Lab 1: Getting Started with your Micro

Introduction

Single Sentence Overview

Learn how to use your microcontroller, as well as LED lights and a switch attached to it.

Overview of Lab 1

This three-part lab is designed to introduce the student to a debugging-centric method for developing a program on a microcontroller.

- First, the student will write a simple program, download it onto the board and verify that it is working.
- Second, the student will use the debugger memory access functions to examine whether a button is being pressed and whether the LEDs can be made to turn on.
- Third, the student will write a program in C that turns an LED on and off.

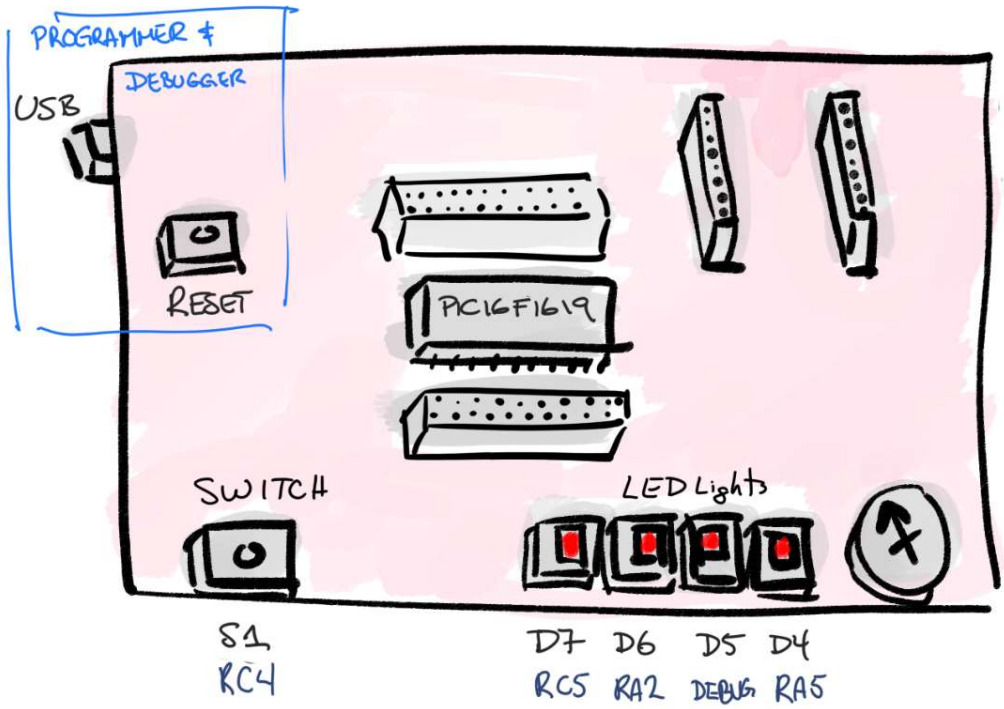


Figure 1 You will use the debugger and a USB connection to your computer to access the internal memory of the PIC microcontroller, as well as a switch and LED lights attached to it.

Learning Outcomes of Lab 1

The student will know how to

1. Write a generic program in C and make sure it works without accessing special chip-specific features.
2. Use the debugger to manually access the inner features (registers) of the microcontroller, as well as the LEDs and Switch on your board.
3. Write a chip-specific program in C that can turn on LEDs on the board and uses a simple delay.

Success Criteria

Review the grading rubric and evaluate how it relates to achieving these criteria:

1. Write a simple program in C and demonstrate that it works on the board.
2. Show that you can use the debugger to manually watch a digital input (switch) and set a digital output (LED).
3. Demonstrate a C program that can flash LED D6 repeatedly

Grading Rubric

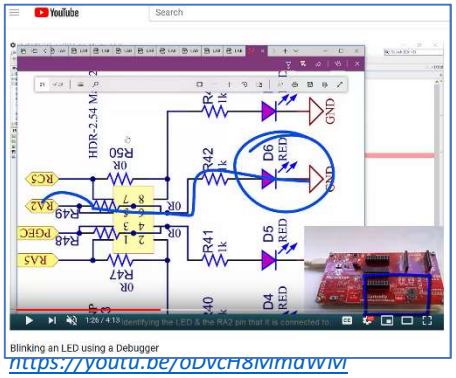
During the TP lab session make sure to conduct all the required demonstrations to the lab instructor. Also, submit a document with the answers to the "Stimulate" and "Explore" questions.

