
Getting Started with GPIO on PIC18

Introduction

Author: Cristina Ionescu, Microchip Technology Inc.

An embedded system is capable of exchanging stimuli with the outside world using General-Purpose Input/Output (GPIO) ports. GPIO pins are clustered in PORTs and the PIC18 devices provide multiple PORT modules.

This technical brief explains the concepts behind PORT modules and their functionality on the PIC18 family of microcontrollers by implementing the use cases presented below.

- **GPIO Read/Write Example:**
This example shows how to read an input pin value, changed by pressing a button, and how to set an output pin value in order to turn an LED on and off.
- **Using GPIO Interrupts:**
This example shows how to enable the Interrupt-on-Change (IOC). The IOC is configured to be triggered on the falling edge detected on the desired input pin.
- **Wake-Up from Sleep:**
This example shows how to reduce power consumption by enabling Sleep mode on the device and waking it up using the IOC, with a button controlling the input pin value.

There are two different implementations for each use case that have the same functionalities: one code generated with [MPLAB® Code Configurator \(MCC\)](#) and one bare metal code. The MCC-generated code offers hardware abstraction layers that ease the use of the code across different devices from the same family. The bare metal code is easier to follow, allowing a fast ramp-up on the use case associated code.



View Code Examples on GitHub

Click to browse repositories

Note: The examples in this technical brief have been developed using the PIC18F47Q10 Curiosity Nano development board. The PIC18F47Q10 pin package present on the board is QFN.

Table of Contents

Introduction.....	1
1. Peripheral Overview.....	3
2. GPIO Read/Write Example.....	6
3. Using GPIO Interrupts.....	9
4. Wake-Up from Sleep.....	12
5. References.....	16
6. Revision History.....	17
The Microchip Website.....	18
Product Change Notification Service.....	18
Customer Support.....	18
Microchip Devices Code Protection Feature.....	18
Legal Notice.....	19
Trademarks.....	19
Quality Management System.....	20
Worldwide Sales and Service.....	21

1. Peripheral Overview

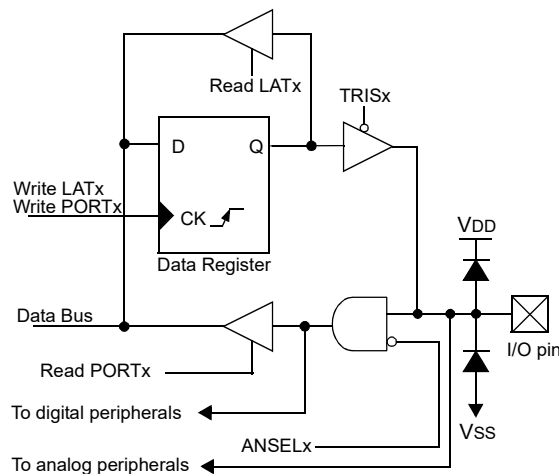
Each of the Input/Output (I/O) available ports is provided with eight registers to control their operation:

- Port registers (PORTx) allow the user to read the logic levels on the pins. For example, reading from PORTA register returns the actual I/O pin values. Pins configured as output can also be driven by writing to PORTx.
- Output Latch registers (LATx) can be used for both writes to the port and read-modify-write operations on the values that the I/O pins are driving. Writes to LATx are equivalent with writes to the corresponding PORTx register. Reads from LATx return register values, not I/O pin values. This prevents inadvertent modification of pins that are configured both as inputs and outputs by the software.
- Tri-State Control registers (TRISx) allow the user to configure the direction of the pins. When a bit from register is set (logic '1'), the port output driver is disabled for that pin. Otherwise, if a pin is cleared (logic '0') the port output driver is enabled.
- Analog Select registers (ANSELx) allow the user to enable/disable the digital input buffers or the ST and TTL input buffers for specific pins. Pins configured for analog inputs will have the digital input buffers disabled to reduce potential shoot-through currents, which can increase the operating current.
- Weak Pull-Up registers (WPUx) are used to enable or disable weak pull-up on specific pins. This option allows the connection of one or more open collector drives to an input pin without the need for an external pull-up resistor. The option also allows the disabling of the pull-ups, when not needed, to minimize the current consumption.
- Input Level Control registers (INLVx) allow the user to configure either ST or TTL input thresholds to be used for port reads and Interrupt-on-Change.
- Slew Rate Control registers (SLRCONx) allow the user to select a lower pin slew rate or configure it at a maximum rate. Slew Rate reduction is useful for limiting the EMI emissions of the pins during state transitions.
- Open-Drain Control registers (ODCONx). The output pins can be configured as open collector outputs (drive-only sink outputs) signals, or both source and sink outputs. The desired setting can be configured using the Open-Drain Configuration registers.

