Isaias Osorio

CPE301 – SPRING 2016

Design Assignment 5

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **SUBMISSION ITEM** | **COMPLETED (Y/N)** | **MARKS**  **(/MAX)** |
| 0. | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS |  |  |
| 1. | INITIAL CODE OF TASK 1/A |  |  |
| 2. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 2/B |  |  |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 3/C |  |  |
| 4. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 4/D |  |  |
| 5. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 5/E |  |  |
| 6. | SCHEMATICS |  |  |
| 7. | SCREENSHOTS OF EACH TASK OUTPUT |  |  |
| 8. | SCREENSHOT OF EACH DEMO |  |  |
| 9. | VIDEO LINKS OF EACH DEMO |  |  |
| 10. | GOOGLECODE LINK OF THE DA |  |  |
|  |  |  |  |
|  |  |  |  |

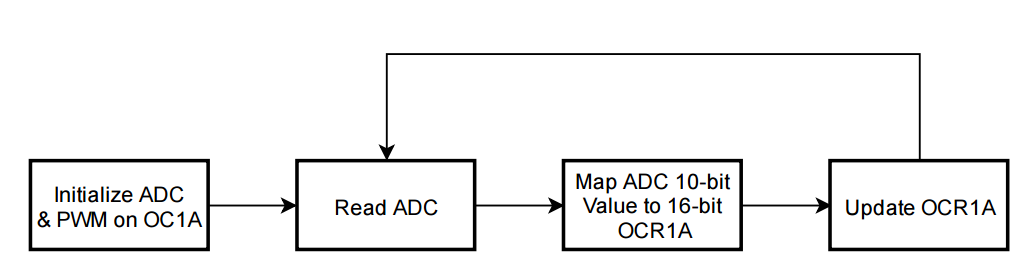
|  |  |  |  |
| --- | --- | --- | --- |
| 0. | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS |  |  |

* Atmega328P
* DC motor
* Servo motor
* Stepper motor
* ULN2803
* 5V power supply

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | INITIAL CODE OF TASK 1 |  |  |

Write an AVR C program to control the speed of the DC Motor using a potentiometer connected to any of the analog-in port.

## Flow chart of Task 1



#define *F\_CPU* 8000000UL // XTAL = 8MHZ

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#define BAUDRATE 9600 // Define baudrate

#define ASYNCH\_NORM\_PRESCALER (*F\_CPU*/16/BAUDRATE - 1) // Calculate prescaler for USART0

void ADC0init(); // Initialize ADC0 input

void PWM\_OC1A\_init(); // Initialize PWM on OC1A at 50Hz

unsigned short readADC(); // read ADC0 analog input and return it

void updateDC\_OC1A(unsigned char); // Change duty cycle on OC1A

int USART0\_sendChar(char, *FILE*\*); // Send character on USART0

void usart0\_init (void); // Initialize USART0

// reset stream pointer

// http://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

*FILE* USART0\_stream = *FDEV\_SETUP\_STREAM*(USART0\_sendChar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

int main()

{

unsigned short adcVal; // Variable to store input ADC Value

unsigned char dc; // Store calculated DC value based on adcVal

*stdout* = &USART0\_stream;// change standard output to point to a USART stream

PWM\_OC1A\_init(); // initialize pwm on OC1A

ADC0init(); // Initialize ADC0 input

usart0\_init(); // Initialize USART0 for debugging and monitoring

while (1)

{

adcVal = readADC(); // read ADC0;

dc = (unsigned short)(100.0\*adcVal / 1023); // get percentage of input voltage from Vcc.

updateDC\_OC1A(dc); // Update OCR1A to update duty cycle of OC1A

*printf*("ADC Value = %u\n", adcVal); // Monitoring output

*printf*("\tDuty cycle = %u%%\n", dc); // Monitoring output

*\_delay\_ms*(100); // Have an imperceivable delay

}

}

void usart0\_init (void)

/\*

\* Procedure to initialize USART0 asynchronous with enabled RX/TX, 8 bit data,

\* no parity, and 1 stop bit.

\*/

{

UCSR0B = (1<<TXEN0) | (1<<RXEN0); // enable transmit/receive

UCSR0C = (1<<UCSZ01) | (1<<UCSZ00); // asynchronous, 8N1

UBRR0L = ASYNCH\_NORM\_PRESCALER; // Set prescaler based on desired baudrate

}

int USART0\_sendChar(char data, *FILE* \*stream)

/\*

\* Procedure to send a single character over USART0. If character is linefeed, reset

\* line.

