EEE174 –CpE185 Introduction to Microprocessors

LAB 4 – MicroChip

**Lab Session: Wednesday 6:30PM - 9:10PM**

**Section 32385**

**Lab Instructor: Sean Kennedy**

**Student Name: Andrew Robertson**

Table of Contents

[Part 1 3](#_Toc508868050)

[Overview 3](#_Toc508868051)

[Lab Discussion 4](#_Toc508868052)

[Work Performed / Solution: 4](#_Toc508868053)

[Listing Files(s): 4](#_Toc508868054)

[Part 2 5](#_Toc508868055)

[Overview 5](#_Toc508868056)

[Lab Discussion 6](#_Toc508868057)

[Work Performed / Solution: 6](#_Toc508868058)

[Listing Files(s): 7](#_Toc508868059)

# Part 1

## Overview

This is the part of any lab where you are supposed to use known working code to test your setup and make sure all is working fine. Sort of like a blink LED or hello world idea.

## Lab Discussion

### Work Performed / Solution:

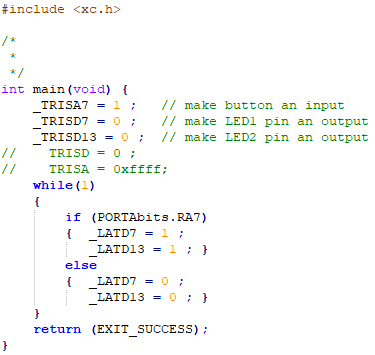
Unfortunately for me part 1 of this lab is the part I took a long time to try and get working but could not resolve. The reason is still unknown to me but I was able to get part two working very fast so I’ll explain what I tried and what my results were despite this failure.

With the given documentation I was initially under the impression that a new bootloader needed to be manually uploaded before being able to use the MAX32 as a dedicated PIC32 but I was mistaken, this process is automated the first time using the serial pins and the manual bootloader is only to be applied to return the device to an Arduino firmware after completed.

Given this, the first step is download the given project files and connect the PICKit 3 to the MAX32 as described. After this, both the PICKit 3 and the MAX32 board need their own independent power sources. I used USB power for them both for ease of use and to not drain the batteries I keep aside for things that require added juice like the servo motor.

Once everything is connected properly, we need to open the given project “Push Button.x” in MPLAB X. Once opened we then build the solution to be sure the given code does not have any syntax issues. Once verified we then connect a breadboard with a push button and LED connected as instructed to the MAX32 to the proper pins. I believe this is where I had my issues, likely translating the pins from PIC32 to ChipKit. I think this is the case because I was able to upload the project to the board without issue but could not get the LED to light, despite using the same breadboard configuration for the second part and having it work just fine.

### Listing Files(s):



# Part 2

## Overview

Now we learn how to take a program and debug it using MPLAB X. We will use a combination of breakpoints and watches to stop the program, step through it, and watch the values stored in different registers during program execution.

## Lab Discussion

### Work Performed / Solution:

I learned MPLAB X has a bit of a hierarchy quirk. When opening another project, it’s best to close any open project first before opening a second one. If another project is opened while another is active it gets nested in the current project instead of closing the previous one automatically.

Anyway, after opening the second provided project “BlinkWithButton.x” we then need to re-connect or continue using the breadboard setup with the pull up pushbutton and current limited resistor. This time I had no issue at all with this setup due to the comments and diagram showing the MAX32 pinouts. The lab instructions say to “make and program” next but I decided to build first just to be sure everything is syntactically correct. After this I then did the make and program as directed and verified that the LED lit when the button was pressed.

Now that we know the correct pins are in use and the code works as designed we can examine the code and see what happens within to make the program work. The first part of this examination is to add a simple breakpoint to our main code next to the line that is only executed when the button is pressed. The breakpoint will look like this:



We then want to run the program in debug and press the button. The breakpoint line should turn green and stop execution. We can leave it here or press the green triangle to resume debugging with the breakpoint still in place.

The second tool is to use is watch points, when combined with the breakpoints we can now see exactly what is stored in the registers during execution. After following the lab instructions to set up my watch list, it looked like this:

The capture above was at the breakpoint, but watches can be used for any part of the program. In order to get a better idea of the way this program works, we then use the step debug function above to step through the program one line at the time while watching the register values. The important thing to pick up during this step through process is to see that highlighted code is basically the line to be executed next when hitting step. After hitting step it then executed and highlights the next line to be executed. I expected a register to hold the value 0 or 1 for high but interpretation seems to be different, maybe bit masked where only one bit related to high or low…

### Listing Files(s):

