



---

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](http://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](http://www.cypress.com).

## FR, MB91460, External Interrupts

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

### Contents

1	Introduction.....	1	4.1	Basic Functionality.....	4
1.1	Key Features .....	1	4.2	Disturbed Signals .....	5
2	External Interrupts .....	2	4.3	External Interrupts and Wake Up from Stop Mode.....	9
2.1	Block Diagram .....	2	5	Additional Information.....	13
2.2	Connection Diagram .....	2		Document History.....	14
2.3	Register .....	3			
3	External Interrupt Timing .....	3			
4	External Interrupt Examples .....	4			

## 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 - rising and falling selectable
- Level - low and high selectable
- A minimum pulse width of >50ns while waking up from STOP mode with edge detection
- A minimum pulse width of 3 peripheral clock cycle for detection of edge
- Up to 16 channels

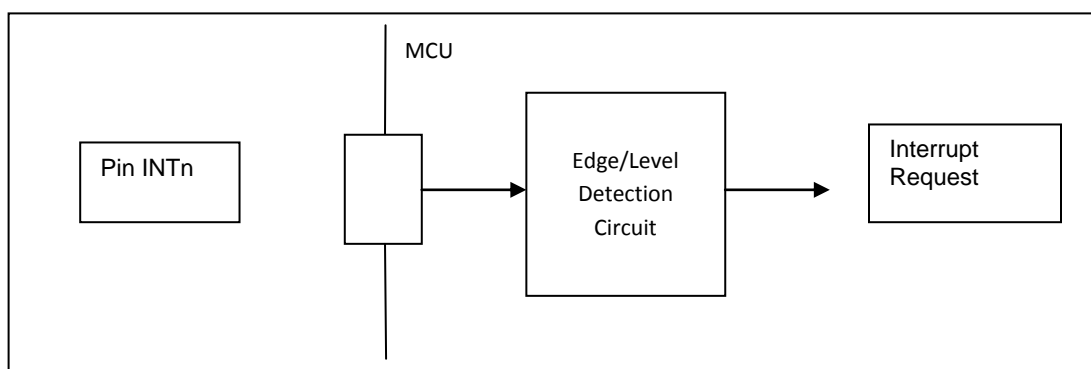
## 2 External Interrupts

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



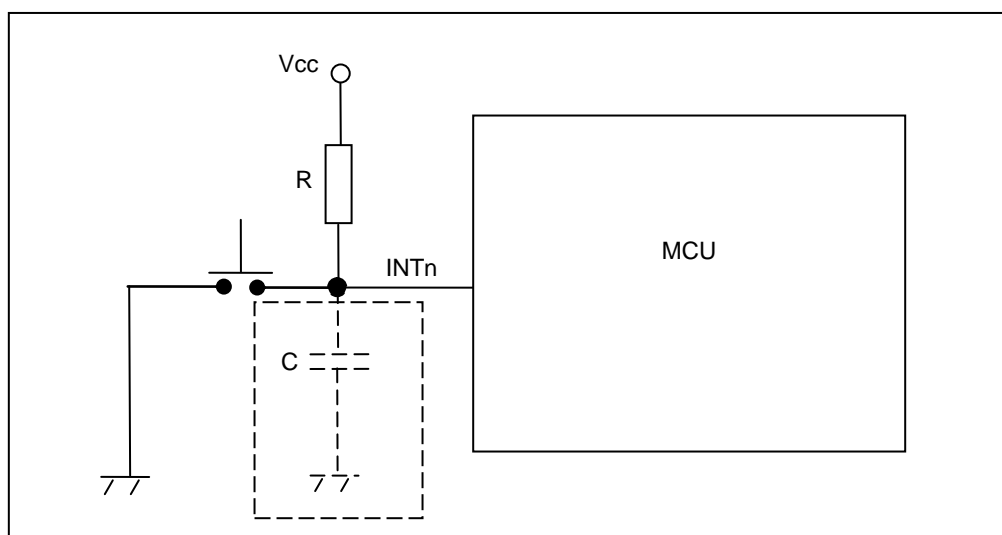
### 2.2 Connection Diagram

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 (ENIR0/1)