Most port pins share functions with device peripherals, both analog and digital. In general, when a peripheral is enabled on a port pin, that pin cannot be used as a General Purpose output. However, the pin can still be read.

A simplified model of a generic I/O port, without interfaces to other peripherals, is shown below.

Figure 1-1. Generic I/O Port Operation



Individual pins can be independently configured to generate an interrupt. The source of the interrupt can be either a rising or falling edge of the signal on the port. The Interrupt-on-Change can also be used to wake the device from Sleep mode.

The GPIO registers and the configurations they provide are described below, along with some basic settings using these registers.

- **Set the Direction of an I/O Pin**

In order to configure an I/O pin as an input or an output, the TRISx register is used. A logic '1' sets the corresponding bit as an input, while a logic '0' sets the bit as an output.

For example, in order to set TRISA4 pin as an output without changing the rest of the register, the following code must be used:

```
TRISAbits.TRISA4 = 0; /* Configure the TRISA4 pin as output */
```

The following code configures the TRISA3 pin as an input:

```
TRISAbits.TRISA3 = 1; /* Configure the TRISA4 pin as input*/
```

- **Drive Output Pins Values**

To set the output values, in order to drive the output of the pins high or low, either PORTx or LATx registers can be used.

There is, however, an advantage to using the LATx register. For example, writing a value to the PORTx register can be translated into a read-modify-write low level operation. This can cause a problem when a pin is configured sometimes as an input or as an output. If a read-modify-write is executed when the pin is an input, the current state of the pin (as an input) is copied into the output latch. When the pin is again set as an output, the state of that output is unknown.

To avoid this issue, the user must write to the LATx registers and the values will be written directly to the PORTx registers. By reading from the LATx registers, register values will be returned.

For example, to drive the RA4 pin high, the following code is recommended:

```
LATABits.LATA4 = 1; /* Drive the output high */
```

To drive the RA4 pin low, the following code is recommended:

```
LATABits.LATA4 = 0; /* Drive the output low */
```

- **Enable/Disable the Digital Input Buffers**

If an I/O pin is used as a digital input, the ANSELxn corresponding bit must be cleared. This will allow the digital input signals to reach the digital peripherals, as shown in [Figure 1-1](#). If the respective pin is used as an analog input, the ANSELxn corresponding bit must be set in order to limit current consumption.

For example, to configure the RA0 pin as a digital input and have the RA1 pin as an analog input, the following code must be used (considering the pins are already configured as inputs):

```
ANSELAbits.ANSELA0 = 0; /* ST and TTL input buffers are enabled */
ANSELAbits.ANSELA1 = 1; /* Digital input buffers are disabled*/
```

- **Enable Weak Pull-up**

When using an I/O pin as an input, the pin state must be reliable. In order to avoid an input pin to enter a floating state, the user must enable the weak pull-up. This way, the pin state will be a logic '1' by default since it will be internally connected to the source voltage, using a weak pull-up resistance value. The main use for WPU is to eliminate the current draw of a pull-up on a grounded input. Since the WPUs can be enabled/disabled in code, the pull-up can be turned on only when it is necessary to read the input. This saves current consumption that can appear when using a fixed external pull-up resistor.

This is an important feature when using, for example, a button and without using an external pull-up resistor.

The weak pull-up can be enabled using the WPUx register, as presented in the following code:

```
WPUAbits.WPUA0 = 1; /* Enable weak pull-up */
```

- **Configure the Open Drain Control**

The PIC® GPIO pins can be configured as sink-only output, or to both source and sink current. The Open-Drain Control Register (ODCONx) controls the open-drain feature of the port. When an ODCONx bit is set, the corresponding port output becomes an open-drain driver capable of sinking current only. When an ODCONx bit is cleared, the corresponding port output pin is the standard push-pull drive capable of sourcing and sinking current.

Some advantages to this mode include the following:

- Multiple outputs can be tied together in a logic OR configuration.
- There are bi-directional open collector buses that both send and receive through a single pin.
- It is possible to create a 555 timer IC using comparators, a CLC Flip Flop and the OD output mode.

