 2.4.2 of Interrupts application note AN205548.

Time required for the context saving is dependent on various factors. Those are described in section 2.4.1 of Interrupts application note AN205548. The above mentioned time of 10 cycles is the minimum timing required for context saving.

4 External Interrupt Examples

Examples for external interrupts

4.1 Basic Functionality

The following example shows how to set up the External Interrupt Channel 0 of the MB96340 Series.

```
/*                                    SAMPLE CODE                                  */
/*---------------------------------------------------------------------------------*/
void InitExtInt0(void)
{
    ADER2 = 0x00;        // Port function
    PIER07_IE0 = 1;      // Enable Port 07_0 input

    ELVRL0_LB0 = 0;      // LB0, LA0 = 00 -> low level
    ELVRL0_LA0 = 0;
    EIRR0_ER0 = 0;       // reset interrupt request
    ENIR0_EN0 = 1;       // enable interrupt request
}
    . . .

_interrupt void IRQHandler_EI0 (void)
{
    EIRR0_ER0 = 0;       // clear interrupt request
        . . .
}
```

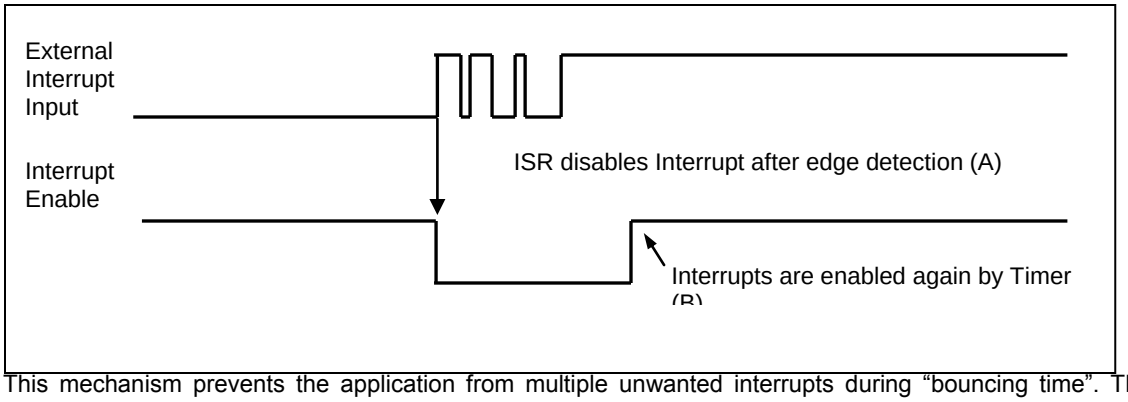
Please note, that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
  /*                                    SAMPLE CODE                                  */
  /*---------------------------------------------------------------------------------*/
    void InitIrqLevels(void)
   {
           . . .

   ICR = ((17 & 0xFF) << 8) | 6;      // Priority Level 6 for External
                                      // Interrupt 0 of MB96340 Series
    }
           . . .

