CPE301 – SPRING 2024

Design Assignment 6

Student Name: Abraham Garcia
Student #: 5005262049
Student Email: garci11@unlv.nevada.edu
Primary Github address: https://github.com/SON-Abe/submission_da.git
Directory: submission_da/Design_Assignments/DA6
Video Playlist: 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**

- Microchip Studio Debugger
- ATmega328PB Microcontroller
- Terminal Window
- Motor Driver
- USB Power Supply
- 7 Segment Display
- Potentiometer
- Multi functional Shield
- TB6612FNG
- Microchip Studio Simulator
- Female-to-Male Wires
- Male to Male Wires
- SerialPlot

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

TASK 1&2:

```
/*
 * TB6612_MotorSpeed.c
 *
 * Created: 3/25/2020 1:27:23 PM
 * Author : VenkatesanMuthukumar
 */

#define F_CPU 16000000UL /* Define CPU Frequency e.g. here its 8MHz */
#include "uart.h"
#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;

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 */
}

// INT0 interrupt
ISR(INT0_vect) {
  // Use for Motor direction one trigger for forward, another for reverse
}

// INT1 interrupt
ISR(INT1_vect) {
  // Use for Motor direction one trigger for stop and go
}

volatile uint32_t revTick; // Ticks per revolution
volatile uint32_t revCtr;  // Total elapsed revolutions
volatile uint16_t T1Ovs2;  // Overflows for small rotations

// Initialize timer
void InitTimer1(void) {
  // Set PB0 as input
  DDRB &= ~(1 << DDB0);
  PORTB |= (1 << DDB0);

  // Set Initial Timer value
  TCNT1 = 0;
  ////First capture on rising edge
  TCCR1A = 0;
  TCCR1B = (0 << ICNC1) | (1 << ICES1);
```

```
  TCCR1C = 0;
  // Interrupt setup
  // ICIE1: Input capture
  // TOIE1: Timer1 overflow
  TIFR1 = (1 << ICF1) | (1 << TOV1);    // clear pending
  TIMSK1 = (1 << ICIE1) | (1 << TOIE1); // and enable
}

void StartTimer1(void) {
  // Start timer without pre-scaler
  TCCR1B |= (1 << CS10);
  // Enable global interrupts
  sei();
}

volatile uint32_t tickv, ticks;
// capture ISR
ISR(TIMER1_CAPT_vect) {
    tickv = ICR1; // save duration of last revolution
    revTickAvg = (uint32_t)(tickv) + ((uint32_t)T1Ovs2 * 0x10000L);
  revCtr++;  // add to revolution count
  TCNT1 = 0; // restart timer for next revolution
  T1Ovs2 = 0;
}
// Overflow ISR
ISR(TIMER1_OVF_vect) {
  // increment overflow counter
  T1Ovs2++;
}

int main(void) {
  char outs[72];
  USART_Init(9600);
  USART_SendString("Connected!\r\n"); // we're alive!
  InitTimer1();
  StartTimer1();
  USART_SendString("TIMER1 ICP Running \r\n");
  /* 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); /* Make OC0 pin as
Output */
  // We are manually setting the direction
  PORTD |= (1 << DDD5);                // CW Direction Set
  PORTD &= ~(1 << DDD4);               // CW Direction Set
  EIMSK |= (1 << INT0) | (1 << INT1); /* enable INT0 and INT1 */
  MCUCR |= (1 << ISC01) | (1 << ISC11) |
           (1 << ISC10); /* INT0 - falling edge, INT1 - raising edge */
  sei();                  /* Enable Global Interrupt */
  // 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 << CS02); /* Set Fast PWM with Fosc/64 Timer0 clock
*/
  OCR0A = 30;
  /* ready start value */
  while (1) {
    // Convert ticks to RPM
       // send Speed value to LCD or USART
      USART_SendString("Tick;Period;Frequency  ");
      snprintf(outs, sizeof(outs), "%f ", (float)revTickAvg); // print it
      USART_SendString(outs);
      USART_SendString(" \r\n");
  }
}
```

