Lab 05

Pulse Width Modulation

Logan Barber, Ian Nail

March 4, 2022

Ian Nail

Logan Barber

ME-4370 - Stephen Canfield

1 Executive Summary

In this lab, the objective is to use the Arduino Mega 2560 to develop a product that makes use of the servo motors with a Pulse Width Modulated (PWM) signal. Two servo motors are used in making a robot arm. A joystick was used as the controller to the robot arm.

Servos are extremely useful in robotics. The motors are small, have built-in control circuitry, and are extremely powerful for their size. The output shaft of the servo is capable of traveling around 180 degrees.

To make use of the PWM functionality of the Mega 2560, the correct clock and compare bits need to be set. The data sheet of the ATMEGA 2560 gives instructions on the register values required for the different options. The fast PWM with a 256 pre-scaler is used. Two PWM signals are used to control two servo motors. One servo motor on PH6 and the other on PB5. The servo motors move according to the duty cycle of the PWM signal. If the duty cycle is 0%, the motor moves to the 0 degree position. If the duty cycle is 50%, it moves to the 180 degree position. The duty cycle is changed by setting the OCRxx bit to a new value, were xx is the timer and pin connection. OCR1A is Timer 1 and channel A that is mapped to PB5.

The joystick is what controls the movement of the motors. The joystick values are read from the analog to digital pins. The joystick is capable of y direction and x direction. The voltages are sent to the ADC pins and converted into signals to use in software. The software take the signals and makes decisions on which direction to move the servo motors.

2 Analog to Digital Conversion

The Joystick is powered with 5 Vdd and GND. When the stick is moved analog voltages change on Vrx and Vry according to the movement of the stick. These analog values are sent to the ADC pins and the conversion to a digital number happens. That digital number is displayed onto the LCD. The software makes decisions on if the servo motor should be moved when the joystick is pressed.