\* Assumes ASCII code.

\*/

{

if(data == '\n') // If character is linefeed,

{ // First send return.

while(! (UCSR0A & (1<<UDRE0)) );

UDR0 = '\r';

}

while(! (UCSR0A & (1<<UDRE0)) ); // Wait for last data to be transmitted.

UDR0 = data; // send data.

return 0;

}

void updateDC\_OC1A(unsigned char DC)

// Procedure to update PWM duty cycle on OC1A. Given an unsigned character DC, this

// procedure will calculate the appropriate OCR1A value based on the top value of

// Timer1.

{

OCR1A = (unsigned short)(DC \* 2499.0 / 100);

}

unsigned short readADC()

// readADC will read the adcValue after it has been calculated.

{

ADCSRA |= (1<<ADSC); // Begin conversion

while((ADCSRA & (1<<ADIF)) == 0 ); // Wait for conversion to finish.

return ADC;

}

void PWM\_OC1A\_init()

{

//Set PORTB1 pin as output

DDRB |= (1<<DDB1); // make OC1A as output.

// Output compare mode on OC1A. Fast PWM with top = ICR1.

// Clear OC1A on Compare match and set at bottom.

TCCR1A |= (1<<COM1A1)|(0<<COM1A0)|(0<<COM1B1)|(0<<COM1B0)|(0<<FOC1A)|(0<<FOC1B)|(1<<WGM11)|(0<<WGM10);

// Start timer with prescaler 64

TCCR1B |= (0<<ICNC1)|(0<<ICES1)|(1<<WGM13)|(1<<WGM12)|(0<<CS12)|(1<<CS11)|(1<<CS10);

ICR1 = 2499; // F\_CPU / (N \* F\_pwm) - 1, where N is the prescaler = 64, and F\_pwm is the desired 50Hz frequency.

}

void ADC0init()

// ADC0init will initialize analog input on ADC0, set voltage reference to Vcc, with

// data right justified on data register.

{

DDRC &= ~(0<<DDC0);

ADCSRA = 0x87; // Make ADC enable and select ck/128

ADMUX = (1<<REFS0); // VCC reference, ADC0 single ended input

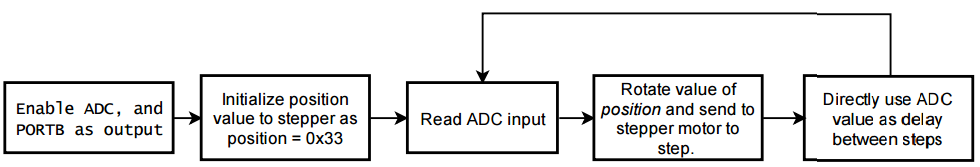
// data will be right-justified

}

|  |  |  |  |
| --- | --- | --- | --- |
| 2. | INITIAL CODE OF TASK 2 |  |  |

Write an AVR C program to control the speed of the Stepper Motor using a potentiometer connected to any of the analog-in port.

## Flow chart of task 2



#define *F\_CPU* 8000000UL // XTAL = 8MHZ

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#define BAUDRATE 9600

#define ASYNCH\_NORM\_PRESCALER (*F\_CPU*/16/BAUDRATE - 1)

void ADC0init(); // Initialize ADC0 input

unsigned short readADC(); // read ADC0 analog input and return it

void delay\_ms(unsigned int); // shell procedule to call \_delay\_ms on variable input

void step\_clockwise(unsigned int, unsigned int); // step stepper motor desired number of times with delay

unsigned char rotateLeft(unsigned char); // rotate bits of input to the left.

int USART0\_sendChar(char, *FILE*\*); // Write character to USART0

void usart0\_init (void); // Initialize USART0

// reset stream pointer

// http://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

*FILE* USART0\_stream = *FDEV\_SETUP\_STREAM*(USART0\_sendChar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

// Current position signal of stepper motor

unsigned char positionSig = 0x33;

int main()

{

unsigned short adcVal;

DDRB = 0xFF; // make portB output pins.

*stdout* = &USART0\_stream; // change standard output to point to a USART stream

usart0\_init(); // Initialize USART0 for debugging and monitoring

ADC0init(); // Initialize ADC0 input

while (1)

{

adcVal = readADC(); // read ADC0;

step\_clockwise(1, adcVal); // Step stepper motor 1 step with an adcVal delay

// print monitoring message

*printf*("ADC Value: %u | Position signal: 0x%X\n", adcVal, positionSig);

}

}

void usart0\_init (void)

/\*

\* Procedure to initialize USART0 asynchronous with enabled RX/TX, 8 bit data,

\* no parity, and 1 stop bit.

