Lab 10: Programming the PIC Microcontroller

(adapted from documents prepared by ECE331 student, Kristopher Micinski)

Summary:

Complete two state machine projects with the PIC microcontroller using the C programming language in order to observe the similarities and differences between different microcontrollers and programming languages.

Learning Objectives:

- Gain familiarity with the PIC microcontroller
- Experience the use of C language programs for compilation into assembly-level code

Resources and Supplies:

- PC with the PIC Integrated Development Environment (IDE)
- DC power supply and protoboard
- Microchip MPLAB ICD 2

Important Reminders:

- It is your responsibility to save the programs you create.
- Please read this entire lab assignment before coming to the lab.

Background:

Introduction

By now, you should be familiar with the workings of the 68HC12 and how it can be used to perform computations and interface to peripheral devices. Although the 68HC12 is a popular microcontroller used in many industry, research, and hobbyist projects, it is important to learn about how other microcontrollers work to get a general understanding of how they may be used to accomplish various tasks. For example, you may find that certain microcontrollers cost less than others, one may have desirable processing capabilities (such as Digital Signal Processing functionality), or perhaps the company you work for uses a specific brand of controllers. This lab will introduce you to another common family of microcontrollers, the PIC my Microchip Technology Inc. You should find that much of the knowledge you have acquired about the 68HC12 will carry over to the PIC microcontroller, but you will also notice some fundamental differences between the two. This lab should help you apply your knowledge of microcontrollers and the 68HC12 to any other family of controllers.

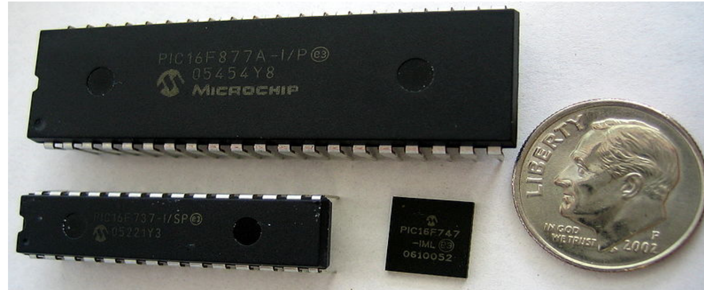
The PIC Microcontroller

The PIC microcontroller relies on a minimized architecture to save power and maintain a compact footprint. It includes only a single general purpose data register but has many memory-mapped registers. Because the PIC has only one register, the instruction set does not explicitly encode which register is being used (though RAM addresses mapped registers are explicitly specified). The PIC also uses mapped I/O, and RAM addresses can be used freely as operands to instructions. Unlike the Von Neumann architecture of the HC12, the PIC utilizes the Harvard architecture, where program code and memory occupy different data space. The instruction set of the PIC is similar in nature to the HC12, although slightly reduced. The PIC can be programmed in assembly, however C is the most common programming language for the PIC18.

The C Programming Language

The C Programming language grew out of the language B (which grew out of another language called BCPL) in the 1960s at Bell Laboratories. It was originally created by Dennis Ritchie (later Brian Kernighan wrote a book along with Ritchie, though he was not the main contributor to the language). C

was developed to help generate the UNIX operating system, which had become too cumbersome to write entirely in assembly code. Because an operating system needs to run as quickly as possible, C was designed to map directly onto assembly code. In this lab, we will also investigate using the C programming language to control the PIC.



Common PIC microcontrollers in different package formats.

Pre-lab Assignment:

- Read this entire lab assignment so you know what to expect in the lab.
- Complete the steps described in the Pre-lab sheet at the end of this document. Each student must complete his/her own pre-lab before coming to the lab and show it to the TA at the beginning of the lab.

