**CPE301 – SPRING 2023**

# Design Assignment 6

Student Name: David Lenzin
Student #: 2001654470
Student Email: lenzin@unlv.nevada.edu
Primary Github address: https://github.com/dlenzin15/submissions
Directory: submissions/DA6

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

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

List of Components used

- Atmega328PB and Multi-Functional Shield
- DG01D-E-PH Motor
- TB6612FNG dual motor driver

See schematics for pinout

## 2. INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1

Insert initial code here

```
/*
 * DA6
 * David Lenzin, 2001654470
 */

#define F_CPU 16000000UL                                    // Define F_CPU to 16 MHz
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <util/delay.h>

volatile uint8_t Direction = 0;

// Functions:
void ADC_Init(void);
int ADC_Read(char);
ISR(INT0_vect);

//UART functions for debugging
void UART_init(unsigned int);
void UART_transmit_string(char *);


void ADC_Init()                                                           // ADC Initialization function
{
       DDRC = 0x00;                                                // Make ADC port as input
       ADCSRA = 0x87;                                                     // Enable ADC, with freq/128
       ADMUX = 0x40;                                               // Vref: Avcc, ADC channel: 0
}

int ADC_Read(char channel)                                  // ADC Read function
{
       ADMUX = 0x40 | (channel & 0x07);              // set input channel to read
       ADCSRA |= (1<<ADSC);                                 // Start ADC conversion
       while (!(ADCSRA & (1<<ADIF)));                       // Wait until end of conversion by polling ADC interrupt
flag
       ADCSRA |= (1<<ADIF);                                 // Clear interrupt flag
       _delay_us(1);                                               // Wait a little bit
       return ADCW;                                            // Return ADC word
}

ISR(INT0_vect)
{
       TCCR0B |= (0<<CS00)|(0<<CS01);                   // Set Fast PWM with Fosc/64 Timer0 clock
       _delay_us(5000);                                        // Software de-bouncing control delay
       TCCR0B |= (1<<CS00)|(1<<CS01);                   // Set Fast PWM with Fosc/64 Timer0 clock

}

void UART_init(unsigned int ubrr)
{
       //Set baud rate
       UBRR0H = (unsigned char)(ubrr>>8);
       UBRR0L = (unsigned char)ubrr;

       //Enable transmitter and receiver and reciever interrupt
       UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0);

       //Set frame format: 8 bits data, 1 stop bit
       UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);

       sei();
}

void UART_transmit_string(char *data) {
       while ((*data != '\0')) {   // Check if NULL char
              while (!(UCSR0A & (1 <<UDRE0)));   // Wait for register to be
              UDR0 = *data; // Store data in the data register
              data++;
       }
}

int main(void)
{
       DDRD &= ~(1<<PD2);                               // Make INT0 pin as Input
       PORTD |= (1 << PD2);                      // turn On the Pull-up
       DDRD |= (1<<PD6) | (1<<PD5) | (1<<PD4) | (1<<PD1); // Set AIN2, AIN1, STBY to outputs
       PORTD &= ~(1 << PD5); //set AIN2 low
       PORTD |= (1 << PB4) | (1 << PD1); //set AIN1 and STBY high

       EICRA |= (1 << ISC01);  // set INT0 to trigger to falling edge
       EIMSK |= (1 << INT0);   // Turns on INT0
       sei();                            // Enable Global Interrupt

       ADC_Init();                       // Initialize ADC
       UART_init(MYUBRR);
       TCNT0 = 0;                        // Set timer0 count zero
       TCCR0A |= (1<<WGM00)|(1<<WGM01)|(1<<COM0A1);
       TCCR0B |= (1<<CS00)|(1<<CS01);                   // Set Fast PWM with Fosc/64 Timer0 clock

       while(1)
       {
              OCR0A = (ADC_Read(0)/4);                  // Read ADC and map it into 0-255 to write in OCR0
register

              // Transmit to UART for debugging
```

```
		char buffer[100];		//Buffer to read ADC
		sprintf(buffer, "%d mV\r\n", OCR0A);	//Read the adc value into the buffer
		UART_transmit_string(buffer);	//Send the adc value to the terminal
		_delay_ms(100);		//Delay for 0.10 seconds
	}
}
```

## 3. DEVELOPED MODIFIED CODE OF TASK 2

Insert only the modified sections here

```
/*
 * DA6_Task2.c
 *
 * Created: 4/25/2023 7:38:26 PM
 * Author : david
 */

#define F_CPU 16000000UL                              // Define F_CPU to 16 MHz
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1
#define PERIOD 1/F_CPU

#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <util/delay.h>

// capture Flag
volatile uint8_t Flag;
volatile uint8_t Direction = 0;
volatile uint32_t revTickAvg;

volatile uint32_t revTick; // Ticks per revolution
volatile uint32_t revCtr;  // Total elapsed revolutions
volatile uint32_t T1Ovs2;  // Overflows for small rotations

void ADC_Init() /* ADC Initialization function */
{
	DDRC = 0x00;   /* Make ADC port as input */
	ADCSRA = 0x87; /* Enable ADC, with freq/128  */
	ADMUX = 0x40;  /* Vref: Avcc, ADC channel: 0 */
}

int ADC_Read(char channel) /* ADC Read function */
{
	ADMUX = 0x40 | (channel & 0x07); /* set input channel to read */
	ADCSRA |= (1 << ADSC);           /* Start ADC conversion */
	while (!(ADCSRA & (1 << ADIF))); /* Wait until end of conversion by polling ADC interrupt flag */
	ADCSRA |= (1 << ADIF); /* Clear interrupt flag */
	_delay_us(1);          /* Wait a little bit */
	return ADCW;           /* Return ADC word */
}

void UART_init(unsigned int ubrr)
{
        //Set baud rate
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;

        //Enable transmitter and receiver and reciever interrupt
        UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0);

        //Set frame format: 8 bits data, 1 stop bit
        UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01);

}

void UART_transmit_string(char *data) {
        while ((*data != '\0')) {   // Check if NULL char
                while (!(UCSR0A & (1 <<UDRE0)));   // Wait for register to be
                UDR0 = *data; // Store data in the data register
                data++;
        }
}

ISR(INT0_vect)
{
        TCCR0B |= (0<<CS00)|(0<<CS01);                    // Set Fast PWM with Fosc/64 Timer0 clock
        _delay_us(5000);                                         // Software de-bouncing control delay
        TCCR0B |= (1<<CS00)|(1<<CS01);                    // Set Fast PWM with Fosc/64 Timer0 clock
}

// Initialize timer
void InitTimer3(void) {
        // Set PE2 as input
        DDRE &= ~(1 << DDE2);
        PORTE |= (1 << DDE2);

        // Set Initial Timer value
        TCNT3 = 0;

        ////First capture on rising edge
        TCCR3A = 0;
        TCCR3B = (0 << ICNC3) | (1 << ICES3);
        TCCR3C = 0;

        // Interrupt setup
        // ICIE3: Input capture
        // TOIE3: Timer1 overflow
        TIFR3 = (1 << ICF3) | (1 << TOV3);    // clear pending
        TIMSK3 = (1 << ICIE3) | (1 << TOIE3); // and enable
}

void StartTimer3(void) {
        // Start timer without pre-scaler
        TCCR3B |= (1 << CS30);
}

volatile uint32_t tickv, ticks;

// capture ISR
ISR(TIMER3_CAPT_vect) {
        tickv = ICR3; // save duration of last revolution
```

```
       revTickAvg = (uint32_t)tickv + ((uint32_t)T1Ovs2 * 0x10000L);
       revCtr++;  // add to revolution count
       TCNT3 = 0; // restart timer for next revolution
       T1Ovs2 = 0;
}

// Overflow ISR
ISR(TIMER3_OVF_vect) {
       // increment overflow counter
       T1Ovs2++;
}

int main(void) {
       char outs[72];
       UART_init(MYUBRR);
       sei();
       UART_transmit_string("Connected!\n"); // we're alive!
       _delay_ms(100);
       InitTimer3();
       StartTimer3();
       UART_transmit_string("TIMER3 ICP Running \r\n");
       _delay_ms(100);

       /* set PD2 and PD3 as input */
       DDRD &= ~(1 << DDD2);                            /* Make INT0 pin as Input */
       DDRD &= ~(1 << DDD3);                            /* Make INT1 pin as Input */
       PORTD |= (1 << DDD2) | (1 << DDD3);              // turn On the Pull-up
       DDRD |= (1 << DDD6) | (1 << DDD4) | (1 << DDD5) | (1<<DDD1); /* Make PWM, AIN1, AIN2, STBY outputs */

       // We are manually setting the direction
       PORTD &= ~(1 << PD5); //set AIN2 low
       PORTD |= (1 << PB4) | (1 << PD1); //set AIN1 and STBY high
       EIMSK |= (1 << INT0) | (1 << INT1); /* enable INT0 and INT1 */
       MCUCR |= (1 << ISC01) | (1 << ISC11) |
       (1 << ISC10); /* INT0 - falling edge, INT1 - raising edge */

       // WE are not using the ADC for speed - just manually setting the value
       ADC_Init(); /* Initialize ADC */
       TCNT0 = 0;  /* Set timer0 count zero */
       TCCR0A |= (1 << WGM00) | (1 << WGM01) | (1 << COM0A1);
       TCCR0B |=
       (1 << CS00) | (1 << CS01); /* Set Fast PWM with Fosc/64 Timer0 clock */
       OCR0A = 30;

       float last_reading = 0;
       while (1) {
              OCR0A = (ADC_Read(0)/4);                 // Read ADC and map it into 0-255 to write in OCR0
register

              // Convert ticks to RPM
              float rpms = (float)PERIOD * (float)revTickAvg * 1000.0 * 2.0;

              // send Speed value to LCD or USART
              UART_transmit_string("RPMS =  ");
              sprintf(outs, "%.2f \n", rpms);
              UART_transmit_string(outs);
              _delay_ms(100);
       }
```

```
}
```