\*/

{

UCSR0B = (1<<TXEN0) | (1<<RXEN0); // enable transmit/receive

UCSR0C = (1<<UCSZ01) | (1<<UCSZ00); // asynchronous, 8N1

UBRR0L = ASYNCH\_NORM\_PRESCALER; // Set prescaler based on desired baudrate

}

int USART0\_sendChar(char data, *FILE* \*stream)

/\*

\* Procedure to send a single character over USART0. If character is linefeed, reset

\* line.

\* Assumes ASCII code.

\*/

{

if(data == '\n') // If character is linefeed,

{ // First send return.

while(! (UCSR0A & (1<<UDRE0)) );

UDR0 = '\r';

}

while(! (UCSR0A & (1<<UDRE0)) ); // Wait for last data to be transmitted.

UDR0 = data; // send data

return 0;

}

unsigned char rotateLeft(unsigned char x)

/\*

\* Given an unsigned character x, rotateLeft will do a logic rotatation of

\* the bits of x to the right.

\*/

{

unsigned char shiftIn = 0;

if ((x & 0x80) == 0x80)

shiftIn = 0x01;

return ((x<<1) | shiftIn);

}

void step\_clockwise(unsigned int steps, unsigned int delay)

/\*

\* Given the unsigned integers steps, and delay, step\_clockwise will send the appropriate

\* signal to PORTB[7:0] to step a stepper motor in the clockwise direction.

\* A global variable positionSig must be initialized to 0x33.

\*/

{

for (; steps > 0; steps--) // loop steps times.

{

positionSig = rotateLeft(positionSig); // Rotate value of positionSig

PORTB = positionSig; // send data to PORTB

delay\_ms(delay); // Delay a given value of milliseconds.

}

}

unsigned short readADC()

// readADC will read the adcValue after it has been calculated.

{

ADCSRA |= (1<<ADSC); // Begin conversion

while((ADCSRA & (1<<ADIF)) == 0 ); // Wait for conversion to finish.

return ADC;

}

void ADC0init()

// ADC0init will initialize analog input on ADC0, set voltage reference to Vcc, with

// data right justified on data register.

{

DDRC &= ~(0<<DDC0);

ADCSRA = 0x87; // Make ADC enable and select ck/128

ADMUX = (1<<REFS0); // VCC reference, ADC0 single ended input

// data will be right-justified

}

void delay\_ms(unsigned int count)

/\*

\* Procedure to perform a delay based on an unsigned short

\* since the \_delay\_ms macro will not accept parameters

\* other than constant values.

\*/

{

int i;

for(i = 0; i < count; i++)

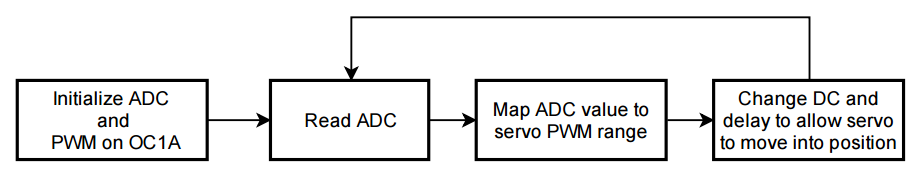
*\_delay\_ms*(1);

}

|  |  |  |  |
| --- | --- | --- | --- |
| 3. | INITIAL CODE OF TASK 3 |  |  |

Write an AVR C program to control the position of the Servo Motor using a potentiometer connected to any of the analog-in port. When pot value is 0 the servo is at position 0 deg. and when pot value is max (approx. 5V) the servo is at position 180 deg.

## Flow chart of task 3



#define *F\_CPU* 8000000UL // XTAL = 8MHZ

#include <stdio.h>

#include <avr/io.h>

#include <util/delay.h>

#define SERVO\_MIN 65

#define SERVO\_MAX 285

#define BAUDRATE 9600

#define ASYNCH\_NORM\_PRESCALER (*F\_CPU*/16/BAUDRATE - 1)

void ADC0init(); // Initialize ADC0 input

unsigned short readADC(); // read ADC0 analog input and return it

void delay\_ms(unsigned int); // shell procedule to call \_delay\_ms on variable input

void PWM\_OC1A\_init(); // Initialize PWM on OC1A at 50Hz

int USART0\_sendChar(char, *FILE*\*); // Send character on USART0

void usart0\_init (void); // Initialize USART0

// reset stream pointer

// http://www.gnu.org/savannah-checkouts/non-gnu/avr-libc/user-manual/group\_\_avr\_\_stdio.html

*FILE* USART0\_stream = *FDEV\_SETUP\_STREAM*(USART0\_sendChar, *NULL*, *\_FDEV\_SETUP\_WRITE*);

int main()

{

unsigned short adcVal; // Variable to store input ADC Value

unsigned short newVal; // new value calculated based on a range for servo

DDRB = 0xFF; // make portB output pins.

