CPE301 – SPRING 2019

Design Assignment 4b

Student Name: Do Le

Student #: 2001183621

Student Email: led2@unlv.nevada.edu

Primary Github address: <https://github.com/DoVietLe>

Directory: <https://github.com/DoVietLe/assignments>

The goal of the assignment is to develop the above code to do the following:

1. Write an AVR C program to control the speed of the Stepper Motor using a potentiometer connected to PC0. Use a timer in CTC mode to control the delay.

2. Write an AVR C program to control the position of the Servo Motor using a potentiometer connected to PC0. 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.

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Components

• XPlained Mini Board

* ATmega328PB
* Debugger
* Programmer

• Atmel Studios 7.0

* Compiler

• 28BYJ-48 Stepper Motor

• ULN2003 Stepper Motor Driver

• 3001 HB Servo Motor

• Potentiometer

Block Diagram

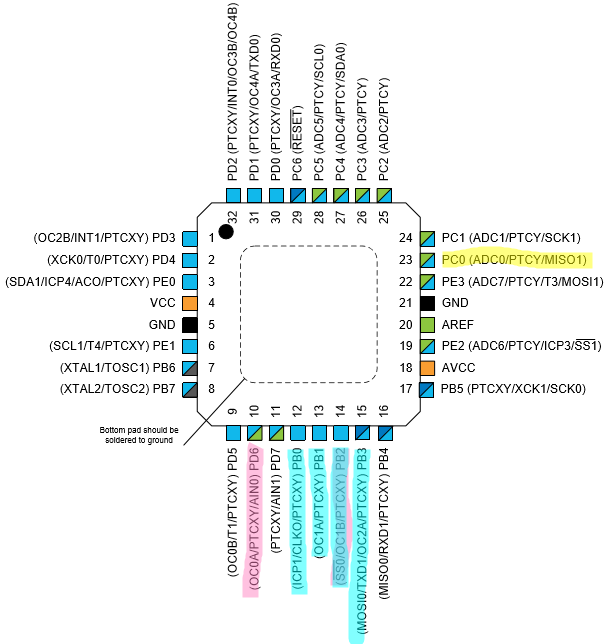


Figure : Pinout

PD6 & PB2 used for PWM.

PB[3:0] used to control stepper motor.

PC0 used for ADC.

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

C Code

#include <avr/interrupt.h>

#include <avr/io.h>

volatile *uint16\_t* currentCycle = 374;

volatile *uint8\_t* currentSequence = 0;

volatile *uint8\_t* sequences[] = {

0b00000001,

0b00000010,

0b00000100,

0b00001000,

};

/\*

volatile uint8\_t sequences[] = {

0b00000001,

0b00000011,

0b00000010,

0b00000110,

0b00000100,

0b00001100,

0b00001000,

0b00001001

};

\*/

void setTimer() {

// Sets time parameters.

TCCR1A |= 0b00000000; // Sets timer to CTC mode.

TCCR1B |= 0b00001100; // Sets pre-scaler to 256.

TCNT1 = 0;

// Sets interrupts.

TIMSK1 |= (1<<OCIE1A); // Enables interrupt on compare A match.

TIFR1 |= (1<<OCF1A); // Clears interrupt flag.

sei(); // Enables global interrupt.

}

void setDelay(int cycles) {

// Updates the delay.

OCR1A = cycles;

}

/\*

void delay() {

TCNT1 = 1; // Sets TCNT1 to 1.

while (TCNT1 != 0); // Waits until TCNT1 is 0 (OCR1A value reached).

}

\*/

void enableADC() {

// Sets PINC0 as input.

DDRC &= ~(1<<PINC0);

// Sets up ADC settings.

ADMUX = 0x40; // Reads from ADC0. Right justified. Internal 1.1V reference.

ADCSRB = 0x00; // Free running mode.

ADCSRA = 0b10000111; // Enables ADC with 128 pre-scaler.

}

void disableADC() {

ADCSRA &= 0x7F; // Disables the ADEN bit.

}

float analogRead() {

ADCSRA |= (1<<6); // Enables ADSC to start AD conversion.

while (!(ADCSRA&(1<<4))); // Waits until ADIF is set indicating ADC is done.

ADCSRA |= (1<<4); // Clears ADIF flag.

return ADC; // Returns value.

}

*uint16\_t* getCycles() {

// Calculates cycles. Minimum delay is 100ms and maximum is 1s.

*uint16\_t* cycles = 374 + (3749-374)\*analogRead()/1023;

return cycles;

}

void setPorts() {

DDRB = 0x0F; // Enables PINB[3:0] to control the motor.

PORTB = sequences[currentSequence];

}

int main(void)

{

setPorts();

enableADC();

setDelay(currentCycle);

setTimer();

while (1) {

// Calculates cycles based on ADC value.