## 4. DEVELOPED MODIFIED CODE OF TASK 3

```
#define F_CPU 16000000UL                               // Define F_CPU to 16 MHz
#define PERIOD 1/F_CPU

#define SHIFT_REGISTER DDRB
#define SHIFT_PORT PORTB
#define DATA (1<<PB3) //MOSI (SI)
#define LATCH (1<<PB2) //SS (RCK)
#define CLOCK (1<<PB5) //SCK (SCK)

#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <util/delay.h>

// capture Flag
volatile uint8_t Flag;
volatile uint8_t Direction = 0;
volatile uint32_t revTickAvg;

volatile uint32_t revTick; // Ticks per revolution
volatile uint32_t revCtr;  // Total elapsed revolutions
volatile uint32_t T1Ovs2;  // Overflows for small rotations

void ADC_Init() /* ADC Initialization function */
{
       DDRC = 0x00;   /* Make ADC port as input */
       ADCSRA = 0x87; /* Enable ADC, with freq/128  */
       ADMUX = 0x40;  /* Vref: Avcc, ADC channel: 0 */
}

int ADC_Read(char channel) /* ADC Read function */
{
       ADMUX = 0x40 | (channel & 0x07); /* set input channel to read */
       ADCSRA |= (1 << ADSC);           /* Start ADC conversion */
       while (!(ADCSRA & (1 << ADIF))); /* Wait until end of conversion by polling ADC interrupt flag */
       ADCSRA |= (1 << ADIF); /* Clear interrupt flag */
       _delay_us(1);          /* Wait a little bit */
       return ADCW;           /* Return ADC word */
}

ISR(INT0_vect)
{
       TCCR0B |= (0<<CS00)|(0<<CS01);                  // Set Fast PWM with Fosc/64 Timer0 clock
       _delay_us(5000);                                       // Software de-bouncing control delay
       TCCR0B |= (1<<CS00)|(1<<CS01);                  // Set Fast PWM with Fosc/64 Timer0 clock
}

// Initialize timer
void InitTimer3(void) {
       // Set PE2 as input
       DDRE &= ~(1 << DDE2);
       PORTE |= (1 << DDE2);

       // Set Initial Timer value
       TCNT3 = 0;

       ////First capture on rising edge
       TCCR3A = 0;
       TCCR3B = (0 << ICNC3) | (1 << ICES3);
       TCCR3C = 0;

       // Interrupt setup
       // ICIE3: Input capture
       // TOIE3: Timer1 overflow
       TIFR3 = (1 << ICF3) | (1 << TOV3);    // clear pending
       TIMSK3 = (1 << ICIE3) | (1 << TOIE3); // and enable
}

void StartTimer3(void) {
       // Start timer without pre-scaler
       TCCR3B |= (1 << CS30);
}

volatile uint32_t tickv, ticks;

// capture ISR
ISR(TIMER3_CAPT_vect) {
       tickv = ICR3; // save duration of last revolution
       revTickAvg = (uint32_t)tickv + ((uint32_t)T1Ovs2 * 0x10000L);
       revCtr++;  // add to revolution count
       TCNT3 = 0; // restart timer for next revolution
       T1Ovs2 = 0;
}

// Overflow ISR
ISR(TIMER3_OVF_vect) {
       // increment overflow counter
       T1Ovs2++;
}

void init_IO(void){
       //Setup IO
       SHIFT_REGISTER |= (DATA | LATCH | CLOCK); //Set control pins as outputs
       SHIFT_PORT &= ~(DATA | LATCH | CLOCK); //Set control pins low
}
void init_SPI(void){
       //Setup SPI
       SPCR0 = (1<<SPE) | (1<<MSTR); //Start SPI as Master
}
void spi_send(unsigned char byte){
       SPDR0 = byte; //Shift in some data
       while(!(SPSR0 & (1<<SPIF))); //Wait for SPI process to finish
}

/* Segment byte maps for numbers 0 to 9 */
const uint8_t SEGMENT_MAP[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99,
0x92, 0x82, 0xF8, 0X80, 0X90};
/* Byte maps to select digit 1 to 4 */
```

