# Interrupts example in Kinetis Design Studio with FDRM-K64f

**By:**
**Augusto Panecatl**
**Paul Garate**

### Description

In this document you will find a detailed step by step guide of how to configure interrupts on Kinetis K devices using Kinetis Design Studio using a Switch 2, the Red and Blue Led included in the FRDM-K64F120 evaluation board.

## 1. Pin Control Register configuration

Once the clock gating has been setup we need to configure the pin function using the multiplexor, according to the FRDM-K64's schematic the Red and Blue led are assigned to pins:

| LED | K64 |
|---|---|
| RED | **PTB22**/SPI2_SOUT/FB_AD29/CMP2_OUT |
| BLUE | **PTB21**/SPI2_SCK/FB_AD30/CMP1_OUT |
| GREEN | **PTE26**/ENET_1588_CLKIN/UART4_CTS_b/RTC_CLKOUT/USB0_CLKIN |

Switch 2 is assigned to:

| Switch | GPIO Function |
|---|---|
| SW2 | PTC6/SPI0_SOUT/PD0_EXTRG/I2S0_RX_BCLK/FB_AD9/I2S0_MCLK/LLWU_P10 |
| SW3 | PTA4/FTM0_CH1/NMI_b/LLWU_P3 |

The next step is to configure the Pin Control Register to define the pin's functions:

**Pin Control Register n (PORT*x*_PCR*n*)**

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | | | | | | | ISF | 0 | | | | IRQC | | | |
| W | | | | | | | | w1c | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | LK | 0 | | | | MUX | | | 0 | DSE | ODE | PFE | 0 | SRE | PE | PS |
| W | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | * | * | * | 0 | * | 0 | * | 0 | * | * | * |

The only bits we need to configure in the Red and Blue led are those assigned to the MUX field, in order to set them as GPIOs. As for Switch 2 you need to configure the ISF, IRQC and MUX fields.

| | |
|---|---|
| 24<br>ISF | Interrupt Status Flag<br>The pin interrupt configuration is valid in all digital pin muxing modes.<br>0 Configured interrupt is not detected.<br>1 Configured interrupt is detected. If the pin is configured to generate a DMA request, then the corresponding flag will be cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic one is written to the flag. If the pin is configured for a level sensitive interrupt and the pin remains asserted, then the flag is set again immediately after it is cleared. |
| 19–16<br>IRQC | Interrupt Configuration<br>The pin interrupt configuration is valid in all digital pin muxing modes. The corresponding pin is configured to generate interrupt/DMA request as follows:<br>0000 Interrupt/DMA request disabled.<br>0001 DMA request on rising edge.<br>0010 DMA request on falling edge.<br>0011 DMA request on either edge.<br>0100 Reserved.<br>1000 Interrupt when logic zero.<br>1001 Interrupt on rising edge.<br>1010 Interrupt on falling edge.<br>1011 Interrupt on either edge.<br>1100 Interrupt when logic one.<br>Others Reserved. |
| 10–8<br>MUX | Pin Mux Control<br>Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot.<br>The corresponding pin is configured in the following pin muxing slot as follows:<br>000 Pin disabled (analog).<br>001 Alternative 1 (GPIO).<br>010 Alternative 2 (chip-specific).<br>011 Alternative 3 (chip-specific).<br>100 Alternative 4 (chip-specific).<br>101 Alternative 5 (chip-specific).<br>110 Alternative 6 (chip-specific).<br>111 Alternative 7 (chip-specific). |

First we configure the pin function as a GPIO by writing 001 to the MUX field.

The IRQC field is used to enable the interrupt function as well as selecting the type of interrupt detection. The code will be performing a continuous task, when Switch2 is pressed the event will be detected as an interrupt to the program, an interrupt routine will be executed and once finished the main code will continue its execution at the point it was interrupted.

According to the board's schematic we need to configure the following pins as GPIOs: ***R= PortB pin 22, B= PortB pin 21, Switch 2= PortC pin 6***.

In configuration register ***PORTx_PCRn*** "***x***" corresponds to the port whilst "***n***" corresponds to the pin

```
PORTB_PCR21 = 0x100;            /*Blue Led, configured as Alternative 1 (GPIO)*/
PORTB_PCR22 = 0x100;            /*Red Led, configured as Alternative 1 (GPIO)*/
PORTC_PCR6  = 0x90100;          /*PORTC_PCR6: ISF=0,IRQC=9,MUX=1 */
PORTA_PCR4  = 0x100;            /*Changing the NMI to GPIO*/
```

*(0x100 = 100000000)
*(0x90100 = 10010000000100000000)
*The 4th pin of Port A is configured as GPIO to avoid Switch 3 triggering the NMI.