*uint16\_t* cycles = getCycles();

// If cycles is different than the current cycle, update the delay.

if (cycles <= currentCycle - 15 || cycles >= currentCycle + 15) {

currentCycle = cycles;

setDelay(currentCycle);

}

}

}

// Occurs when CTC value is reached.

ISR(TIMER1\_COMPA\_vect) {

// Updates the current sequence.

currentSequence = (currentSequence + 1) % 4;

// Sets the pins to the current sequence based on the table.

PORTB = sequences[currentSequence];

// Clears the interrupt flag.

TIFR1 |= (1<<OCF1A);

}

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

C Code

#include <avr/io.h>

#define cycles\_20ms 4999

#define cycles\_1ms 249

#define cycles\_2ms 499

volatile *uint16\_t* currentCycles = (cycles\_2ms+cycles\_1ms)/2;

void enableADC() {

// Sets PINC0 as input.

DDRC &= ~(1<<PINC0);

// Sets up ADC settings.

ADMUX = 0x40; // Reads from ADC0. Right justified. AVCC reference.

ADCSRB = 0x00; // Free running mode.

ADCSRA = 0b10000111; // Enables ADC with 128 pre-scaler.

}

void disableADC() {

ADCSRA &= 0x7F; // Disables the ADEN bit.

}

float analogRead() {

ADCSRA |= (1<<6); // Enables ADSC to start AD conversion.

while (!(ADCSRA&(1<<4))); // Waits until ADIF is set indicating ADC is done.

ADCSRA |= (1<<4); // Clears ADIF flag.

return ADC; // Returns value.

}

void enablePWM() {

DDRB |= (1<<PINB2); // Uses PINB2 as output for OC1B.

OCR1A = cycles\_20ms; // Sets the frequency of the PWM.

TCCR1A = 0b00100011; // Use Fast PWM on OC1B in non-inverting mode.

TCCR1B = 0b00011011; // Use Fast PWM with 64 prescaler.

}

void setDuty(*uint16\_t* cycles) {

OCR1B = cycles;

}

int main(void)

{

// Enable settings.

enableADC();

setDuty(currentCycles);

enablePWM();

while (1) {

// Calculates cycles based on analog signal. (min is cycles\_1ms, max is cycles\_2ms).

*uint16\_t* cycles = cycles\_1ms + (analogRead()/1023)\*(cycles\_2ms - cycles\_1ms);

// Changes cycle if a change is detected.

if (cycles <= currentCycles - 1 || cycles >= currentCycles + 1) {

currentCycles = cycles;

setDuty(currentCycles);