Laboratory Assignment:

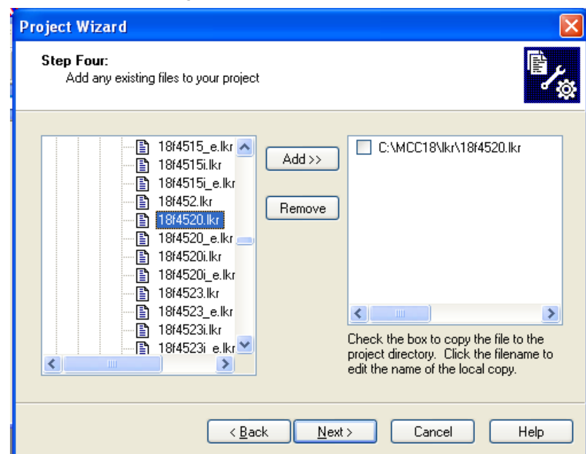
- Print the check off sheet. Where indicated, you must record your results on the check-off sheet. After you successfully finish each section of the lab, show the TA your results and get your check-off sheet signed.

Part 1: Getting Started with the PIC

Part 1 of this lab, will get you started programming in the C language with the PIC microcontroller. You will learn how to use the Integrated Development Environment (IDE) for the PIC, an integral part of any programming system which allows quick development and debugging.

<u>Environment Preparation:</u>

1. To open the IDE, under **Start->All Programs**, pick **Microchip->MPLAB IDE v8.43->MPLAB IDE**. Once the application starts, select **Project**, start the **Project Wizard**. The select **Next**.
2. You will be using the **PIC18F4520**. Make sure this device is selected, and hit **Next**.
3. Make sure the Active Toolsuite is set to **Microchip C18 Toolsuite**. Select **Next**.
4. Under "Create New Project File", select **Browse**. Create a new folder titled **LAB10EX** on your **M:** drive. Save the project file as **LAB10EX** in this folder. Select **Next**.
5. The window will prompt you to add an existing file. You need to select the file located in **C:\ >Program Files** > **Microchip** > **MCC18** > **bin** > **LKR** > **18f4520_g.lkr**. Select **Add**. Your



screen should look like the figure to the right. Check the box next to the file name you just added and click **Next**. In the next window, click **Finish.**

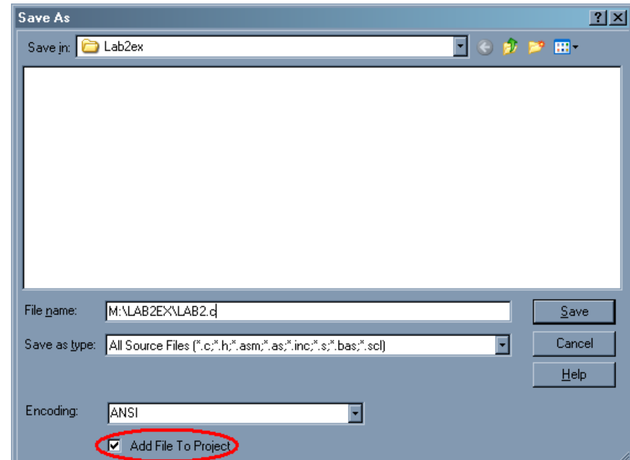
6. Verify the library search path. Go to **Project > Build Options > Project**. Under the Directories tab, select "Library Search Path" next to where it says "Show directories for:." The path should be **C:\Program Files\ Microchip\MCC18\lib**. If it is not, click **New** and browse to the directory to update the path. Click **Apply** then **OK**.

7. Go to **File** > **New**. Enter the block of code below into the new window.

```
#include <p18cxxx.h>

#pragma config WDT=OFF
long int count;
void main()
{
   TRISD = 0x00;
         PORTD = 0;
   while(1)
   {
         PORTD = 0x01;
   }
}
```

Note that "0x" is a format notation for C indicating a hex value.

