CPE301 – SPRING 2020

MIDTERM 2

Student Name: Xianjie Cao

Student #: 5004222179

Student Email: [caox2@unlv.nevada.edu](mailto:caox2@unlv.nevada.edu)

Primary Github address: <https://github.com/c1029324620/Mocha.git>

Directory: Mocha/Midterm2

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

Atmel Studio 7: Debugger, simulator and assembler

Atmega328pb Xmini-PC

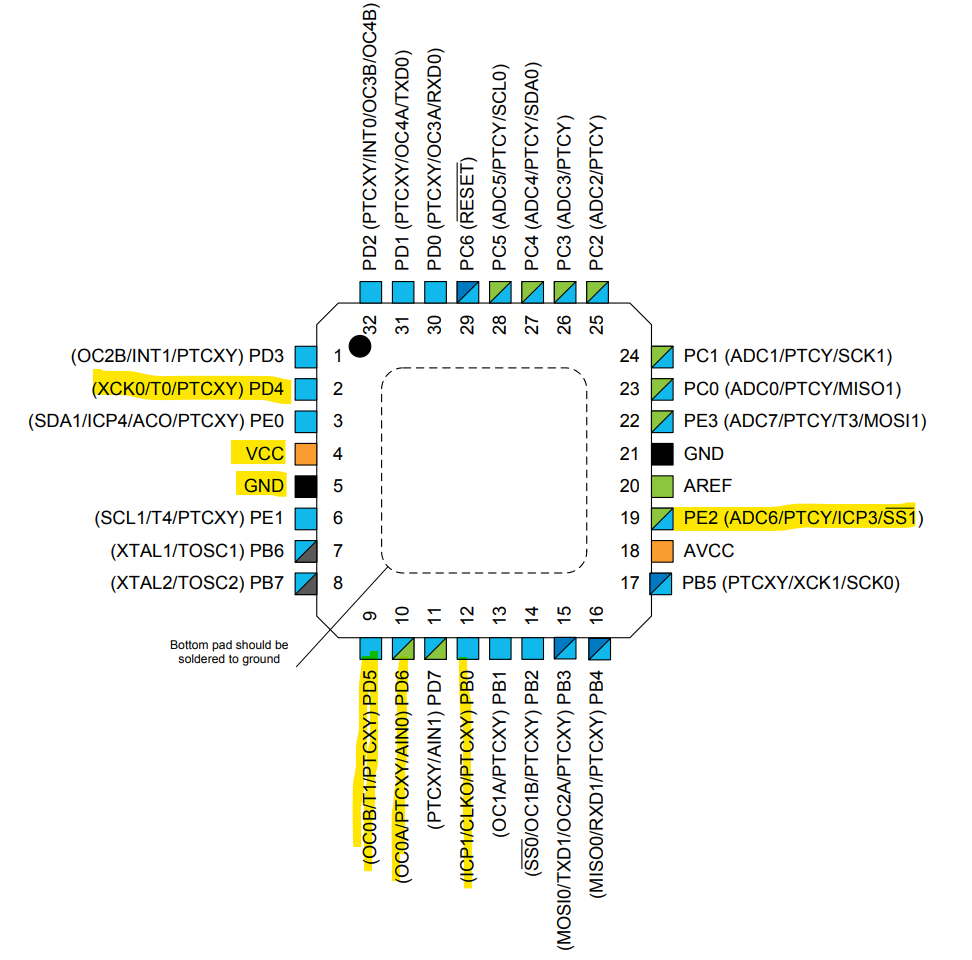
DC motor with decoder

Breadboard

Jump wires

Arduino multi-functional shield

TB6612FNG



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

Task 1:

/\*

\* Task1.c

\*

\* Created: 4/8/2020 10:34:53 AM

\* Author : c1029

\*/

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

int main(void)

{

int Flag;

DDRD |= (1<< DDRD4) | (1<< DDRD5) | (1<< DDRD6); //make PD4,PD5, and PD6 as outputs.

OCR0A = 128; //50% duty cycle

TCCR0A |= (1<< COM0A1) | (1<<WGM01) | (1<< WGM00); //non-inverting and fast PWM

TCCR0B |= (1<< CS00); //no prescaler.

