CPE301 – SPRING 2024

Design Assignment 6

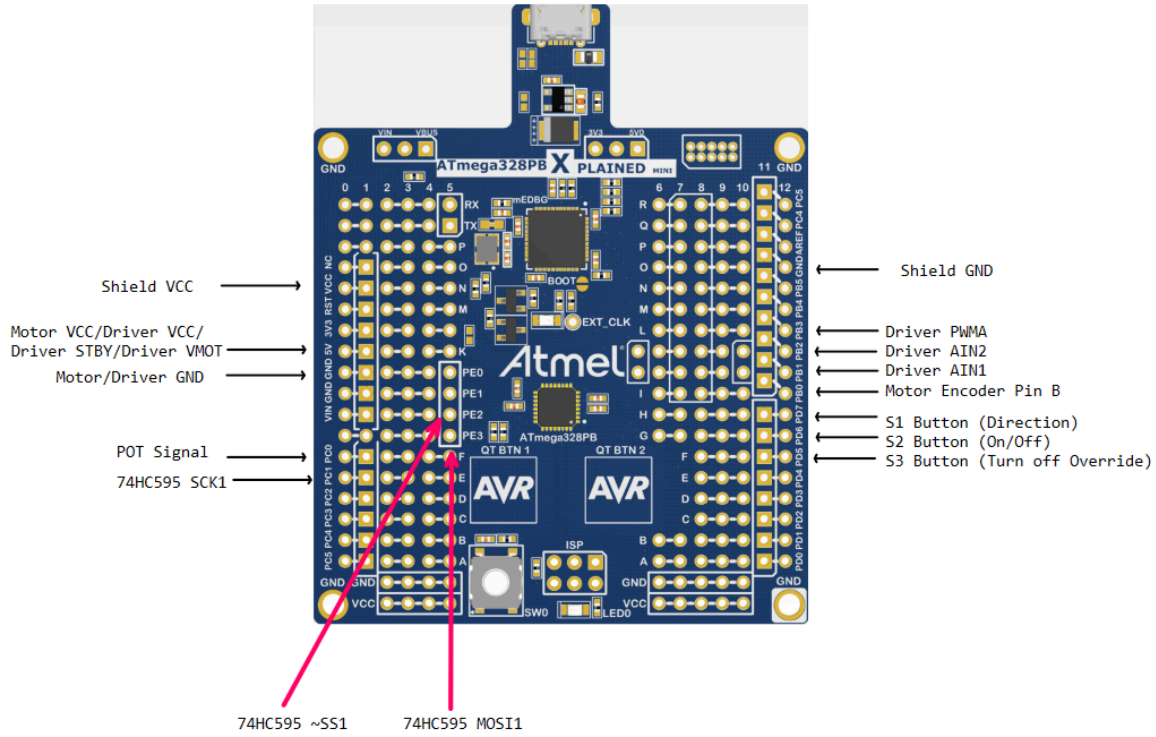
Student Name: Nathan Ramos
Student #: 5006437353
Student Email: ramosn8@unlv.nevada.edu
Primary Github address: https://github.com/n8ramos/
Directory: /atmega328pb
Video Playlist:
https://www.youtube.com/playlist?list=PL2RpCRW8TC6YOj-NnLPqqfRTV48RcUe48

1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

Components Used:

- DG01D-E DC Motor with Encoder
- Potentiometer (On-shield)
- TB6612FNG Dual Motor Driver
- 74HC595 shift register (On-shield)
- atmega328pb Xplained Mini
- microchip studio 7

Block diagram with pins used in the Atmega328PB (only)