- *Part 1 demo: no grade*
- *Part 2 demo:*
  - *0 pts: no demonstration attempted or neither worked at all.*
  - *5 pts: One of the two demonstrations worked, or both worked only partly.*
  - *10 pts: Both demonstrations worked (LED7 / Port C bit 5 and Switch Port C Bit 4)*
- *Part 3 demo:*
  - *0 pts: no demonstration attempted.*
  - *5 pts: LED D6 only partially worked or was not fully implemented in C and / or in debug mode.*
  - *10 pts: LED D6 turned on / off using C in debug mode.*

Prerequisites

You are expected to have done the following before attending the TP lab session:

- Having attended "TD" classes on the PIC16 microcontroller.
- Watched the introductory video: https://youtu.be/oDvcH8MmdWM



https://youtu.be/oDvcH8MmdWM

More Resources and Information

- *Curiosity board getting started PDF: [https://bit.ly/2PYjtzE]*
- *PIC16F1619 datasheet [https://bit.ly/2OsQtQ1]*
- *MikroE 7-segment Click Board datasheet: [https://bit.ly/2MY0LGx]*
- *Video: [https://youtu.be/oDvcH8MmdWM]*

Part 1: Introduction to Debugging

One Sentence Summary

Learn to use the Curiosity board's built-in debugger to turn on an LED on the PIC16 board.

Overview

In the first hour of your three-hour "Temps Pratique" (TP) lab session, you will get familiar with the hardware for your PIC16 microcontroller and the software we use to write software for the PIC16.

Learning Objectives

The general learning objective is to show you that programming for embedded devices (*systèmes embarquées)* like the PIC16 *can* and *should* be done from a debugger-centric perspective. This method is compatible with the goal of self-directed learning that is typical of many engineering programs.[1]

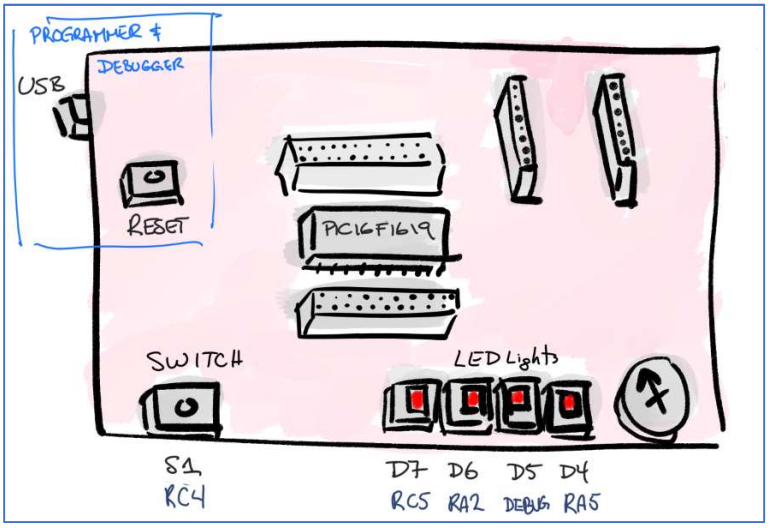


Figure 3 Change the LED on the board using MPLAB X on your computer. You will use the built-in debugger to connect the PIC16 chip to your computer.

In the process of learning to develop from a "debugging-centric" perspective, you will explore the tools for developing programs on a microcontroller, including:

- The PIC16 chip
- The board the chip sits on
- The built-in debugger that connects to the board to your computer and, finally,
- the MPLAB X software for writing programs for the PIC16.

Success Criteria

When you have completed this learning module you will be able to demonstrate that you can write a simple program in C and run a debugging session that connects MPLAB X to your board.

[1] Labs should encourage structured and facilitated, but self-directed learning. To this end, this document is designed around a modified version of the Process-Oriented Guided Inquiry Learning (POGIL) model, and is compatible with the North American ABET and CEAB models for engineering education. A good applied example of this process is in the "Laboratory Short Course" series produced for Freescale's 9s12 processors by Fred Cady, Natasha Kholgade and Ken Hsu. Dr. Cady has given permission to modify his original material.

