USC Viterbi School of Engineering

Ming Hsieh Department of Electrical Engineering

EE 109L - Introduction to Embedded Systems

Spring 2015 Lab Assignment #9
Pulse Width Modulation

1 Introduction

In this lab exercise you will use a rotary encoder to control the generation of a pulse width modulation (PWM) signal from the Arduino Uno. In the first part of this lab we will use an 8-bit Counter/Timer to generate a PWM signal to control the brightness of an LED. The second part of the lab will involve using the higher resolution 16-bit Counter/Timer to generate a more precise PWM signal to control a servo motor. By adjusting the encoder position you'll be able to control the position or speed of the servo.

2 PWM Control of an LED

The Atmel microcontroller used on the Arduino Uno can generate PWM signals through the use of the Counter/Timer modules. The counter is used to perform two tasks for generating the proper PWM signal. First it has to generate pulses with a constant period. The period of the pulses is independent of the pulse width and is the time from the $0 \rightarrow 1$ transition of one pulse until the next $0 \rightarrow 1$ transition. To generate pulses at a fixed period the counter has to count the clock signal until the pulse period has been reached and then reset back to zero and start over.

In the PWM mode we will be using, each period of the pulse is limited to the length of time it takes the counter to count from 0 to 255. The counter simply counts up to the maximum count value and then starts over. We will use the prescaler to divide the 16Mhz clock by 1024, so the period of our pulses will be

$$256 \times \frac{1}{16Mhz/1024} = 16.4msec$$

The second task for the counter is to control the width of the pulse. The pulse output goes to the one state at the beginning of the pulse period, then at some point during the pulse period the counter must terminate the pulse by setting the pulse output to zero. The time that the pulse spends in the one state is the pulse width, and varying this width is what controls the brightness of the LED.

The counter can be configured so that at the start of the pulse period (counter = 0) the `OC0A` output signal, which appears on Arduino port D6 (Port D, pin 6), will go to the high state to start the pulse and then will go back to zero at some point while the counter continues to count to 255. In Counter/Timer0, register "`OCR0A`" is used to store a value that determines the pulse width by telling the counter when to terminate the pulse. The pulse output will go high at the start of the pulse period and when the counter reaches the value in `OCR0A` the output signal goes back to zero.

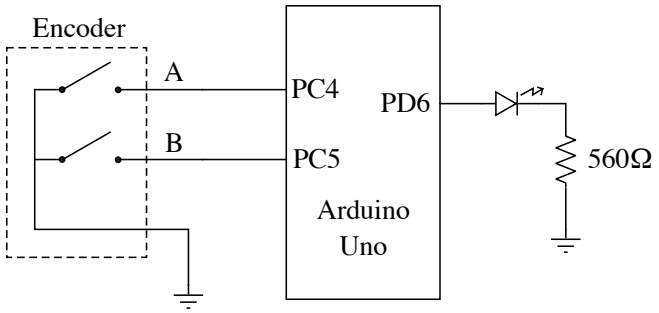

Figure 1: Rotary encoder controlling generation of a PWM signal

3 Program 9A

For lab program 9A you should be able to start with a copy of the lab 8B program and then modify it. The following tasks will need to be done.

1. Remove the LCD from the Arduino and remove all the LCD code from the program. If you had all your LCD routines in a separate file this should be relatively easy.

2. Add the following function to your program to initialize Counter/Timer0 to operate as described above for PWM generation.

```
void init_pwm(void)
{
    TCCR0A |= (0b11 << WGM00);  // Fast PWM mode, modulus = 256
    TCCR0A |= (0b10 << COM0A0); // Turn D6 on at 0x00 and off at OCR0A
    OCR0A = 128;                // Initial pulse duty cycle of 50%
    TCCR0B |= (0b101 << CS00);  // Prescaler = 1024 => 16ms period
}
```

3. In your program make the following changes or additions.

   (a) Set the DDR bit for Port D, bit 6 to a one to make it the PWM pulse output.

   (b) Add a line to call the `init_pwm` function.

   (c) Initialize the variable that is controlled by the encoder to the initial value that was put in the `OCR0A` register in the `init_pwm` function. This will prevent the value from jumping around the first time the encoder state is changed.

   (d) Add code in the main loop to limit the value that is changed by the encoder to stay between 0 and 255. This number has to fit in the eight bit `OCR0A` register

   (e) Add code in the main loop to copy this pulse width count value into the `OCR0A` register whenever it has changed.

4. To confirm that the pulses are being created as specified, use an oscilloscope to look at the pulses on the `OC0A` output (Arduino port D6). You should be able to see the pulse width change as you rotate the encoder. If you feel it's taking too many turns of the encoder to cover the full range, change your program to increment and decrement the count value by more than one each time the encoder changes state.

5. If all looks correct, hook the D6 output pin to the LED according the schematic in Fig. 1 and see if it works. You should be able to adjust the LED brightness from full off to full on by turning the encoder knob.

4 PWM Control of a Servo Motor

The next thing we want to do is use the PWM signal to control a servo motor. Servos are a bit more picky than LEDs about the PWM signals. The LED worked fine with a PWM signal with a duty cycle that varied from 0% to 100%, and the pulse period of 16.4ms seemed to work fine also. A typical servo requires a PWM pulse period of 20ms and the pulse width should only vary between 0.75ms to 2.25ms. The eight-bit timer used above can't generate a pulse with a 20ms period since that would require a count value greater than 255. So for this part of the lab we will use the 16-bit Counter/Timer1.