__interrupt void IRQHandler_EI0 (void);     // Prototype
           . . .

#pragma intvect IRQHandler_EI0    17    // EXT0 of MB96340 Series

           . . .
```

4.2 Disturbed Signals

If signal on an External Interrupt input pin is disturbed (such as from a bouncing switch), it is recommended to use edge detection instead of level sensitivity. Also, if the bouncing of the switch needs to be taken care in the software (i.e. the optional capacitor as shown the Figure 2 is not used), then the external interrupt service routine should disable the interrupt for a time slot greater than the bouncing time. This can be done by a further interrupt using a timer (e.g. Reload Timer 0).

Figure 4. Interrupt Timing – Disturbed Input Signal



This mechanism prevents the application from multiple unwanted interrupts during “bouncing time”. The same is accomplished using the software for the External Interrupt Channel 0 of the MB96340 Series in the example on the next page.

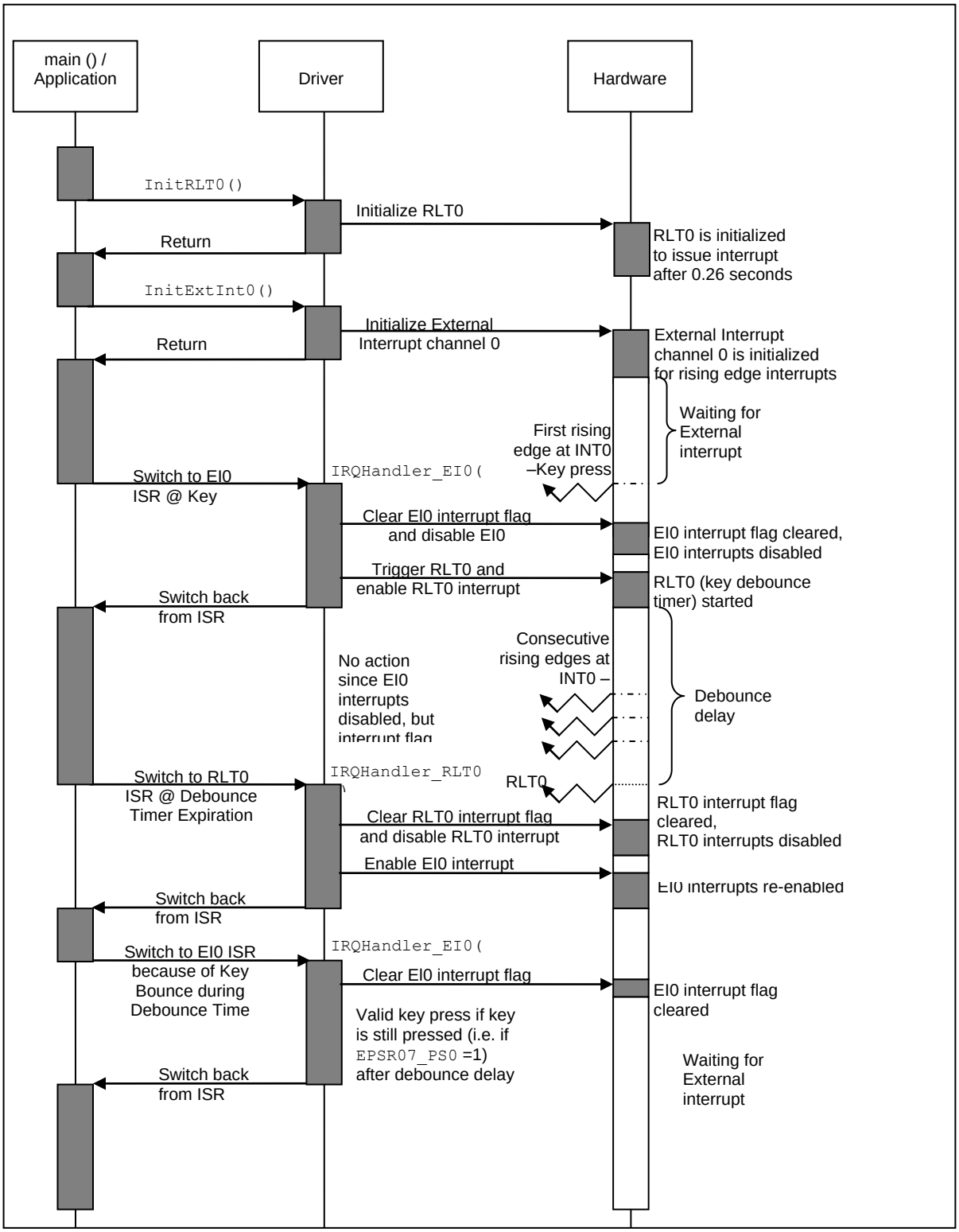
In the below example, at the first rising edge of the signal at External Interrupt input pin INT0, the interrupt service routine IRQHandler_EI0() will be executed and since the initial value of FLAG variable is TRUE, the Reload Timer RLT0 will be started (A in the above figure) and no further processing would happen.

After 0.26 seconds (debouncing time, this can be chosen depending upon the type of key/switch) the interrupt service routine IRQHandle_RLT0() will be executed and it will enable External Interrupt request again and make value of FLAG variable FALSE. Now by this time (B in the above figure) since there are multiple edges already appeared at pin INT0, again interrupt service routine IRQHandler_EI0() will be executed. Within this interrupt service routine value of FLAG variable will be made TRUE for the next key press and then the user can do the desired processing considering a “valid” key press, if the key is still pressed (i.e. if EPSR07_PS0 =1).

It should be noted that the above mentioned logic would work properly only when the key bounces a few times before settling at a position after a key press.

4.2.1 Sequence Diagram

Figure 5. Sequence Diagram – Disturbed Input Signal



```
 /*                              SAMPLE CODE                                    */
 /*-----------------------------------------------------------------------------*/
   #define TRUE      1
   #define FALSE     0

volatile unsigned char FLAG = TRUE;
void InitExtInt0(void)
{
    ADER2 = 0x00;    // Port function
    PIER07_IE0 = 1;  // enable Port 07_0 input

    ELVRL0_LB0 = 1;  // LB0, LA0 = 10 -> Rising edge
    ELVRL0_LA0 = 0;
    EIRR0_ER0 = 0;   // reset interrupt request
    ENIR0_EN0 = 1;   // enable interrupt request

void InitRLT0(void)
{
    TMRLR0 = 0xFFFF; // Reload value (about 0.26 s @ 16 MHz)
    TMCSR0 = 0x0800; // Prescaler 1:64
}
void main(void)
{
. . .