2. DEVELOPED CODE OF TASK 1,2,3,& 4 COMBINED

```
#define F_CPU 16000000UL
// UART
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1
#define VREF 5
#define STEPS 1024
#define STEPSIZE VREF/STEPS
#define BUFFERSIZE 100
#define ASCII_NUM 48

// dc motor
#define ENABLE 3
#define MTR_1 1
#define MTR_2 2
#define BUTTON1 (PIND & (1<<PIND7)) // S1 controls direction of motor spin
#define BUTTON2 (PIND & (1<<PIND6)) // S2 controls stop and go of motor
#define BUTTON3 (PIND & (1<<PIND5)) // S3 turns off USART override
#define DELAY_BTN 200
#define SAMPRATE 100

// 7seg 74HC595
#define DATA (1<<PE3) //MOSI (SI)
#define LATCH (1<<PE2) //SS (RCK)
#define CLOCK (1<<PC1) //SCK (SCK)
#define DELAY_SEG 1

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

/* Segment byte maps for numbers 0 to 9 */
const uint8_t SEG_digit[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99,
0x92, 0x82, 0xF8, 0X80, 0X90};
/* Byte maps to select digit 1 to 4 */
const uint8_t SEG_select[] = {0xF1, 0xF2, 0xF4, 0xF8};

int dout, doutOLD;
int numDigits;
unsigned int STALL, OVERRIDE, DIRECTION, ONOFF;
float rpm;
unsigned int numBuffer[BUFFERSIZE];

// capture Flag
volatile uint32_t revTickAvg, revTickSig;
volatile uint32_t tickv, ticks;
volatile uint32_t revTick[SAMPRATE]; // Ticks per revolution
volatile uint32_t revCtr = 0;  // Total elapsed revolutions
```

```
volatile uint16_t T1Ovs2;  // Overflows for small rotations

// capture ISR
ISR(TIMER1_CAPT_vect) {
	cli();
	tickv = ICR1; // save duration of last revolution
	revTickSig = (uint32_t)(tickv) + ((uint32_t)T1Ovs2 * 0x10000L);
	revTick[revCtr] = revTickSig;
	revCtr++; // add to revolution count
	if (revCtr == SAMPRATE) {
		for (int i = 0; i < SAMPRATE; i++) {
			revTickAvg += (float) revTick[i];
		}
		revTickAvg /= SAMPRATE;
		rpm = (float) (60*1000000) / (144 * revTickAvg*0.0625);
		revCtr = 0;
	}
	TCNT1 = 0; // restart timer for next revolution
	T1Ovs2 = 0;
	sei();
}

// Overflow ISR
ISR(TIMER1_OVF_vect) {
	cli();
	// increment overflow counter
	T1Ovs2++;
	if (T1Ovs2 > 10) {
		STALL = 1;
	} else {
		STALL = 0;
	}
	sei();
}

ISR (ADC_vect)
{
	cli();
	doutOLD = dout;
	dout = ADC;
	// start ADC conversion=
	ADCSRA |=(1<<ADSC);
	sei();
}

ISR(USART0_RX_vect) {
	cli();
	static unsigned int i;
	char input = UDR0;

	switch (input) {
		case '0':
```

```
                        case '1':
                        case '2':
                        case '3':
                        case '4':
                        case '5':
                        case '6':
                        case '7':
                        case '8':
                        case '9':
                                numBuffer[i] = (int) input - ASCII_NUM;
                                i++;
                                break;
                        case '\n':
                        case '\r':
                                numDigits = i;
                                i = 0;
                                if (numDigits > 0) { OVERRIDE = 1; }
                                break;
/*                      default:*/

                }
                if (i == BUFFERSIZE) {
                        i = 0;
                }
                sei();
}

void init_IO(void){
                //Setup IO
                //Set control pins as outputs
                DDRE |= (DATA | LATCH);
                DDRC |= (CLOCK);
                 //Set control pins low
                PORTE &= ~(DATA | LATCH);
                PORTC &= ~CLOCK;
}

void init_SPI(void){
                //Setup SPI
                SPCR1 = (1<<SPE) | (1<<MSTR); //Start SPI as Master
}

void SPI_send(unsigned char byte){
                SPDR1 = byte; //Shift in some data
                while(!(SPSR1 & (1<<SPIF))); //Wait for SPI process to finish
}

void SEG_display(int thou, int hun, int ten, int one) {
                cli();
                PORTE &= ~LATCH;
                thou %= 10;
                hun %= 10;
```