TASK 4:

```
/*
 * Latch_HWSPI_Example.c
 *
 * Created: 3/19/2022 12:51:10 PM
 * Author : venkim
 */

#ifndef F_CPU
#define F_CPU 16000000UL
```

```
#endif


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


#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)

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)
{
    init_IO();
    init_SPI();
```

```
    //unsigned int binary_counter = 0;

    while(1)
    {
        //Pull LATCH low (Important: this is necessary to start the SPI
transfer!)
        SHIFT_PORT &= ~LATCH;

        //spi_send((unsigned char)(0xA4));
        //spi_send((unsigned char)0xF8);
    /* Show 0-9 pattern for 1 second @ different digits */
    for (uint8_t i = 0; i < 10; i++) {
        spi_send((unsigned char)SEGMENT_MAP[i]);
        spi_send((unsigned char)0xF4);
        //Toggle latch to copy data to the storage register
        SHIFT_PORT |= LATCH;
        SHIFT_PORT &= ~LATCH;
        //wait for a little bit before repeating everything
        _delay_ms(1000);
    }

        //Toggle latch to copy data to the storage register
    //  SHIFT_PORT |= LATCH;
    //  SHIFT_PORT &= ~LATCH;
        //wait for a little bit before repeating everything
    //  _delay_ms(200);
    }
}
```

3. **DEVELOPED/MODIFIED CODE OF TASK 3/A**

```
#define F_CPU 16000000UL

#define BAUD 9600

#define BAUD_PRESCALE (((F_CPU / (BAUD * 16UL))) - 1)  //BAUD_PRESCALE
FORMULA

#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 <stdio.h>

#include <util/delay.h>

#include <avr/pgmspace.h>

#include <avr/interrupt.h>



// capture Flag

char rT[20];

char outs[20];

volatile unsigned int ctov;

volatile uint8_t revCtr = 0;

volatile uint16_t T1Ovs2;

volatile uint32_t tickv, ticks, revTick[200];

volatile float revTime, revTickAvg, CountAvg = 0;

const uint8_t SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8}; /* Byte maps to
select digit 1 to 4 */

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




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

}



void USART_Init(void)

{

    UBRR0H = (uint8_t)(BAUD_PRESCALE >> 8);    //LOAD UBRR0 HIGH 8 BITS

    UBRR0L = (uint8_t)(BAUD_PRESCALE);         //LOAD UBBR0 LOW BITS

    UCSR0B = (1 << RXEN0) | (1 << TXEN0);       //USART TRANSMITTER AND
RECEIVER ENABLED

    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);     //8 DATA BITS, 1 STOP BIT,
NO PARITY

}
```

```
char USART_RxChar()                                /* Data receiving function
*/
{
    while (!(UCSR0A & (1 << RXC0)));               /* Wait until new data
receive */
    return (UDR0);                                 /* Get and return received
data */
}


void USART_TxChar(char data)                       /* Data transmitting
function */
{
    UDR0 = data;                                   /* Write data to be
transmitting in UDR */
    while (!(UCSR0A & (1 << UDRE0)));              /* Wait until data transmit
and buffer get empty */
}


void USART_SendString(char *str)                   /* Send string of USART
data function */
{
    int i = 0;
    while (str[i] != 0)
    {
        USART_TxChar(str[i]);                      /* Send each char of string
till the NULL */
        i++;
```

```
    }
}


ISR(TIMER3_CAPT_vect)                          // capture ISR
{
    tickv = ICR3; // save duration of last revolution
    uint32_t revTickSig = tickv + (T1Ovs2 * 0x10000L);
    revTick[revCtr] = revTickSig;
    revCtr++;  // add to revolution count


    if (revCtr == 200)
    {
        for (int i = 0; i <= 200; ++i)
        revTickAvg += revTick[i];
        revTickAvg = (float)revTickAvg / 200;
        revTime = (float)(60 * 1000000) / (144 * revTickAvg * 0.0625);
        snprintf(outs, sizeof(outs), "RPM: %.2f \r\n", revTime);
        snprintf(rT, sizeof(outs), "%05.2f", revTime);
        USART_SendString(outs);
        revCtr = 0;
    }
    TCNT3 = 0; // restart timer for next revolution
    T1Ovs2 = 0;
}
```