/\* Replace with your application code \*/

while (1)

{

if(Flag == 1)

{

PORTD|=(1<<DDD5);// CW Direction Set

PORTD&=~(1<<DDD4);// CW Direction Set

*\_delay\_ms*(5000);

Flag = 0;

}

else

{

PORTD&=~(1<<DDD5);// CCW Direction Set

PORTD|=(1<<DDD4);// CCW Direction Set

*\_delay\_ms*(5000);

Flag = 1;

}

}

}

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

Task 2:

/\*

\* Task2.c

\*

\* Created: 4/8/2020 1:29:52 PM

\* Author : c1029

\*/

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

unsigned int adc\_value;

void init\_adc()

{

ADMUX = (0<<REFS1) | //AVref

(1<<REFS0) |

(1<<ADLAR) | //Left justified

(0<<MUX2) | //ADC0 channel 0

(0<<MUX1) |

(0<<MUX0);

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

(0<<ADSC) |

(0<<ADATE)|

(0<<ADIF) |

(0<<ADIE) |

(1<<ADPS2)| //prescaler of 128

(1<<ADPS1)|

(1<<ADPS0);

}

void read\_adc()

{

ADCSRA |= (1 << ADSC); //enable start conversion

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

adc\_value = ADCH; // potentiometer value stored in adc\_value

}

int main(void)

{

int Flag = 1;

init\_adc();

DDRD |= (1<< DDRD4) | (1<< DDRD5) | (1<< DDRD6); //make PD4,PD5, and PD6 as outputs.

TCCR0A |= (1<< COM0A1) | (1<<WGM01) | (1<< WGM00); //non-inverting and fast PWM

TCCR0B |= (1<< CS00); //no prescaler.

/\* Replace with your application code \*/

while (1)

{

read\_adc();

OCR0A = adc\_value; //duty cycle are based on the adc value read.

if(Flag == 1)

{

PORTD|=(1<<DDD5);// CW Direction Set

PORTD&=~(1<<DDD4);// CW Direction Set

*\_delay\_ms*(5000);

Flag = 0;

}

else

{

PORTD&=~(1<<DDD5);// CCW Direction Set

PORTD|=(1<<DDD4);// CCW Direction Set

*\_delay\_ms*(5000);

Flag = 1;

}

}

}

Task 3:

/\*

\* Task3.c

\*

\* Created: 4/8/2020 1:50:39 PM

\* Author : c1029

\*/

#define *F\_CPU* 16000000UL

#define BAUD 9600

#include <avr/io.h>

#include <util/setbaud.h>

#include <stdlib.h>

#include <stdio.h>

#include <util/delay.h>

#include <avr/interrupt.h>

int Flag = 0;

unsigned long revTick = 0;

unsigned long Capt2 = 0;

unsigned long Capt3 = 0;

unsigned long Capt4 = 0;

//uint32\_t T1Ovs1 = 0;

//uint32\_t T1Ovs2 = 0;

//uint32\_t T1Ovs3 = 0;

//uint32\_t T10vs4 = 0;

unsigned overflow\_cnt= 0;

int speed = 0;

unsigned long PulseWidth, Period;

volatile float DutyCycle;

unsigned int adc\_value;

void USART\_init(void)

{

UBRR0H = *UBRRH\_VALUE*;

UBRR0L = *UBRRL\_VALUE*;

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

UCSR0B = \_BV(RXEN0) | \_BV(TXEN0); //enable receiver and transmitter

}

void USART\_tx\_string(char\* data)

{

while((\*data !='\0'))

{

while(!(UCSR0A & (1<<UDRE0)));

UDR0 = \*data;

data++;

}

}

void adc\_read()

{

ADCSRA |= (1 << ADSC); //enable start conversion

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

adc\_value = ADCH; // potentiometer value stored in adc\_value

}

void display\_adc()

{

adc\_read();

char array1[32];

*sprintf*(array1,"%i", adc\_value); //convert int to string

USART\_tx\_string("ADC Value: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void display\_duty()

{

char array1[32];

*sprintf*(array1,"%f", DutyCycle); //convert int to string

USART\_tx\_string("Duty Cycle: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void display\_period()

{

char array1[32];

*sprintf*(array1,"%lu", Period); //convert int to string

USART\_tx\_string("Period: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void display\_pulse()

{

char array1[32];

*sprintf*(array1,"%lu", PulseWidth); //convert int to string

USART\_tx\_string("Pulsewidth: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void init\_adc()