}

}

}

1. **SCHEMATICS**

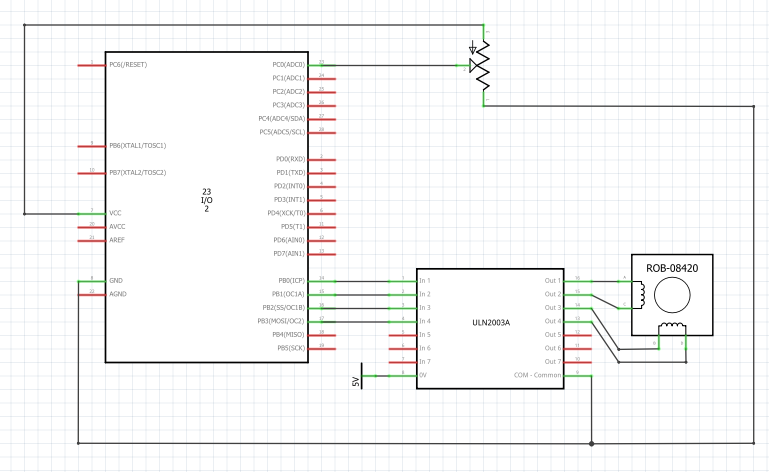
Task 1 Schematic

Figure : Task 1 schematic

Task 2 schematic

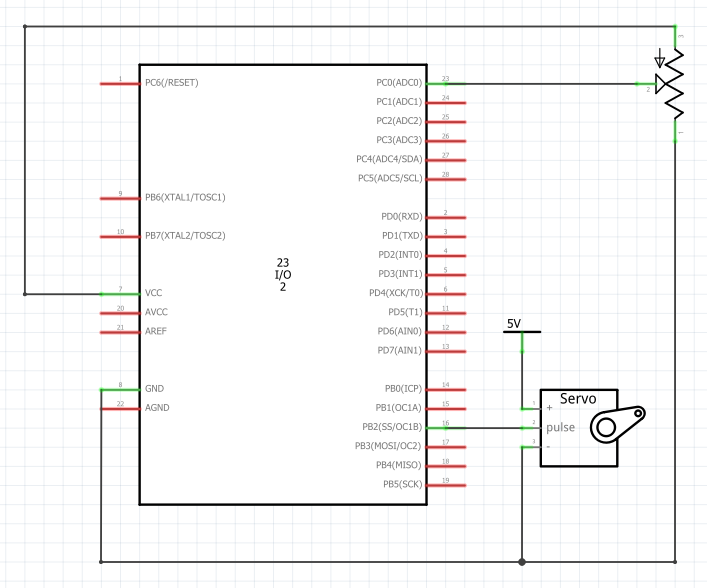


Figure : Task 2 schematic

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

Task 1

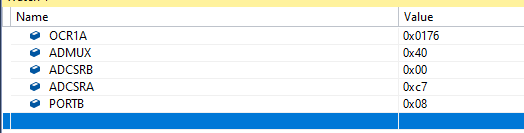


Figure : Task 1 settings

Task 2

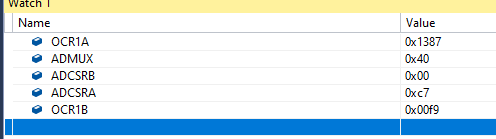


Figure : Task 2 settings

1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

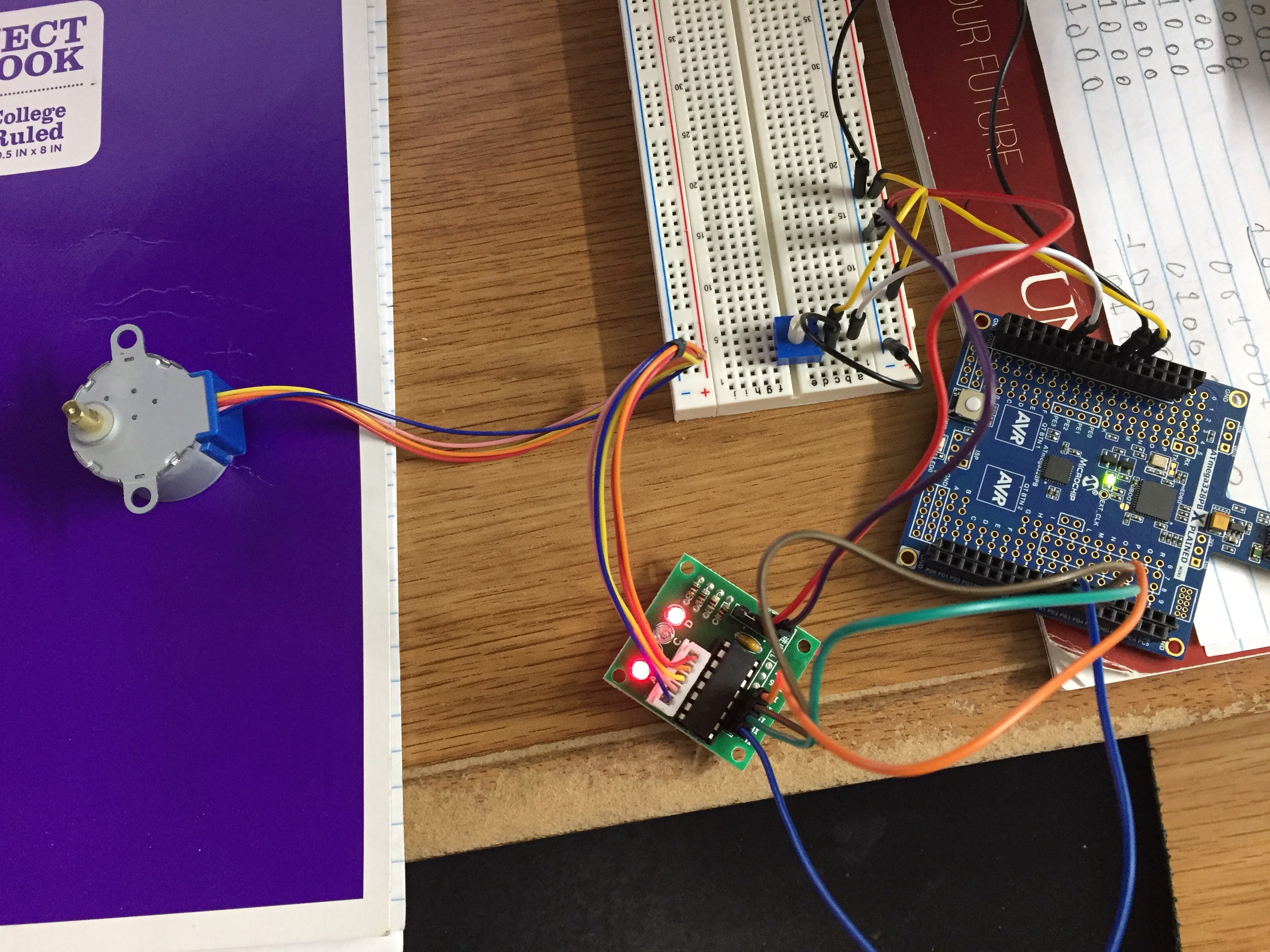