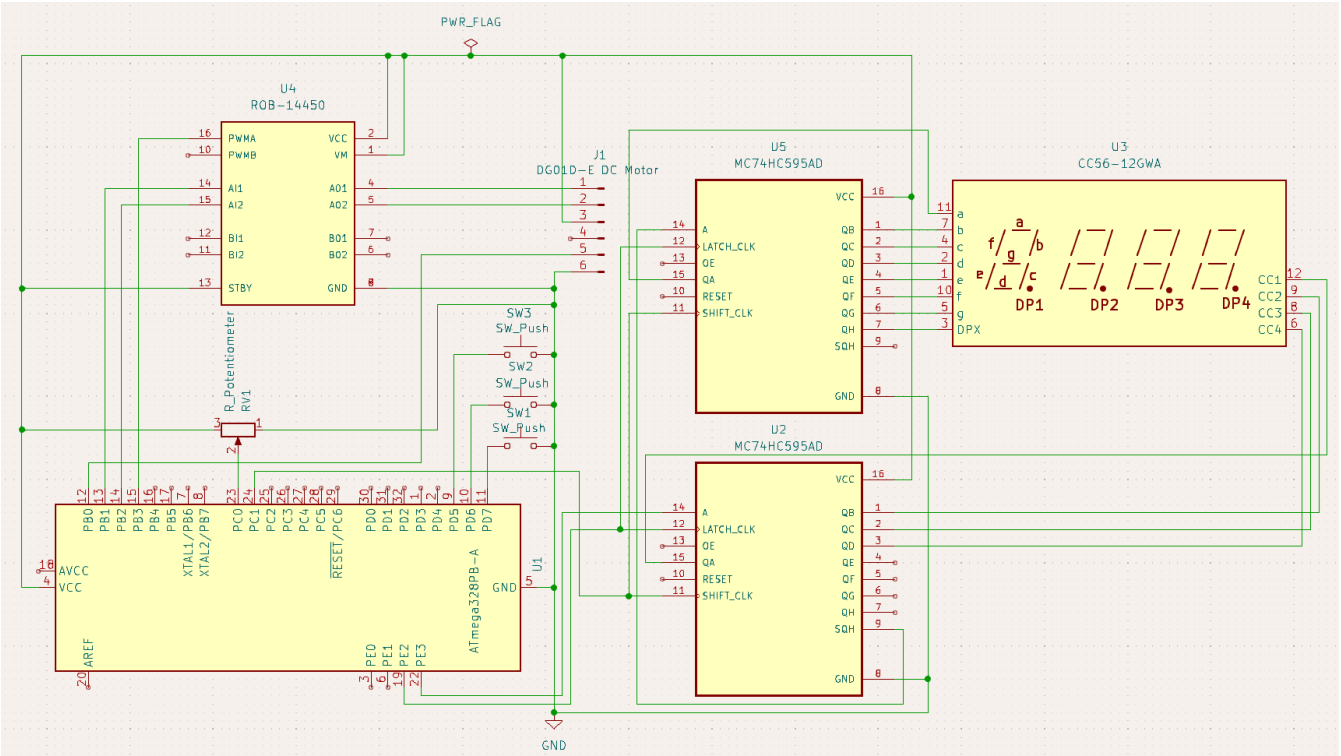
```
		ten %= 10;
		one %= 10;
//		// sending thousands place
//		SPI_send((unsigned char) SEG_digit[thou]);
//		SPI_send((unsigned char) SEG_select[0]);
//		PORTE |= LATCH;
//		PORTE &= ~LATCH;
//		_delay_ms(DELAY_SEG);
//		// sending hundreds place
//		SPI_send((unsigned char) SEG_digit[hun]);
//		SPI_send((unsigned char) SEG_select[1]);
//		PORTE |= LATCH;
//		PORTE &= ~LATCH;
//		_delay_ms(DELAY_SEG);
		// sending tens place
		SPI_send((unsigned char) SEG_digit[ten]);
		SPI_send((unsigned char) SEG_select[2]);
		PORTE |= LATCH;
		PORTE &= ~LATCH;
		_delay_ms(DELAY_SEG);
		// sending ones place
		SPI_send((unsigned char) SEG_digit[one]);
		SPI_send((unsigned char) SEG_select[3]);
		PORTE |= LATCH;
		PORTE &= ~LATCH;
		_delay_ms(DELAY_SEG);
		sei();
}

void ADC_init() {
		DDRC &= ~(1 << PINC0); // set PC0 as input
		//set channel to take input for ADC0,right justified, AVcc with external cap at AREF
		ADMUX = (1 << REFS0) | (0 << MUX0); // Also defaults ADC0 reading
		//set prescaler to 64, enable ADC interrupt,enable ADC,start conversion
		ADCSRA |= (1 << ADEN) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1) | (0 << ADPS0) | (1 << ADSC);
}

void USART_init(unsigned int ubrr)
{
		//Set baud rate
		UBRR0H = (unsigned char)(ubrr>>8);
		UBRR0L = (unsigned char) ubrr;
		// enable transmitter
		UCSR0B = (1<<TXEN0) | (1 << RXEN0) | (1 << RXCIE0);
		// Set frame format: async, no parity, 1 stop bit, , 8 data bits
		UCSR0C =
(0<<UMSEL01)|(0<<UMSEL00)|(0<<UPM01)|(0<<UPM00)|(0<<USBS0)|(1<<UCSZ01)|(1<<UCSZ00);
}

void USART_transmit(const char* data)
{
		while (*data) {
```

```
                //check if buffer is empty so that data can be written to transmit
                while (!(UCSR0A & (1 << UDRE0)));
