ECE 3720

Microcomputer Interfacing Laboratory

Section 005

Aaron Bruner

Date Performed: 03 / 11 / 2021

Lab 8

**ABSTRACT**:

A lab designed to demonstrate how to program the PIC32 microcontroller to utilize the pulse width modulation (PWM) and how to apply a PWM to drive a motor. Similar to the previous labs, this will also enhance skills in reading documentations for new and existing parts.

**INTRODUCTION:**

The goal of lab 7 is to program our microcontroller to understand how to program our PIC32MX to drive a motor using PWM. We will be setting up the internal interrupt to control the duty cycle of the PWM. A button will be what triggers the interrupt and results in an increase of 25% to the duty cycle. The output of our PWM will be connected to a motor driver chip (L293DNE) which will determine how fast to spin the motor. To perform our lab, we required a variety of materials including:

* A breadboard
* Wires
* Push Button
* Resistor (220 Ω)
* Diode (x4)
* L293DNE
* DC Motor
* PIC32MX150F128D
* Analog Discovery 2 (AD2)
* Digilent WaveForms
* MPLAB Snap Debugger
* MPLAB X IDE (Programming Software)

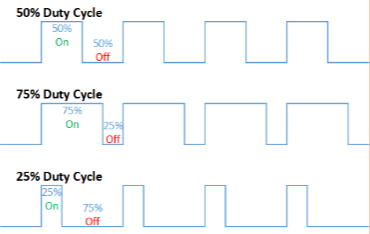
The AD2 will power the PIC32 chip along with partially powering the MPLab Snap Debugger so that it can write to the PIC32. The AD2 will also be supplying a 5V source to the L293DNE chip which will drive the motor. The majority of the power for the MPLAB Snap Debugger will come from the Micro USB cord.

**EXPERIMENTAL PROCEDURES:**

We must assume that the individual reproducing this lab has already setup their breadboard in a manner that their PIC32 chip can be programmed using the MPLAB X IDE software.

To start we can begin by examining the circuit diagram in figure 1. The PIC32 is setup in such a way that we could drive the motor in either the clockwise direction or the counterclockwise direction. The output of B9 and B8 will instruct the motor driver chip to output the specific duty cycle to the motor on the H-bridge. When the button on B7 is pressed it is intended to increase the duty cycle by 25% which in turn will make the motor spin faster. Once the duty cycle exceeds 100% we will return back to being in an idle position at 0% duty cycle.

However, since our motor draws more current that we’re able to supply this setup does not work. Instead, we opted instead to remove both the DC motor and the motor driver circuitry and view the PWM signal in our oscilloscope. To do this we remove the B9 connection and instead of powering the button off of the Vcc2 connection we connect it to the 3.3V source powering the PIC32. Now, when we press the button we can view the oscilloscope and see a result similar to what is shown in the image below.



Ideally we would have no noise and the duty cycle would have perfect edges but that is unrealistic. However, we did get a result similar to what is shown above but instead with more noise infused readings. One duty cycle that is now shown is the 100% value. This reading is almost a perfectly horizontal line but with a small dip to 0V when it resets.

For our code it is mostly self-documenting. We begin by including our header files which allow us to utilize many features while programming. Then, we initialize our pwm integer which will be either 0, 25, 50, 100 or 125. The 125 value is what returns us back to 0. Then we initialize our interrupt for the button so we can increase the duty cycle. It’s a very simple interrupt using INT0. After we either increase the duty factor by 25 or reset it back to 0 we set our output signal to the PWM and then reset the interrupt. Next, we initialize our main function starting with provided code. Then, we set B8 to be an output pin which was intended for the L293DNE but now is a hanging pin with nothing to connect to. We will read what is on this pin using our oscilloscope. Next we set B7 to be an input pin for the button and enable the pull up resistor as well. Next, we start setting up our internal structure to generate the PWM signal. We will be using output compare to generate the PWM signal. We start by using peripheral pin select to assign B8 to be OC2. The next four lines of code are setting up the INT0 interrupt. INT0IP will assign polarity to the interrupt, anything other than 0. INT0IE is the enable bit for the INT0 interrupt. INT0IF is how we clear the interrupt so it can be triggered again. And lastly INT0EP is the edge polarity to which the interrupt triggers to, we want rising edge in this case. Next we continue by setting up our timer control register. First we select the internal clock using TCS=0, then use the 1:1 prescale. Lastly we opt for a 16-bit timer and disable the option for a gated timer. Next we clear the timer register and load our initial compare value into PR2 with a value of 100 and then start the timer with T2CONbits.ON=1. Now, we can setup our output compare starting by selecting PWM mode without fault detection. To do this we assign a value of 0x6 to the OCM register of OC3CONbits. Then, we opt for a 16 bit register instead of 32 and finally use timer two as the clock source. Lastly we set the initial compare register to be at 0% duty cycle and enable the module.

