**Jared Hayes**

**CPE 301 Microcontroller Based Systems**

**Motor Speed Encoder and Controller**

May 12, 2014

**Final Project\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Objective

Create a C program to control DC motor speed and direction using PWM signals and an optical encoder. By default, the motor will run clockwise gradually increasing speed from 0% to 100%, wait, and then do the same thing in the counter-clockwise direction using PWM. A quaderature encoder will monitor the motor’s direction and position for display to an LCD. The LCD screen will display the direction of the motor, its relative position after every direction change and the duty cycle that it is operating at. A Bluetooth module will also be used as an alternative means to control or rewrite the PWM signal; and therefore, controlling the motor speed.

Project Video Link: https://www.youtube.com/watch?v=WxVQC9fV\_lQ

2. Equipment List

* Atmel AVR Atmega328P or Arduino Uno
* Potentiometer
* L293D Dual H-Bridge Motor Driver
* DC Gear motor
* 48 CPR Quaderature Encoder
* 1602 16X2 LCD Screen
* HC-05 Bluetooth Module
* Android phone or Android PC simulation device
* Breakout board

3. Connections

I’m using an Arduino Uno for this project; however, all of the code is implemented using only standard AVR C libraries. A dc motor will be connected to the Atmega328P microcontroller through use of a L293D chip, which contains an H-bridge motor driver circuit. Four data pins from PortB of the AVR will be connected to a 16X2 LCD screen, along with two control pins. The motor is connected to a quaderature encoder which transmits to two signals to the INT0 and INT1 pins of the AVR. Two LEDs are setup to capture the output of these signals. Finally, a Bluetooth module is connected to the TX and RX pins of the AVR.

4. Theory

In order to control the speed of the motor, I needed to use pulse-width modulation (PWM) to send a signal to the motor. The AVR sends square waves of identical frequencies, but differing duty cycles in order to output a range of potential speeds. This program uses timer0 in fast PWM mode to control the signal. In fast PWM mode, the timer counter repeatedly counts from 0 to 255 and turns off when the count matches the value stored in the output compare register, OCR0A. The higher the compare value, the higher the duty cycle; and therefore, the faster the motor spins. My AVR is operating at a system clock frequency of 16MHz and uses a prescaler value of 64 to divide the clock.

The quaderature encoder is mounted on the motor and provides a resolution of 48 counts per revolution. The gearbox connected to my motor has a gear ratio of 9.7:1, which allows for 465 encoder counts per revolution. However, my code only uses the rising edge of the signals, so there are only 24 counts per revolution. The two signals sent from the encoder are 90 degrees out of phase, which allows for determination of the motor’s direction. When the pulse of signal A is followed by the pulse of signal B, then the motor is going forward and when the pulses are reversed, the motor is going in the opposite direction. Incrementing a counter for pulses detected in one direction and decrementing for pulses of the other direction can allow for tracking of the motor’s position.

5. Calculations

*-PWM Calculations:*

* Output A frequency: Sys Clock Freq / prescaler / 256 = fout
  + Output A frequency: 16 MHz / 64 / 256 = 976.5625Hz
* Output A duty cycle: (OCRnX +1) / 256 = duty cycle (decimal)
  + Output A duty cycle: (128+1) / 256 = 50.39%

The system clock frequency divided by the prescaler, divided by the number of cycles it takes for the timer to count gives the output frequency of the PWM signal. The value of the OCR0A register plus one, all divided by 256 gives the duty cycle of the signal.

*-Encoder Calculations:*

* Counts per Revolution: Encoder counts \* gearbox ratio
  + Counts per Revolution : 24\*9.7 =233

6. Code Breakdown

*Pin Definitions*

#define led1 3 // SigA led -PC3-

#define led2 4 // SigB led -PC4-

#define motorPin1 4 // Motor Pin + -PD4-

#define motorPin2 5 // Motor Pin - -PD5-

#define pwm 6 // PWM for motor speed -PD6-

#define sw 7 // Switch/Button -PD7-

int sigA = 0; // Encoder Signal A, int0 -PD2-

int sigB = 0; // Encoder Signal B, int1 -PD3-

*Clockwise/Counter-Clockwise*

The same code is used to turn the motor counter-clockwise, except that the motor pins are reversed. Also prints the current direction on the LCD.

void cw(void){

PORTD |= (1<<motorPin1); // Motor Pin + turned high

PORTD &= ~(1<<motorPin2); // Motor Pin - turned low

lcd\_setCursor(13,0); // Row 1, Column 14

lcd\_print(" "); // Clear

lcd\_setCursor(13,0);

lcd\_print("CW");

}

*Initialize PWM*

Set Timer0 registers based on aforementioned PWM calculations. Sets OCR0A on compare match when down counting.

