ENCM 369 ILS PIC Lab 2

Department of Electrical & Computer Engineering

University of Calgary

Introduction

This lab guides you through setting up the circuit that will eventually become your audio player for the course project. The project is based on an 8-bit PIC16F1778 microcontroller. Many of the concepts learned for MIPS processors will apply to the PIC, through the two processor architectures and instruction sets are very different. However, most of your PIC code will be written in C so you will not have to worry about these details.

D2L Documentation

Documents common to all three PIC labs is posted in the PIC Labs folder on D2L. This includes documentation on the development board used in the lab, instructions for importing or creating MPLABX projects, and links to the data sheets of the parts used in the lab.

Exercise A: Debugging a PIC16F1778 program

Read This First

Debugging your code is an extremely useful skill. When writing relatively simple programs in the lab, you can usually find mistakes by looking over the code yourself. But as your programs become more complicated, it is very useful to pause the code and make sure it is doing everything you expect it to. This exercise will introduce you to the debugging process by having you reverse-engineer the functions provided in the Digital to Analog Converter (DAC) library DAC.X.a.

Before you begin, you should read the MPLABX Debugging Guide posted on D2L.

What to do, Part I

1. This lab uses the same circuit as the previous lab. If you need to rebuild or repair your circuit, refer to the PIC Lab 1 manual. Make sure pins RB6 and RB7 are not connected to LEDs. Connect an oscilloscope to measure the voltage at pin RA2. Hint: Put a short piece of wire into the RA2 header and clip the scope probe to it. You can connect the oscilloscope ground to the heatsink on the 7805 voltage regulator.

2. Download the Lab 2 demonstration project (ENCM369_Lab2.zip) from D2L and import it into MPLABX. Read through Main.C and Lab2_Library.h. Source code to control the timer is provided in Timer.c, but the functions controlling the digital to analog converter are in a precompiled library DAC.X.a.

3. Run the ENCM369_Lab2 project. The oscilloscope should show a sawtooth wave with positive slope. When the button is pressed, the sawtooth wave should change to a negative slope.

4. Set a breakpoint at line 19: the __nop() instruction before Lab2_ConfigureDAC1(). Debug the project and the code should halt just before executing __nop(). When you step over Lab2_ConfigureDAC1(), the register browser should show changes to five registers (PCL, BSR, WREG, TRISA, and DAC1CON0). Record the new values of these registers.

5. Based on the register values you recorded, replace Lab2_ConfigureDAC1() with your own code. If you need to, look up the individual registers in the PIC16F1778 User Manual and only change the registers necessary to make the DAC function. Run the project to make sure your code behaves the same as Lab2_ConfigureDAC1().

Demonstrate your code to a TA and be ready to answer the following questions:

1. During Step 4 of this exercise, Lab2_ConfigureDAC1() caused changes in registers unrelated to the DAC. Look up these registers in the PIC16F1778 User Manual and explain why each register changed.

2. The DAC can't output a continuous sawtooth wave, since it only has 1024 discrete output levels. Zoom in with the oscilloscope and use cursors to measure the interval between each voltage step, and how long each step lasts. Include a screenshot from the scope in your lab report. If the oscilloscope zoom is insufficient, measure the total duration and total voltage change of the sawtooth, and divide by 1024 to find the voltage interval and step duration. In this case, include a screenshot from the oscilloscope showing the overall voltage and duration.

Exercise B: Writing to the DAC

Read this first

The PIC16F1778 has a 10-bit DAC, which allows for $2^{10} = 1024$ different output voltages between Vss and Vdd. However, all of the PIC's registers are 8-bit, so it is impossible to write the entire 10-bit value in one instruction. Writing to the DAC twice creates a timing problem, since the DAC would have an incorrect output during the period between each write. In this exercise, you will investigate the PIC's method for writing to the DAC to avoid this problem.

What to do

1. The registers DAC1REFH and DAC1REFL are used to set the output value of the DAC. Look up these registers in the PIC16F1778 user manual to find the rules for writing to the DAC.

2. Based on this information, complete the following function and add it to your project:

```
// REQUIRES: DAC1 is configured using Lab2_ConfigureDAC1 or student
code.
// PROMISES: Writes the provided WriteValue to pin RA2 via DAC1.
void Lab2_WriteDAQ(short WriteValue)
{
     // Your code here
}
```

   Replace Lab2_OutputSawtooth() with your own code that uses Lab2_WriteDAQ() and the provided function SineArray() to output a sine wave pattern on pin RA2.

When you reach this point, you should demonstrate your code to a teaching assistant. If all the teaching assistants are busy you can move on to Exercise C until someone is available.

Be ready to answer the following questions:

1. The code you wrote for Exercise B does not use pins RC-7 or RB0-7, and the code you wrote for Lab 1 does not use the DAC or pin RA2. If you wanted, you could combine the two programs into one. Explain what changes you would make to your Lab 1 code to work without interfering with the timers in Lab 2. You don't actually have to make the code work, just explain what changes would be necessary.

Exercise C: Writing a Musical Scale

Read this first

You may have noticed that the Lab 2 project no longer uses the __delay_ms() function. Even though the PIC also has a __delay_us() function for smaller time periods, these methods are not suitable for more precise timing control.

Suppose you want your code to run every 100µs, like we have done in the Lab 2 project. You couldn't just use __delay_us(100), because your code takes some time to execute before the delay begins. You would have to calculate the execution time of your code and reduce the delay to compensate. But then you would have to recalculate your delay every time you changed your program. Even worse, if your program uses jumps, loops, or branches, the same code may not have the same execution speed each time it runs. Clearly this isn't practical.

Instead, most microcontroller programs run with a timer peripheral providing a "system tick" or time base of some sort for the system. Timer peripherals are dedicated hardware modules inside the microcontroller that are configured through registers and can interact with the rest of the processor. They are clocked from one of the available clock sources on the microcontroller. The timer can run in parallel with the processor, or it can run while the processor is sleeping.

The typical process is to start a timer, execute the main program loop once, then wait for the timer to expire before moving on to the next loop iteration. The timer always starts a new period at the start of the loop and since it is on an independent peripheral, it is not impacted by any code executed by the core in the loop. The execution time of the main loop code may vary depending on what functions run in that particular loop iteration. Any remaining time left is spent waiting for the timer period to expire. As long as the total execution time of the code in the main loop isn't longer than the timer period, then the system automatically balances itself so every loop will have the same duration.

What to do

Before you begin, read through Timer.c. Modify the code you wrote in Exercise B to change the frequency of the sine wave when you press the button. It must meet the following requirements:

- When the PIC first starts up, it outputs a 220Hz sine wave, which is the musical note A_3.
- Each time the button is pressed, the sine wave changes to the next note in the list found at http://pages.mtu.edu/~suits/notefreqs.html
- Once the PIC reaches note A_4, pressing the button again resets the frequency to A_3.

Demonstrate your code to a TA and be ready to answer the following questions:

1. The function Lab2_ConfigureTimer2 can only configure the timer period to values from 1µs to 255µs. Look up the T2CLKCON, T2CON, and T2PR registers in the PIC16F1778 User Manual, and find a combination of values that will result in a timer period of 1s. Show how you calculated these values.

2. Find the smallest value of TimerPeriod_us you can give to Lab2_ConfigureTimer2() before the frequency of your sine wave no longer increases. Include the value of TimerPeriod_us, the observed maximum frequency, and a screenshot from the oscilloscope in your report.