A code example on how to use the ODCONx registers is presented below (assuming the pins are already configured as output pins)

```
ODCONAbits.ODCA4 = 1; /* Sink current only */
ODCONAbits.ODCA5 = 0; /* Source and sink current */
```

- **Configure the Slew Rate**

The SLRCONx registers change the slew rate option for each pin. It allows the user to limit the slew rate, or to drive a slew rate at the maximum possible rate. Decreasing the slew rate reduces the amount of noise induced by capacitive coupling to adjacent signals (a capacitor is formed when two conductors are in proximity).

A code example on how to use the SLRCONx registers is presented below (assuming the pins are already configured as output pins):

```
SLRCONAbits.SLRA4 = 1; /* Slew rate is limited */
SLRCONAbits.SLRA5 = 0; /* Maximum slew rate */
```

- **Configure the Input Threshold Voltage**

The INLVLx registers control the input voltage threshold for each of the available PORTx input pins. A selection between the Schmitt Trigger CMOS and the TTL compatible thresholds is available.

The input threshold is important in determining the value of a read of the PORTx register and the level at which an interrupt-on-change occurs, if that feature is enabled.

Table 1-1. Input Level Configurations

	TTL Levels	Schmitt Trigger (ST)
Logic '0'	$V_{IN} < 0.8V$	$V_{IN} < 0.2V_{DD}$
Logic '1'	$V_{IN} > 2.0V$	$V_{IN} > 0.8V_{DD}$

TTL uses fixed voltages but has a lower noise margin (1.2V), while Schmitt Trigger inputs are proportional to the supply voltage (2V for 3.3V supply and 3V for 5V supply).

A code example on how to use the SLRCONx registers is presented below (assuming the pins are already configured as input pins):

```
/* ST input used for port reads and interrupt-on-change */
INLVLABits.INLVLA0 = 1;
/* TTL input used for port reads and interrupt-on-change */
INLVLABits.INLVLA1 = 0;
```

2. GPIO Read/Write Example

This example describes a basic configuration of an input pin (connected to a button and an output pin) used to turn an LED on and off. To implement this example, one PORT pin is configured as a digital input and another pin is configured as an output.

The input pin is connected to a button and the output pin is connected to an LED. The value of the input pin is read continuously through polling.

There are two possibilities:

- The read value is logic '1' (the input pin is pulled high), meaning the button is released. The microcontroller drives the output high, therefore the LED is off.
- The read value is logic '0' (the input pin is pulled low), meaning the button is pressed. The microcontroller drives the output low and the LED is on.

For this example, the RE2 pin is configured as input and connected to a button while the RE0 pin is configured as output and connected to an LED.

To achieve the functionality described by the use case, the following actions will need to be performed:

- System clock initialization
- Port initialization

Note: The weak pull-up is enabled for the input pin so that it does not enter a floating state.

2.1 MCC Generated Code

To generate this project using MPLAB Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (for details on how to install the MCC plug-in, see [References](#)).
3. Go to *Project Resources* → *System* → *System Module* and do the following configurations:
 - Oscillator Select: HFINTOSC
 - HF Internal Clock: Select 1_MHz
 - Clock Divider: 1
 - In the Watchdog Timer Enable field in **WWDT** tab, ensure '**WDT Disabled**' is selected
 - In the **Programming** tab, ensure **Low-voltage Programming Enable** is checked
4. Open the *Pin Manager* → *Grid View* window, select UQFN40 in the MCU package field and make the following pin configurations:

Figure 2-1. Pin Mapping

Package:	UQFN40		Pin No:	17	18	19	20	21	22	29	28	8	9	10	11	12	13	14	15	30	31	32	33	38	39	40	1	34	35	36	37	2	3	4	5	23	24	25	16
				Port A ▼							Port B ▼							Port C ▼							Port D ▼							Port E ▼							
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	
OSC	CLKOUT	output																																					
Pin Module ▼	GPIO	input																																					
	GPIO	output																																					
RESET	MCLR	input																																					

- Go to *Project Resources* → *System* → *Pin Module* and do the following configurations:
 - RE2 Custom Name: SW0
 - RE2 WPU: checked
 - RE0 Custom Name: LED0
 - RE0 Output: checked

The names used in this examples are 'LED0' for the RE0 pin and 'SW0' for the RE2 pin.

- In the *Project Resources* window, press the **Generate** button so that the MCC can generate the code capable of configuring the microcontroller as specified.