                UDR0 = *data; //copy “data” to be sent to UDR0
                ++data;
        }
}

void USART_transmitChar(const char data)
{
        //check if buffer is empty so that data can be written to transmit
        while (!(UCSR0A & (1 << UDRE0)));
        UDR0 = data; //copy character to be sent to UDR0
}

// 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();
}

int main()
{
        sei();
        OVERRIDE = 0;
        DIRECTION = 0;
        ONOFF = 0;
        int DISPTOGGLE = 0;
        int pwm = 0;
        // For the USART output
        USART_init(MYUBRR);
        InitTimer1();
        StartTimer1();
        ADC_init();
```

```
init_IO();
init_SPI();
DDRD &= ~((1 << PIND7) | (1 << PIND6) | (1 << PIND5));
PORTD |= (1 << PIND7) | (1 << PIND6) | (1 << PIND5); //enable pull-up
DDRB |= 0b00001110; //PB3, PB1, and PB2 as outputs
PORTB &= ~(1<<ENABLE); //Enable = 0
PORTB &= ~(1<<MTR_1); //MTR_1 = 0
PORTB &= ~(1<<MTR_2); //MTR_2 = 0
//Fast PWM, non-inverted
TCCR2A = (1<<COM2A1)|(1<<WGM21)|(1<<WGM20);
TCCR2B = 0x02; //N = 8
while (1)
{
	if (BUTTON1 == 0) {
		_delay_ms(DELAY_BTN);
		DIRECTION ^= 1;
	}
	if (BUTTON2 == 0) {
		_delay_ms(DELAY_BTN);
		ONOFF ^= 1;
	}
	if (BUTTON3 == 0) {
		_delay_ms(DELAY_BTN);
		OVERRIDE = 0;
	}
	if (ONOFF) {
		if(DIRECTION)
		{
			// Clockwise Rotation
			_delay_ms(10);
			PORTB |= (1<<MTR_1); //MTR_1 = 1
			PORTB &= (~(1<<MTR_2)); //MTR_2 = 0
		}
		else
		{
			// Anti-Clockwise Rotation
			_delay_ms(10);
			PORTB &= (~(1<<MTR_1)); //MTR_1 = 0
			PORTB |= (1<<MTR_2); //MTR_2 = 1
		}
	} else {
		PORTB &= ~((1 << MTR_1) | (1 << MTR_2)); // set MTR1 and MTR2 to 0, STOP
		STALL = 1;
	}

	rpm = (STALL) ? 0 : rpm;

	int rpm1000 = (int) (rpm/1000) % 10;
	int rpm100 = (int) (rpm/100) % 10;
	int rpm10 = (int) (rpm/10) % 10;
	int rpm1 = (int) (rpm/1) % 10;
```