    InitRLT0();      // initialize RLT0
    InitExtInt0();   // initialize EXT0

. . .
     while (1);
}

 __interrupt void IRQHandler_EI0(void)
{
EIRR0_ER0 = 0;       // clear interrupt request

    if (FLAG == TRUE)
    {
    ENIR0_EN0 = 0;          // disable interrupt request, Figure 3-1:(A)
    TMCSR0_UF = 0;          // clear Reload Timer 0
    TMCSR0_TRG = 1;  // trigger Reload Timer 0
    TMCSR0_INTE = 1; // enable Reload Timer 0 interrupt
    }
    else
    {       FLAG = TRUE;     // for next key press
            if (1 == EPSR07_PS0)// key still pressed?
            {
                        // valid key press, take required action
            }
     }
}
__interrupt void IRQHandler_RLT0(void)
{
    TMCSR0_UF = 0;   // clear Reload Timer 0
    TMCSR0_INTE = 0; // disable Reload Timer 0 interrupt

    ENIR0_EN0 = 1;        // enable External Interrupt request again,
                          // Figure 3-1:(B)
    FLAG = FALSE;
}
```

Please note, that the ISRs for the External Interrupts and the Reload Timer also has to be defined in *vectors.c*.

```
/*                                  SAMPLE CODE                                  */
/*-------------------------------------------------------------------------------*/
void InitIrqLevels(void)
 {
             . . .

      ICR = ((17 & 0xFF) << 8) | 6;         // Priority Level 6 for External
                                            // Interrupt 0 of MB96340 Series
      ICR = ((51 & 0xFF) << 8) | 5;        // Priority Level 5 for Reload
                                           // Timer 0 of MB96340 Series
 }
__interrupt void IRQHandler_EI0 (void);  // Prototype
__interrupt void IRQHandler_RLT0 (void); // Prototype


#pragma intvect IRQHandler_EI0    17       // EXT0 of MB96340 Series
#pragma intvect IRQHandler_RLT0   51       // RLT0 of MB96340 Series
```

Also make sure that the RLT0 interrupt priority is higher than that of External Interrupt 0, in order to avoid nesting of interrupts.

4.2.2 External Interrupts and DMA

The user has to take special care when using DMA transfer triggered by an External Interrupt pin connected to disturbed signal source (as explained before in the section 4.2).

During the bouncing time of the switch, multiple DMA transfers can happen which is not desired. The DMA as such is not able to disable the interrupts during this bouncing time, so the software debounce mechanism discussed in section 4.2 cannot be realized in this case.

Hence it is required to use the capacitor (hardware debounce) circuit as shown in Figure 2. By using the same, the bouncing of the switch is eliminated at the hardware level itself and the clean signal (without noise) would appear at the external interrupt pin.

4.3 External Interrupts and Wake Up from Stop Mode

4.3.1 Example I

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

External Interrupt 0 is used to wake up the MCU and External Interrupt 1 is used to request a stop mode. Before stop mode “0x0F“ is written to Port00, during run mode this Port is counting.

```
/*                          SAMPLE CODE                                    */
/*-------------------------------------------------------------------------*/

// initialise external int 0
void Init_extint0 (void)
{
    ADER2_ADE16 = 0;    // select IO mode
    PIER07_IE0 = 1;
    ENIR0_EN0 = 0;      // disable ext int 0
    ELVR0_LA0 = 1;
    ELVR0_LB0 = 1;      // LB0,LA0 = 11 -> falling edge
}

// initialise external int 1
void Init_extint1 (void)
{
     ADER2_ADE17 = 0;    // select IO mode
     PIER07_IE1 = 1;
     ENIR0_EN1 = 0;      // disable interrupt request
     ELVR0_LA1 = 1;
     ELVR0_LB1 = 1;      // LB1,LA1 = 11 -> falling edge
 }
```

```
void main (void)
{
  PDR00 = 0x00;       // clear port data
  DDR00 = 0xFF;       // set port 0 to output

  Init_extint0();
  Init_extint1();
  InitIrqLevels();

