CPE301 – SPRING 2021

MIDTERM 2

Student Name: Elmer Mejia

Student #: 5003824808

Student Email: mejiae4@unlv.nevada.edu

Primary Github address: https://github.com/cpemejia/design\_assignments.git

Directory: design\_assignments

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

-Atmega328pb -DC motor w/encoder -potentiometer

-Microchip Studio - TB6612

Table

Description automatically generated with medium confidence



1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASKS**

**1.Use the motor driver, program the ATMega328pb to drive the geared DC motor in CW(5sec) and CCW(5sec) direction for a given PWM. Repeat the above forever. (20)**

#include <avr/io.h>

#define *F\_CPU* 16000000UL

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

#define AIN2 5

#define AIN1 4

// PD6 PWMA

// PD5 AIN2

// PD4 AIN1

// ADC0/PC0

int main(void)

{

*uint8\_t* flag = 0;

//speed PWMA

DDRD |= (1<<6); // set PD6 output

// direction AIN2/AIN1

DDRD |= (1<<5); // set PD5 output

DDRD |= (1<<4); // set PD4 output

PORTD &= ~(1<<AIN2);

PORTD &= ~(1<<AIN1);

// Set timer0

OCR0A = 45; // initial

// Fast PWM, non inverted

TCCR0A |= ((1<<COM0A1) | (1<<WGM01) | (1<<WGM00));

TCCR0B |= (1<<CS01); // n = 8;

while (1)

{

if (flag == 1)

{

*\_delay\_ms*(500);

PORTD |= (1<<AIN1); // H

PORTD &= (~(1<<AIN2)); // L

*\_delay\_ms*(5000);

flag = 0;

}

else{

PORTD &= (~(1<<AIN1)); // L

PORTD |= (1<<AIN2); // H

*\_delay\_ms*(5000);

flag = 1;

}

}

}

**2.Using the Potentiometer connected to PC0, translate the ADC value (0~1023) to PWM value/speed of the motor (0~255 ifusingTimer0/2). Verify the operation. (10)**

/\*

\* MID2.c

\*

\* Created: 4/17/2021 2:19:23 PM

\* Author : ElmerOMejia

\*/

#include <avr/io.h>

#define *F\_CPU* 16000000UL

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

#define AIN2 5

#define AIN1 4

// PD6 PWMA

// PD5 AIN2

// PD4 AIN1

// ADC0/PC0

volatile unsigned int newADC;

*uint16\_t* read\_adc(*uint8\_t* channel);

void adc\_init(void);

int main(void)

{

*uint8\_t* flag = 0;

//speed PWMA

DDRD |= (1<<6); // set PD6 output

// direction AIN2/AIN1

DDRD |= (1<<5); // set PD5 output

DDRD |= (1<<4); // set PD4 output

PORTD &= ~(1<<AIN2);

PORTD &= ~(1<<AIN1);

adc\_init();

// Set timer0

OCR0A = 45; // initial

// Fast PWM, non inverted

TCCR0A |= ((1<<COM0A1) | (1<<WGM01) | (1<<WGM00));

TCCR0B |= (1<<CS01); // n = 8;

while (1)

{

if (flag == 1)

{

*\_delay\_ms*(500);

PORTD |= (1<<AIN1); // H

PORTD &= (~(1<<AIN2)); // L

*\_delay\_ms*(5000);

flag = 0;

}

else{

PORTD &= (~(1<<AIN1)); // L

PORTD |= (1<<AIN2); // H

*\_delay\_ms*(5000);

flag = 1;

}

newADC = (read\_adc(0)/4); // 10 bit ADC to 8 bit

*\_delay\_ms*(500);

if (newADC < 45){ // if MIN OCRA

OCR0A = 45;

}

else if (newADC > 243){ // MAX OCRA

OCR0A = 243;

}

else {

OCR0A = newADC;

}

}

}

void adc\_init(void)

{

/\*\* Setup and enable ADC \*\*/

ADMUX = (0<<REFS1)| // Reference Selection Bits

(1<<REFS0)| // AVcc - external cap at AREF

(0<<ADLAR)| // ADC Left Adjust Result

(0<<MUX2)| // Analog Channel Selection Bits

(0<<MUX1)| // ADC0 (PC0 PIN23)

(0<<MUX0);

ADCSRA = (1<<ADEN)| // ADC ENable

(0<<ADSC)| // ADC Start Conversion

(0<<ADATE)| // ADC Auto Trigger Enable

(0<<ADIF)| // ADC Interrupt Flag

(0<<ADIE)| // ADC Interrupt Enable

(1<<ADPS2)| // ADC Prescaler Select Bits

(0<<ADPS1)|

(1<<ADPS0);