- In the `main.c` file generated by MCC, change or add the following code:

```
while (1)
{
    if(SW0_GetValue())
    {
        LED0_SetHigh();
    }
    else
    {
        LED0_SetLow();
    }
}
```



View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

2.2 Bare Metal Code

The necessary code and functions to implement the example are analyzed in this section.

The first step is to configure the microcontroller to disable the Watchdog Timer and enable the Master Clear external pin.

```
#pragma config WDTE = OFF      /* disable Watchdog */
#pragma config LVP = ON        /* Low voltage programming enabled */
```

The main clock of the microcontroller must be initialized. The following function initializes the system clock to have as input clock the HFINTOSC oscillator and to run at 1 MHz:

```
static void CLK_Initialize(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
    /* set HFFRQ to 1 MHz */
    OSCFRQbits.HFFRQ = 0;
}
```

The peripheral initialization must be added to the project. The PORT registers allow the user to configure a pin as input or output, enable the weak pull-up and enable the digital input buffer for the desired pins.

The RE0 pin is configured as output and the RE2 pin is configured as digital input, with weak pull-up enabled. The desired configuration for this example translates to the following code:

```
static void PORT_Initialize(void)
{
    /* Set RE0 (LED) pin as output */
    TRISEbits.TRISE0 = 0;
    /* Set RE2 (button) pin as input*/
    TRISEbits.TRISE2 = 1;

    /* Enable weak pull-up for pin RE2 (button) */
    WPUEbits.WPUE2 = 1;

    /* Enable digital input buffer for pin RE2 (button) */
    ANSELEbits.ANSELE2 = 0;
}
```

After initializing the PORT peripheral, the microcontroller is constantly checking the input pin value (or polling). As described above, if the pin value is '0' (the button is pressed), the microcontroller will turn on the LED by driving the output pin low. Otherwise, if the pin value is '1' (the button is released), the microcontroller will turn off the LED by driving the output pin high. This translates to the following code:

```
while (1)
{
    if(PORTEbits.RE2)    /* Read the input pin value */
    {
        LATEbits.LATE0 = 1; /* Turn off LED */
    }
    else
    {
        LATEbits.LATE0 = 0; /* Turn on LED */
    }
}
```



View the PIC18F47Q10 Code Example on GitHub

[Click to browse repositories](#)

3. Using GPIO Interrupts

This example consists of toggling the output pin value when an IOC is detected on the input pin. An interrupt can be generated by detecting a signal at the port pin that has either a rising or falling edge.

In this example, the interrupt is configured to be triggered on the falling edge of the input signal (the value is changed from logic '1' to logic '0'), in order to generate the interrupt when the button is pressed. The interrupt routine consists of toggling the output pin value, which causes the LED to turn on and off.

For this example, the RA0 pin is configured as input with IOC enabled on falling edge and connected to a button while the RE0 pin is configured as output and connected to an LED.

To achieve the functionality described by the use case, the following actions will need to be performed:


































- System clock initialization
- Port initialization
- Interrupts initialization

3.1 MCC Generated Code

To generate this project using MPLAB Code Configurator (MCC), follow the next steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (for details on how to install the MCC plug-in, see [References](#)).
3. Go to Project *Resources* → *System* → *System Module* and do the following configurations:
 - Oscillator Select: HFINTOSC
 - HF Internal Clock: Select 1_MHz
 - Clock Divider: 1
 - In the Watchdog Timer Enable field in **WWDT** tab, ensure '**WDT Disabled**' is selected
 - In the **Programming** tab, ensure **Low-voltage Programming Enable** is checked
4. Open the *Pin Manager* → *Grid View* window, select UQFN40 in the MCU package field and make the following pin configurations:

Figure 3-1. Pin Mapping

Package:	UQFN40	Pin No:	17	18	19	20	21	22	29	28	8	9	10	11	12	13	14	15	30	31	32	33	38	39	40	1	34	35	36	37	2	3	4	5	23	24	25	16
			Port A ▼							Port B ▼							Port C ▼							Port D ▼							Port E ▼							
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3
OSC	CLKOUT	output																																				
Pin Module ▼	GPIO	input																																				
	GPIO	output																																				
RESET	MCLR	input																																				

