**EET 240 Microcontroller I**

**Lab : Basic Input and Output**

**Instructor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Introduction:** The lab is a further exploration of input and output functions with assembly and an introduction to what embedded C projects look like and how to see the assembly code after the compiler has done the work. The same process of sending values out to a port and being able to bring a value in based on a key-press for example allows you to control your environment via a microcontroller. When you hear the term “fly by wire” or “brake by wire” are examples of a system controlled with a microcontroller. We will start are experience with turning on lights with switches and then graduate to sensors and motors later in the term. Many of the input and output programs will be the same the difference will be the external interface circuitry that you are controlling by the program; this is why embedded systems and the field called mechatronics are so important and will continue to grow and expand.

An embedded C/C++ project is done by selecting an executable C/C++ project at the beginning of a new project identification instead of assembler. The rest of the process is the same for selecting location of project and device. In the example program codes given to you for this lab in the comments you will notice a selection for included files and Macro definitions. **The included files are the same thing in assembly** they just start with a hash-tag instead of a .include. The **macro definitions can be done in assembly also and allows us to create a definition or constant value (i.e. frequency for the cpu or memory address).** If the file is surrounded by **chevrons (<filename> then it is a system file that was included in the installation of Microchip Studio including pre-defined folders.** If the file is surrounded by quotation marks then it is a local or user-defined file (“filename”) that is in either the source or include folders of the project.

In the main code you will notice labels for port locations just like in assembly the use of 0xFF is used always instead of given you an option for the dollar sign as is done in assembly. In the main code the functions are called and a delay loop is used inside and outside the function to slow down the LEDs to give it a pattern change and speed that you can see. The three project codes are:

1. Blink LED (think back to EET129)
2. Display Lights project (displays different patterns like a POV system)
3. Display an Array of values for a hexadecimal display

Some of the code will be further explained such as the bitwise OR, AND, NOT, and XOR operations that are used to set, clear, and toggle the bits in a register in later labs.

Short note on the difference between Embedded C/C and C++: The syntax between the languages for making decisions, setting up and calling functions is the same. Fantastic! The biggest difference between any language (i.e., Python, C, C++, Pearl, etc.) and Embedded… is that in the embedded world we are controlling our environment. There are actuators, sensors, IoT devices, and motors for example that are different than an interactive database program run on a PC, Mac, etc. The main difference between C and C++ is the use of Classes and Objects that can be done fairly easily with larger microcontrollers but with 8-bit microcontrollers we have to be careful with memory consumption and you may see more structures. Both structures and classes are used as user-defined data types and encapsulated functions for security purposes and management practices.

External Interface Circuits for Input/Output:

The majority of pre-built modules or peripheral modules will have a reference manual and schematics that you can refer to when connecting it up to the microcontroller and determining what logic level enables it to work. Most modules regardless of manufacturer will work the same way but always check both references to verify operation. In addition, most vendors will also give you example code or a website where a developer has “test” code to check the operation.

The SSD PMOD is a common cathode display that has a set of control lines (C1 & C2) connected to C line to control which display is on. The displays share the individual segments and the use of two Schmitt Triggered inverts allows the user to toggle the displays on and off to switch values. If you “toggle” at approximately 30 Hz it will seem to show two different values on both displays. So you send one value out and turn on segment 1 to show value and then turn it off; alternate the same process for segment 2. The persistent LED brightness will make it seem like you are seeing both numbers at the same time. This “technology” is how digital wrist watches work.

|  |  |
| --- | --- |
|  | Un-used port pins on the STK200 board can be used for VCC, GND, and C lines for 1 display to work.  The voltage level must remain between 2.7V and 5.25V (maximum)  Recommended voltage 3.3V |

SSD PMOD

Before moving unto the next program code to explore make sure to check the questions at the end of each section.

1. Use the external Bar LED circuit for the following program codes.
   1. Write a program to toggle PB3, PB7 continuously without disturbing the rest of the bits.
   2. Write a program to monitor PA0 bit. When it is LOW, send $55 to PortB.

Questions:

1. What possible commands can be used to program to a specific bit location? Provide an example of each and how it works?
2. How do configure a port for input versus output?
   1. To configure a port for input you set it to all zeros and to configure it for output you set it to all ones.
3. What four lines of code must be in a program to use a RCALL or CALL command? What do they do?
   1. ldi r16, high(ramend)

out sph, r16

ldi r16, low(ramend)

out spl, r16

It sets up the stack.

1. Use the external SSD PMOD for the indirect addressing code.
   1. Download the indirect addressing code from Blackboard, compile the project, and run it on the development board. Answer question A. Edit the code to use the WaitRelease Function. Re-compile the code, and run it on the development board. Answer question B.
2. Change the CAT (C) line from a logic 1 (VCC) connection to a logic 0 (GND). Answer question C.

Questions:

1. What are your observations about how the code is working?
   1. The value displayed is jumping all around not working as intended.
2. After adding the WaitRelease Function what are your observations about how the code is working? What is the difference? What would be a hardware solution to the problem you are seeing?
   1. The code is now counting by 1 each time properly.
   2. The WaitRelease Function waits for the button to be released before counting again.
   3. NAND Latch circuit
3. What does the CAT line control? What makes this part of the circuit work? What would happen if the CAT line was floating? If you want to control both displays what do you need to do?
   1. It controls which display is on.
   2. The inverters.
   3. You would only be using C2
   4. You need to toggle CAT from HIGH to LOW quickly
4. Leave the 7-segment PMOD connected for this part of the lab. Download the embedded C codes listed and run each program on the development board to verify operation. Answer Question D.
   1. Blinky
   2. DisplayLights1
   3. 7segmentarrayC

Questions:

1. Which primary GPR registers are used to manipulate the data in your codes? In the array code where is the look up table stored for the values that go out to the display?
   1. SRAM

Extra Credit:

3. Write a program to monitor PD5 and PD6 bits. When either of them is HIGH, send $AA to PortB; otherwise, send $55 to PortA.

Simulate your work to verify it will operate correctly.

**Submission Details:**

**Lab Check:**

Either develop a short video for one of the program codes using the 7-segment display and LED bar to upload or demonstrate in the lab the two codes working with your development board.

**Lab Synopsis:**

1. Develop an IPO Chart for the indirect 7-segment display.
2. Develop your program and comment your codes to reflect the processes being executed. Make sure to clean each project before zipping up the work.
3. Type up a brief response to each question.
   1. What register is used to configure a port for input or output?
   2. What value must be placed in the register for input versus output function?
   3. What register is used to read a value from the outside into the general purpose register?
   4. Compare and contrast the SBIC and SBIS commands.
   5. Are there any questions that you still have after this lab?
4. Zip up your work in a folder labeled LabInOut\_yourname and attach it to the submission box.