}

*uint16\_t* read\_adc(*uint8\_t* channel){

ADMUX &= 0xF0; //Clear the older channel that was read

ADMUX |= channel; //Defines the new ADC channel to be read

ADCSRA |= (1<<ADSC); //Starts a new conversion

while(ADCSRA & (1<<ADSC)); //Wait until the conversion is done

return ADCW; //Returns the ADC value of the chosen channel

}

**3.Using the CCP capture pin of PWM1, in mode 1x and 2x determine the speed of the DC Motor for a set ADC Pot value/position. (35)**

/\*

\* tryq3.c

\*

\* Created: 4/24/2021 4:58:46 PM

\* Author : ElmerOMejia

\*/

#define BAUD 9600

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdio.h>

#define AIN2 5

#define AIN1 4

// PD6 PWMA

// PD5 AIN2

// PD4 AIN1

// ADC0/PC0

volatile *uint32\_t* T1Ovs1, T1Ovs2;

volatile *uint32\_t* revTick1, revTick2, revTick3, revTick4;

volatile *uint8\_t* Flag = 0;

*uint32\_t* period;

*uint16\_t* revCtr;

volatile float RPM;

float calc\_spd\_1x(*uint32\_t* p);

void USART\_send(char data); // Used to send integer to terminal

void USART\_putstring(char\* StringPtr); // Used to take in every character in the string and sends it to the terminal

void USART\_init(void); // Initializes the analog to digital functions, as well as OVF interrupt

char outs[20];

int main(void)

{

//speed PWMA

DDRD |= (1<<6); // set PD6 output

// direction AIN2/AIN1

DDRD |= (1<<5); // set PD5 output

DDRD |= (1<<4); // set PD4 output

DDRB &= ~(1<<0); // set SIG input

PORTB |= (1<<0);

PORTD &= ~(1<<AIN2);

PORTD &= ~(1<<AIN1);

init\_timer0\_pwm();

USART\_init();

TCNT1 = 0; // counter = 0

TIMSK1 |= (1<<ICIE1) | (1<<TOIE1); // enable ICP1&ovf interrupts

TCCR1B |= (1<<ICES1) | (1<<CS10); // rsing edge, prescaler 1

*sei*();

while (1)

{

*\_delay\_ms*(500);

PORTD |= (1<<AIN1); // H

PORTD &= (~(1<<AIN2)); // L

if (Flag==3)

{

period=((*uint32\_t*)(revTick3-revTick1)+((*uint32\_t*)T1Ovs2\*0x10000L));

// for 2x

// period=((uint32\_t)(((revTick3-revTick1)+(revTick4-revTick2))/2)+((uint32\_t)T1Ovs2\*0x10000L));

*snprintf*(outs, sizeof(outs),"%f rpm", calc\_spd\_1x(period));

USART\_putstring(outs);

USART\_putstring("\r\n");

*\_delay\_ms*(500);

//clear variables and registers

Flag=0;

T1Ovs2=0;

TIFR1=(1<<ICF1)|(1<<TOV1);

TIMSK1|=(1<<ICIE1)|(1<<TOIE1);

}

}

}

float calc\_spd\_1x(*uint32\_t* p)

{

// average of 10 samples

float rpm1 = 0;

for (int i = 0; i<10; i++){

rpm1 = rpm1 + ((p\*60)/(96))/50;

}

return rpm1/10; // return rpm

}

ISR(TIMER1\_CAPT\_vect){

if (Flag==0)

{

//first capture

revTick1=ICR1;

//falling edge

TCCR1B&=~(1<<ICES1);

// ovf = 0

T1Ovs2=0;

}

if (Flag==1)

{

// second capture

revTick2=ICR1;

//rising edge

TCCR1B|=(1<<ICES1);

}

if (Flag==2)

{

// third capture

revTick3=ICR1;

TIMSK1&=~((1<<ICIE1)|(1<<TOIE1));

// falling edge for 2x

//TCCR1B&=~(1<<ICES1);

}

/\*

// last flag check for 2x

if (Flag == 3)

{

// fourth capture

revTick4 = ICR1;

TIMSK1&=~((1<<ICIE1)|(1<<TOIE1));

}

\*/

Flag++;

}

ISR(TIMER1\_OVF\_vect){

//increment overflow counter

T1Ovs2++;