Prerequisites

You are expected to have done the following before attending the TP lab session:

- Having attended "TD" classes on the PIC16 microcontroller.
- Watched video 1: https://youtu.be/oDvcH8MmdWM

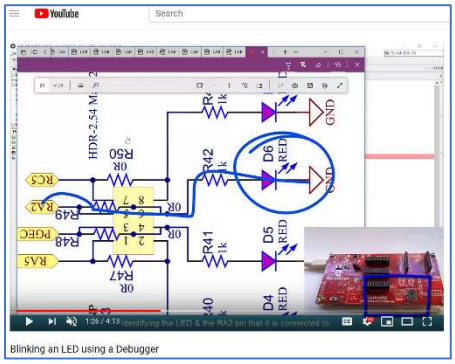


Figure 4 Watch the Youtube video: https://youtu.be/oDvcH8MmdWM

More Resources

- Benson, David. *Easy PIC'n: A Beginner's Guide to Using PIC Microcontrollers 3.1*. Square 1 Electronics, 1996. (URL: https://archive.org/details/EasyPicing, last viewed September 2018). *This is a good book on programming the PIC16F84 in assembler. Distills important data sheets.*
- Datasheet: PIC16F1619 (document number: DS40001770D) by Microchip. (URL: https://bit.ly/2OsQtQ1, last viewed September 2018). *While it's long, you can search through it for key words, as necessary.*

The Main Topic: The Debugger as your access into the Microcontroller.

Debuggers[2] like the one built-in to the Curiosity board (see Figure 3) allow you to examine the inner workings of your microcontroller. They are professional-grade tools and allow you to

- Program your chip
- Read existing programs or data on the chip
- Change settings manually on the chip
- Analyze behaviour of your program

They are widely used by embedded systems programmers to ensure that the programs they write and the hardware they design work as intended. They are also used by security professionals to find weaknesses in electronic systems.

Figure 5 Debuggers are like Star Trek's Vulcans. They can "mind-meld" with your chip.

[2] External debuggers like the PICKit4 and the Segger JLink are inexpensive and can be used when your board doesn't have one built in. (https://binged.it/2y9iq9F and https://youtu.be/Bhd8644wvf8?t=386)

Writing your first program

Your first program won't do much, but it will allow you to connect your PIC16 to your computer. Here are the steps to follow to get started

1. Watch the intro video (https://youtu.be/oDvcH8MmdWM)
2. Connect your Curiosity Board to your PC.
3. Open MPLAB X & start a New Project (then Microchip Embedded -> Standalone Project)
4. For "Device", type in **PIC16F1619** ("none" for supported debug header)
5. Choose the **Curiosity** debugger (Microchip Starter Kits -> Starter Kit (PKOB) -> Curiosity)
6. Choose the XC8 (v2.00) compiler and name the project anything you wish.

Now that you've got a project started, you need to start programming in C:

1. After the main project window appears, right-click on the Sources folder in your Project Pane.
2. Add "Other" -> "C Main File"
3. Open up the resulting C file
4. Modify the C file
    a. Replace the two #include lines with `#include <xc.h>`
    b. Make sure that the "main" line says `int main(void){`
    c. Make sure that the last line in the main function says `return 0;`
    d. Add a line below the main line that says int `my_variable =0;`
    e. Add a line below that one with `my_variable = my_variable +1;`
    f. Add another line below that says `asm("NOP");`
5. Make sure that your project is selected as the "main" one.
6. Add a breakpoint. Click on the line number to the left of the `my_variable = 0;` line. A red box should appear where the number was.
7. Compile your project with the Hammer icon.

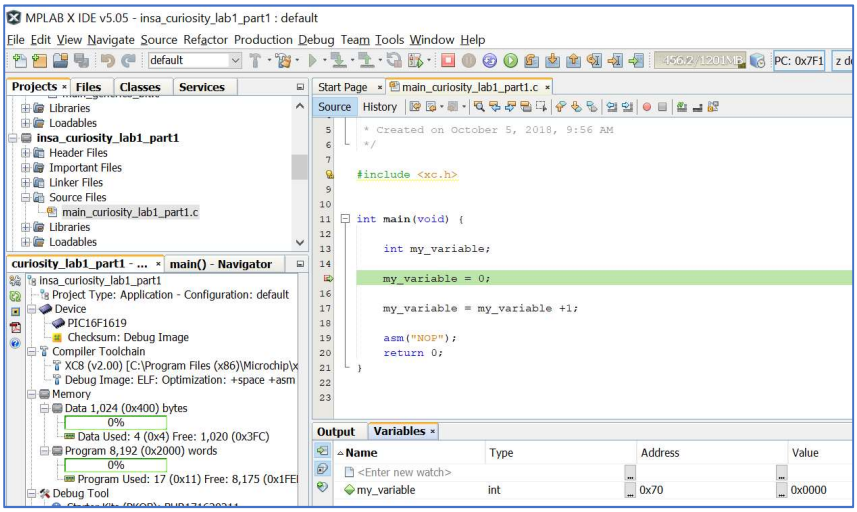


Figure 6 Code written in MPLAB X and debugging session started, with a breakpoint on line 13. The variable my_variable is being watched.