**RESULTS and DISCUSSION:**

Once the microcontroller was programmed it was apparent that things were working as expected when we pressed the push button we could see, in the oscilloscope, the PWM signal is being displayed.

It’s unfortunate that we were not able to utilize the DC motor or the L293DNE chip since the DC motor wants more current that we can provide. If we were going to use the motor driver and the DC motor we would have setup our circuit as shown in figure 1. The H bridge allows us to drive the motor in either clockwise direction or counterclockwise direction. The purpose of the motor driver is to allow varying speeds of rotation depending on the duty cycle. Since we did not use the motor driver chip or the DC motor we instead attached scope 1 to the output on B8. As we watched the scope readings it showed the different duty cycles as we pressed the button.

One problem during this experiment was that the motor would not spin when we had everything setup as intended in figure 1. It wasn’t until the lab TA instructed us that our motors were trying to draw more power that could be supplied so they would not spin under normal conditions. Thus, to resolve this problem we removed the L293DNE chip and the DC motor and instead watched the voltage reading in scope 1.

This week we demonstrated how to utilize PWM signals, DC motors and motor drivers. Being able to read the datasheet and determine what pin everything needs to be connected to and how to properly setup your circuit is crucial in the real world. Managing high power DC motors using PWM signals will be common in microcontroller programming careers.

**CONCLUSION:**

To conclude, lab 8 taught us many useful techniques on generating PWM signals, setting up motor controllers, H-bridges with DC motors and combining those with interrupts on the PIC32 microcontroller. Also, as with many other labs, this one teaches us how to debug our circuit to check if everything is working. This lab also taught us more complex ways to analyze problems and how to solve them.

**Diagram

Description automatically generatedFIGURES AND TABLES:**

**Figure 1: Wiring for lab 8 (Pin connections described in experimental procedures)**

**CODE:**

#include <xc.h>

#include <sys/attribs.h>

int pwm;

void \_\_ISR(3)pushbutton(void){ // Interrupt for button press

pwm+=25;

if (pwm > 100) { // If PWM gets above 100 then go back to 0

pwm = 0;

}

OC2RS=pwm; // Set OC2R to be our pwm

IFS0bits.INT0IF = 0; // Clear the INT0 interrupt flag

}

int main(void) {

INTCONbits.MVEC = 1;

\_\_builtin\_enable\_interrupts();

CFGCONbits.JTAGEN = 0;

TRISBbits.TRISB8 = 0; // Output for L293DNE

TRISBbits.TRISB7 = 1; // Input from button

CNPUBbits.CNPUB7 = 1;

RPB8R = 0b0101; // Assign B8 to OC2

IPC0bits.INT0IP=1; // Interrupt polarity

IEC0bits.INT0IE=1; // Enable INT0 interrupt

IFS0bits.INT0IF = 0; // Clear INT0 interrupt

INTCONbits.INT0EP=1; // Edge Polarity (1 Rising, 0 Falling Edge)

T2CONbits.TCS=0; // Use internal clock

T2CONbits.TCKPS=0; // 1:1 Prescale

T2CONbits.TGATE=0; // Gated time accumulation disabled

T2CONbits.T32=0; // 16-bit timer (TMRx and TMRy separate)

TMR2=0; // Clear timer register

PR2=100; // Load period value

T2CONbits.ON=1; // Enable timer 2

OC2CONbits.OCM=0x6; // PWM mode on OC, fault disabled

OC2CONbits.OC32=0; // 16-bit Mode

OC2CONbits.OCTSEL=0; // Timer 2 is clock source

OC2RS=0; // Initialize compare register with 0% duty cycle

OC2CONbits.ON=1; // Enable module

while(1){

}

}