  __set_il(7);        // allow all levels
  __EI();             // globally enable interrupts

  EIRR0_ER1 = 0;      // clear ext. int 1 request flag
  ENIR0_EN1 = 1;      // enable ext. int1
  status = RUNMODE;

  while(1)
  {
    if (status == STOPREQUEST) // stop mode request
    {
      PDR00 = 0x0F;
      SMCR = 0x03;    // goto stop mode, preserve pin state
    }

    else              // run mode
    {
      for (i = 0; i < 50000; i++)      // wait loop
      {
        __asm(" NOP");
        __asm(" NOP");
      }

      PDR00++;     // show, that MCU is running
    }
  }
}

// ISR external Int 0
__interrupt void IRQ_extint0 (void)
{
    status = RUNMODE;
    EIRR0_ER0 = 0;    // clear ext. int 0 request flag
    ENIR0_EN0 = 0;    // disable ext. int0
    EIRR0_ER1 = 0;    // clear ext. int 1 request flag
    ENIR0_EN1 = 1;    // enable ext. int1
 }

// ISR external Int 1
__interrupt void IRQ_extint1 (void)
{
    status = STOPREQUEST;
    EIRR0_ER0 = 0;    // clear ext. int 0 request flag
    ENIR0_EN0 = 1;    // enable ext. int0
    EIRR0_ER1 = 0;    // clear ext. int 1 request flag
    ENIR0_EN1 = 0;    // disable ext. int1
 }
```

Please note, that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
/*                              SAMPLE CODE                                   */
/*----------------------------------------------------------------------------*/
void InitIrqLevels(void)
 {
     ICR = ((17 & 0xFF) << 8) | 6;          // Priority Level 6 for External
                                            // Interrupt 0 of MB96340 Series
     ICR = ((18 & 0xFF) << 8) | 6;          // Priority Level 6 for External
                                            // Interrupt 1 of MB96340 Series
 }
            . . .
__interrupt void IRQ_extint0 (void);        // Prototype EXT0
__interrupt void IRQ_extint1 (void);        // Prototype EXT1
   . . .

#pragma intvect IRQ_extint0    17           // EXT0 of MB96340 Series
#pragma intvect IRQ_extint1    18           // EXT1 of MB96340 Series
         . . .
```

4.3.2 Example II

The following example shows to request a stop mode and to wake the MCU up from this mode using External Interrupt 0. After the MCU is in stop mode, if the falling edge appears at External Interrupt 0 pin, then the MCU wakes up and continues executing the program. There is no need to have an interrupt service routine in this case.

```
/*                            SAMPLE CODE                                  */
/*-------------------------------------------------------------------------*/

// Initialise external int 0
void Init_extint0 (void)
{
  ADER2_ADE16 = 0;    // select IO mode
  PIER07_IE0 = 1;
  ENIR0_EN0 = 0;      // disable ext int 0
  ELVR0_LA0 = 1;
  ELVR0_LB0 = 1;      // LB0,LA0 = 11 -> falling edge
  EIRR0_ER0 = 0;      // clear ext. int 0 request flag
}

// Request stop mode
void Request_stop (void)
{
  __DI();             // globally disable interrupts
  EIRR0_ER0 = 0;      // clear ext. int 0 request flag
  ENIR0_EN0 = 1;      // enable ext int 0
  SMCR = 0x03;        // goto stop mode
  EIRR0_ER0 = 0;      // clear ext. int 0 request flag after wakeup
  __EI();             // globally disable interrupts

}

void main (void)
{
  InitIrqLevels();
  __set_il(7);        // allow all levels
  __EI();             // globally enable interrupt
  Init_extint0();
  Request_stop()

  . . .

  . . .

}
```

5 Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

http://www.cypress.com/cypress-mcu-product-softwareexamples

The software examples related to this application note is:

96340_irq_ext

Document History

Document Title: AN205496 - F²MC-16FX, All Series, External Interrupts

Document Number:002-05496

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	04/25/2006	First Version; MWi
			12/13/2006	V1.1, Reviewed the document and updated with review findings, MPi
			02/21/2007	V1.2, Updated with re-review findings from PHu, MPi
			03/01/2007	V1.3, Updated with re-review findings from HWe, MPi
			06/05/2007	V1.4, Updated with re-review findings from PHu, MPi
			08/23/2007	V1.5, Fixed typos, added flowchart and modified code for disturbed signal, MPi
			08/28/2007	V1.6, Updated key features and added external interrupt timing, MPi
			06/24/2008	V1.7; add information on Port Input Enable; PHu
*A	5076591	MKEA	05/18/2016	Converted Spansion Application Note “MCU-AN -300203-E-V17” to Cypress format
*B	5869276	AESATP12	08/31/2017	Updated logo and copyright.

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
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

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

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

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