- Go to *Project Resources* → *System* → *Pin Module* → *Easy Setup* and do the following configurations:
 - RA0 Custom Name: RA0
 - RA0 WPU: checked
 - RA0 IOC: negative
 - RE0 Custom Name: LED0
 - RE0 Output: checked
- In the *Project Resources* window, press the **Generate** button so that the MCC can generate the code capable to configure the microcontroller as specified.
- In the `main.c` file which has been generated by MCC, change or add the following code:
 - Enable the global interrupts
 - Add the IOC0 interrupt function
 - Set the IOC0 interrupt handler initializer

```
static void IOC0_ISR()
{
```

```

        if(IOCAFbits.IOCAF0 == 1)
        {
            LED0_Toggle();
        }
    }

    void main(void)
    {

        // Enable the Global Interrupts
        INTERRUPT_GlobalInterruptEnable();

        // Disable the Global Interrupts
        //INTERRUPT_GlobalInterruptDisable();

        // Enable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptEnable();

        // Disable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptDisable();

        IOCAF0_SetInterruptHandler(IOC0_ISR);

        while (1)
        {
            // Add your application code
        }
    }

```



View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

3.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section.

The first step is to configure the microcontroller to disable the Watchdog Timer and enable the Master Clear external pin.

```

#pragma config WDTE = OFF           /* disable Watchdog */
#pragma config LVP = ON             /* Low voltage programming enabled */

```

The main clock of the microcontroller must be configured. The following function initializes the system clock to have as input clock the HFINTOSC oscillator and to run at 1 MHz:

```

static void CLK_Initialize(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
    /* set HFFRQ to 1 MHz */
    OSCFRQbits.HFFRQ = 0;
}

```

For this example, the RA0 pin from PORTA is configured as an input pin and connected to a button in order to generate the desired interrupt. The PORT initialization function is listed below:

```

static void PORT_Initialize(void)
{
    TRISEbits.TRISE0 = 0;    /* Set RE0 pin as output (LED) */
    TRISAbits.TRISA0 = 1;    /* Set RA0 pin as input */
}

```

```
ANSELAbits.ANSELA0 = 0; /* Enable Digital Input buffers for RA0 */  
  
WPUAbits.WPUA0 = 1;      /* Enable weak pull-up for RA0 */  
}
```

The next step is to initialize the IOC. For this example, the IOC is configured to be triggered on the falling edge of the input signal. The code is listed below:

```
static void IOC_Initialize(void)  
{  
    IOCAFbits.IOCAF0 = 0; /* Clear interrupt flag */  
  
    IOCANbits.IOCAN0 = 1; /* Enable IOC on negative change */  
  
    PIE0bits.IOCIE = 1;   /* Enable IOC interrupt */  
}
```

In order to work with interrupts, the user must enable the global interrupts by setting the corresponding bit in the INTCON register. The function is listed below:

```
static void INTERRUPT_Initialize(void)  
{  
    INTCONbits.GIE = 1; /* Enable global interrupts */  
}
```

The IOC Interrupt Service Routine (ISR) function that will be called when receiving an IOC interrupt is listed below:

```
static void IOC0_ISR (void)  
{  
    /* Clear the interrupt flag */  
    IOCAFbits.IOCAF0 = 0;  
    /* Toggle the LED */  
    LATEbits.LATE0 = ~LATEbits.LATE0;  
}
```

In this function, the IOC interrupt flag must be cleared and, in this instance, the output value is toggled to turn an LED on or off every time an interrupt is triggered.

An interrupt handler is needed to manage all the desired interrupts. For this example, the interrupt handler will call the IOC0_ISR interrupt routine function as listed below:

```
void __interrupt() INTERRUPT_InterruptManager (void)  
{  
    if(IOCAFbits.IOCAF0) /* Check the interrupt flag */  
    {  
        IOC0_ISR();  
    }  
}
```



View the PIC18F47Q10 Code Example on GitHub

[Click to browse repositories](#)

4. Wake-Up from Sleep

This use case presents how to put the microcontroller in Sleep mode and wake it up using the IOC. When a falling edge is detected on the input pin, the device will wake up. The exit from Sleep mode is signaled using an LED. If the microcontroller wakes up, it will turn the LED on, keep the LED on for 100 ms and then turn it off. The Sleep mode is used to provide power efficiency.

For this example, the RA0 pin is configured as input with IOC enabled on falling edge and connected to a button while the RE0 pin is configured as output and connected to an LED.