// ----Enable PWM on OC0A----

TCCR0A |= ( 1 << COM0A1 ) | ( 1 << WGM01 ) | ( 1 << WGM00 ); // Fast PWM - PD6 to Enable on H-bridge

TCCR0B |= (1<<CS01) | (1<<CS00); // 64 Prescaler

*Motor Speed Control*

The function will increment OCR0A for the full range of motor speed values and convert them into duty cycles to print to the LCD. A direction is specified and once the loop is finished, the motor will turn off and wait for 3 seconds. Then it will update the motor’s current position and repeat for the opposite direction.

void setSpeed(){

uint8\_t dutyCycle; // Set OCR0A to desired motor speed connected to En on H-bridge

for (int i = 0; i < 256; i++) // Increment motor speed from 0-255

{

cw(); // Turn Motor clockwise

OCR0A = i; // OC0RA = speed

//dutyCycle = i/256;

//dutyCycle = dutyCycle\*100;

dutyCycle = dutyConvert(i, 0, 255, 0, 100); // Convert speed to duty cycle

lcd\_setCursor(13,1); // Column 14, Row 2

lcd\_print(" "); // Clear

lcd\_setCursor(13,1); // Column 14, Row 2

lcdData(dutyCycle); // Display duty cycle to lcd

\_delay\_ms(50);

}

\_delay\_ms(3000); // Wait 3 seconds

PORTD &= ~(1<<motorPin1); // Motor Pin + turned low ==> Motor off

DisplayPosition(1); // Display current position

*Encoder Interrupts*

The interrupt vectors are triggered by any rising edge change in the two encoder signals sent to the int0 and int1 pins as external interrupts. If signal A is rising, it determines what direction the motor is moving based on whether signal B is high or low. The same applies for when signal B is rising. The function will either increment or decrement the current position counter depending on the direction determined by the signals.

ISR(INT0\_vect){

sigA ^= 1; // Signal A Rising Edge

PORTC ^= (1<<led1); // Toggle led1

if(sigB == 0) // If Signal B is low...

CurrentPosition++; // moving forward

if(sigB == 1) // If Signal B is high...

CurrentPosition--; // moving reverse

}

ISR(INT1\_vect){

sigB ^= 1; // Signal B Rising Edge

PORTC ^= (1<<led2); // Toggle led2

if(sigA == 1) // If Signal A is high..

CurrentPosition++; // moving forward

if(sigA == 0) // If Signal A is low...