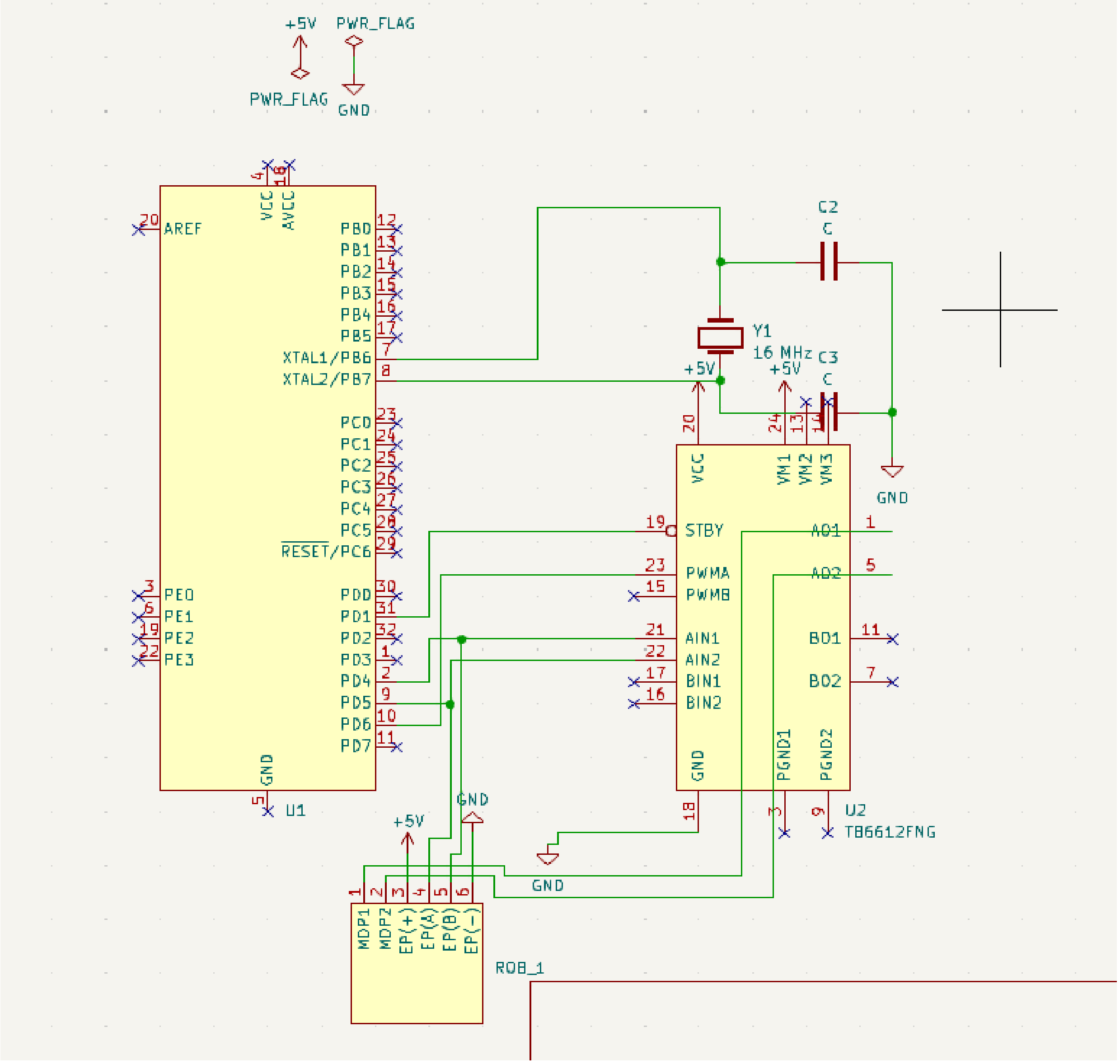
```
const uint8_t SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8};

int main(void)
{
       char outs[72];
       sei();
       InitTimer3();
       StartTimer3();

       /* set PD2 as input */
       DDRD &= ~(1 << DDD2);                /* Make INT0 pin as Input */
       PORTD |= (1 << DDD2);                // turn On the Pull-up

       /* Make PWM, AIN1, AIN2, STBY outputs */
       DDRD |= (1 << DDD6) | (1 << DDD5) | (1 << DDD1);
       DDRC |= (1<<DDC4);

       // We are manually setting the direction
       PORTD &= ~(1 << PD5); //set AIN2 low
       PORTD |= (1 << PD1); //set AIN1 and STBY high
       PORTC |= (1 << PC4);
       EIMSK |= (1 << INT0) | (1 << INT1); /* enable INT0 and INT1 */
       MCUCR |= (1 << ISC01) | (1 << ISC11) |
       (1 << ISC10); /* INT0 - falling edge, INT1 - raising edge */

       // WE are not using the ADC for speed - just manually setting the value
       ADC_Init(); // Initialize ADC
       TCNT0 = 0;  // Set timer0 count zero
       TCCR0A |= (1 << WGM00) | (1 << WGM01) | (1 << COM0A1);
       TCCR0B |=
       (1 << CS00) | (1 << CS01); // Set Fast PWM with Fosc/64 Timer0 clock
       OCR0A = 30;

       init_IO();
       init_SPI();
       while(1)
       {
              OCR0A = (ADC_Read(0)/4);                 // Read ADC and map it into 0-255 to write in OCR0
register

              // Convert ticks to RPM
              float rpms = (float)PERIOD * (float)revTickAvg * 1000.0 * 4.0;
              int rpms7seg_tens = (int)rpms / 10;
              int rpms7seg_ones = (int)rpms % 10;

              for (int i = 0; i < 10; i++)
              {
                     //Pull LATCH low (start the SPI transfer!)
                     SHIFT_PORT &= ~LATCH;
                     //Send the tens digit to sevenseg
                     spi_send((unsigned char)SEGMENT_MAP[rpms7seg_tens]);
                     spi_send((unsigned char)0xF4);
                     SHIFT_PORT |= LATCH;
                     SHIFT_PORT &= ~LATCH;
                     _delay_ms(10);

                     //Send the ones digit to sevenseg
                     //SHIFT_PORT &= ~LATCH;
```

```
                spi_send((unsigned char)SEGMENT_MAP[rpms7seg_ones]);
                spi_send((unsigned char)0xF8);
                SHIFT_PORT |= LATCH;
                SHIFT_PORT &= ~LATCH;
                _delay_ms(10);
            }
        }
}
```

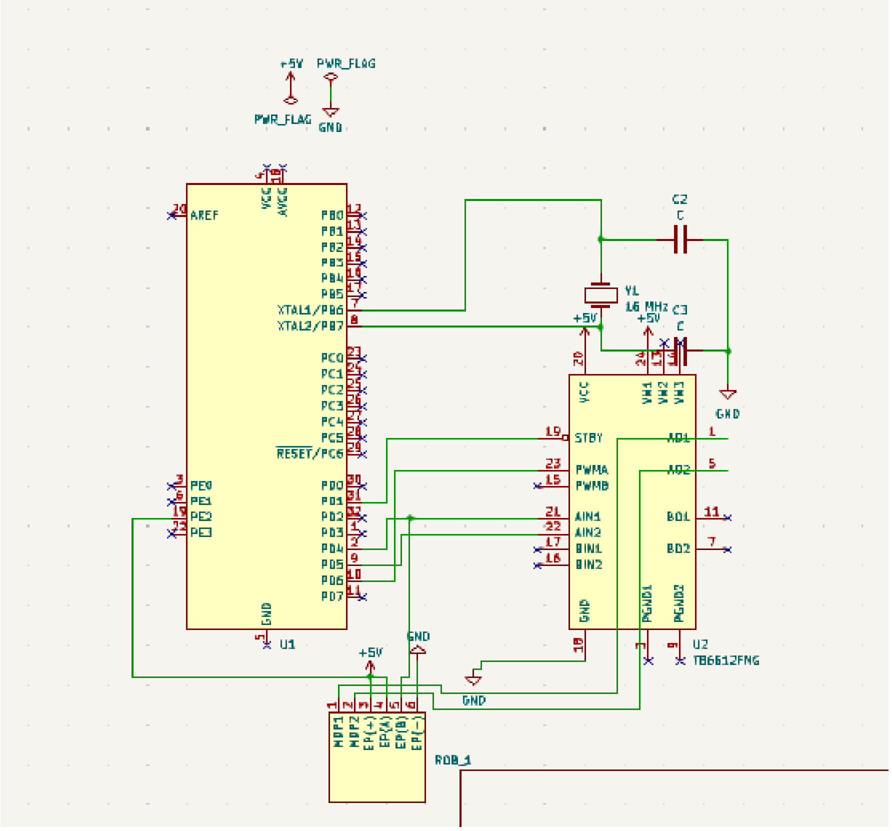
## 5. SCHEMATICS

Use fritzing.org

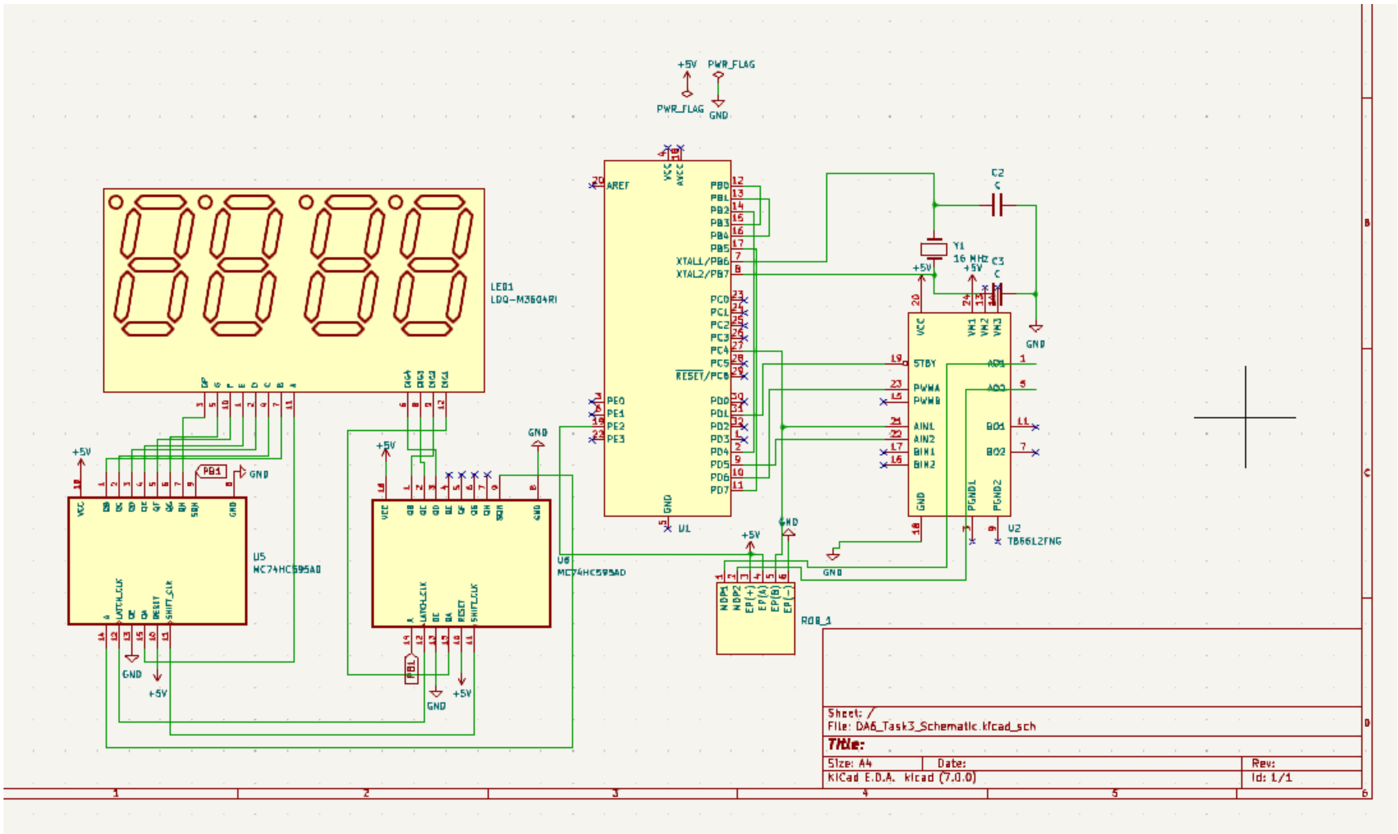
Task 1:



Task 2:


+5V
PWR_FLAG
PWR_FLAG
GND
VCC
AVCC
AREF
PB0
PB1
PB2
PB3
PB4
PB5
XTAL1/PB6
XTAL2/PB7
PC0
PC1
PC2
PC3
PC4
PC5
RESET/PC6
PE0
PE1
PE2
PE3
PD0
PD1
PD2
PD3
PD4
PD5
PD6
PD7
GND
U1
C2
C
Y1
16 MHz
C3
C
+5V
+5V
VCC
VM1
VM2
VM3
GND
STBY
PWMA
PWMB
AIN1
AIN2
BIN1
BIN2
AO1
AO2
BO1
BO2
GND
PGND1
PGND2
U2
TB6612FNG
+5V
GND
GND
MDP1
MDP2
EP(+)
EP(A)
EP(B)
EP(-)
ROB_1

Task 3:



Sheet: /
File: DA6_Task3_Schematic.kicad_sch

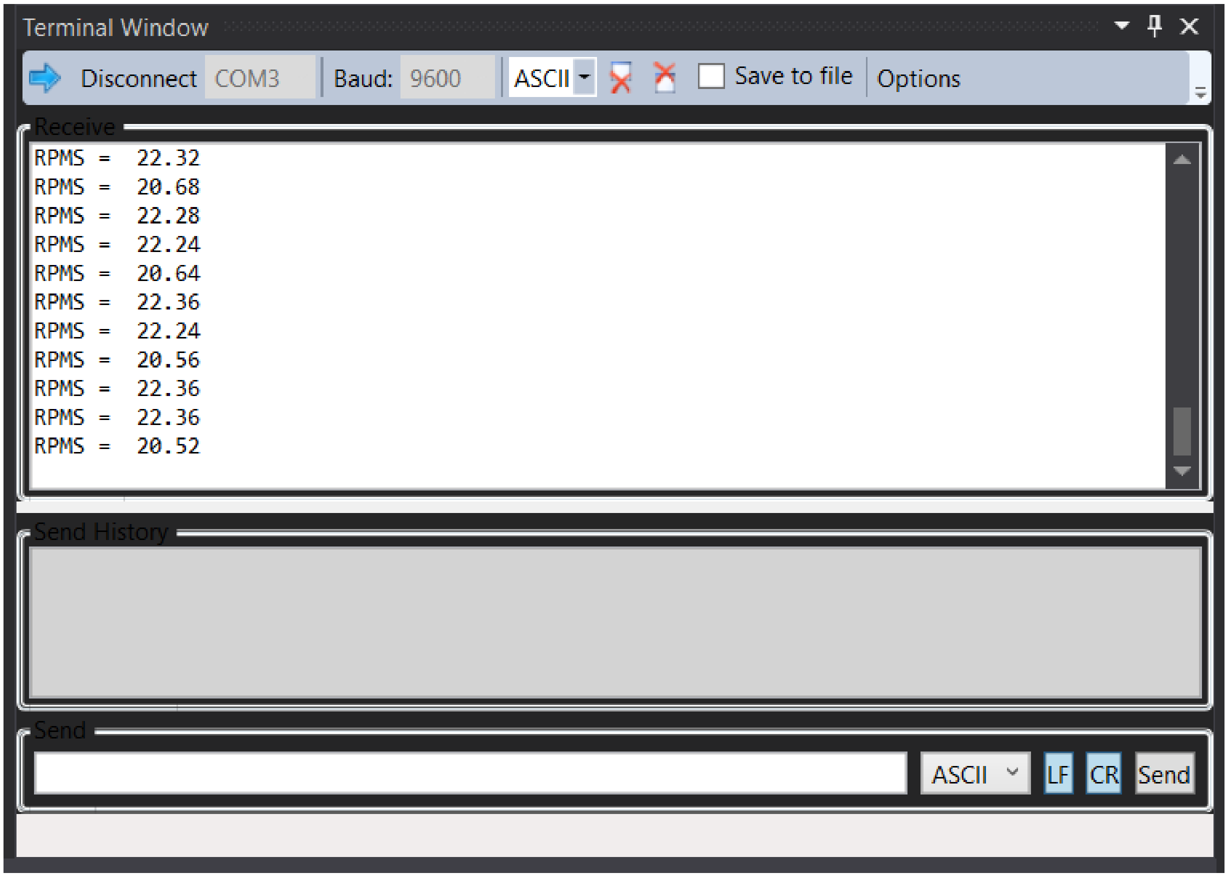
Title:

Size: A4 Date: Rev:
KiCad E.D.A. kicad (7.0.0) Id: 1/1

## 6. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

**Task 1:** No Atmel Studio Output

**Task 2:**



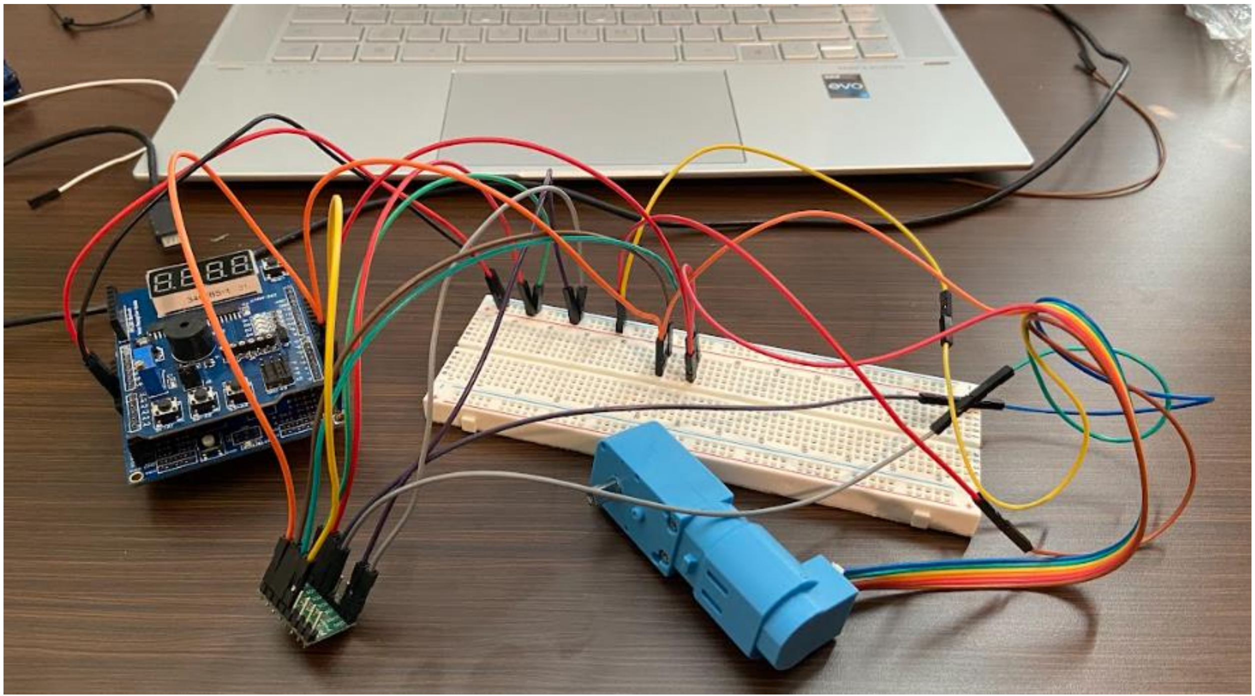
COM3 Baud: 9600 ASCII □ Save to file

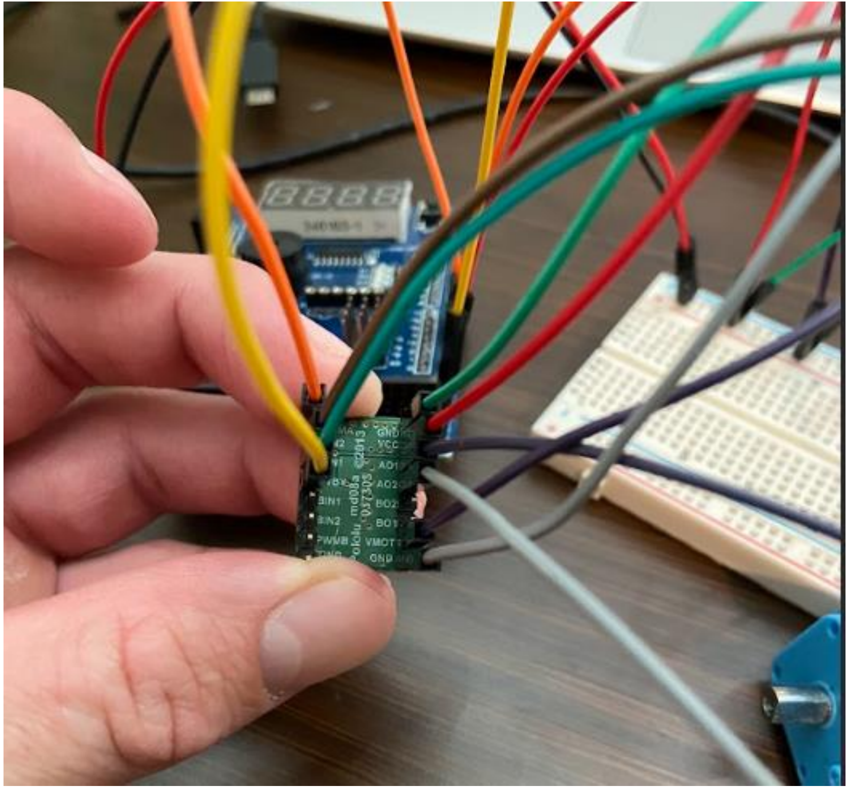
```
RPMS =  22.32
RPMS =  20.68
RPMS =  22.28
RPMS =  22.24
RPMS =  20.64
RPMS =  22.36
RPMS =  22.24
RPMS =  20.56
RPMS =  22.36
RPMS =  22.36
RPMS =  20.52
```

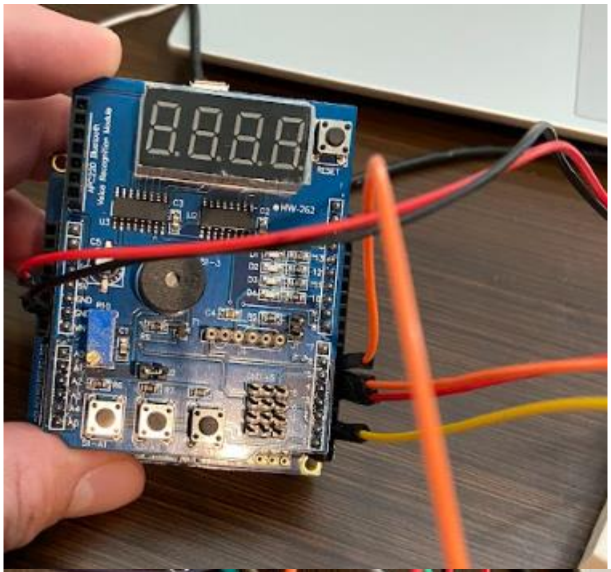
**Task 3:** No Atmel Studio output

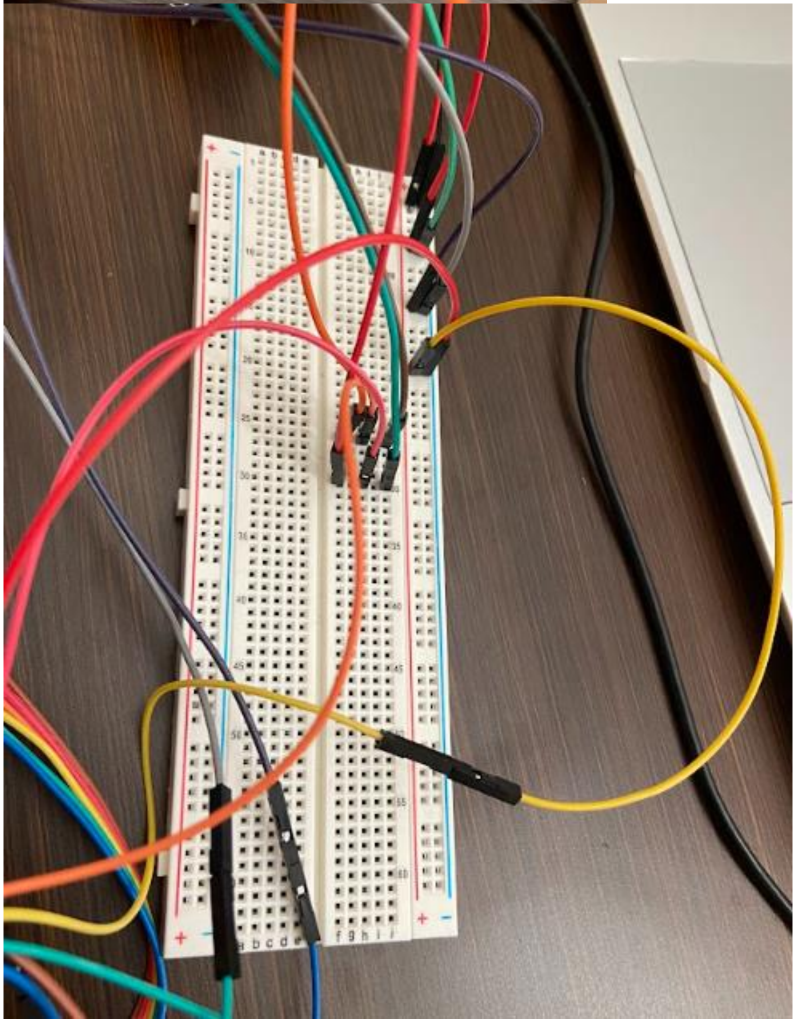
## 7. SCREENSHOT OF EACH DEMO (BOARD SETUP)

Task 1:

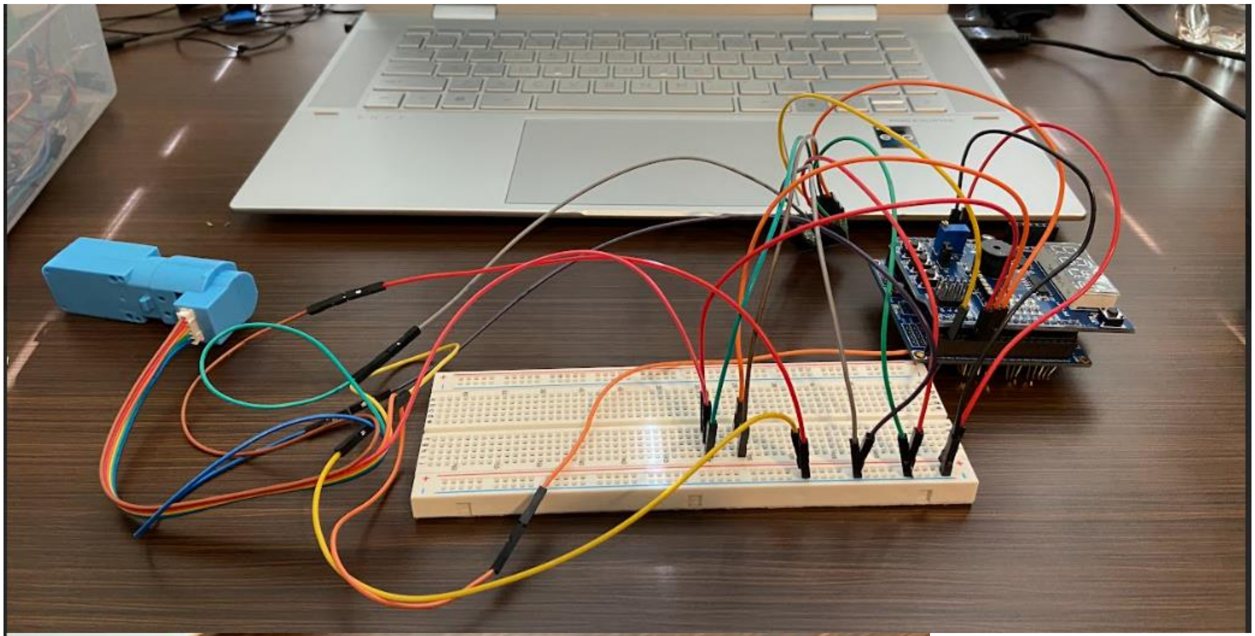


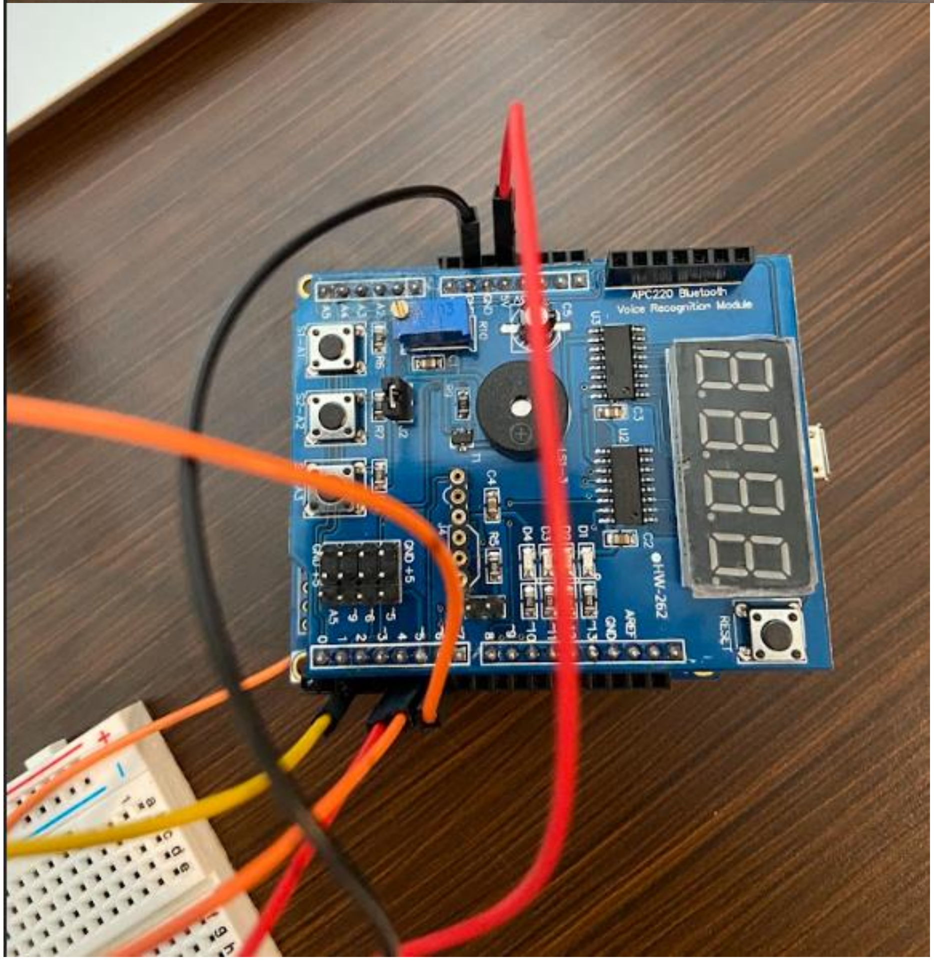

GND
VCC
AO1
AO2
BO2
BO1
VMOT
GND
BIN1
BIN2
PWMB
GND
Pololu md08a ©2013