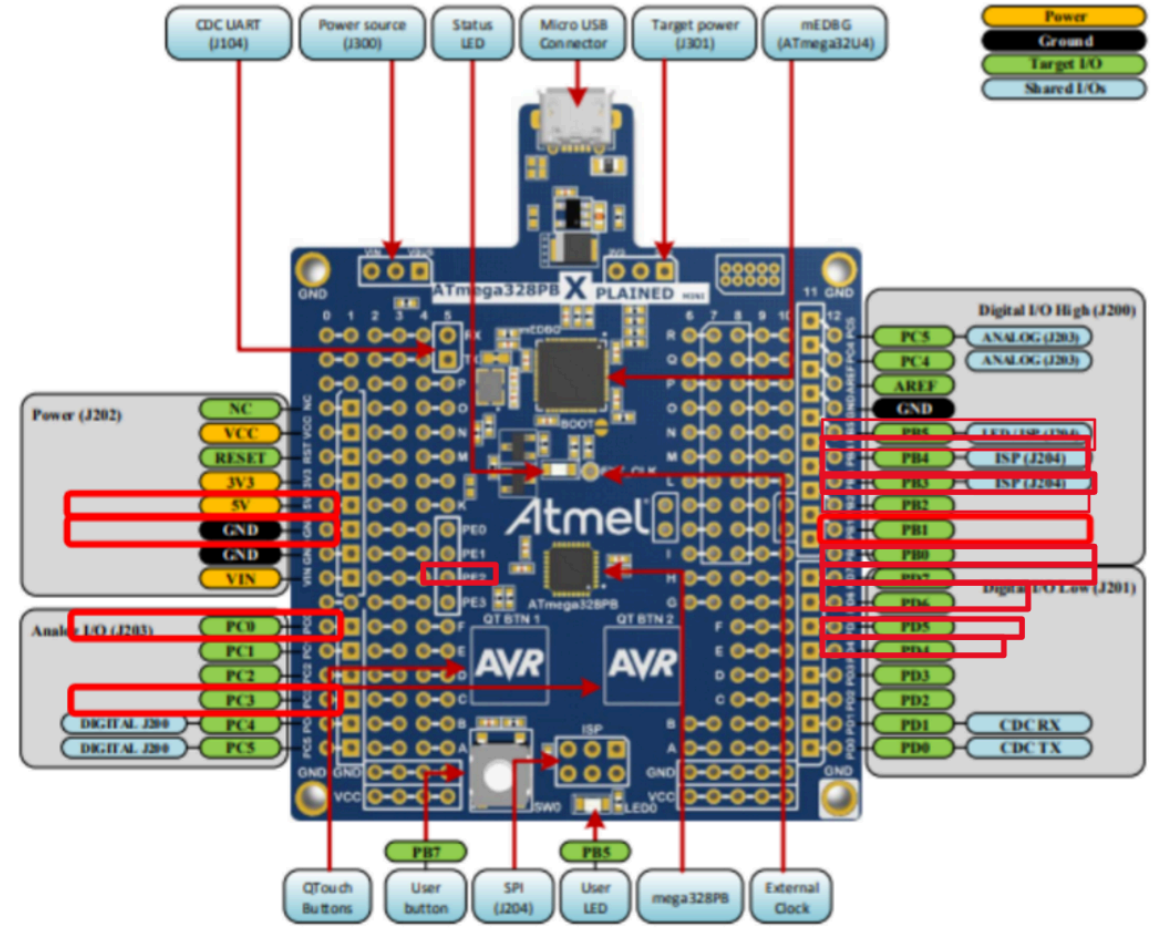
```
ISR(TIMER3_OVF_vect)    // Overflow ISR

{

    // increment overflow counter

    T1Ovs2++;

}



void StartTimer3(void)

{

    // Start timer without pre-scaler

    TCCR3B |= (1 << CS30);

    // Enable global interrupts

    sei();

}



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);
```

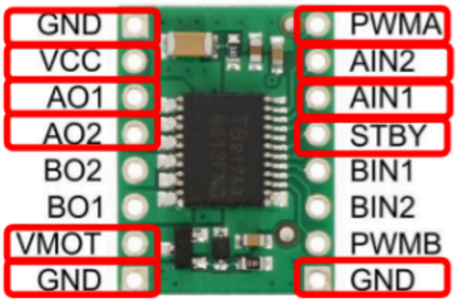
```
    TCCR3C = 0;

    // Interrupt setup

    // ICIE3: Input capture

    // TOIE3: Timer3 overflow

    TIFR3 = (1 << ICF3) | (1 << TOV3);    // clear pending

    TIMSK3 = (1 << ICIE3) | (1 << TOIE3); // and enable

}



void read_adc(void)

{

    ADMUX |= (1 << REFS0);                 //Set ADC reference voltage to
AVCC

    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS0); //Enable ADC, Auto
Trigger, Interrupt, Set ADC prescale



    ctov = 0;

    unsigned char i = 4;

    while (i--)

    {

        ADCSRA |= (1 << ADSC);

        while (ADCSRA & (1 << ADSC));

        ctov += ADC;

        _delay_ms(50);

    }

    ctov = ctov / 4;  // Average a few samples

    ctov = ctov * 50 / 1024; // Convert to volts
```