These 8 bit registers contain 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 (EIRR0/1)

These 8 bit registers contain 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 (ELVR0/1)

These 16 bit registers contain a pair of control bits LBx:LAx for each channel for the detection type. The following table shows the possible settings:

Table 1. ELVR

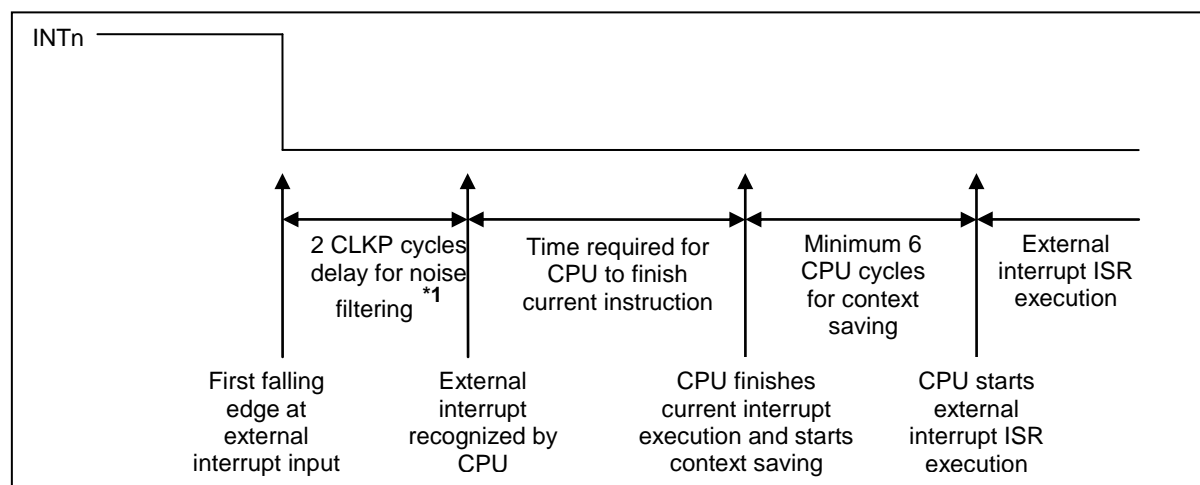
LBx	LAx	Functionality
0	0	"L" Level Input
0	1	"H" Level Input
1	0	Rising Edge Pin Input
1	1	Falling Edge Pin Input

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

## 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.

Time required for the context saving is dependent on various factors. Those are described in section 2.4.1 of Interrupts application note MCU-AN-300055. The above mentioned time of 6 cycles is the minimum timing required for context saving.

Time required to detect a valid edge depends on level of INT signal, the time window is two CLKP cycles before and two CLKP cycle after the falling edge. In case of STOP mode the trigger pulse must be > 50ns.

## 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 MB91460 Series.

```

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

void InitExtInt0(void)
{
    DDR24_D0 = 0;    // Port 24_0 Data direction as input
    PFR24_D0 = 1;    // Enable Port 24_0 as an external interrupt input

    ELVR0_LB0 = 0;   // LB0, LA0 = 00 -> low level
    ELVR0_LA0 = 0;
    EIRR0_ER0 = 0;   // clear interrupt flag
    ENIR0_EN0 = 1;   // enable interrupt request
}

. . .

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

    . . .

```

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)
{
    . . .

    ICR00 = 23; /* Priority Level 23 for External Interrupt 0 & 1 */

    . . .
}

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

. . .

