CoE 115 Lab 1A: General Purpose I/O (GPIO)

January 2019

Topics/Objectives:

- Learn about the microcontroller's GPIO module, as well as how to access and interface with it.
- Apply assembly programming knowledge to utilize the microcontroller's GPIO module.

Pre-lab:

- Read PICkit3 User Guide
- Review instructions in ISA Reference Manual

A. Overview

The general purpose I/O (GPIO) pins can be considered the simplest of peripherals. They allow the PIC MCU to monitor and control other devices. All I/O ports have the following registers directly associated with the operation of the port, where 'x' is a letter that denotes the particular I/O port. Each I/O pin on the device has an associated bit in the TRIS, PORT, LAT register.

TRISx	**PORTx Data Diretion Control register** The TRISx register control bits determine whether each pin associated with the I/O port is an input or an output. If the TRIS bit for an I/O pin is a '1', then the pin is an input. If the TRIS bit for an I/O pin is a '0', then the pin is configured for an output. **1 - Input** **0 - Output**
PORTx	**I/O Port Register** Data on an I/O pin is accessed via a PORTx register. A read of the PORTx register reads the value of the I/O pin, while a write to the PORTx register writes the value to the port data latch. This will also be reflected on the PORTx pins if the TRISx is configured as an output and the multiplexed peripherals (if any) are disabled.
LATx	**PORTx Data Latch register** The LATx register associated with an I/O pin eliminates the problems that could occur with read-modify-write instructions. A read of the LATx register returns the values held in the port output latches instead of the values on the I/O pins. A read-modify-write operation on the LATx register, associated with an I/O port, avoids the possibility of writing the input pin values into the port latches. A write to the LATx register has the same effect as a write to the PORTx register.

Rule of Thumb: READ on PORTx; WRITE on LATx

- Reading PORTx reads the value or voltage seen in the I/O port.
- Reading LATx reads the value that was held (or last value written) onto the port, instead of the actual values on the pins.

The **PIC24FJ64GB002** microcontroller has 2 GPIO ports. See p. 42 of the data sheet to see register mapping.

Port A	PORTA[4:0]
Port B	PORTB[15:13], PORTB[11:7], PORTB[5:0]

B. Note on driving LEDs

LEDs can be connected to a microcontroller such that they are (1) turned on with a logic LOW or (2) turned on with a logic HIGH. These are referred to as ACTIVE LOW and ACTIVE HIGH respectively. Note that the LEDs in your reference circuit are configured to be active low and are connected to **RA0** and **RA1** pins.

C. "Hello World!" LED Blinking Program

1. Create a new MPLAB project named **CoE115[LastName]_Lab1a** and type in the sample program below.

```
1   .include "p24FJ64GB002.inc"
2   .global __reset
3   config __CONFIG2, FNOSC_FRC
4   .bss
5   i: .space 2
6   j: .space 2
7   .text
8   __reset:
9       mov #__SP_init, W15
10      mov #__SPLIM_init,W0
11      mov W0, SPLIM
12      clr i
13      clr j
14      mov #0xFFFC, W0
15      mov W0, TRISA
16      mov #0x0003, W1
17
18  blinky:
19      clr PORTA
20      call delay
21      mov W1, PORTA
22      call delay
23      bra blinky
24
25  delay:
26      mov #0x000b,W0      ;11
27      mov W0,i
28      mov #0x2C23,W0      ;11299
29      mov W0,j
30
31  loop:
32      dec j
33      bra nz, loop
34      dec i
35      bra nz, loop
36      return
37
38  .end
39
```

2. Build and debug your code using the MPLAB simulator. Observe the values of the special function registers (TRISA, PORTA) as you step through the code.
3. **[QUESTION]** Try replacing PORTA with LATA. Observe what happens to the values of PORTA and LATA. What is the difference between using PORTA and LATA in this code?
4. Simulating the delay:
   (a) The Fast Internal RC (FRC) provides an 8 MHz clock source. Search the datasheet for the relationship of instruction cycle frequency and the oscillator frequency.
   (b) **[QUESTION]** Given this relationship, calculate the corresponding instruction cycle frequency. Write the equation and the answer on a piece of paper.
   (c) Change the simulator settings to this value.
   (d) **[QUESTION]** Using the stopwatch utility, how long is the delay used in our sample code?

D. Run code on target device

1. Connect the PICkit3 to your computer. For Windows users, you might need to install additional USB drivers for the PICkit3.

2. Open the project properties dialog box. In the hardware tools, select PICkit3 instead of the Simulator. Click OK.

3. Build your code. Once it builds without errors, you may run your code by clicking the "Make and Program Device " button ( ).

4. If successful, you will have an output message similar to the message below:

```
Connecting to MPLAB PICkit 3...
Currently loaded firmware on PICkit 3
Firmware Suite Version.....01.32.09
Firmware type..............dsPIC33F/24F/24H
Programmer to target power is enabled - VDD = 3.250000 volts.
Target device PIC24FJ64GB002 found.
Device ID Revision = 3046
Device Erased...
Programming...
The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x3ff
Programming/Verify complete
```

5. The LEDs on you reference board should now blink. If so, you have successfully programmed your PIC microcontroller!

6. **[CHECK]** Verifying the delay. Show the results to your instructor.

   (a) Using an oscilloscope, display the waveform generated by RA0. Measure the time that your signal stays in the HIGH/LOW state.

   (b) Compare with the value reported by the stopwatch utility.

Push Button Exercise

Part 1:

In this exercise, will use a push button make our LED turn on/off. Note that the push button is connected to the **RB0** pin and we will control the LED connected to the **RA0** pin. When the push button is pressed, **RB0** is driven LOW via the direct connection to the ground.

1. Create a new project named **CoE115[LastName]_Lab1a_pb**. Type in the sample code below.

```
.include "p24FJ64GB002.inc"
.global __reset
config __CONFIG2, FNOSC_FRC
.bss
.text
__reset:
    mov #__SP_init, W15
    mov #__SPLIM_init,W0
    mov W0, SPLIM
    mov #0xFFFF, W0
    mov W0, AD1PCFG     ;set to digital (instead of analog
    mov #0xFFFE, W0
    mov W0, TRISA
    mov #0xFFFF, W0
    mov W0, TRISB
main:
waitdown:
    bset LATA, #0
    btsc PORTB, #0
    goto waitdown
waitup:
    bclr LATA, #0
    btss PORTB, #0
    goto waitup
    goto main
.end
```

2. Run the main project. Observe what happens to the LED connected to RA0 when you press and release the push button.
3. **[CHECK]** Modify the code to control both LEDs. Present this to your instructor for checking.

Part 2:

[CHECK] Using the two LEDs and the push button, program your microcontroller so that the following behavior is achieved:

1. By default the two LEDs are OFF.
2. When the LEDs are OFF and the push button is pressed, LED1 turns ON and remains ON for as long as the push button is pressed.
3. When LED1 is ON and the push button is released, LED1 turns OFF and LED2 turns ON for 1 second.
4. During the duration that LED2 is ON, when the push button is pressed, LED1 should NOT turn ON.
5. Pressing and releasing the push button when LED2 is ON do not extend/refresh the 1 second ON time of LED2.
6. If the ON time for LED2 expires while the push button is pressed, LED1 turns ON in a similar manner as in (2).

Have your instructor check Part 2 for Lab 1a completion.