ADC0init(); // Initialize ADC0 input

PWM\_OC1A\_init(); // initialize pwm on OC1A

usart0\_init(); // Initialize USART0 for debugging and monitoring

*stdout* = & USART0\_stream;// change standard output to point to a USART stream

while (1)

{

adcVal = readADC(); // read ADC0;

// Map ADC value to a range from 0 to SERVO\_MAX

newVal = (unsigned short)((float)adcVal / ((1UL<<10) - 1) \* SERVO\_MAX);

*printf*("adcVal = %u\n", adcVal); // Print monitoring data

*printf*("\tnewVal = %u\n", newVal);

if (newVal <= SERVO\_MIN) // If newVal is less than minimum servo value (0 degrees)

OCR1A = SERVO\_MIN; // then set OCR1A to minimum value.

else

{

OCR1A = newVal; // else, update OCR1A to change duty cycle.

}

*\_delay\_ms*(50); // Delay to allow servo to move

}

}

int USART0\_sendChar(char data, *FILE* \*stream)

/\*

\* Procedure to send a single character over USART0. If character is linefeed, reset

\* line.

\* Assumes ASCII code.

\*/

{

if(data == '\n')

{

while(! (UCSR0A & (1<<UDRE0)) );

UDR0 = '\r';

}

while(! (UCSR0A & (1<<UDRE0)) );

UDR0 = data;

return 0;

}

void usart0\_init (void)

/\*

\* Procedure to initialize USART0 asynchronous with enabled RX/TX, 8 bit data,

\* no parity, and 1 stop bit.

\*/

{

UCSR0B = (1<<TXEN0) | (1<<RXEN0); // enable transmit/receive

UCSR0C = (1<<UCSZ01) | (1<<UCSZ00); // asynchronous, 8N1

UBRR0L = ASYNCH\_NORM\_PRESCALER; // To set 9600 baud rate with 8MHz clock

}

unsigned short readADC()

/\*

\* Procedure to send a single character over USART0. If character is linefeed, reset

\* line.

\* Assumes ASCII code.

\*/

{

ADCSRA |= (1<<ADSC); // Begin conversion

while((ADCSRA & (1<<ADIF)) == 0 ); // Wait for conversion to finish.

return ADC;

}

void ADC0init()

// ADC0init will initialize analog input on ADC0, set voltage reference to Vcc, with

// data right justified on data register.

{

DDRC &= ~(0<<DDC0);

ADCSRA = 0x87; // Make ADC enable and select ck/128

ADMUX = (1<<REFS0); // VCC reference, ADC0 single ended input

// data will be right-justified

}

void delay\_ms(unsigned int count)

/\*

\* Procedure to perform a delay based on an unsigned short

\* since the \_delay\_ms macro will not accept parameters

\* other than constant values.

\*/

{

int i;

for(i = 0; i < count; i++)

*\_delay\_ms*(1);

}

void PWM\_OC1A\_init()

{

//Set PORTB1 pin as output

DDRB |= (1<<DDB1); // make OC1A as output.

// Output compare mode on OC1A. Fast PWM with top = ICR1.

// Clear OC1A on Compare match and set at bottom.

TCCR1A |= (1<<COM1A1)|(0<<COM1A0)|(0<<COM1B1)|(0<<COM1B0)|(0<<FOC1A)|(0<<FOC1B)|(1<<WGM11)|(0<<WGM10);