{

ADMUX = (0<<REFS1) | //AVref

(1<<REFS0) |

(1<<ADLAR) | //Left justified

(0<<MUX2) | //ADC0 channel 0

(0<<MUX1) |

(0<<MUX0);

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

(0<<ADSC) |

(0<<ADATE)|

(0<<ADIF) |

(0<<ADIE) |

(1<<ADPS2)| //prescaler of 128

(1<<ADPS1)|

(1<<ADPS0);

}

void init\_timer1()

{

DDRB &= ~(1<<DDB0);

PORTB |= (0<<DDB0);

TCNT1 = 0;

TCCR1A = 0;

TCCR1B = (0<<ICNC1) | (1<<ICES1); //RISING EDGE CAPTURE.

TCCR1C = 0;

TIFR1 =(1<<ICF1)|(1<<TOV1);// clear pending

TIMSK1 = (1<<ICIE1)|(1<<TOIE1);// and enable

}

void start\_timer1()

{

//Start timer without prescaller

TCCR1B|=(1<< CS10);

}

void timer0\_pwm()

{

DDRD |= (1<<DDD6);

TCNT0 = 0;

adc\_read();

OCR0A = adc\_value;

PORTD|=(1<<DDD5);// CW Direction Set

PORTD&=~(1<<DDD4);// CW Direction Set

TCCR0A |= (1<< COM0A1) | (1<<WGM01) | (1<< WGM00); //non-inverting and fast PWM

TCCR0B |= (1<< CS00) |(1<<CS02); //prescaler to 1024

}

void speed\_rpm()

{

speed = (1/(Period\* 0.0000000625)) \* 60 /120L;

}

void display\_speed()

{

char array1[20];

*sprintf*(array1,"%i", speed); //convert int to string

USART\_tx\_string("Speed: "); //print

USART\_tx\_string(array1);

USART\_tx\_string(" rpm\n");

}

int main(void)

{

init\_adc();

timer0\_pwm();

USART\_init();

init\_timer1();

start\_timer1();

sei();

/\* Replace with your application code \*/

//USART\_tx\_string("Connected\n");

while (1)

{

adc\_read(); //update adc value

OCR0A = adc\_value; //set speed.

if(Flag == 4)

{

Period= ((Capt3 - revTick + Capt4 - Capt2) / 2); //get the average period

PulseWidth = (Capt2 - revTick + Capt4 - Capt3) / 2; //get the average pulse width

DutyCycle = PulseWidth/(float)Period;

Flag = 0;

display\_duty();

display\_pulse();

speed\_rpm(); //calculate speed in rpm.

display\_period();

display\_speed();

TCNT1 = 0;

overflow\_cnt = 0;

*\_delay\_ms*(2000); //delay for printing result to terminal

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

}

}

}

//capture ISR

ISR(TIMER1\_CAPT\_vect)

{

//2x encoding

if(Flag == 0)

{

revTick = ICR1 + overflow\_cnt \* 65536;

//TCNT1=0;// restart timer for next revolution

//T1Ovs1=overflow\_cnt;

TCCR1B&=~(1<<ICES1); //switch to falling edge

}

if(Flag == 1)

{

Capt2 = ICR1 + overflow\_cnt \* 65536;

TCCR1B|=(1<<ICES1); //switch back to rising edge

//T1Ovs2=overflow\_cnt;

}

if(Flag ==2)

{

Capt3 = ICR1 + overflow\_cnt \* 65536;

TCCR1B&=~(1<<ICES1); //switch to falling edge

//T1Ovs3 = overflow\_cnt;

}

if(Flag == 3)

{

Capt4 = ICR1 + overflow\_cnt \* 65536;

//T10vs4 = overflow\_cnt;

TIMSK1&=~((1<<ICIE1)|(1<<TOIE1)); //stop input capture and overflow interrupt.