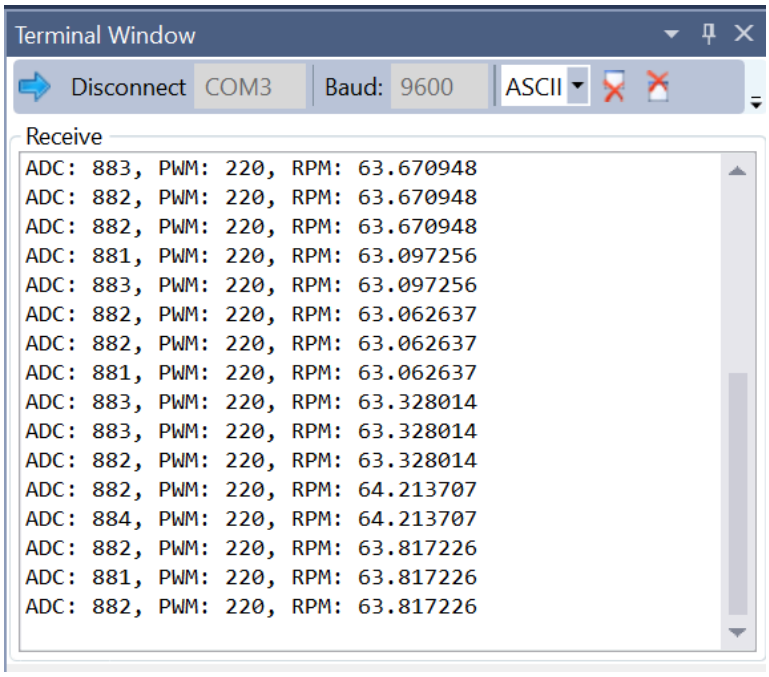
```
                for (int i = 0; i < 10; i++) {
                        SEG_display(rpm1000, rpm100, rpm10, rpm1);
                }

/*              char str[10];*/
//              USART_transmit("ADC: ");
//              itoa(dout, str,10);
//              USART_transmit(str);
//              USART_transmit(", PWM: ");
                if (OVERRIDE) {
                        int num = 0;
                        for (int i = 0; i < numDigits; i++) {
                                num = (num * 10) + numBuffer[i];
                        }
                        float rpmIN = (float) num * (8.0 / 3.0) + 55;
                        pwm = (numDigits) ? (int) rpmIN : pwm;
                        OCR2A = pwm % 256;
                } else {
                        OCR2A = dout / 4;
                }
//              itoa(OCR2A, str,10);
//              USART_transmit(str);
//              USART_transmit(", RPM: ");
//              snprintf(str, sizeof(str), "%2f ", rpm);
//              itoa((int) rpm, str, 10);
//              USART_transmit(str);
//              USART_transmit(",");
//              USART_transmit("\n");
                int rpmIN = (3 * (OCR2A - 55)) / 8;
                USART_transmitChar((unsigned char) rpmIN);
                USART_transmitChar((unsigned char) rpm);
        }
        return 0;
}
```