#pragma intvect IRQHandler_EI0    16    /* EXT0 of MB91460 Series */

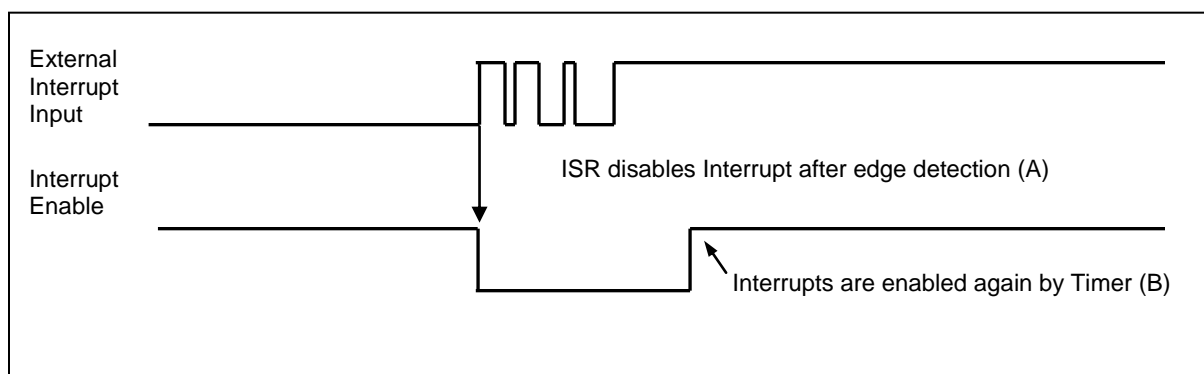
. . .

```

## 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 4 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 MB91460 Series in the example on the next page.

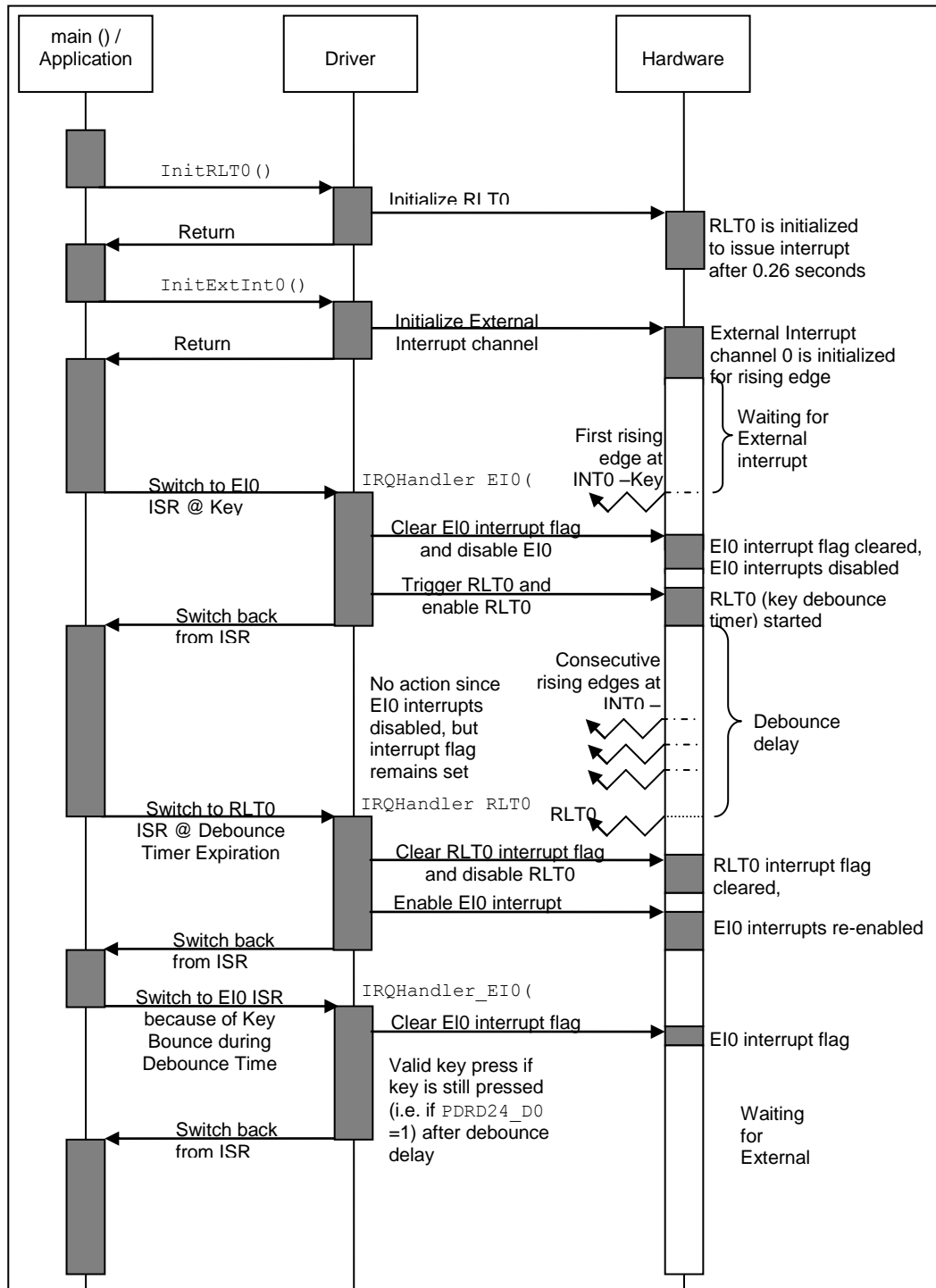
In the below example, at the first rising edge of the signal at External Interrupt input pin INT0, the interrupt service routine IRQHandler\_EI0() will be executed and since the initial value of FLAG variable is TRUE, the Reload Timer RLT0 will be started (A in the above figure) and no further processing 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 PDRD24\_D0 =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)
{
    DDR24_D0 = 0;    // Port 24_0 Data direction as input
    PFR24_D0 = 1;    // Enable Port 24_0 as an external interrupt input
    ELVR0_LB0 = 1;    // LB0, LA0 = 10 -> Rising edge
    ELVR0_LA0 = 0;
    EIRR0_ER0 = 0;    // clear interrupt flag
    ENIR0_EN0 = 1;    // enable interrupt request
}

void InitRLT0(void)
{
    TMR0 = 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 flag

    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 == PDRD24_D0) // 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)
{
    . . .

    ICR00 = 23; /* Priority Level 23 for External Interrupt 0 & 1 */
    ICR08 = 21; /* Priority Level 21 for RLT 0 & 1 */

    . . .

    __interrupt void IRQHandler_EI0 (void); // Prototype
    __interrupt void IRQHandler_RLT0 (void); // Prototype

    . . .

#pragma intvect IRQHandler_EI0    16    // EXT0 of MB91460 Series
#pragma intvect IRQHandler_RLT0   33    // RLT0 of MB91460 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 1

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 Port25, during run mode this Port is counting.

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

// initialize external int 0
void Init_extint0 (void)
{
    DDR24_D0 = 0;      // Port 24_0 Data direction as input
    PFR24_D0 = 1;      // Enable Port 24_0 as an external interrupt input
    ENIRO_EN0 = 0;     // disable ext int 0
    ELVR0_LA0 = 1;
    ELVR0_LB0 = 1;     // LB0,LA0 = 11 -> falling edge
}

// initialize external int 1
void Init_extint1 (void)
{