}

Flag++;

}

ISR(TIMER1\_OVF\_vect)

{

overflow\_cnt++;

}

Task 4:

/\*

\* Task4.c

\*

\* Created: 4/8/2020 1:50:39 PM

\* Author : c1029

\*/

#define *F\_CPU* 16000000UL

#define BAUD 9600

#include <avr/io.h>

#include <util/setbaud.h>

#include <stdlib.h>

#include <stdio.h>

#include <util/delay.h>

#include <avr/interrupt.h>

int Flag = 0;

int Flag\_2 = 0;

unsigned long Cpa1 = 0;

unsigned long Capt2 = 0;

unsigned long Capt3 = 0;

unsigned long Capt4 = 0;

unsigned long Capt5 = 0;

unsigned long Capt6 = 0;

unsigned long Capt7 = 0;

unsigned long Capt8 = 0;

unsigned overflow\_cnt= 0;

unsigned overflow\_cnt\_2 = 0;

int speed = 0;

unsigned long PulseWidth, Period;

volatile float DutyCycle;

unsigned int adc\_value;

void USART\_init(void)

{

UBRR0H = *UBRRH\_VALUE*;

UBRR0L = *UBRRL\_VALUE*;

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

UCSR0B = \_BV(RXEN0) | \_BV(TXEN0); //enable receiver and transmitter

}

void USART\_tx\_string(char\* data)

{

while((\*data !='\0'))

{

while(!(UCSR0A & (1<<UDRE0)));

UDR0 = \*data;

data++;

}

}

void adc\_read()

{

ADCSRA |= (1 << ADSC); //enable start conversion

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

adc\_value = ADCH; // potentiometer value stored in adc\_value

}

void display\_adc()

{

adc\_read();

char array1[32];

*sprintf*(array1,"%i", adc\_value); //convert int to string

USART\_tx\_string("ADC Value: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void display\_duty()

{

char array1[32];

*sprintf*(array1,"%f", DutyCycle); //convert int to string

USART\_tx\_string("Duty Cycle: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void display\_period()

{

char array1[32];

*sprintf*(array1,"%lu", Period); //convert int to string

USART\_tx\_string("Period: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void display\_pulse()

{

char array1[32];

*sprintf*(array1,"%lu", PulseWidth); //convert int to string

USART\_tx\_string("Pulsewidth: "); //print

USART\_tx\_string(array1);

USART\_tx\_string("\n");

}

void init\_adc()

{

ADMUX = (0<<REFS1) | //AVref

(1<<REFS0) |

(1<<ADLAR) | //Left justified

(0<<MUX2) | //ADC0 channel 0

(0<<MUX1) |

(0<<MUX0);

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

(0<<ADSC) |

(0<<ADATE)|

(0<<ADIF) |

(0<<ADIE) |

(1<<ADPS2)| //prescaler of 128

(1<<ADPS1)|

(1<<ADPS0);

}

void init\_timer1()

{

DDRB &= ~(1<<DDB0); //PB0 as input for timer1 input capture

PORTB |= (0<<DDB0);

TCNT1 = 0;

TCCR1A = 0;

TCCR1B = (0<<ICNC1) | (1<<ICES1); //RISING EDGE CAPTURE.

TCCR1C = 0;

TIFR1 =(1<<ICF1)|(1<<TOV1);// clear pending

TIMSK1 = (1<<ICIE1)|(1<<TOIE1);// enable interrupt

}

void init\_timer3()

{

DDRE &= ~(1<<DDE2); //PE2 as input for timer3 input capture

PORTE |= (0<<2);

TCNT3 = 0;

TCCR3A = 0;

TCCR3B = (0<<ICNC3) | (1<<ICES3); //rising edge capture.

TCCR3C = 0;

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

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

}

void start\_timer1()

{

//Start timer without prescaller

TCCR1B|=(1<< CS10);

}

void start\_timer3()

{

TCCR3B |=(1<<CS30); //start timer3 without prescaller

}

void timer0\_pwm()

{

DDRD |= (1<<DDD6);

TCNT0 = 0;

adc\_read();

OCR0A = adc\_value;

PORTD|=(1<<DDD5);// CW Direction Set

PORTD&=~(1<<DDD4);// CW Direction Set

TCCR0A |= (1<< COM0A1) | (1<<WGM01) | (1<< WGM00); //non-inverting and fast PWM

TCCR0B |= (1<< CS00) |(1<<CS02); //prescaler to 1024

}

void speed\_rpm()

{

speed = (1/(Period\* 0.0000000625)) \* 60 /120L;

}

void display\_speed()

{

char array1[20];

*sprintf*(array1,"%i", speed); //convert int to string

USART\_tx\_string("Speed: "); //print

USART\_tx\_string(array1);