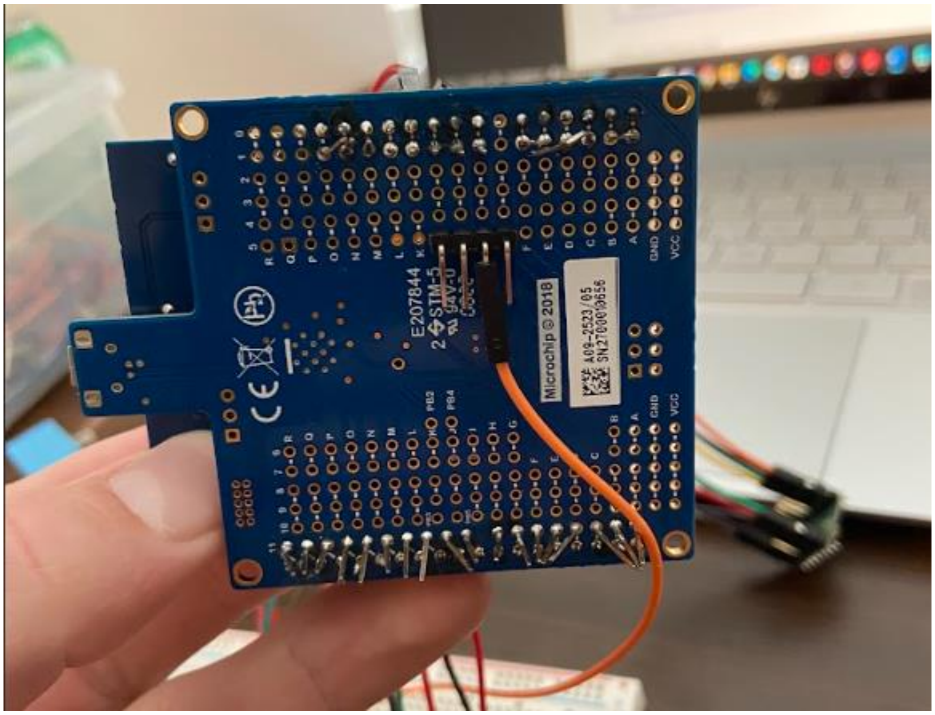


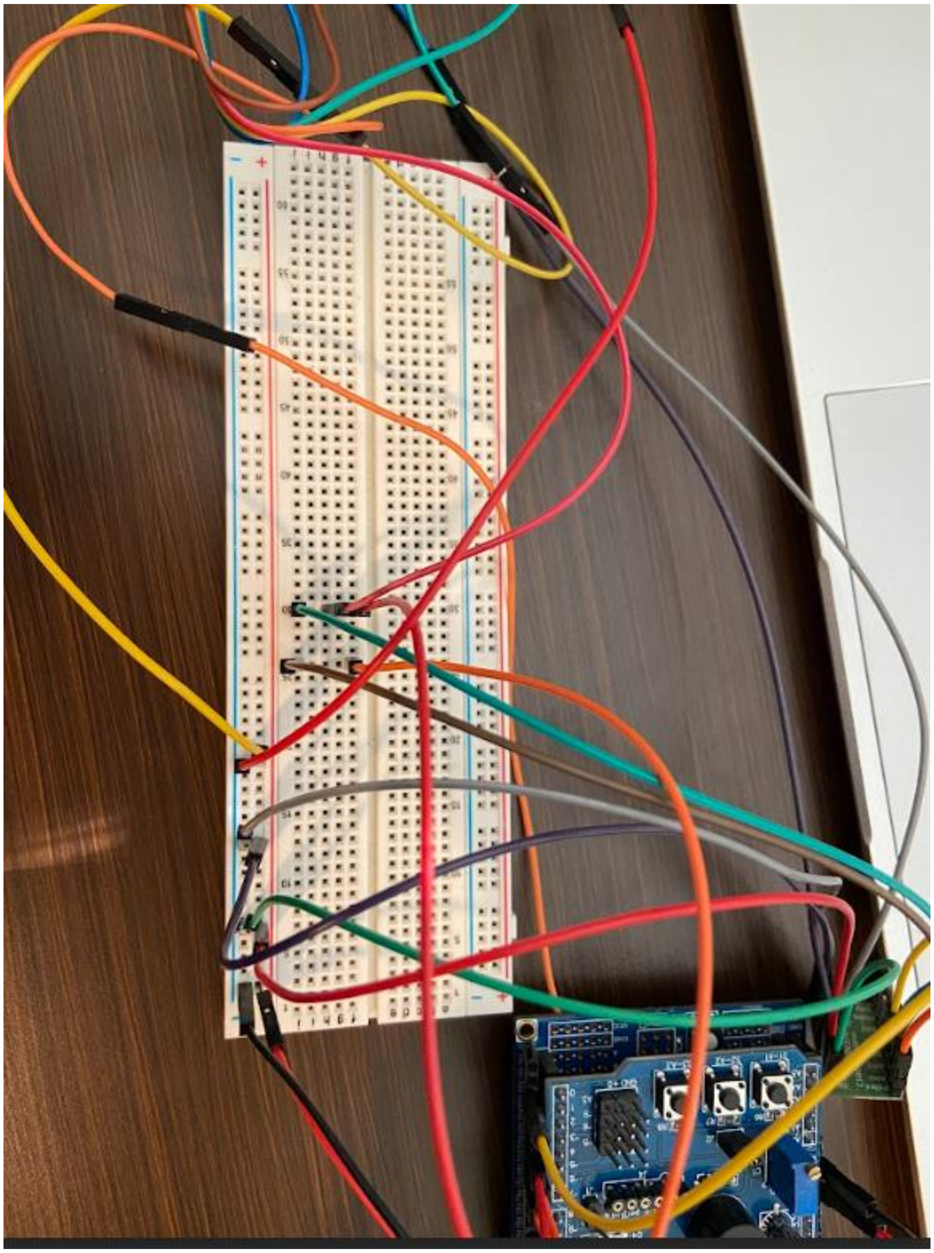


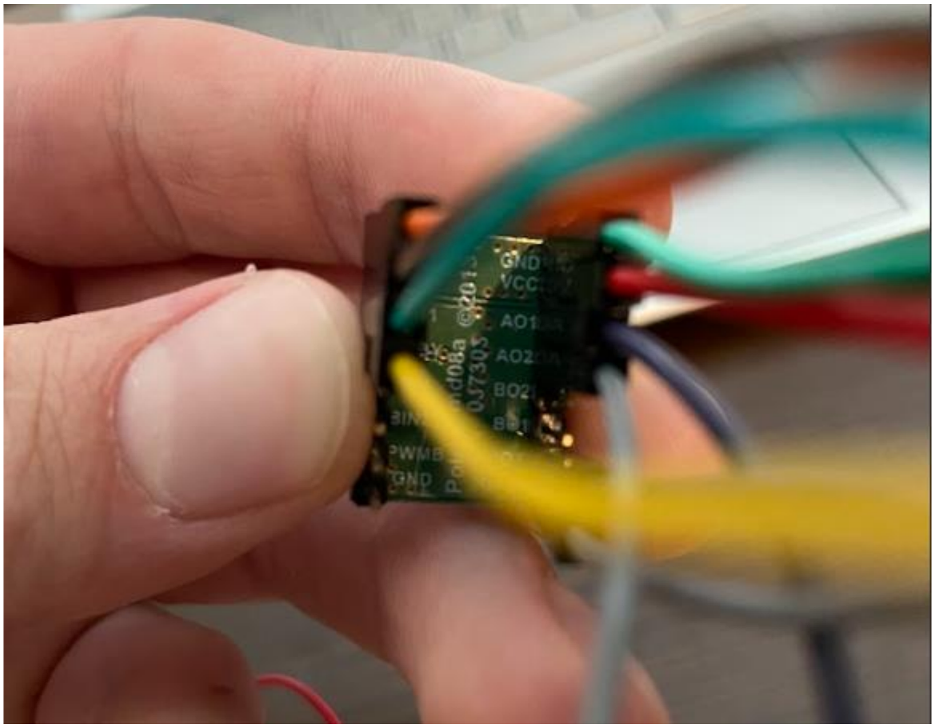
Task 2:



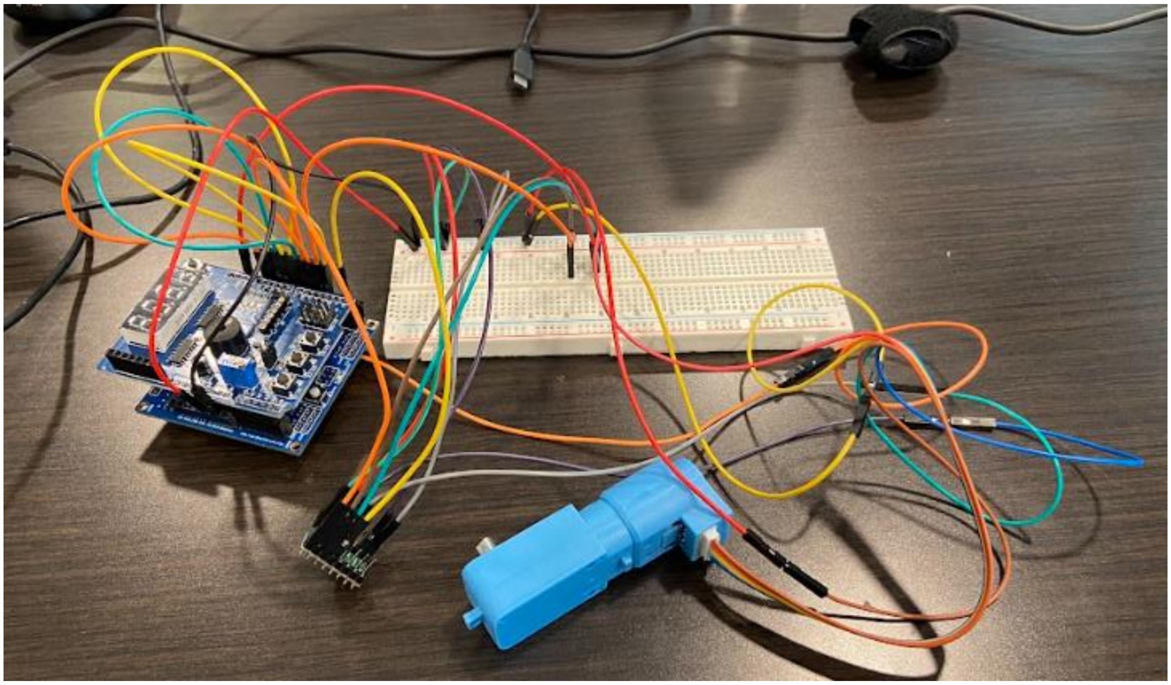

APC220 Bluetooth
Voice Recognition Module
HW-262
RESET
GND +5

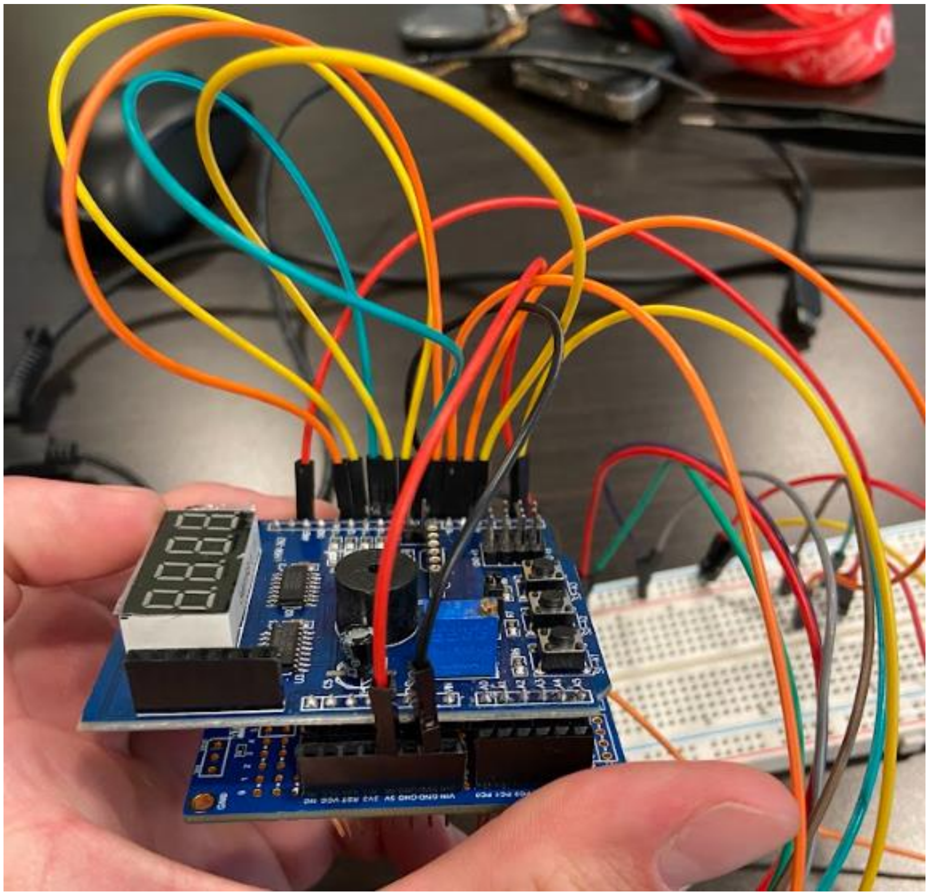

E207844
Microchip © 2018
A09-2523/05
SN2700010656
GND
VCC
PB2
PB4

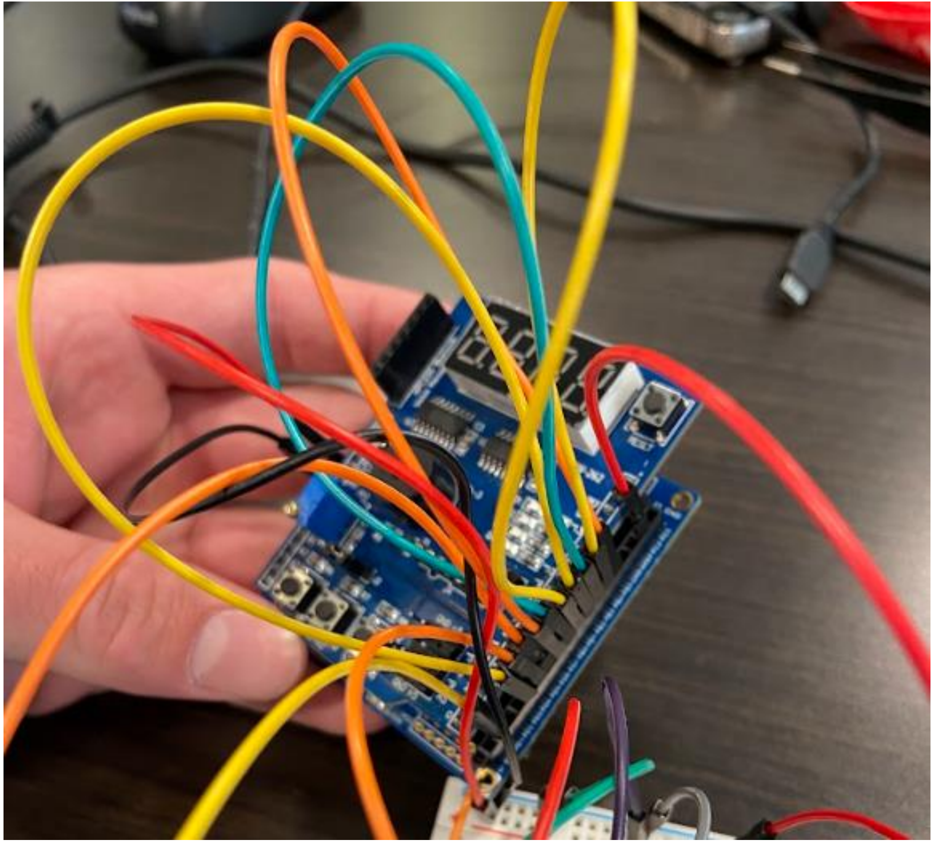


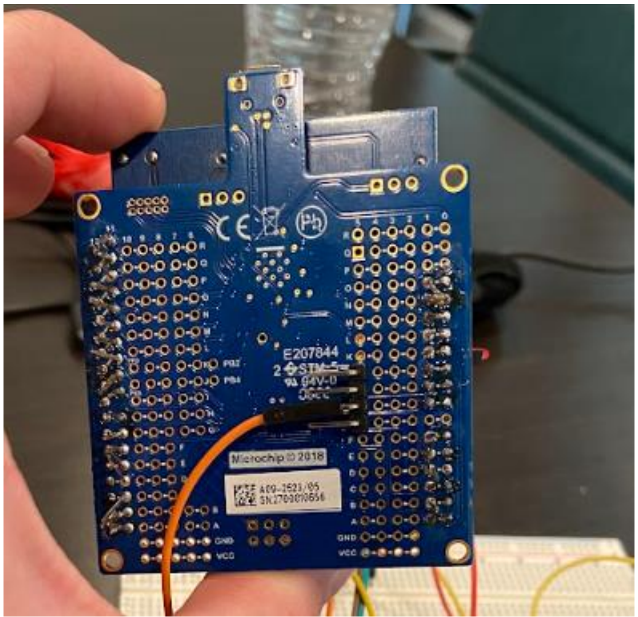


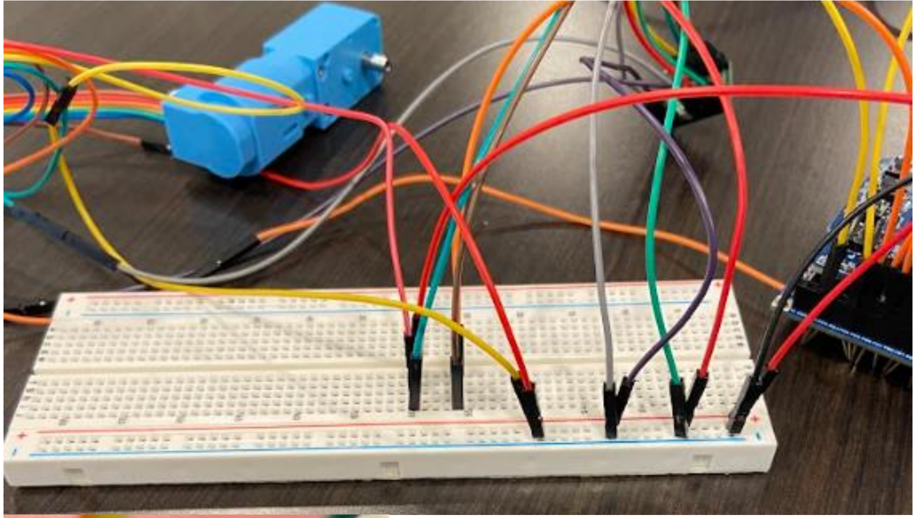
Task 3:

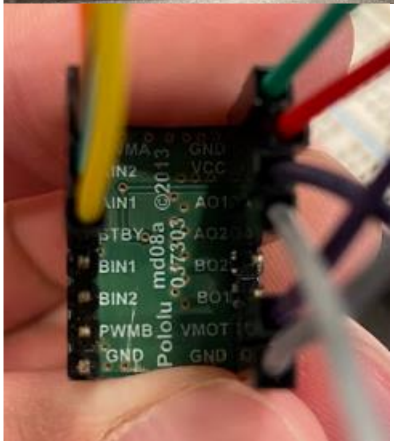












## 8. VIDEO LINKS OF EACH DEMO

**Playlist:** https://www.youtube.com/playlist?list=PLlHKEZIJ23uD-ZNniV7pnYYlqsTPrwvjY
**Task 1:** https://youtu.be/V-T3fuNzXjk
**Task 2:** https://youtu.be/8gL_M5XQzzM
**Task 3:** https://youtu.be/6q-x8n12X6o

## 9. GITHUB LINK OF THIS DA

https://github.com/dlenzin15/submissions/tree/main/DA6

**Student Academic Misconduct Policy**
http://studentconduct.unlv.edu/misconduct/policy.html

*"This assignment submission is my own, original work"*.
David Lenzin