// Start timer with prescaler 64

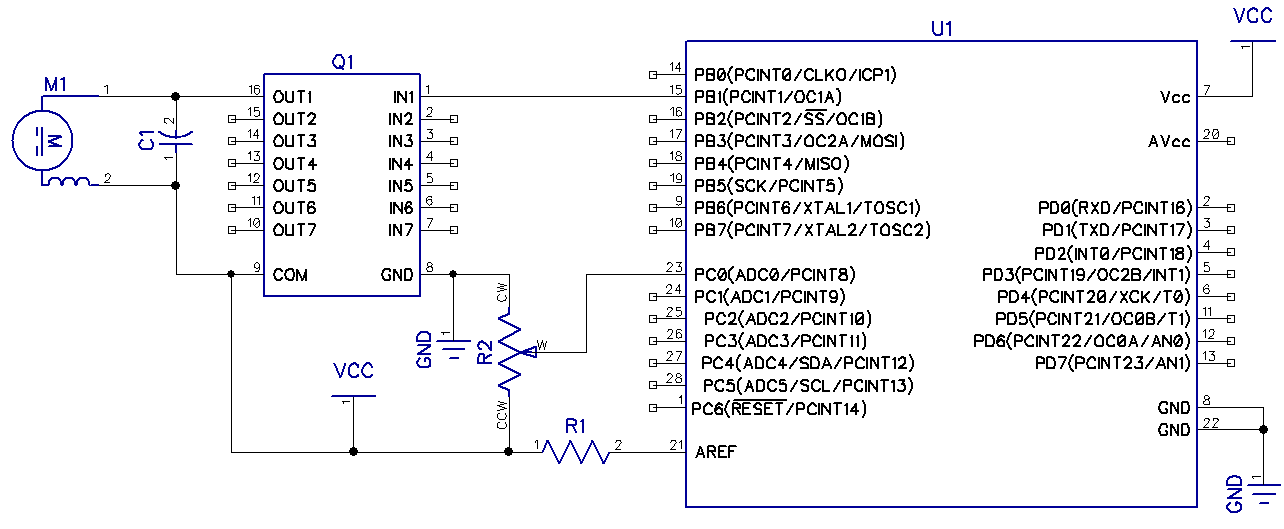
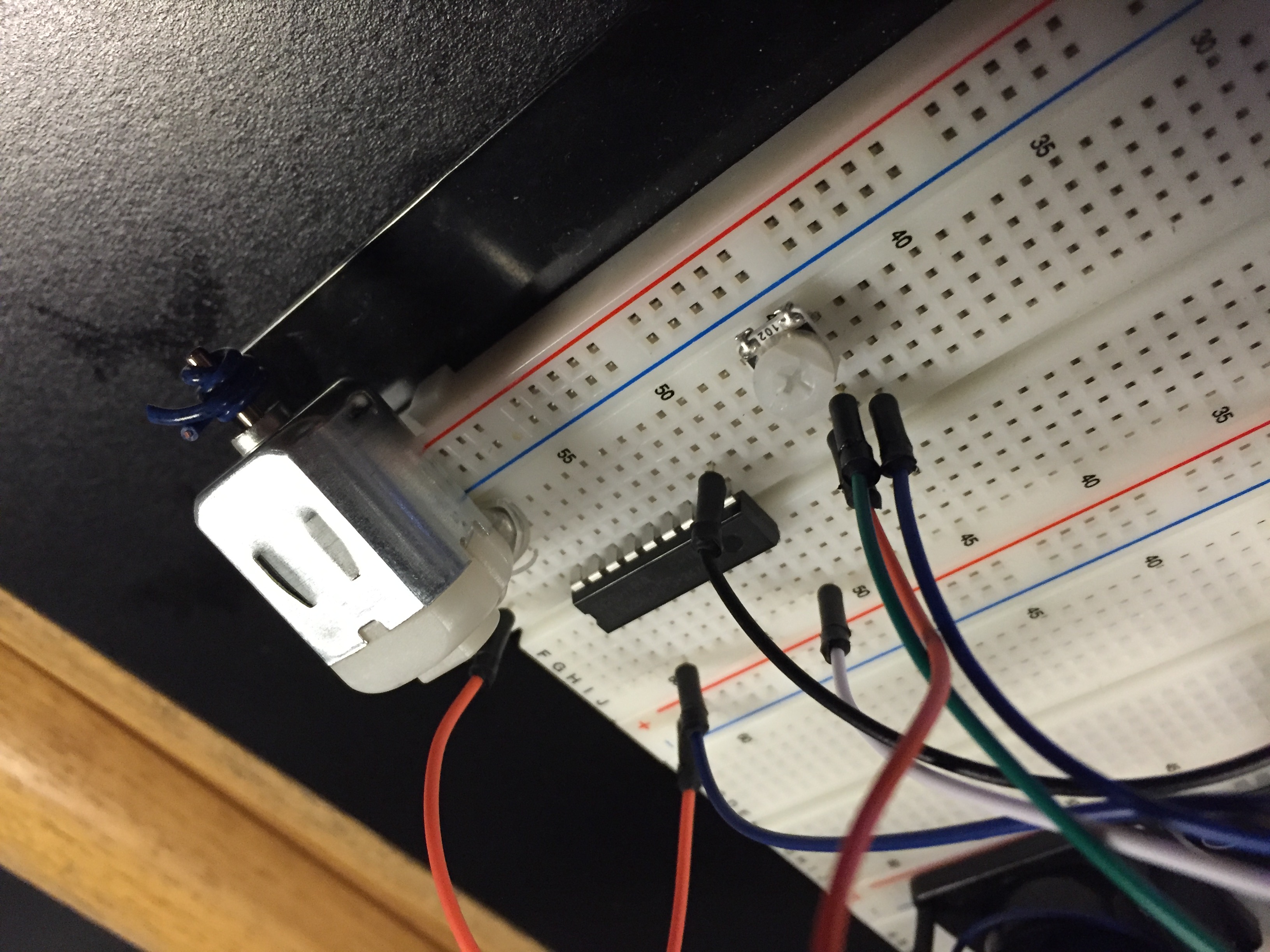
TCCR1B |= (0<<ICNC1)|(0<<ICES1)|(1<<WGM13)|(1<<WGM12)|(0<<CS12)|(1<<CS11)|(1<<CS10);