}

void init\_timer0\_pwm(void){

DDRD |= (1<<DDD6); // PD6 out

TCNT0 = 0;

OCR0A = 250; // 50% duty

TCCR0A |= (1<<COM0A1); // non-inverting

TCCR0A |= (1<<WGM01) | (1<<WGM00); // fast pwm mode

TCCR0B |= (1<< CS02); // prescaler 256

}

void USART\_init( void )

{

UBRR0H = 0;

UBRR0L = *F\_CPU*/16/BAUD - 1; // Used for the BAUD prescaler

UCSR0C = \_BV(UCSZ01) | \_BV(UCSZ00); /\* 8-bit data \*/

UCSR0B = \_BV(RXEN0) | \_BV(TXEN0); /\* Enable RX and TX \*/

}

void USART\_send(char data)

{

while (!(UCSR0A & (1 << UDRE0))); // Until UDRE0 goes high, it will keep looping

UDR0 = data; // UDR0 register grabs the value given from the parameter

}

void USART\_putstring(char \*StringPtr)

{

while ((\*StringPtr != '\0')){ // Until it reaches the end of the line, it will keep looping

while (!(UCSR0A & (1 << UDRE0))); // Until UDRE0 goes high, it will keep looping

UDR0 = \*StringPtr; // UDR0 register grabs the value given from the parameter

StringPtr++; // but it does it by every character as shown here

}

}

1. **For a given/setspeed of the DC Motor, control the speed using PID algorithm. (35)**

/\*

\* PID.c

\*

\* Created: 4/25/2021 6:59:04 PM

\* Author : ElmerOMejia

\*/

#define BAUD 9600

#define *F\_CPU* 1000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdio.h>

// PID

#define P\_val 1.0

#define I\_val 0.0

#define D\_val 0.0

#define AIN2 5

#define AIN1 4

#define I\_memory 20

// PD6 PWMA

// PD5 AIN2

// PD4 AIN1

// ADC0/PC0

// global variables for PID

volatile *int32\_t* mean\_I\_err[I\_memory],

PID=0,

err=0,

err\_old=0,

mean\_I\_error=0,

I\_err=0.0,

P\_err=0,

D\_err=0;

volatile float set\_speed = 100.0,

current\_speed = 0.0;

volatile *uint16\_t* duty = 0, // initial

duty\_max = 250; // about 100 RPM

volatile *uint8\_t* flag = 0;

// global variables for RPM

volatile *uint32\_t* T1Ovs1, T1Ovs2;

volatile *uint32\_t* revTick1, revTick2, revTick3, revTick4;

volatile *uint8\_t* Flag;

volatile *uint32\_t* period;

volatile *uint16\_t* revCtr;

volatile float RPM;

float calc\_spd\_1x(*uint32\_t* p);// func to calc speed

void pid\_controller(void);

void USART\_send(char data); // Used to send integer to terminal

void USART\_putstring(char\* StringPtr); // Used to take in every character in the string and sends it to the terminal

void USART\_init(void); // Initializes the analog to digital functions, as well as OVF interrupt

char outs[20];

int main(void)

{

//speed PWMA

DDRD |= (1<<6); // set PD6 output

// direction AIN2/AIN1

DDRD |= (1<<5); // set PD5 output

DDRD |= (1<<4); // set PD4 output

DDRB &= ~(1<<0); // set SIG input

PORTB |= (1<<0);

PORTD &= ~(1<<AIN2);

PORTD &= ~(1<<AIN1);

for(int i=0;i<I\_memory-1;i++){

mean\_I\_err[i]=(*int32\_t*)0.0;

}

init\_timer0\_pwm();

init\_timer2();

USART\_init();

TCNT1 = 0; // counter = 0

TIMSK1 |= (1<<ICIE1) | (1<<TOIE1); // enable ICP1&ovf interrupts

TCCR1B |= (1<<ICES1) | (1<<CS10); // rsing edge, prescaler 1

*sei*();

while (1)

{

*\_delay\_ms*(500);

PORTD |= (1<<AIN1); // H

PORTD &= (~(1<<AIN2)); // L

if (Flag==3)

{

period=((*uint32\_t*)(revTick3-revTick1)+((*uint32\_t*)T1Ovs2\*0x10000L));

// period=((uint32\_t)(((revTick3-revTick1)+(revTick4-revTick2))/2)+((uint32\_t)T1Ovs2\*0x10000L));

current\_speed = calc\_spd\_1x(period);

*snprintf*(outs, sizeof(outs),"%f rpm", calc\_spd\_1x(period));