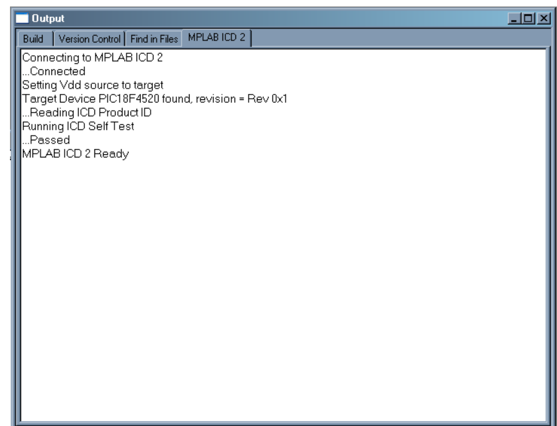


8. Go to **File** > **Save**. Type the file name as **LAB10_1.c** in your **M:\LAB10EX** directory. Select the check box at the bottom labeled **Add File To Project**. Click **Save**. You can also add files from **Project** > **Add Files to Project** in case you forget to check the box.

Hardware Preparation:

9. Turn on the LASCAR DC Power Supply. Set this to 5V using the **dial**. Connect the power supply to the protoboard.

10. Plug the **Microchip MPLAB ICD 2** into the adaptor on the protoboard. Then plug USB cord into the **Microchip MPLAB ICD 2**. In *MPLAB*, select under **Debugger** > **Select Tool** > **MPLAB ICD 2**. The output should look like the figure here. If that does not happen, there must be a problem in your wiring. Turn off the power supply and locate the problem. Setup Wizard may open. Click **Next**. Select **USB** for your Com Port. Click **Next**. Select "Target has own power supply". Click **Next**. Select "MPLAB IDE automatically connects to MPLAB ICD 2." Click **Next**. Select "MPLAB automatically downloads the required operating system." Click **Finish**.



11. Go to **Project** > **Build All**. You should not get any red errors. If you do, you did not type the code in correctly so go back to step 7 and check the code. Otherwise, proceed.

12. Go to **Debugger** > **Program**. This will upload the program to the microcontroller. This can also be achieved by pressing button **1** highlighted below. (Disable low voltage programming, if necessary, to proceed.)

13. Run the program by going to **Debugger** > **Run**. This can also be achieved by pressing button **2** highlighted above. You should now see the red LED lit.
14. Stop the program by pressing button **3** highlighted above. NOTE: You must always follow this pattern when trying new code:

STOP > SAVE > BUILD ALL > PROGRAM > RUN

If you have any questions about C programming in the Microchip IDE, please talk to the TA before continuing.

Part 2: Creating a Moore-type state machine in C Language

1. The general structure of a C program takes the following form:

```
/*
 * The basic structure of a C program
 */

#include <p18cxxx.h> /* PIC specific definitions */

#pragma config WDT=OFF

/*
 * The code that will be executed when the program starts.
 */
void main()
{
   /* Insert main code here... */
}
```

2. You will now modify this program to make the red and green LEDs on the development board continuously flash in the following sequence:

   red on and green off
   red off and green on
   both on

   When you want to do something over and over again in C, use an infinite loop as follows:

```
/* Example infinite loop in C */
while(1)
{
   // Do something...
   }
```

   There is also a “for” loop in C, which says: initialize a variable to some base value, execute the contents of the loop while some condition is true, and after each run of the loop perform some operation. Typically this is used in the following fashion:

```
/* A for loop in C, count to 500,000 */
long int variable;
for(variable = 0; variable < 500000; variable++)
{
   ; /* Do nothing, think of this as a NOP. */
   }
```

   We want to construct an infinite loop, and in the middle we want to sequence the LEDs. The control output to the LEDs is a memory mapped register, as in the 68HC12. This register is already defined as a variable (actually, a macro) in the header file *p18cxxx.h* as **PORTD**. There

is also a Data Direction Register for D on the PIC, **TRSID**. This DDR functions such that a 0 defines an output bit, and a 1 defines an input bit. So, for example, if we wanted to turn on the green LED, we would use the following code:

```
/* Set the DDR on port D to all outputs */
TRSID = 0x00;
/* Turn on green LED by writing a 1 to bit 0 of port D */
PORTD = 0x02;
```

We also want to wait for a while after turning on lights on the development board: we couldn't notice a flash that only lasts a few microseconds. Recall how you achieved this in previous labs and implement that feature here.