If your program compiled, then move on to the debugging stage.

1. Make sure that you have connected the Curiosity board to your PC.
2. Click on the **Debug Main Project** button.
3. The bottom of your MPLAB X screen should be very active with text now. The program is recompiling and being sent to the board. You will get a warning about a "watchdog." Click on yes.

4. You will be told that your program has halted. A little green arrow will appear on top of the red breakpoint box. The program is now paused (halted) at this line in the code. Good!

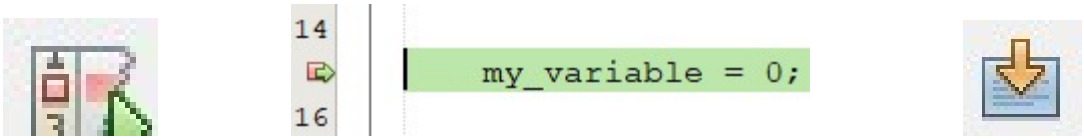


Figure 7 The Debug Main Project engages the debugger (left icon). When you add a breakpoint (red box, middle image), the debugger will halt on it (green arrow, middle image), allowing you to Step Into (right icon) through your code.

At this point you will need to verify that your program is running properly. Put a watch on `my_variable`. Highlight the name of the variable with your mouse. Then right-click and choose "New Watch". This will open a pane allowing you to watch the value of the variable as you step through the code. Click on the "Step Into" icon a few times until you reach the "asm" line. You'll see the value of your variable change from 0 to 1.

There is nothing to submit for this portion of the lab. Just make sure that it works and ask for clarification if it does not or does not work as you thought it might.

Part 2: Directly Accessing the Microcontroller's Resources

Single Sentence Overview

You will access the internal resources directly on the microcontroller with almost no programming required.

Overview

The debugger is a powerful tool that allows you to not only program the chip but also to monitor and change resources inside the chip. Here we'll see how the Special Function Register View allows us to modify the values and functions of the chip's pins without writing commands in C or Assembler. This approach can be used with more complex programming problems to verify that your program is executing as intended and, if it is not, why.

Background material

The debugger is capable of directly examining and setting up the hardware on your PIC16 chip. This is useful when either exploring your hardware for the first time or for understanding the chip's behaviour when something doesn't go right at a later stage. To get started, you should generally look at two documents:

1. The board's schematic
2. The chip's datasheet

The board schematic helps narrow down the chip's pins that are of direct importance to you. You then use the labels to search through the chip datasheet for hints as to which registers need direct manipulation by the debugger. Usually, these are:

1. Port input value register (e.g. PORTA, PORTB, or PORTC)
2. Data Direction register (e.g. TRISA, TRISB or TRISC)
3. Analogue vs. Digital function selection (e.g. ANSELA, ANSELB or ANSELC)
4. Latch digital output value register (e.g. LATA, LATB, or LATC)

You typically set up (1) to (3) first, and then change values using the register in (4). You can do all of these steps in Assembler programming, C programming, direct debugger register manipulation, or a combination of any of these three. Here, we will do this directly via the debugger and a simple C program.

Explore

1. When the green debug arrow arrives at a line in your C code, does it mean that
    a. The code ran in the past?
    b. The code is running in the present?
    c. The code will run in the future?

Problem

Write a simple program as you did in Part 1, with a main function that contains only these lines:

```
while(1)
{
      asm("NOP");
}
return 0;
```