USART\_putstring(outs);

USART\_putstring("\r\n");

*\_delay\_ms*(5000);

//clear variables and registers

Flag=0;

T1Ovs2=0;

TIFR1=(1<<ICF1)|(1<<TOV1);

TIMSK1|=(1<<ICIE1)|(1<<TOIE1);

}

if (flag == 1){

pid\_controller();

OCR0A = (duty/256);

}

}

}

ISR(TIMER1\_CAPT\_vect){

if (Flag==0)

{

//save captured timestamp

revTick1=ICR1;

//change capture on falling edge

TCCR1B&=~(1<<ICES1);

//reset overflows

T1Ovs2=0;

}

if (Flag==1)

{

revTick2=ICR1;

//change capture on rising edge

TCCR1B|=(1<<ICES1);

}

if (Flag==2)

{

revTick3=ICR1;

// //stop input capture and overflow interrupts

TIMSK1&=~((1<<ICIE1)|(1<<TOIE1));

//change capture on falling edge

//TCCR1B&=~(1<<ICES1);

}

/\*

if (Flag == 3)

{

revTick4 = ICR1;

TIMSK1&=~((1<<ICIE1)|(1<<TOIE1));

}

\*/

//increment Flag

Flag++;

}

ISR(TIMER1\_OVF\_vect){

//increment overflow counter

T1Ovs2++;

}

ISR(TIMER2\_COMPA\_vect){

if (flag == 1)

{

flag = 0;

}

else{

flag = 1;

}

}

void init\_timer0\_pwm(void){

DDRD |= (1<<DDD6); // PD6 out

TCNT0 = 0;

OCR0A = 250; // duty

TCCR0A |= (1<<COM0A1); // non-inverting

TCCR0A |= (1<<WGM01) | (1<<WGM00); // fast pwm mode

TCCR0B |= (1<< CS02); // prescaler 256

}

void init\_timer2(void){

TCCR2A |= (1 << WGM21)| (1<<COM2A0);//Set TCCR2A to CTC and toggle compare match

OCR2A = 194; // Set OCR2A for 10hz

TCNT2 = 0; // begin counter at 0

TCCR2B |= (1<<CS22)|(1<<CS21); // set prescaler to 256

}

float calc\_spd\_1x(*uint32\_t* p)

{

// average of 10 samples

float rpm1 = 0;

for (int i = 0; i<10; i++){

rpm1 = rpm1 + ((p\*60)/(96))/50;

}

return rpm1/10; // return rpm

}

void pid\_controller (void){

// Make sure that this is executed @ 10Hz (dt=0.1)

// current\_speedin rpm is updated by the encoder

//PID

err\_old=err;

err=set\_speed-current\_speed;

P\_err=err;

//Proportional is directly the difference times P\_val

//save new value to last

mean\_I\_err[I\_memory-1]=(*int32\_t*)err;

mean\_I\_error=0;

//move all samples left by one in array and sum them up

for(int i=0;i<I\_memory-1;i++){

mean\_I\_err[i]=(*int32\_t*)mean\_I\_err[i+1];

mean\_I\_error+=(*int32\_t*)mean\_I\_err[i];

}

mean\_I\_error+=(*int32\_t*)err; //take the latest value in account

I\_err=(*int32\_t*)mean\_I\_error;

D\_err=(*int32\_t*)err-err\_old;

PID=(*int32\_t*)P\_val\*P\_err+I\_val\*I\_err\*0.1+D\_val\*D\_err/0.1;

if(PID>duty\_max){

PID=duty\_max;

}

else if(PID<0){

PID=0;

}

duty=(*uint16\_t*)PID;

}

void USART\_init( void )

{

UBRR0H = 0;

UBRR0L = *F\_CPU*/1/BAUD - 1; // Used for the BAUD prescaler

UCSR0C = \_BV(UCSZ01) | \_BV(UCSZ00); /\* 8-bit data \*/

UCSR0B = \_BV(RXEN0) | \_BV(TXEN0); /\* Enable RX and TX \*/

}

void USART\_send(char data)

{

while (!(UCSR0A & (1 << UDRE0))); // Until UDRE0 goes high, it will keep looping

UDR0 = data; // UDR0 register grabs the value given from the parameter

}

void USART\_putstring(char \*StringPtr)