To achieve the functionality described by the use case, the following actions will need to be performed:

- System clock initialization
- Port initialization
- Interrupts initialization

4.1 MCC Generated Code

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open MCC from the toolbar (for details on how to install the MCC plug-in, see [References](#)).
3. Go to Project *Resources* → *System* → *System Module* and do the following configurations:
 - Oscillator Select: HFINTOSC
 - HF Internal Clock: Select 1_MHz
 - Clock Divider: 1
 - In the Watchdog Timer Enable field in **WWDT** tab, ensure 'WDT Disabled' is selected
 - In the **Programming** tab, ensure **Low-voltage Programming Enable** is checked
4. Open the *Pin Manager* → *Grid View* window, select UQFN40 in the MCU package field, and make the following pin configurations:

Figure 4-1. Pin Mapping

Package:	UQFN40		Pin No:		17	18	19	20	21	22	29	28	8	9	10	11	12	13	14	15	30	31	32	33	38	39	40	1	34	35	36	37	2	3	4	5	23	24	25	16
					Port A ▼							Port B ▼							Port C ▼							Port D ▼							Port E ▼							
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3		
OSC	CLKOUT	output																																						
Pin Module ▼	GPIO	input																																						
	GPIO	output																																						
RESET	MCLR	input																																						

- Go to *Project Resources* → *System* → *Pin Module* → *Easy Setup* and do the following configurations:
 - RA0 Custom Name: RA0
 - RA0 WPU: checked
 - RA0 IOC: negative
 - RE0 Custom Name: LED0
 - RE0 Output: checked
- In the *Project Resources* window, press the **Generate** button so that the MCC can generate the code capable of configuring the microcontroller as specified.
- In the `main.c` file generated by MCC, add the following code:
 - Enable the global interrupts
 - Add the IOC0 interrupt function
 - Set the IOC0 interrupt handler initializer
 - Put the device in Sleep

```
#define DELAY_MS 100

static void IOC0_ISR()
{
```

```

        if(IOCAFbits.IOCAF0 == 1)
        {
            ;
        }
    }

    void main(void)
    {
        // Initialize the device
        SYSTEM_Initialize();

        // Enable the Global Interrupts
        INTERRUPT_GlobalInterruptEnable();

        // Disable the Global Interrupts
        //INTERRUPT_GlobalInterruptDisable();

        // Enable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptEnable();

        // Disable the Peripheral Interrupts
        //INTERRUPT_PeripheralInterruptDisable();

        IOCAF0_SetInterruptHandler (IOC0_ISR);

        while (1)
        {
            LED0_SetLow();
            delay_ms(DELAY_MS);
            LED0_SetHigh();

            SLEEP();
        }
    }

```



View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4.2 Bare Metal Code

The necessary code and functions to implement the presented example are analyzed in this section.

The first step is to configure the microcontroller to disable the Watchdog Timer and enable the Master Clear external pin.

```

#pragma config WDTE = OFF      /* disable Watchdog */
#pragma config LVP = ON       /* Low voltage programming enabled */

```

The `_XTAL_FREQ` macro needs to be defined in order to use the `__delay_ms` function. Also, the `_XTAL_FREQ` value must be the same as the system clock.

```

#define _XTAL_FREQ 1000000UL

```

The main clock of the microcontroller must be configured. The following function initializes the system clock to have as input clock the HFINTOSC oscillator and to run at 1 MHz:

```

static void CLK_Initialize(void)
{
    /* set HFINTOSC Oscillator */
    OSCCON1bits.NOSC = 6;
}

```

```
/* set HFFRQ to 1 MHz */
OSCFRQbits.HFFRQ = 0;
}
```

For this example, the RA0 pin from PORTA is configured as a digital input and connected to a button in order to generate the desired interrupt. The RE0 pin is configured as output and connected to an LED. The PORT initialization function is listed below:

```
static void PORT_Initialize(void)
{
    TRISEbits.TRISE0 = 0; /* Set RE0 pin as output (LED) */
    TRISAbits.TRISA0 = 1; /* Set RA0 pin as input */

    ANSELbits.ANSELA0 = 0; /* Enable Digital Input buffers for RA0 */

    WPUAbits.WPUA0 = 1; /* Enable weak pull-up for RA0 */
}
```

The next step is to initialize the IOC. For this example, the IOC is configured to be triggered on the falling edge of the input signal. The code is listed below:

```
static void IOC_Initialize(void)
{
    IOCAFbits.IOCAF0 = 0; /* Clear interrupt flag */

    IOCANbits.IOCAN0 = 1; /* Enable IOC on negative change */

    PIE0bits.IOCIE = 1; /* Enable IOC interrupt */
}
```