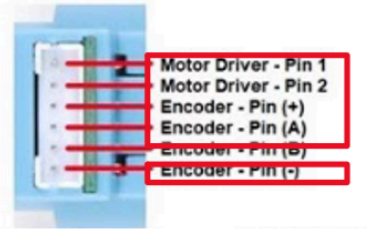
```
}


int main()
{
    USART_Init();
    InitTimer3();
    StartTimer3();
    init_IO();
    init_SPI();


    OCR0A = 200;
    TCCR0B |= (1 << CS02) | (1 << CS00); // Prescale 128
    TCCR0A = (1 << COM0A1) | (1 << WGM01) | (1 << WGM00); // TODO: Fast PWM, non-inverted
    DDRD |= (1 << DDD6); // PD2, PD1, and PD0 as outputs :  Rotate Clockwise


    while (1)
    {
        read_adc();
        OCR0A = ctov * 15;


        for (int i = 0; i < 4; i++)
        {
            // Extract digit from the string
            char digit_char = rT[i];
```

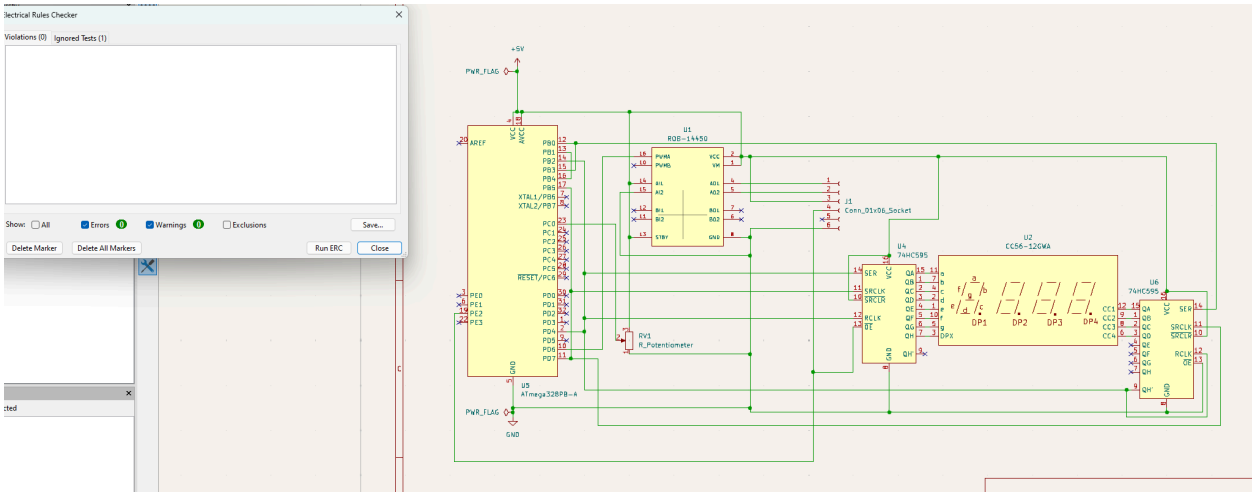
```
            if (digit_char == '.')
            {
                // If it's a decimal point, set the corresponding segment
                spi_send((unsigned char)0x7F);
            }
            else
            {
                // Convert the character digit to integer and get its
segment byte map
                int digit = digit_char - '0';
                spi_send((unsigned char)SEGMENT_MAP[digit]);
            }
            // Send byte map to select the digit
            spi_send((unsigned char)SEGMENT_SELECT[i]);
            // Toggle latch to copy data to the storage register
            SHIFT_PORT |= LATCH;
            SHIFT_PORT &= ~LATCH;
            // Delay to control the display rate
            _delay_ms(50);
        }
    }
}
```

4. **SCHEMATICS**


CDC UART (J104)
Power source (J300)
Status LED
Micro USB Connector
Target power (J301)
mEDBG (ATmega32U4)
Power
Ground
Target I/O
Shared I/Os
Digital I/O High (J200)
Power (J202)
NC
VCC
RESET
3V3
5V
GND
GND
VIN
Analog I/O (J203)
PC0
PC1
PC2
PC3
DIGITAL J200
PC4
DIGITAL J200
PC5
ANALOG (J203)
ANALOG (J203)
AREF
GND
ISP (J204)
ISP (J204)
Digital I/O Low (J201)
CDC RX
CDC TX
ATmega328PB
QT BTN 1
QT BTN 2
ISP
PB7
PB5
QTouch Buttons
User button
SPI (J204)
User LED
mega328PB
External Clock