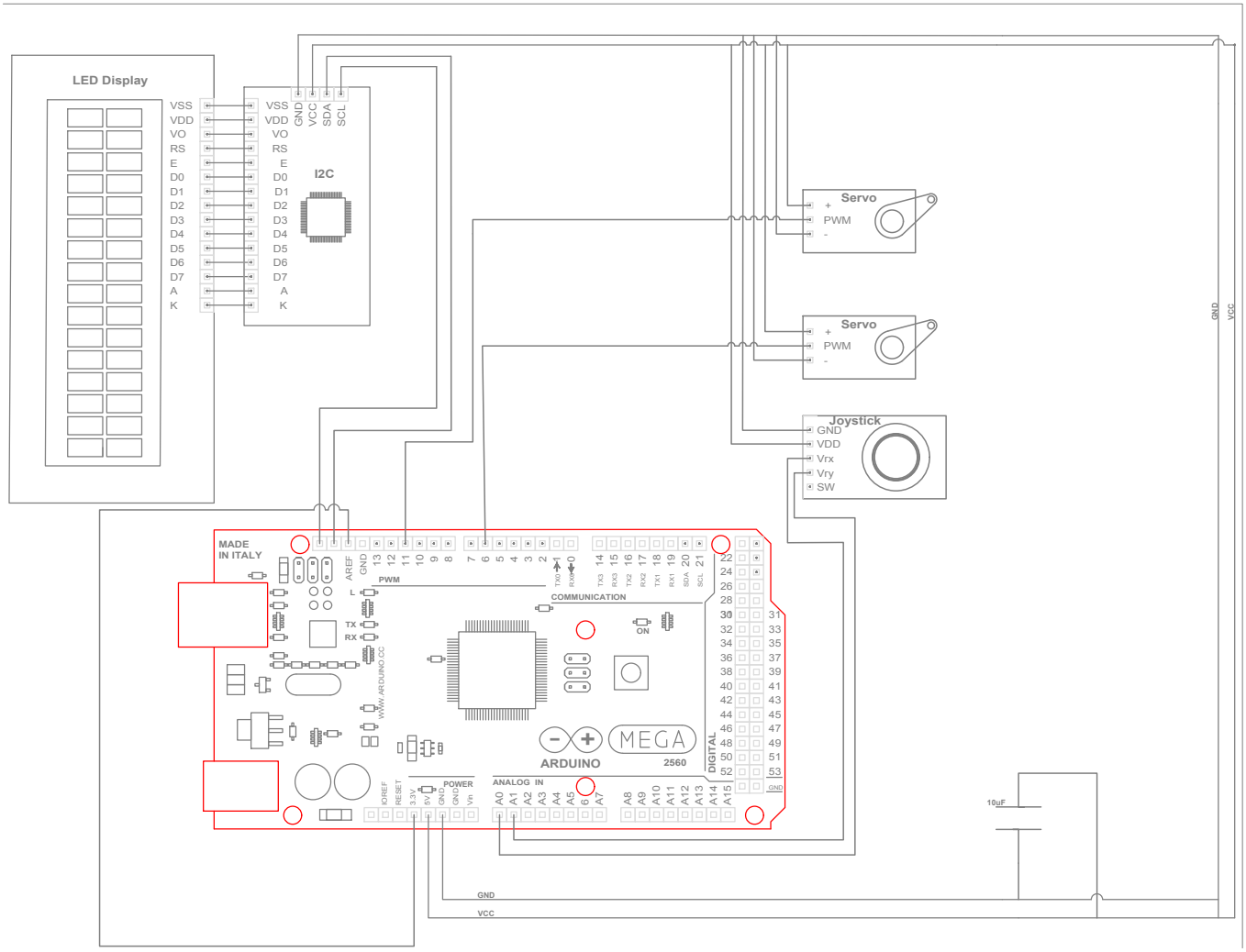

Figure 1: Circuit Diagram

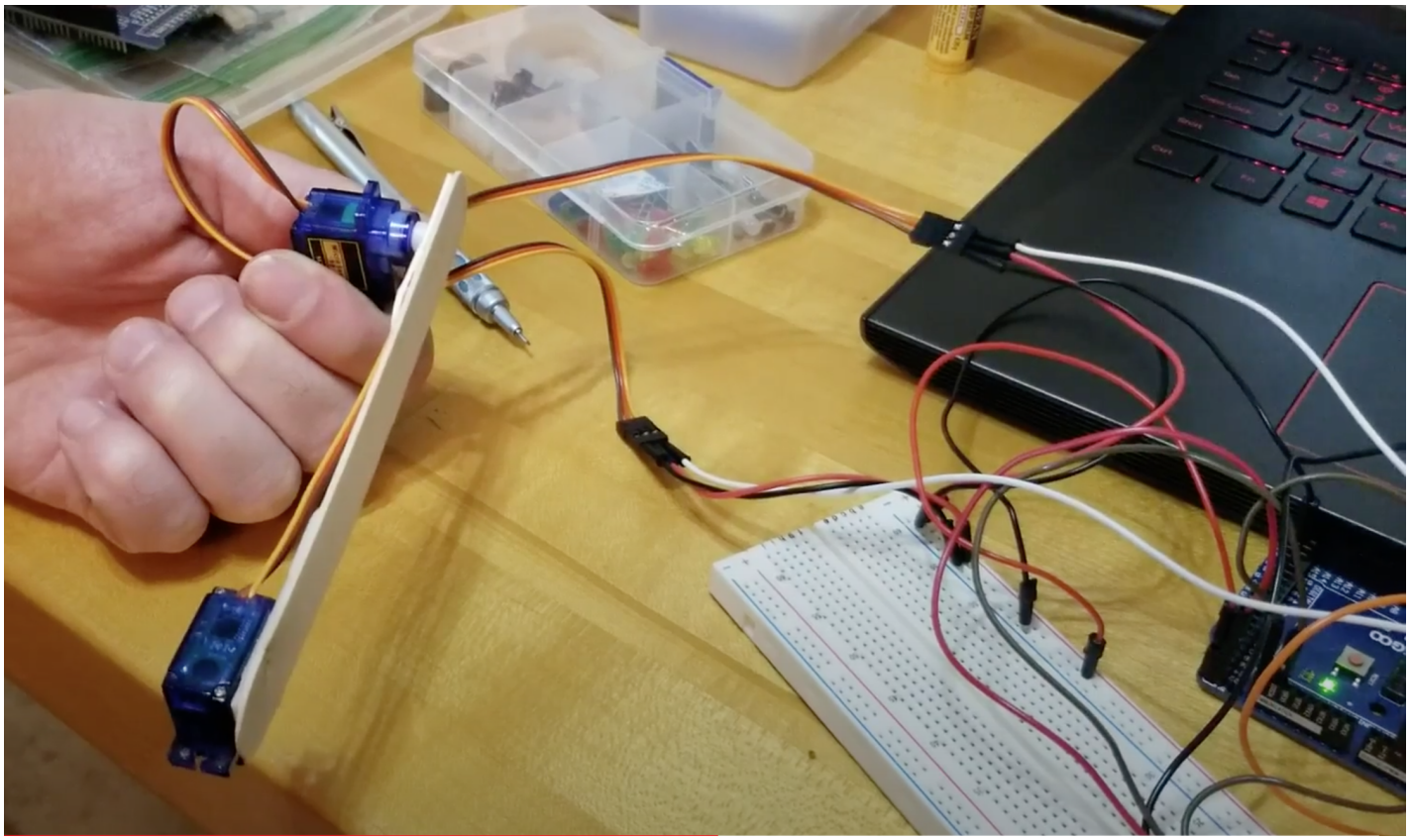

Figure 2: Breadboard and Conections

3 Source Code

```
/* AUTHORS: A. LOGAN BARBER; IAN NAIL
 * FILE NAME: Lab05.ino
 * LAST UPDATED: 4 MARCH 2022
 *
 * PURPOSE: MOVE TWO SERVO MOTORS WITH PWM FROM A JOYSTICK
 *
 */

 /********* G L O B A L  V A R I A B L E S *********/
 uint16_t valueADC0 = 0; // THE VALUE ON ANANLOG CHANNEL 0
 uint16_t valueADC1 = 0; // THE VALUE ON ANANLOG CHANNEL 1
 uint8_t servo1 = 25; // HOLDS THE POSITION FOR SERVO 1
 uint8_t servo2 = 32; // HOLDS THE POSITION FOR SERVO 2

// INCLUDE LIBRARIES
#include <LiquidCrystal_I2C.h>

// DEFINE MACROS FOR LCD SERIAL
#define ADDRESS 0x27
#define LCDCOLS 16
#define LCDROWS 2

// CREATE LiquidCrystal OBJECT
LiquidCrystal_I2C lcd(ADDRESS, LCDCOLS, LCDROWS);

void setup()
{
    // DISABLE DIGITAL BUFFER FOR ALL ANALOG PINS
    DIDR0 = 0xFF;
    DIDR2 = 0xFF;

    // SET UP ANALOG TO DIGITIAL CONVERSION IN FREE RUNNING MODE
    ADCSRB = 0x00; // 0b00000000

    // SET UP PINS FOR OUTPUT
    DDRB |= 1<<PB5;
    DDRH |= 1<<PH3;

    // USE TCNT1
    TCCR1A = 0b10000011; // non-inverted mode, 10bit fast pWM
    TCCR1B = 0b00001100; // (I/O clk)/256 (From prescaler)

    // USE TCNT2
    TCCR4A = 0b10000011; // non-inverted mode, 10bit fast pWM
    TCCR4B = 0b00001100; // (I/O clk)/256 (From prescaler)
    OCR1A = 40;
    OCR4A = 40;

    // INITIALIZE THE LCD SCREEN
    lcd.begin();
}

void loop()
{
    // READ THE INPUT ON ANALOG CHANNEL 0
    ADMUX = 0x40; // 0b01000000

```

```
    // ENABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
58  ADCSRA = 0x97; // 0b10010111

60  // START THE CONVERSION
    ADCSRA = 0xD7; // 0b11010111
62
    // WAIT HERE DURING THE CONVERSION
64  while((ADCSRA & 0x10) == 0x00)
    {
66      asm("NOP");
    }
68
    // IF THE CONVERSION IS COMPLETE