3. Construct your program to sequence the LEDs continuously. Your program should roughly act like the following pseudo code (remember to include the appropriate header file and pragma described above!):

```
main procedure {
   Set the data direction register appropriately
   forever {
         write the value 0x01 on the output port
         wait for a while
         write the value 0x02 on the output port
         wait for a while
         write the value 0x03 on the output port
         wait for a while
   }
```

4. Save and compile the code, and run it on the PIC. When the program is functioning correctly, <u>demonstrate it to the TA and have your check-off sheet signed</u>.

<u>Adding Functionality:</u>

5. Note that the code above implements an incomplete state machine with 3 states (green; red; both on: 1,0; 0,1; 1,1; but no 0,0). Now, modify your code so that it implements a Moore-type state machine, maintaining a state variable and oscillating through the sequence of 4 states: S0, S1, S2, S3, and back again.
   For S0 no LEDs should be lit
   For S1 only the red should light
   For S2 only the green should light
   For S3 both LEDs should light

   To keep track of the states in programs, “state variables” are often used. This can be a simple variable such as “state variable” which you can declare as a global variable. For this part of the lab, you will have to use a construction similar to the following:

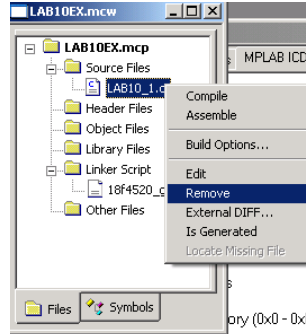
```
// Before or at the beginning of the main function
int currentstate = 0;

// Somewhere in the main function.
if (currentstate == 0){
   // display some value on the LEDs
   // pause for some amount of time
   currentstate = nextstate; // where nextstate is some new state
}
```

!!! – Note: You may have noticed that you can accomplish the task above without using a state variable, by just putting values on the LEDs, waiting, then setting new values, waiting, etc..., then looping back to the beginning. Although this also works, it is not good coding practice and will not work when we use a Mealy machine in the next section

6. Once you have updated your code to the new Moore-type state machine, compile it and run it. Debug as necessary until the code is correctly controlling the hardware.

7. Go to **File** > **Save**. Type the file name as **M:\LAB10EX\LAB10_2.c**. Select the check box at the bottom labeled **Add File To Project**. Select **Save**. Remember to remove LAB10_1.c from the project because you cannot have two 'main' functions in the same project. See image below.



8. Print the code for inclusion in your lab report. Demonstrate functional operation to the TA and have your check-off sheet signed.

Checking Compiled Assembly Code:

9. Go to **View > Disassembly Listing** in the IDE.

10. Look at the assembly code that was generated when the program was compiled and note the difference between this instruction set and the 68HC12 instruction set. Record a comment of your observations on the check-off sheet.

Part 3: Adding Inputs for a Mealy-type State Machine

In this part of the lab, your code from the last section will be modified to add the ability to read inputs into the PIC. To read an input from a pin on the microcontroller, simply modify the PortD data direction register to make bit #2 as an input. You will need the answer to Pre-lab question 4 for your program to work correctly!

Consider how you could make branch decisions based on logic values at the input port. To do this, we can read the global variable PORTDbits. This variable is actually a C "struct" datatype. If you haven't taken a course in C, this simply means that the variable PORTDbits is just a collection of different variables that share a common name because they are related in some way (for example, we could have a structure which described a person and contained name, social security number, age, etc...). All you'll need to know is how to check if bit #2 is a 1 or a 0. For this operation we will again use an if-else structure as follows:

```
if (PORTDbits.RD2)
{
      // Then pin two is high, switch is open
      // Turn on the red and green LED, respectively …
      PORTDbits.RD0 = 1;
      PORTDbits.RD1 = 1;
}
else
{
      // Switch is closed, turn off LEDs
      PORTDbits.RD0 = 0;
      PORTDbits.RD1 = 0;
}
// Wait a while... and reiterate the loop, if you wish...
```

Note that, "if (a)" is equivalent to "if (a ==1)"

1. Modify your program to make a full Mealy-type state machine, where the next state and output are determined by the current state *and* the input. Your Mealy-type state machine should have the following state transitions:

   X = 0: S0 → S1 → S3 → S2 → S0 → …
   X = 1: S0 → S3 → S2 → S1 → S0 → …

   where the X input is the switch input connected to PortD bit 2. For example, if we start the PIC with the switch open (X = 1), we should start in the S0 state, then the S3 state, then S2, etc... If we then close the switch (X = 0), we should go in the sequence S0→ S1 → S2 → S3 → … Our output to the LEDs should be the current state: for example, if the current state is S0, neither LED should be lit, and if the current state is S3, both LEDs should be lit.

   To simplify your coding, start with the code from Part 2 and modify it to implement the following pseudocode:

```
S = 0 // The state variable...
forever {
      if current state is S0 and X = 0 {
            current state = new state // S0 → S3, for example
            Set the LEDs accordingly.
            Pause for a while
      }
      else if current state is S0 and X = 1 {
            current state = new state
            Set LEDs
            Pause
      }
      else if current state is S1 and X = 0 {
            …
      }
}
```

2. Save, compile, test, and debug the Mealy-type state machine. Ensure that the state machine works with both the X = 0 and X = 1 states.
3. To demonstrate that you know how this code is working, add comments to your code to clarify its operation. Include comments for all hardware-related elements (which ports are used for what, what "magic numbers" such as data direction register inputs mean, etc...).
4. <u>Print</u> the final working code for inclusion in your lab report. <u>Demonstrate</u> functional operation to the TA and have your <u>check-off sheet signed</u>.

Final Tasks and Notes

- You will need printed copies of the final C code for Parts 2 and 3 for your report.
- Turn off the power supply and anything else that you might have turned on.
- Disconnect and return the microcontroller interface board to the lab closet or TA.
- Clean up around your lab bench.

Lab Reports

As this is your lab ECE331, you may not see your TA again. Please **submit your reports to the instructor at the beginning of <u>class on the Wednesday</u>** of the week after your lab (i.e., next week).

Discussion Points

As explained in the *Lab Report Guide*, you should address these discussion points in a designated section of your report.

1. In Part 2 of the lab, a delay was added to hold the LEDs in a state long enough to be observed. How could the program be modified so that the hold delay was increased each time the state sequence of repeated?
2. In Part 3 the LEDs were turned on using the PORTDbits structure. Why was this done one bit at a time rather than just writing a byte value Port D once?
3. How could you write directly to Port D instead of using the PORTDbits structure? (Hint: C has bitwise operators & and | that perform a bitwise AND and OR, similar to the HC12 ASM instructions. How could you use the “bitmask” concept to make this work?)
4. Based on your observations of both C code and its compiled ASM code, do you expect that compiled code runs more or less efficiently than original ASM code? Try to explain why this might be true.

PRE-LAB 10

Due: At the beginning of lab.

Student Name: ____________________________ Lab. Section (time): __________

Make sure you read the lab document before you start the pre-lab, especially the Background section.

1. What microcontroller is used during this lab?

2. What language will you be programming in for this lab?

3. In your own words, describe the functionality your program and hardware should demonstrate in Part 2, Step 4 of this lab.

4. How would you setup TRSID to read bit #2 as an input while all other bits are set as outputs?

5. What is the difference between a Moore-type state machine and a Mealy-type state machine?

LAB 10 CHECK-OFF SHEET

Student Name: ____________________________ Lab. Section (time): __________

Complete this sheet as you complete the lab. Remember to have the TA check off each section of the assignment as indicated below. This sheet must be included in your lab report.

Part 2: Creating a Moore-type state machine in C Language

Step 4: Demonstrate to TA

TA sign off Initial________

Step 8: Print code for report. Demonstrate to TA

TA sign off Initial________

Step 10: Comment on your observations (for example, can you identify what the statements are doing and how they were translated from the C code to the compiled code)?

____

____

____

____

____

Part 3: Adding Inputs for a Mealy-type State Machine

Step 4: Print code for report. Demonstrate to TA

TA sign off Initial________