In order to work with interrupts, the users must add an interrupt initialization function that activates the global and the peripheral interrupts. The function is listed below.

```
static void INTERRUPT_Initialize(void)
{
    INTCONbits.GIE = 1; /* Enable global interrupts */
}
```

The IOC ISR function that will be called when receiving an IOC interrupt is listed below: In this function, the only action needed in this example is to clear the interrupt flag.

```
static void IOC0_ISR (void)
{
    IOCAFbits.IOCAF0 = 0; /* Clear the interrupt flag */
}
```

In this function, the only action needed in this example is to clear the interrupt flag.

An interrupt handler is needed to manage all the desired interrupts. For this example, the interrupt handler will call the IOC0_ISR interrupt routine function, as listed below:

```
void __interrupt() INTERRUPT_InterruptManager (void)
{
    if(IOCAFbits.IOCAF0) /* Check the interrupt flag */
    {
        IOC0_ISR();
    }
}
```

Additionally, for this example, the users must add the LED blinking functionality in the `while(1)` loop. After turning the LED on and off, the device enters Sleep mode.

```
while(1)
{
    /* Turn the LED on*/
    LATEbits.LATE0 = 0;
    __delay_ms(DELAY_MS);
    /* Turn the LED off*/
    LATEbits.LATE0 = 1;

    SLEEP();
}
```



View the PIC18F47Q10 Code Example on GitHub

Click to browse repositories

5. References

1. [MPLAB® Code Configurator User's Guide](#)
2. [Getting Started with Writing C-Code for PIC16 and PIC18](#)
3. [Install MPLAB® Code Configurator \(MCC\)](#)

6. Revision History

Table 6-1.

Doc Rev.	Date	Comments
A	05/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Klear, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KlearNet, KlearNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2020, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-6543-0

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com	Australia - Sydney Tel: 61-2-9868-6733 China - Beijing Tel: 86-10-8569-7000 China - Chengdu Tel: 86-28-8665-5511 China - Chongqing Tel: 86-23-8980-9588 China - Dongguan Tel: 86-769-8702-9880 China - Guangzhou Tel: 86-20-8755-8029 China - Hangzhou Tel: 86-571-8792-8115 China - Hong Kong SAR Tel: 852-2943-5100 China - Nanjing Tel: 86-25-8473-2460 China - Qingdao Tel: 86-532-8502-7355 China - Shanghai Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai Tel: 86-756-3210040	India - Bangalore Tel: 91-80-3090-4444 India - New Delhi Tel: 91-11-4160-8631 India - Pune Tel: 91-20-4121-0141 Japan - Osaka Tel: 81-6-6152-7160 Japan - Tokyo Tel: 81-3-6880-3770 Korea - Daegu Tel: 82-53-744-4301 Korea - Seoul Tel: 82-2-554-7200 Malaysia - Kuala Lumpur Tel: 60-3-7651-7906 Malaysia - Penang Tel: 60-4-227-8870 Philippines - Manila Tel: 63-2-634-9065 Singapore Tel: 65-6334-8870 Taiwan - Hsin Chu Tel: 886-3-577-8366 Taiwan - Kaohsiung Tel: 886-7-213-7830 Taiwan - Taipei Tel: 886-2-2508-8600 Thailand - Bangkok Tel: 66-2-694-1351 Vietnam - Ho Chi Minh Tel: 84-28-5448-2100	Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 Denmark - Copenhagen Tel: 45-4485-5910 Fax: 45-4485-2829 Finland - Espoo Tel: 358-9-4520-820 France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 Germany - Garching Tel: 49-8931-9700 Germany - Haan Tel: 49-2129-3766400 Germany - Heilbronn Tel: 49-7131-72400 Germany - Karlsruhe Tel: 49-721-625370 Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 Germany - Rosenheim Tel: 49-8031-354-560 Israel - Ra'anana Tel: 972-9-744-7705 Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781 Italy - Padova Tel: 39-049-7625286 Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340 Norway - Trondheim Tel: 47-72884388 Poland - Warsaw Tel: 48-22-3325737 Romania - Bucharest Tel: 40-21-407-87-50 Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 Sweden - Gothenberg Tel: 46-31-704-60-40 Sweden - Stockholm Tel: 46-8-5090-4654 UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820