3. SCHEMATICS

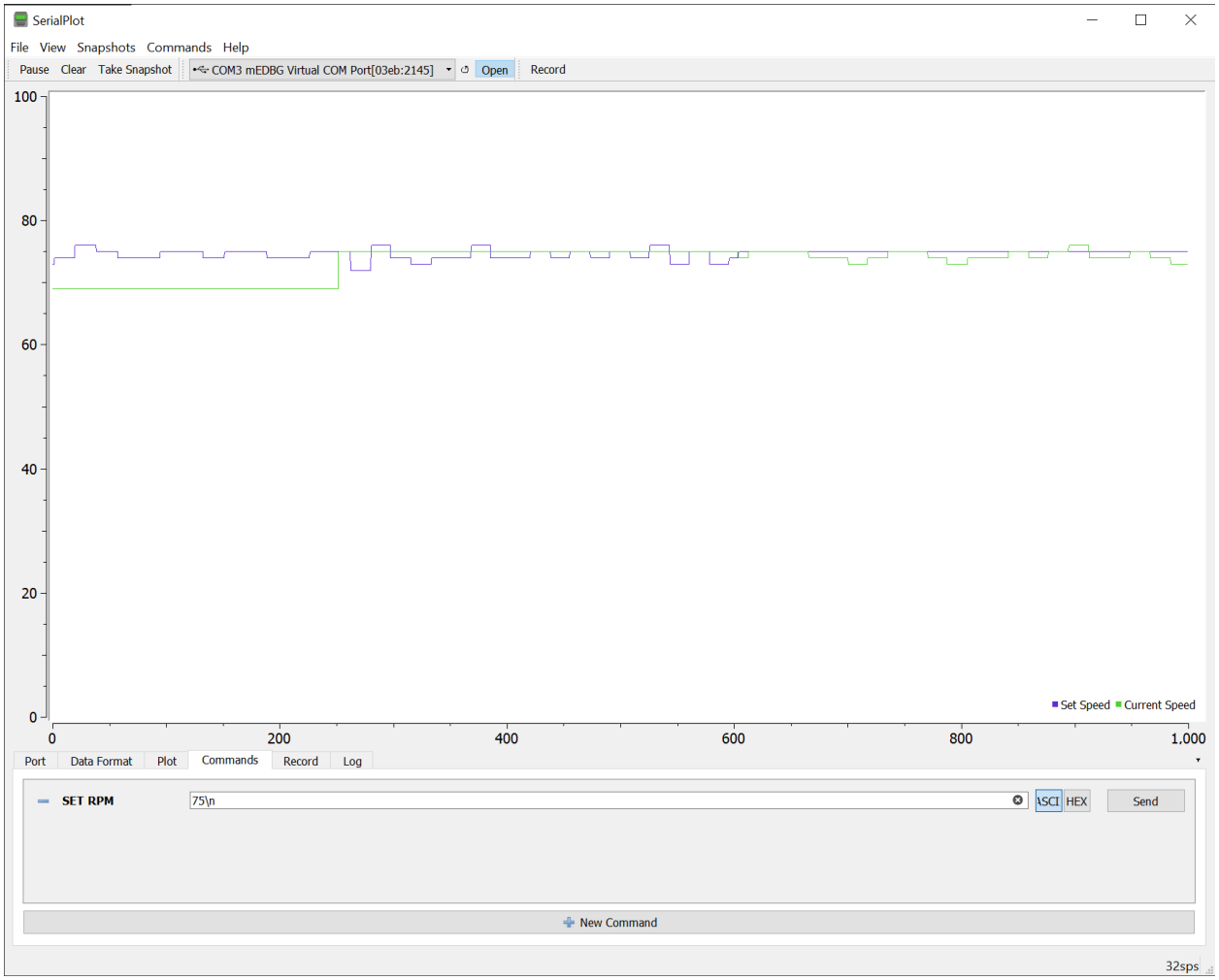


4. SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)

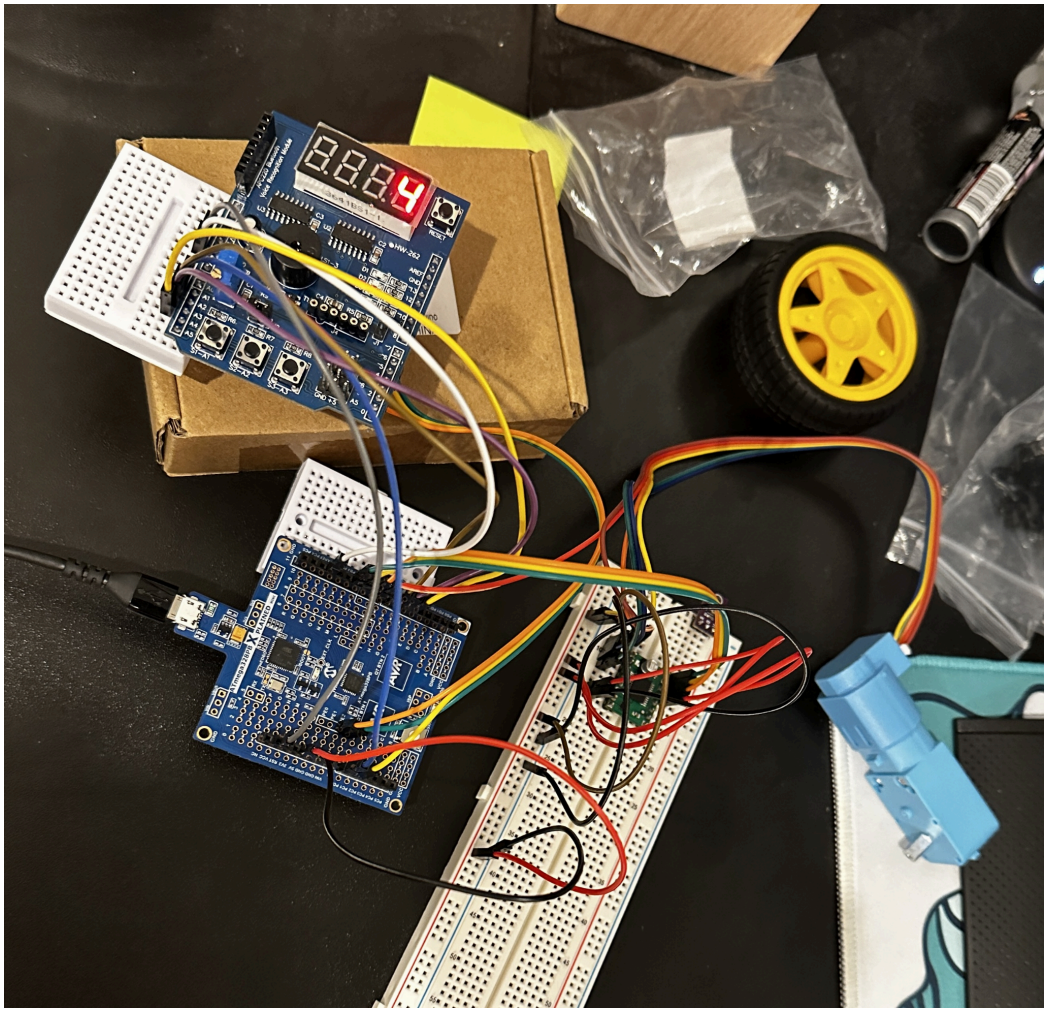
TASK 1 & 2



TASK 4



5. SCREENSHOT OF EACH DEMO (BOARD SETUP)



6. VIDEO LINKS OF EACH DEMO

https://youtu.be/SWg8tQvBVSo Task 1 & 2
https://youtu.be/Q3Yb-G9hxyo Task 3 & 4

7. GITHUB LINK OF THIS DA

https://github.com/n8ramos/atmega328pb/tree/main

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

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