## 2. Setting up the port data direction

Now that the clock gating and the pin functions have been configured we need to establish the pin direction either as Input or Output, this is configured in the ***GPIOx_PDDR*** register:

### 55.2.6 Port Data Direction Register (GPIOx_PDDR)

The PDDR configures the individual port pins for input or output.

Address: Base address + 14h offset

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R<br>W | PDD | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Field | Description |
|---|---|
| 31–0<br>PDD | Port Data Direction<br><br>Configures individual port pins for input or output.<br><br>0 Pin is configured as general-purpose input, for the GPIO function.<br>1 Pin is configured as general-purpose output, for the GPIO function. |

As in the previous registers the "**x**" corresponds to the port, and each of the 32 bits in the register corresponds to a single port pin. The pins can either be setup one by one or the whole port can be configured in a single write.

The two pins connected to the leds must be set as outputs and the switch as an input:

```
GPIOB_PDDR |= (1 << 21);        /*Setting the bit 21 of the port B as Output*/
GPIOB_PDDR |= (1 << 22);        /*Setting the bit 22 of the port B as Output*/
GPIOC_PDDR |= (0 << 6);         /*Setting the bit 6 of the port C as Input*/
```

In this case we configured the pins by shifting the value to the corresponding bit, ***1*** or ***0*** shifted "***n***" number of places.

## 3. Enabling the interrupt and clearing the status flag

**Interrupt Status Flag Register (PORTx_ISFR)**

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | ISF | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| W | w1c | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**PORTx_ISFR field descriptions**

| Field | Description |
|---|---|
| 31–0<br>ISF | Interrupt Status Flag<br>Each bit in the field indicates the detection of the configured interrupt of the same number as the field.<br>0 Configured interrupt is not detected.<br>1 Configured interrupt is detected. If the pin is configured to generate a DMA request, then the corresponding flag will be cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic 1 is written to the flag. If the pin is configured for a level sensitive interrupt and the pin remains asserted, then the flag is set again immediately after it is cleared. |

It is necessary to clear the interrupt status flag:

```
PORTC_ISFR = PORT_ISFR_ISF(0x40);          /* Clear interrupt status flag */
```

Now the status flag is clear we need to enable the interrupt vector, to do that we first need to know which vector is assigned to the port in which the pin we configured as an interrupt is located; you can find it in table 3-5 (Interrupt vector assignment) and search for the right IRQ.

**Table 3-5. Interrupt vector assignments**

| Address | Vector | IRQ[1] | NVIC non-IPR register number [2] | NVIC IPR register number [3] | Source module | Source description |
|---|---|---|---|---|---|---|
| **Non-Core Vectors** | | | | | | |
| 0x0000_012C | 75 | 59 | 1 | 14 | Port control module | Pin detect (Port A) |
| 0x0000_0130 | 76 | 60 | 1 | 15 | Port control module | Pin detect (Port B) |
| 0x0000_0134 | 77 | 61 | 1 | 15 | Port control module | Pin detect (Port C) |
| 0x0000_0138 | 78 | 62 | 1 | 15 | Port control module | Pin detect (Port D) |
| 0x0000_013C | 79 | 63 | 1 | 15 | Port control module | Pin detect (Port E) |

Since Switch2 is assigned to a pin in PORTC, thus the IRQ we need to enable according to the table is vector 61.

Here is the function to enable the IRQ:

```
void NVIC_DisableIRQ(IRQn_Type IRQn)
```

In the function we need to write the IRQ instead of "**IRQn_Type IRQn**"

```
NVIC_EnableIRQ(PORTC_IRQn);                    *PORTC_IRQn = 61
```

## 4. Code

In the main code the Red led will be blinking in an infinite For cycle. The code will keep running until Switch 2 is pressed and triggers the interrupt.

```
for (;;)
{
        GPIOB_PTOR |= (1 << 22);          /*Red LED blinking*/
        DelayFunction();
}
```

The next code, interrupt routine; will execute when the interrupt is triggered. It interrupts the blinking of the red led, then toggles the blue led a single time, once done it returns to the main routine.

```
void PORTC_IRQHandler(void)
{
      DelayFunction();
      GPIOB_PSOR |= (1 << 22);                    /*Turn Off Red Led*/
      GPIOB_PCOR |= (1 << 21);                    /*Turn On Blue Led*/
      DelayFunction();
      GPIOB_PSOR |= (1 << 21);                    /*Turn Off Blue Led*/
      DelayFunction();

      PORTC_ISFR = PORT_ISFR_ISF(0x40);           /* Clear interrupt status flag */
}
```

Here is the Delay Function, which increments the variable "**cnt**" until it gets the value of `1000000`

```
void DelayFunction(void)
{
      int cnt;
      for(cnt=0; cnt<1000000; cnt++)
      {

      }
}
```