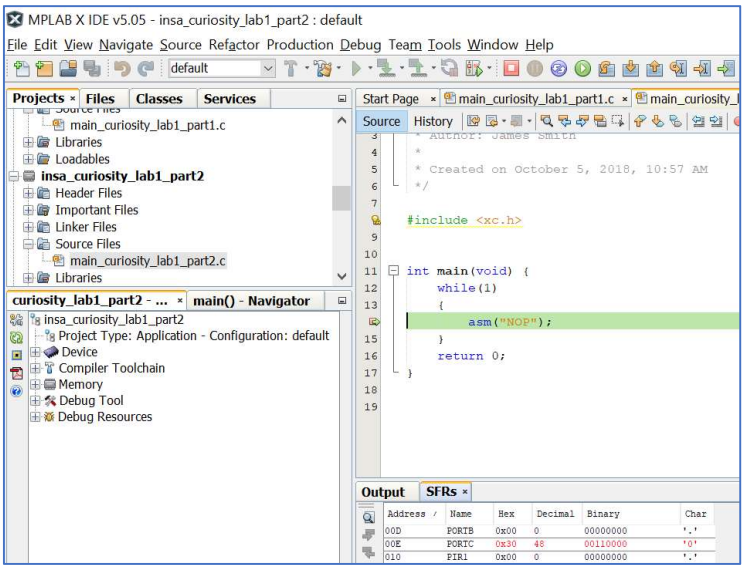


Figure 8 With the program halted at the breakpoint we can modify the ANSEL, TRIS and LAT registers, as well as monitor incoming signals on the PORT registers using the SFRs View. (Window->Target Memory View -> SRFs)

Compile the program to ensure that it is valid C code. Then add a breakpoint to the `NOP` line and run the code on your board using the debug mode in MPLAB X. The code will execute on your board and halt at the `NOP` line. You can now *examine or modify* the internal settings of the PIC microcontroller without running any more C or Assembler program code. <u>You can access each register</u> inside the chip and <u>modify</u> the behaviour of the chip <u>directly</u>.

Here you will use MPLAB X's "SFR View" (*Window->Target Memory View -> SFRs*) to modify the configuration of the pins and then modify the voltage output of the pins so that the LEDs turn on and off.

The datasheet (https://bit.ly/2OsQtQ1) shows us that to make the voltage change on pin RA2 we need to modify four registers in the PIC chip:

1. Analog Select for Port A (ANSELA),
2. Port A (PORTA),
3. Tristate A (TRISA), and
4. Latch A (LATA).

More specifically, particular bits in each of these 8-bit registers need to be changed to reflect the hardware that pins associated with those bits are connected to:

- LED D7 is connected to RC5 (Port C, Bit 5)
- LED D6 is connected to RA2 (Port A, Bit 2)
- LED D4 is connected to RA5 (Port A, Bit 5)
- Switch S1 is connected to RC4 (Port C, Bit 4)

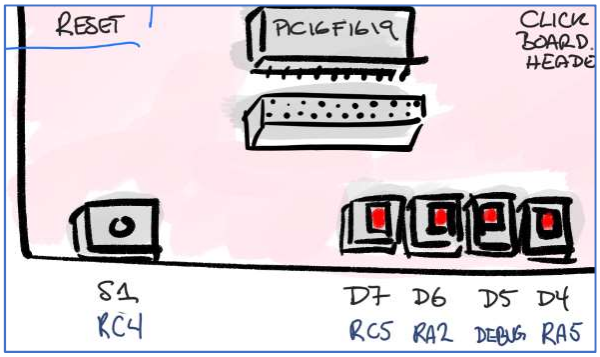
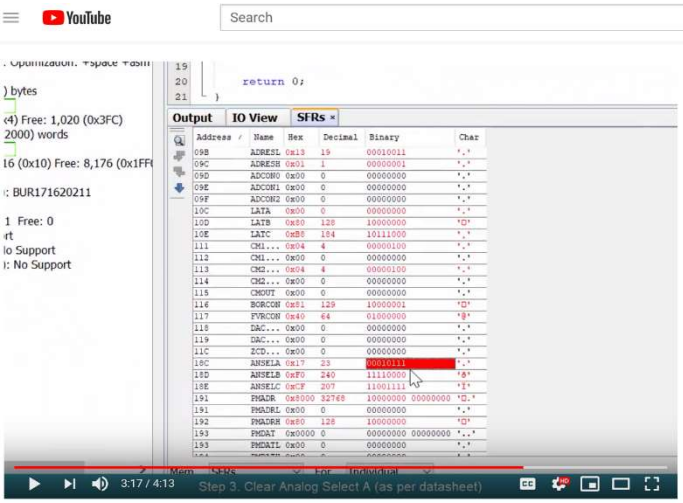


Figure 9 It's important to know which ports and the bits in each port that hardware like LEDs and switches are connected to. Here, we see the hardware we're interested in on the Curiosity board.