CurrentPosition--; } // moving reverse

*Bluetooth*

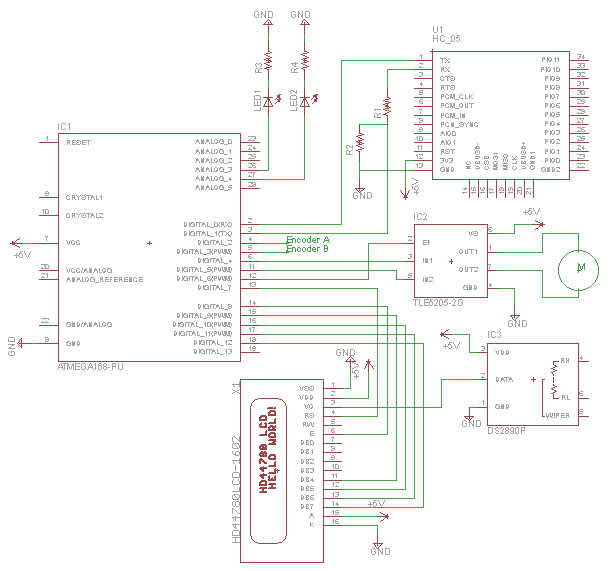
Code to rewrite OCR0A value

if (ard\_command == CMD\_ANALOGWRITE) {

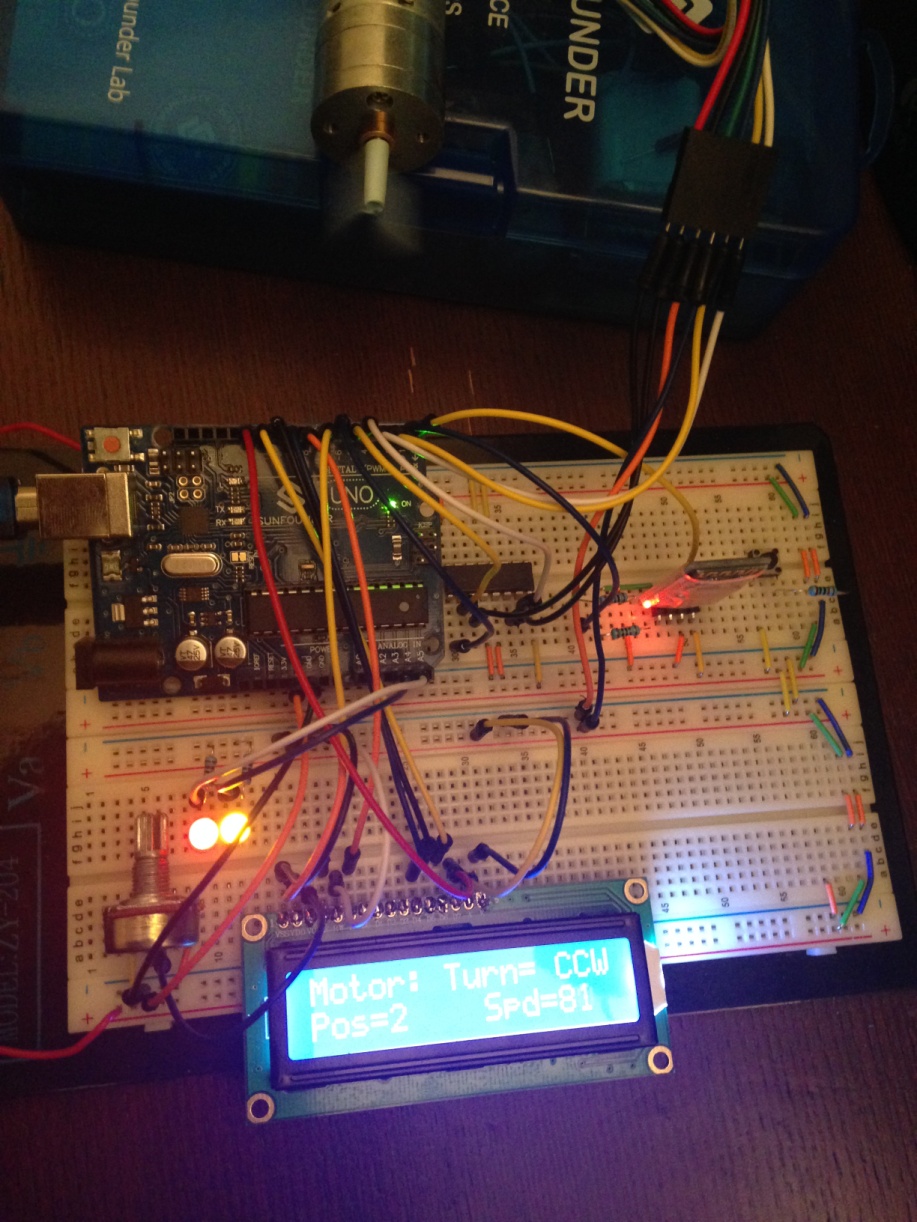
OCR0A = pin\_value;

return; // Done. return to loop(); }

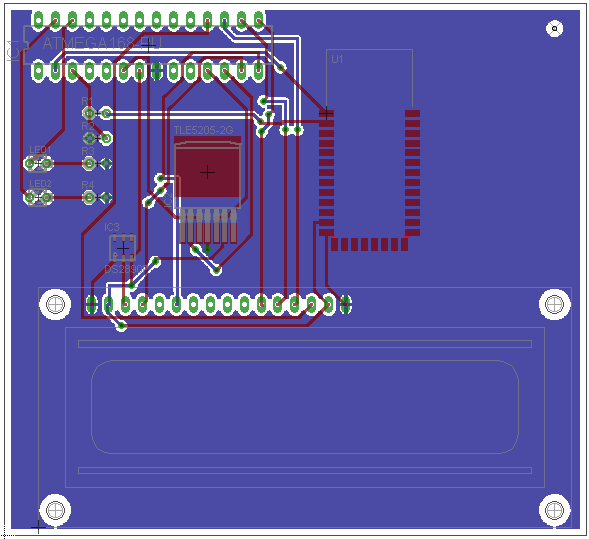
7. Schematic



Breadboard:



8. PCB Layout



9. PCB Netlist

Net Part Pad

+5V IC1 7

IC2 6

IC3 3

U1 12

X1 15

X1 2

GND IC1 8

IC2 4

IC3 1

R2 1

R3 2

R4 2

U1 13

X1 1

X1 16

GND/ANALOG IC1 22

N$1 IC2 1

N$2 IC2 7

N$3 IC1 13

X1 4

N$4 IC1 14

X1 6

N$5 IC1 15

X1 11

N$6 IC1 16

X1 12

N$7 IC1 17

X1 13

N$8 IC1 18

X1 14

N$11 IC3 2

X1 3

N$16 IC1 12

IC2 2

N$17 IC1 6

IC2 3

N$18 IC1 11

IC2 5

N$19 IC1 4

N$20 IC1 5

N$21 IC1 3

R1 1

R2 2

N$22 R1 2

U1 2

N$25 IC1 2

U1 1

N$26 IC1 26

LED1 A

N$27 IC1 27

LED2 A

N$28 LED1 K

R3 1

N$29 LED2 K

R4 1