    DDR24_D1 = 0;      // Port 24_1 Data direction as input
    PFR24_D1 = 1;      // Enable Port 24_1 as an external interrupt input
    ENIRO_EN1 = 0;     // disable ext int 1
    ELVR0_LA1 = 1;
    ELVR0_LB1 = 1;     // LB1,LA1 = 11 -> falling edge
}
```

```

#define RUNMODE      0
#define STOPREQUEST  1
volatile unsigned int status, i;
void main (void)
{
    InitIrqLevels();
    __set_il(31);      // allow all levels
    __EI();            // globally enable interrupts
    PORTEN = 0x3;      /* enable I/O Ports */
                      /* This feature is not supported by MB91V460A */
                      /* For all other devices the I/O Ports must be enabled*/
    PDR25 = 0x00;      // clear port data
    DDR25 = 0xFF;      // set port 0 to output
    Init_extint0();
    Init_extint1();
    EIRR0_ER1 = 0;     // clear ext. int 1 request flag
    ENIR0_EN1 = 1;     // enable ext. int1
    status = RUNMODE;

    while(1)
    {
        HWWD_CL = 0;
        if (status == STOPREQUEST)    // stop mode request
        {
            PDR25 = 0x0F;
            STCR_STOP = 1;             // goto stop mode
            STCR_HIZ = 0;              // preserve output pin state
        }
        else                          // run mode
        {
            for (i = 0; i < 50000; i++) // wait loop
            {
                __asm(" NOP");
                __asm(" NOP");
                HWWD_CL = 0;
            }
            PDR25++;                  // 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)
{
    . . .

    ICR00 = 23; /* Priority Level 23 for External Interrupt 0 & 1 */

    __interrupt void IRQ_extint0 (void); /* Prototype EXT0
    __interrupt void IRQ_extint1 (void); /* Prototype EXT1
    . . .

#pragma intvect IRQ_extint0    16    // EXT0 of MB96340 Series
#pragma intvect IRQ_extint1    17    // 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)
{
  DDR24_D0 = 0;      // Port 24_0 Data direction as input
  PFR24_D0 = 1;      // Enable Port 24_0 as an external interrupt input

  ENIRO_EN0 = 0;     // disable ext int 0
  ELVRO_LA0 = 1;
  ELVRO_LB0 = 1;     // LB0,LA0 = 11 -> falling edge
  EIRRO_ER0 = 0;     // clear ext. int 0 request flag
}

// Request stop mode
void Request_stop (void)
{
  __DI();            // globally disable interrupts
  EIRRO_ER0 = 0;     // clear ext. int 0 request flag