{

while ((\*StringPtr != '\0')){ // Until it reaches the end of the line, it will keep looping

while (!(UCSR0A & (1 << UDRE0))); // Until UDRE0 goes high, it will keep looping

UDR0 = \*StringPtr; // UDR0 register grabs the value given from the parameter

StringPtr++; // but it does it by every character as shown here

}

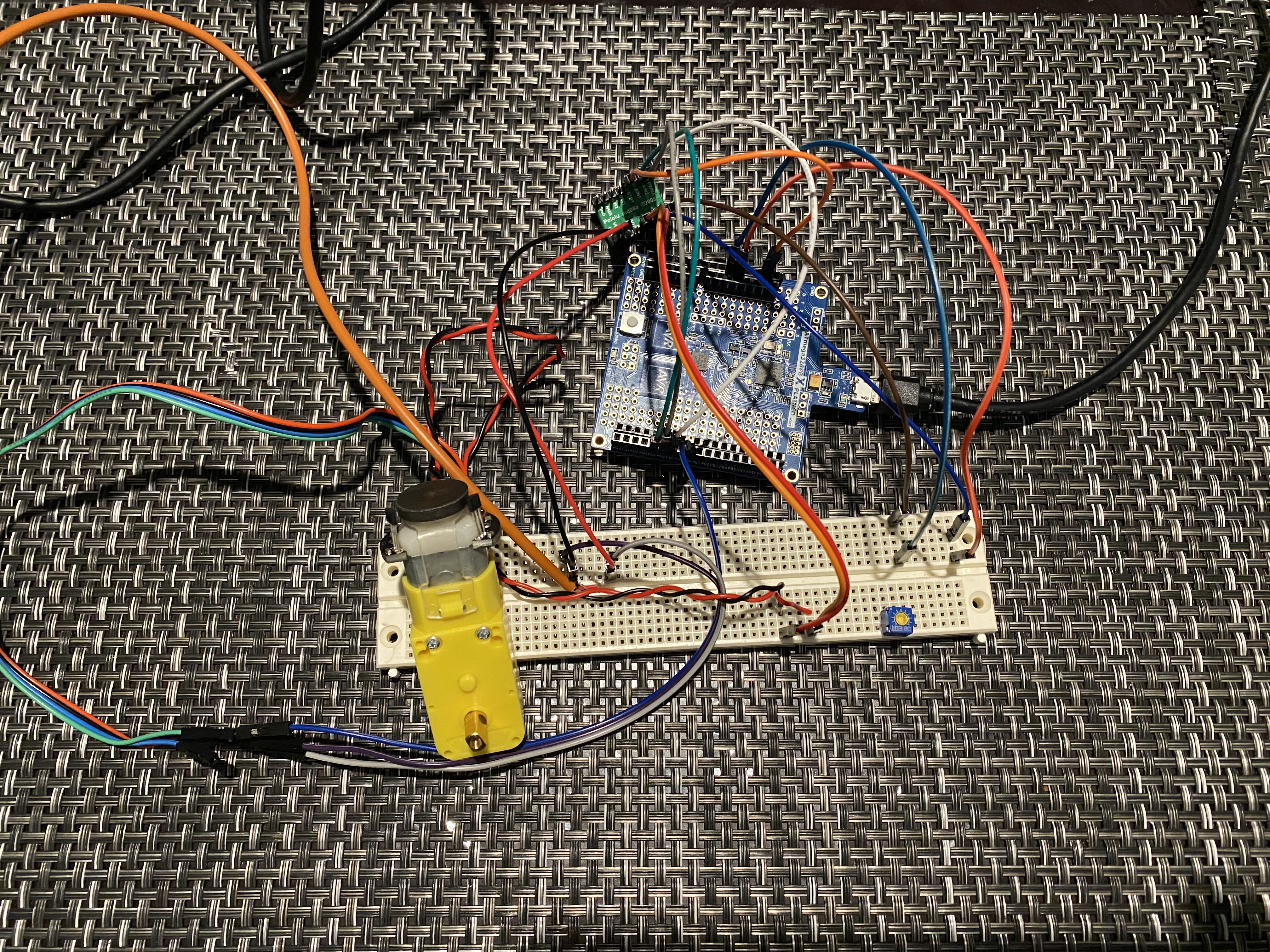
}

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

Graphical user interface, text, application

Description automatically generated

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



1. **VIDEO LINKS OF EACH DEMO**

[demos](https://www.youtube.com/watch?v=SN5ujZp1ugI&list=PL6PbL2NpuYCKbUNnRQzdehoDwvBhUNAA1)

1. **GITHUB LINK OF THIS DA**

[DA](https://github.com/cpemejia/design_assignments.git)

**Student Academic Misconduct Policy**

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

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

Elmer Mejia