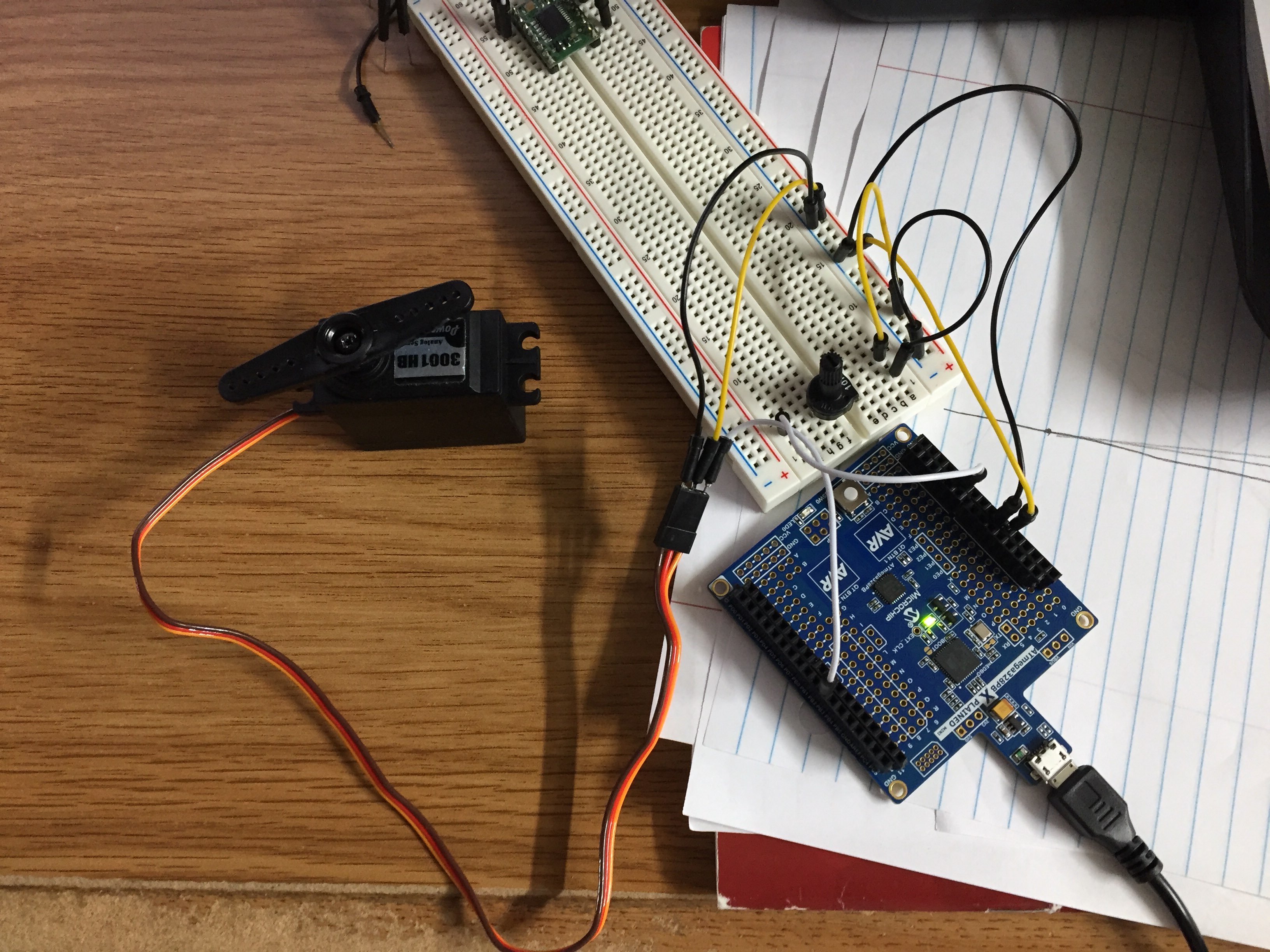


Figure : Task 2 set up

Figure : Task 1 set up

1. **VIDEO LINKS OF EACH DEMO**

Task 1

https://www.youtube.com/watch?v=3A06F05hhEc

Task 2

https://www.youtube.com/watch?v=H4cgJAE-u9Q

1. **GITHUB LINK OF THIS DA**

https://github.com/DoVietLe/assignments/tree/master/ESD301/LAB04b

**Student Academic Misconduct Policy**

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

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

Do V. Le