  ENIRO_EN0 = 1;     // enable ext int 0
  STCR_STOP = 1;     // goto stop mode
  STCR_HIZ = 0;      // preserve output pin state
  EIRRO_ER0 = 0;     // clear ext. int 0 request flag after wakeup
  __EI();            // globally enable 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-microcontrollers>

The software examples related to this application note is:

*91460\_irq\_ext*

*91460\_irq\_ext\_stop\_wakeup*

It can be found on the following Internet page:

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

## Document History

Document Title: AN205264 - FR, MB91460, External Interrupts

Document Number: 002-05264

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	02/14/2008	V1.0 MPi First Version
			04/23/2008	V1.1 MSt Chapter 3 External Interrupt Timing, Valid edge detection timing added
			09/30/2010	V1.2 RSchum Chapter 4: Typo in register naming corrected ELVRL0 -> ELVR0 Disclaimer changed Company name changed
*A	5085785	NOFL	04/06/2016	Converted Spansion Application Note "MCU-AN -300056-E-V12" to Cypress format
*B	5869301	AESATMP9	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	<a href="http://cypress.com/arm">cypress.com/arm</a>
Automotive	<a href="http://cypress.com/automotive">cypress.com/automotive</a>
Clocks & Buffers	<a href="http://cypress.com/clocks">cypress.com/clocks</a>
Interface	<a href="http://cypress.com/interface">cypress.com/interface</a>
Internet of Things	<a href="http://cypress.com/iot">cypress.com/iot</a>
Memory	<a href="http://cypress.com/memory">cypress.com/memory</a>
Microcontrollers	<a href="http://cypress.com/mcu">cypress.com/mcu</a>
PSoC	<a href="http://cypress.com/psoc">cypress.com/psoc</a>
Power Management ICs	<a href="http://cypress.com/pmic">cypress.com/pmic</a>
Touch Sensing	<a href="http://cypress.com/touch">cypress.com/touch</a>
USB Controllers	<a href="http://cypress.com/usb">cypress.com/usb</a>
Wireless Connectivity	<a href="http://cypress.com/wireless">cypress.com/wireless</a>

### 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](http://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, 2008-2017. 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](http://cypress.com). Other names and brands may be claimed as property of their respective owners.