Counter/Timer1 has two internal 16-bit registers that are used to implement the two tasks described above. In the counter mode we will be using, register "`OCR1A`" is used to determine the **pulse period** by storing the value the counter should count to before reseting and repeating. Register "`OCR1B`" is used to store the value that determines the **pulse width** by telling the counter when to terminate the pulse. The pulse output will go high at the start of the pulse period (counter = 0) and when the counter reaches the value in `OCR1B` it will set the pulse output back to zero. The counter continues to count until it reaches the value in `OCR1A` and then resets to zero and starts the process over. Needless to say, the value in `OCR1B` should be less than the value in `OCR1A`. When Counter/Timer1 is used in this mode the pulse output will appear on Arduino port D10 (Port B, bit 2).

5 Program 9B

For lab program 9B, start with a copy of the lab 9A program and then make some modifications to it. The following tasks will need to be done.

1. Using the same process you used in the stopwatch lab, find a combination of an `OCR1A` value that fits in 16-bits ($< 65,536$), and a counter prescaler of either 1, 8, 64, 256 or 1024 that gives a counter period of 20msec. The count value to go in `OCR1A` can be calculated from

$$count = time \times \frac{16Mhz}{prescaler}$$

   There may be more than one combination of count and prescaler that work.

2. With the prescaler selected, use the above equation again to find the values that will go in `OCR1B` to give delays of 0.75msec and for 2.25msec. These are the minimum and maximum pulse widths you will be using and the encoder will be used to change the pulse width value between these limits. Turning the encoder one way will cause the program to increase the value in `OCR1B` making the pulse wider, turning it the other will decrease the value to make the pulse narrower.

3. If the encoder has 64 states per revolution, how many revolutions will be needed to adjust the motor from the minimum to the maximum speed assuming the value in `OCR1B` is incremented or decremented by one for each state change? If you feel the number of revolutions is too high or too low, try going back and find different values for the prescaler and for `OCR1A` and `OCR1B` values that will also work but need fewer revolutions of the knob.

4. Modify the `init_pwm` function to initialize Counter/Timer1.

   (a) Set the DDR bit for Port B, bit 2 to a one since this is the pulse output.

   (b) Set the Waveform Generation Mode bits (`WGM13 - WGM10`) to `1111`. Two of these bits are in register `TCCR1B` and the other two in `TCCR1A`. Check the manual or the class notes to find their location.

   (c) In register `TCCR1A`, set the bit `COM1B1` to a one. This will make the `OC1B` output line (Arduino port D10, or Port B, bit 2) serve as the pulse output.

   (d) Set the `OCR1A` register to the value determined above. For an initial condition for register `OCR1B`, use a value halfway between the two limits which should result in a pulse width of 1.5msec.

   (e) Set the Clock Source bits (`CS12 - CS10`) in `TCCR1B` to the correct values for the prescaler you selected above. Check the manual or the class notes to find these values.

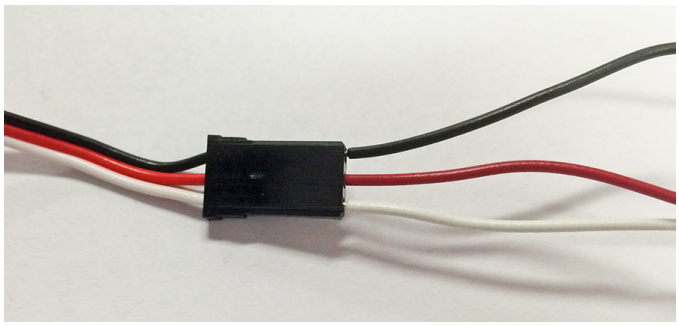

Figure 2: Use breadboard wire to connect to the servo motor

5. Modify the program so that each time an adjustment of the encoder position changes the pulse width count value this value is copied into the `OCR1B` register.

To confirm that the pulses are being created as specified, use an oscilloscope to look at the pulses on the `OC1B` output (Arduino port D10). You should be able to see the pulse width change as you rotate the encoder. Check to make sure that

- The pulse period is 20ms.
- The minimum pulse width is 0.75ms
- The maximum pulse width is 2.25ms

You must demonstrate your working PWM signals by showing them on an oscilloscope to an instructor. Only then will you be issued a servo motor. When connecting the motor to your Arduino, first **disconnect the power to your Arduino before connecting the motor.** The cable to the motor has a three conductor connector with three wires going into it, red for power, black for ground and white for the PWM signal. Using a piece of breadboard hookup wire, make a connection from the $+5V$ on the Arduino to the red wire by inserting the end of the wire into the connector socket for the red wire as shown in Fig. 2. Do the same to connect the Arduino ground to the black wire, and from the D10 output pin to the white wire.

Once you reconnect the power to your Arduino the motor should move to the initial position as set by your PWM signal. Try rotating the encoder knob and see if the motor responds correctly. For a standard servo, check to see that you can move the motor through an arc of about 180 degrees. For a continuously rotating motor, check you can make it rotate at full speed in either direction, and also stop the rotation.

6 Results

Once you have the assignments working demonstrate them to one of the instructors. Turn in a copy of the source code for both programs through the link on the class web site.

EE109 Lab 9B Grading Rubric

Name: _____

TA/Instructor Successful Demo initials: _____

Item	Outcome	Score	Max.
Initialization			
• Program correctly initializes appropriate PORT capabilities	Yes/No		1
• Program correctly initializes timer	Yes/No		2
• Program correctly initializes pin change interrupts	Yes/No		1
Encoder Operation			
• Provided an ISR for timer interrupt	Yes/No		2
• Provided an ISR for pin change interrupts	Yes/No		2
• Program reads input bits and determines state changes	Yes/No		3
• Program adjusts pulse width based on state changes	Yes/No		3
• Program handles min and max pulse widths properly	Yes/No		2
Code Organization			
• Properly indented code	Yes/No		2
• Appropriate coding style	Yes/No		2
Total			20
Open ended comments:			