Let's start with Switch S1 and LED D7, both on Port C. Similar to how Port A is shown to be initialized in the datasheet (ex. 12-2 in the PIC16F1619 datasheet (**https://bit.ly/2OsQtQ1**), we need to update, in order:

1. PORTC, bit 5 to 0 (don't worry about PORTC, bit 4)
2. LATC, bit 5 to 0 and LATC, bit 4 to 0
3. ANSELC, bit 4 and bit 5 to 0 (for digital mode)
4. TRISC, bit 5 to 0 (for output) and TRISC, bit 4 to 1 (for input)

Do this in the SFRs view, similar to the process shown in the video https://bit.ly/2OC6p60. Make sure to hit the "enter" key after each change SFRs view.



Blinking an LED using a Debugger

Figure 10 Modify the registers directly using the SFRs View in MPLAB X, as shown in this video. https://bit.ly/2OC6p60

Now, test the switch. Hit the "Continue" button  and the program will proceed and loop once. The switch is connected to PORTC, Bit 4. Find PORTC in the SFR view. What is the value of PORTC, Bit 4? Now, press the button and hold it. Hit the Continue button again. Did the value of PORTC change? It should have turned red and the value of PORTC, Bit 4 should have changed with it.

Next, test the LED. Find LATC in the SFRs View. Change Bit 5 of LATC to 1. The LED should have changed. Change it back to 0. The LED should have changed again.

All groups: demonstrate to the lab instructor that you can turn LED D7 on and off by changing Bit 5 of the Latch for Port C from 1 to 0 and vice versa. Also, show the instructor that PORTC, Bit 4 updates when you press the switch, S1.

Part 3: A C Program to Turn on an LED

Background material

You can access all of the registers and bits in a C program. The best way to see this is through an example. Here, modify LED D7 (connected to RC5 on the PIC). Write the following program to do so:

```
#include <xc.h>
#include <iso646.h>

#define _XTAL_FREQ 500000      // Define PIC16F1619 clock freq = 500 kHz

int main(void) {

    /* make LED D7 (connected to RC5) blink
     * Initialize Port C's registers first. */
    PORTC = 0b00000000;
    LATC  = 0b00000000;
    ANSELC = 0b00000000;
    TRISC = TRISC & 0b11011111;      // only change RC5 to be 0 (output to LED)

    while(1)
    {
        LATC = LATC bitand  0b11011111;        // make RC5 a zero.
        LATC = LATC bitor   0b00100000;        // make RC5 a one.

    }


    return 0;
}
```

Figure 11 A simple LED blinking program for LED D7 on RC5.

At the top of the file are three lines that do important support work. The first line ("#include <xc.h>") provides background information about the chip. The inclusion of the ios646.h header file permits you to write "bitwise" logical expressions like "and" as "bitand" rather than "&" and "or" as "bitor" instead of "|". Meanwhile the "#define _XTAL_FREQ 500000" informs the program that your board uses a clock frequency of 500 kHz. This is important if you use the __delay_ms() function to create a delay.

Lines 17 to 20 show the initialization of the PORT, LAT, ANSEL and TRIS registers. The TRISC line (line 20) has a bitmask with a "0" at Bit 5 to make RC5 an output. The and operation on that line only updates bit 5 to 0, leaving all other bits as they were before.

Similar bitwise logical operations happen inside the loop for the Latch register. On line 24 the LED will be turned off because Bit 5 of Latch C is made a zero. On line 25 the LED is turned on, because Bit 5 is now made a 1.

Put a breakpoint on one of the two latch assignment lines. Debug and step through. Watch the LEDs as you step.

The Problem

Next, modify the code to toggle LED D6. D6 uses RA2 on the PIC.

All groups: demonstrate to the lab instructor that you can turn LED D6 on and off using a C program in debug mode.

Optional: can you add the __delay_ms(xxxx); function, replacing the xxxx with a time in milliseconds and run this code in regular (non-debugging) mode? If you decide to use the delay function, please make sure that you use the _XTAL_FREQ definition in the ***Appendix****.*

Part 4: Wrap-up

Reflection on Learning

Talk to your lab partner and other students in the lab. Can you see how using the debugger features allows you to better understand the inner workings of the microprocessor? Are there other ways to view memory on the chip that could be useful? If you were to start an engineering project with a microcontroller, how much would it cost you to get a stand-alone debugger like a PICKit4 or a Segger J-Link? Compare that to the standard hourly wage for an Electrical or Mecatronic engineer. Does it sound like it would be worth it to use one in future projects? You can do this outside of TP hours and it will not be graded.

Communication – Reporting

There is no report for this lab. Simply ensure that you perform the demonstrations during the TP lab session.

Appendix

```
#include <xc.h>
#include <iso646.h>

#define _XTAL_FREQ 500000     // Define PIC16F1619 clock freq - 500 kHz
```