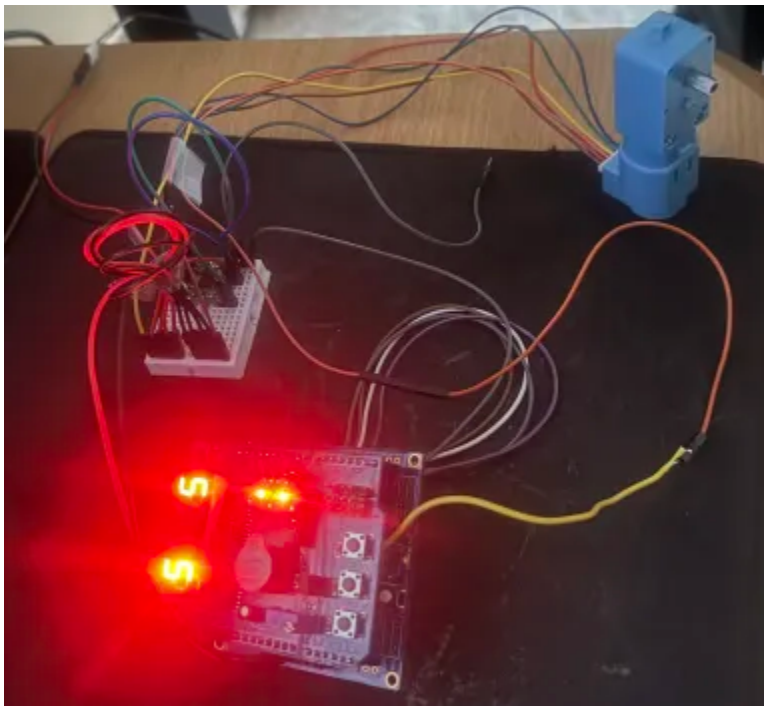

GND
VCC
AO1
AO2
BO2
BO1
VMOT
GND
PWMA
AIN2
AIN1
STBY
BIN1
BIN2
PWMB
GND


Motor Driver - Pin 1
Motor Driver - Pin 2
Encoder - Pin (+)
Encoder - Pin (A)
Encoder - Pin (B)
Encoder - Pin (-)

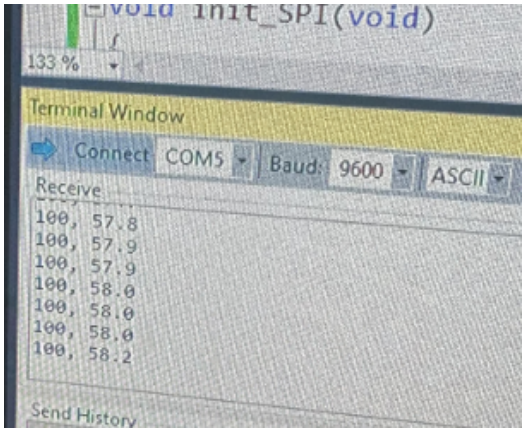


5. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

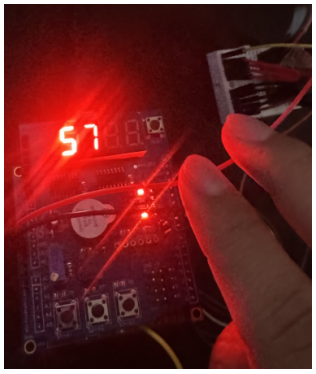
TASK 1:



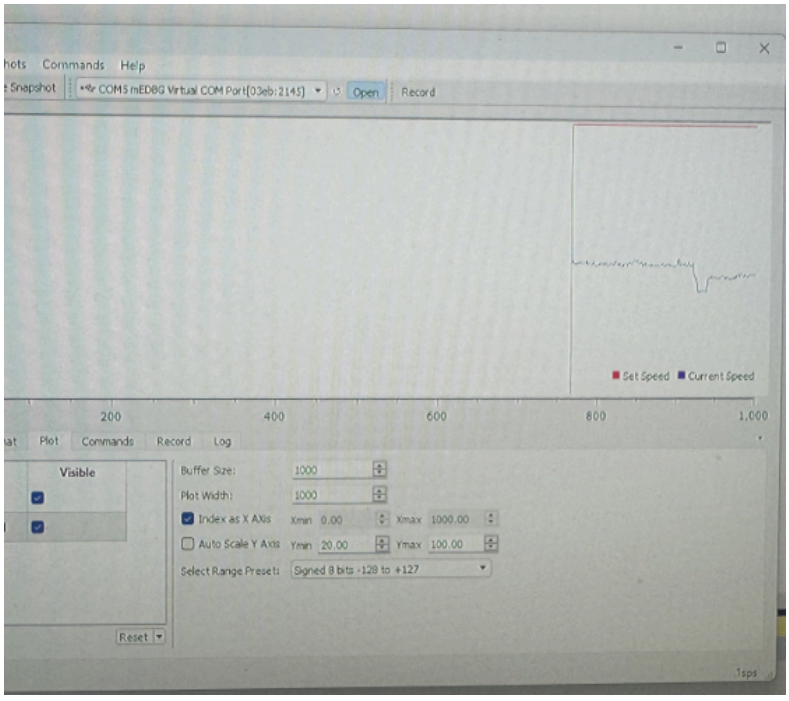
TASK 2:



TASK 3:

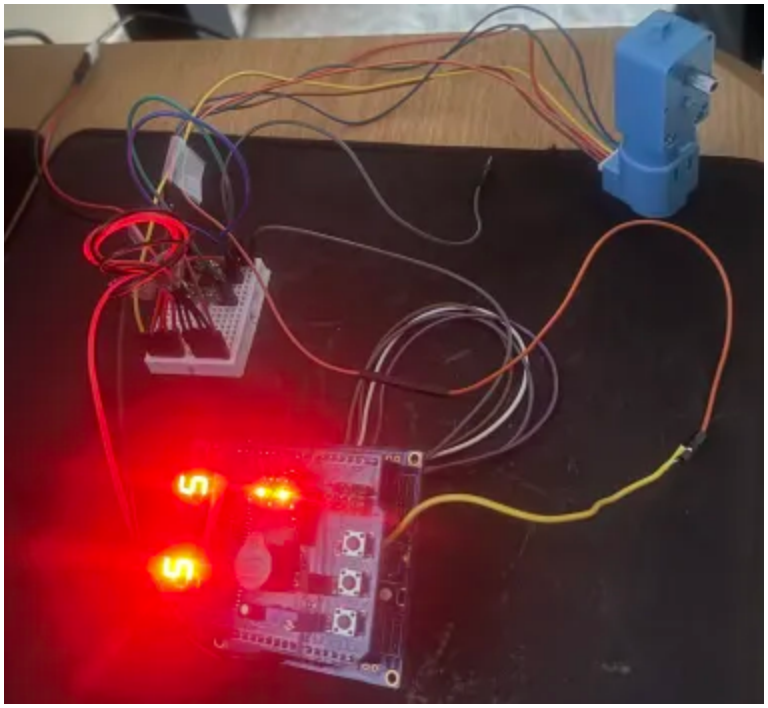


Task 4:



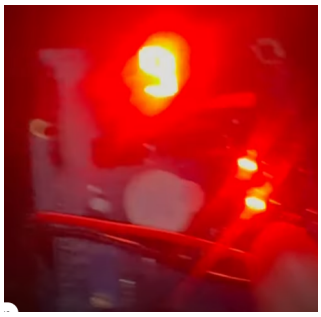
6. SCREENSHOT OF EACH DEMO (BOARD SETUP)

Task 1:

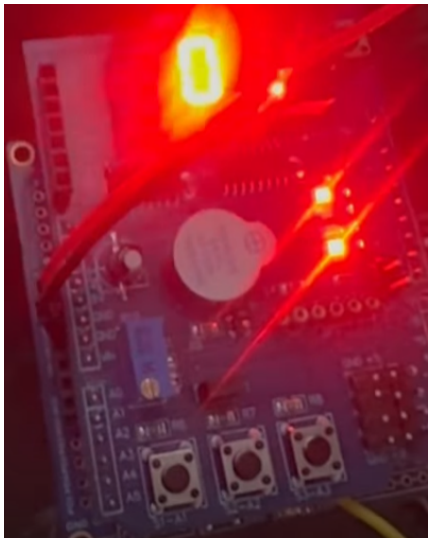


Task 2:

Task 3:



Task 4:



7. **VIDEO LINKS OF EACH DEMO**

DA6_1 DA6_2 DA6_3 DA6_4

8. **GITHUB LINK OF THIS DA**

DA6

Student Academic Misconduct Policy

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

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

Abraham Garcia