ICR1 = 2499; // F\_CPU / (N \* F\_pwm) - 1, where N is the prescaler = 64, and F\_pwm is the desired 50Hz frequency.

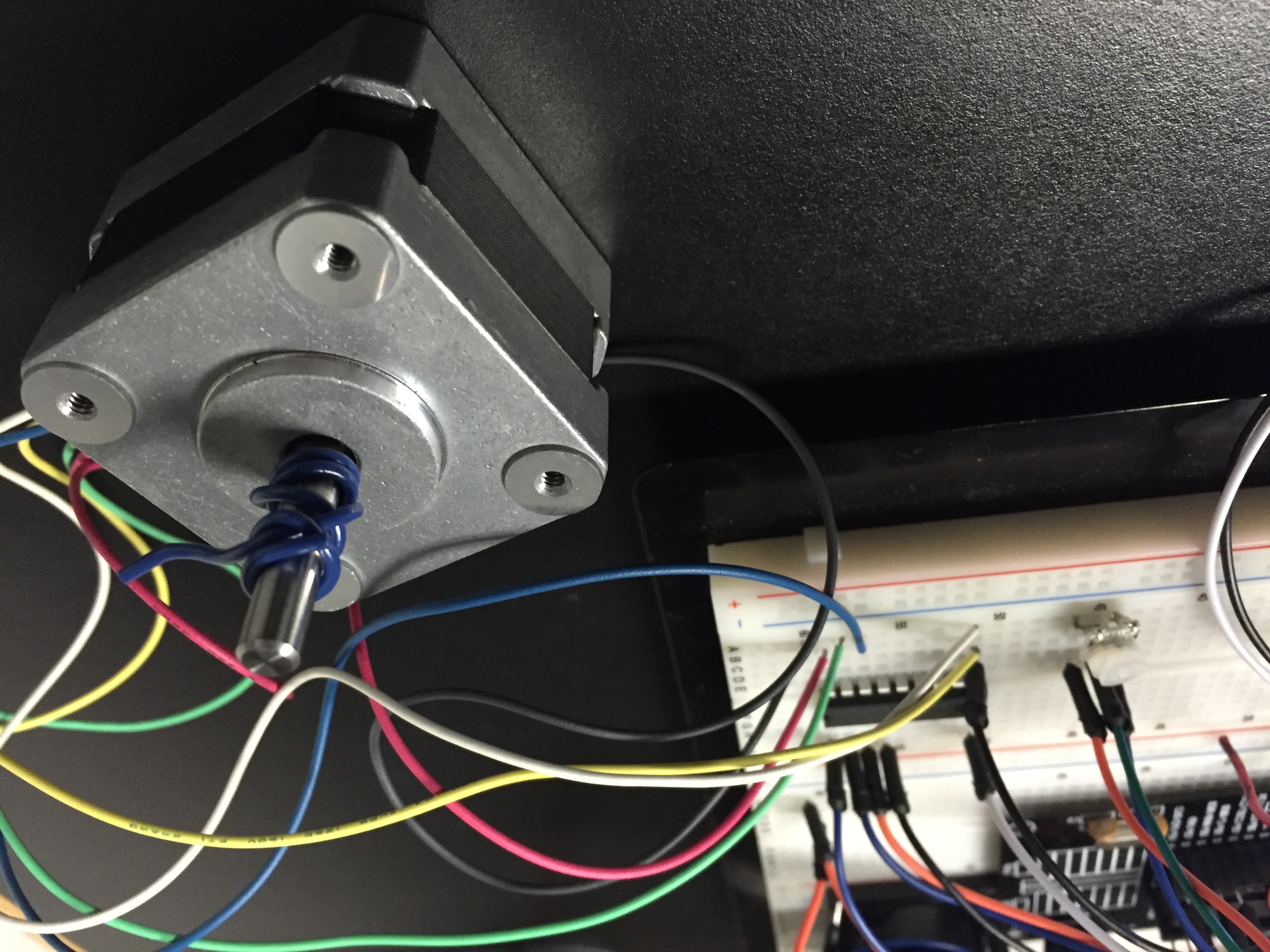
}

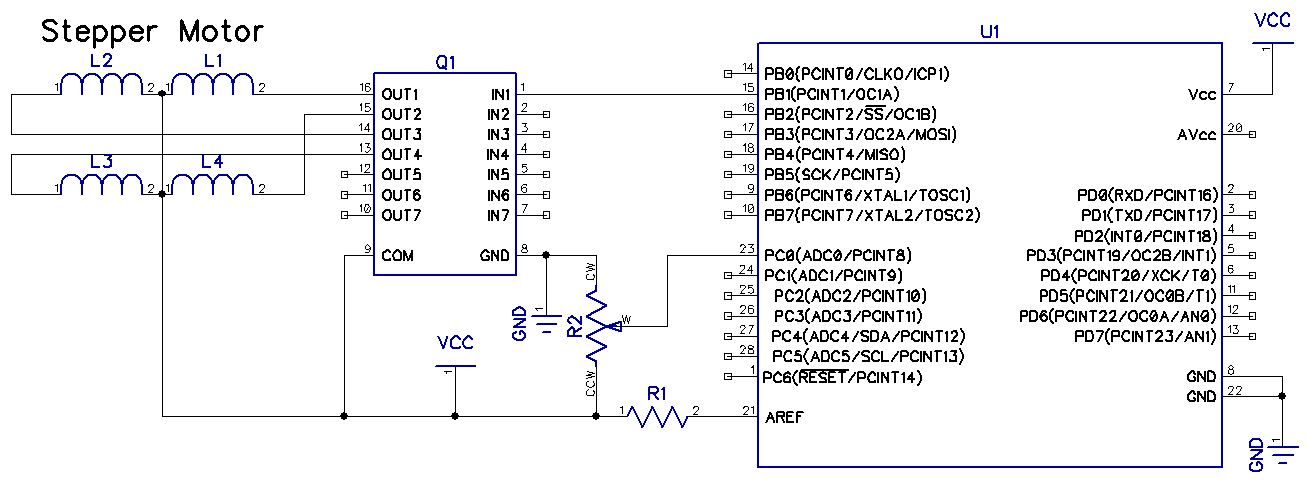
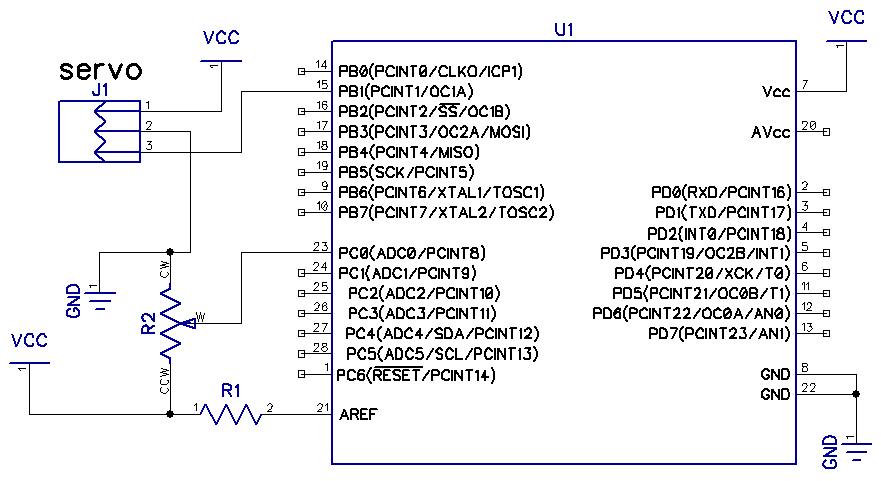
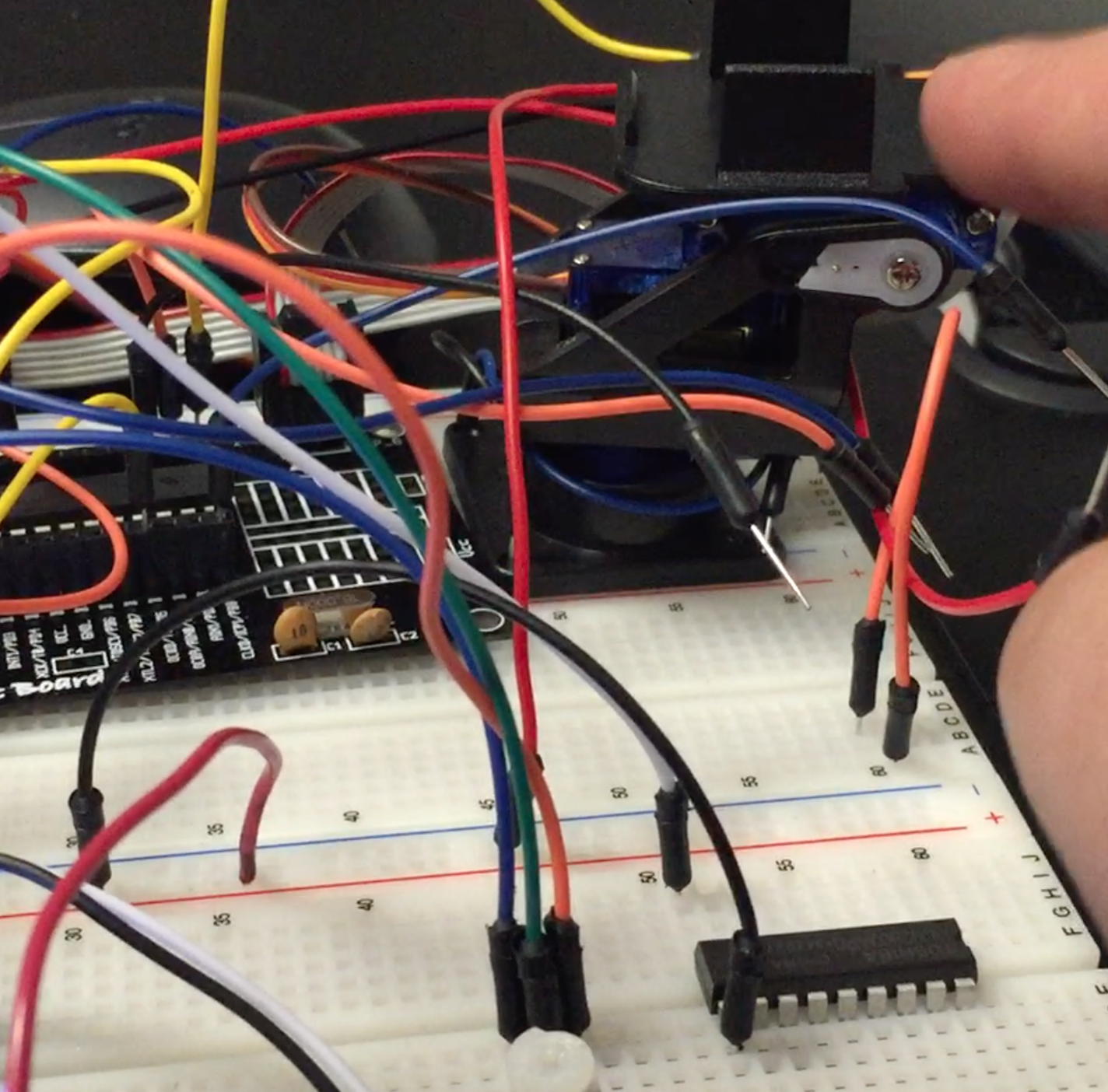
|  |  |  |  |
| --- | --- | --- | --- |
| 6. | SCHEMATICS and SCREENSHOTS |  |  |

Task 1



Task 2



Task 3  


|  |  |  |  |
| --- | --- | --- | --- |
| 9. | VIDEO LINKS OF EACH DEMO |  |  |
| <https://github.com/isaiasmoioso/CPE-301/blob/master/DA-5/da5T1.MOV>  https://github.com/isaiasmoioso/CPE-301/blob/master/DA-5/da5T2.MOV  https://github.com/isaiasmoioso/CPE-301/blob/master/DA-5/da5T3.MOV | | | |
| 10. | GITHUB REPOSITORY |  |  |
| https://github.com/isaiasmoioso/CPE-301.git | | | |

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Isaias Osorio