70  if((ADCSRA & 0x10) == 0x10)
    {
72      // DISABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
        ADCSRA = 0x17; // 0b00010111
74
        // READ THE ADC VALUE
76      valueADC0 = ADCL;
        valueADC0 += (ADCH << 8);
78  }


80
    // READ THE INPUT ON ANALOG CHANNEL 1
82  ADMUX = 0x41; // 0b01000001

84  // ENABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
    ADCSRA = 0x97; // 0b10010111
86
    // START THE CONVERSION
88  ADCSRA = 0xD7; // 0b11010111

90  // WAIT HERE DURING THE CONVERSION
    while((ADCSRA & 0x10) == 0x00)
92  {
        asm("NOP");
94  }

96  // READ THE CONVERSION VALUE
    if((ADCSRA & 0x10) == 0x10)
98  {
        // DISABLE THE ADC, AND RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
100     ADCSRA = 0x17; // 0b00010111

102     // GET THE HIGH BITS
        valueADC1 = ADCL;
104     valueADC1 += (ADCH << 8);
    }
106
    // IF THE VALUE ON A1 IS LESS THAN 101, MOVE THE FIRST ARM FORWARD
108 if((valueADC1 < 101) && (servo1 > 25))
    {
110     servo1 -= 1;
        OCR1A = servo1;
112 }
    // IF THE VALUE ON A1 IS GREATER THAN 899, MOVE THE FIRST ARM BACKWARD
114 else if((valueADC1 > 899) && (servo1 < 150))
    {
116     servo1 += 1;
```

```
        OCR1A = servo1;
    }

    // IF THE VALUE ON A0 IS LESS THAN 101, MOVE THE SECOND ARM COUNTER
    CLOCKWISE
    if((valueADC0 < 101) && (servo2 > 32))
    {
        servo2 -= 1;
        OCR4A = servo2;
    }
    // IF THE VALUE ON A0 IS GREATER THAN 899, MOVE THE SECOND ARM CLOCKWISE
    else if((valueADC0 > 899) && (servo2 < 150))
    {
        servo2 += 1;
        OCR4A = servo2;
    }


    //ADMUX = 0x41; // 0b01000001
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("servo1: ");
    lcd.print(servo1);
    lcd.setCursor(0, 1);
    lcd.print("servo2: ");
    lcd.print(servo2);
    delay(10);
}
```

../Lab05.ino