USART\_tx\_string(" rpm\n");

}

int main(void)

{

init\_adc();

USART\_init();

init\_timer1();

init\_timer3();

start\_timer1();

start\_timer3();

timer0\_pwm();

sei();

unsigned long period\_1 = 0, period\_2 = 0;

unsigned long PulseWidth\_1 = 0, PulseWidth\_2 = 0;

volatile float DutyCycle\_1, DutyCycle\_2;

/\* Replace with your application code \*/

//USART\_tx\_string("Connected\n");

while (1)

{

adc\_read(); //update adc value

OCR0A = adc\_value; //set speed.

if(Flag == 4) //Ch.A finish capturing

{

period\_1 =((Capt3 - Cpa1 + Capt4 - Capt2) / 2); //get the average period

PulseWidth\_1 = (Capt2 - Cpa1 + Capt4 - Capt3) / 2; //get the average pulse width

DutyCycle\_1 = PulseWidth\_1/(float)period\_1;

Flag++;

}

if(Flag\_2 == 4) //Ch.B finish capturing

{

period\_2 =((Capt7 - Capt5 + Capt8 - Capt6) / 2); //get the average period

PulseWidth\_2 = (Capt6 - Capt5 + Capt8 - Capt7) / 2; //get the average pulse width

DutyCycle\_2 = PulseWidth\_2/(float)period\_2;

Flag\_2++;

}

if(Flag == 5 && Flag\_2 == 5)

{

DutyCycle = (DutyCycle\_1 + DutyCycle\_2)/2;

Period = (period\_1 + period\_2) /2;

PulseWidth = (PulseWidth\_1 + PulseWidth\_2) /2;

display\_duty();

display\_pulse();

speed\_rpm(); //calculate speed in rpm.

display\_period();

display\_speed();

Flag = 0;

Flag\_2 = 0;

TCNT1 = 0;

TCNT3 = 0;

overflow\_cnt = 0;

overflow\_cnt\_2 = 0;

*\_delay\_ms*(2000); //delay for printing result to terminal

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

TIMSK3|=(1<<ICIE3)|(1<<TOIE3); //restart interrupt.

}

}

}

//capture ISR

ISR(TIMER1\_CAPT\_vect)

{

//4x encoding

if(Flag == 0)

{

Cpa1 = ICR1 + overflow\_cnt \* 65536;

//TCNT1=0;// restart timer for next revolution

//T1Ovs1=overflow\_cnt;

TCCR1B&=~(1<<ICES1); //switch to falling edge

}

if(Flag == 1)

{

Capt2 = ICR1 + overflow\_cnt \* 65536;

TCCR1B|=(1<<ICES1); //switch back to rising edge

//T1Ovs2=overflow\_cnt;

}

if(Flag ==2)

{

Capt3 = ICR1 + overflow\_cnt \* 65536;

TCCR1B&=~(1<<ICES1); //switch to falling edge

//T1Ovs3 = overflow\_cnt;

}

if(Flag == 3)

{

Capt4 = ICR1 + overflow\_cnt \* 65536;

//T10vs4 = overflow\_cnt;

TIMSK1&=~((1<<ICIE1)|(1<<TOIE1)); //stop input capture and overflow interrupt.

}

Flag++;

}

ISR(TIMER3\_CAPT\_vect)

{

//4x encoding

if(Flag\_2 == 0)

{

Capt5 = ICR3 + overflow\_cnt\_2 \* 65536;

TCCR3B&=~(1<<ICES3); //switch to falling edge

}

if(Flag\_2 == 1)

{

Capt6 = ICR3 + overflow\_cnt\_2 \* 65536;

TCCR3B|=(1<<ICES3); //switch back to rising edge

}

if(Flag\_2 ==2)

{

Capt7 = ICR3 + overflow\_cnt\_2 \* 65536;

TCCR3B&=~(1<<ICES3); //switch to falling edge

}

if(Flag\_2 == 3)

{

Capt8 = ICR3 + overflow\_cnt\_2 \* 65536;

TIMSK3&=~((1<<ICIE3)|(1<<TOIE3)); //stop input capture and overflow interrupt.

}

Flag\_2++;

}

ISR(TIMER3\_OVF\_vect)

{

overflow\_cnt\_2++;

}

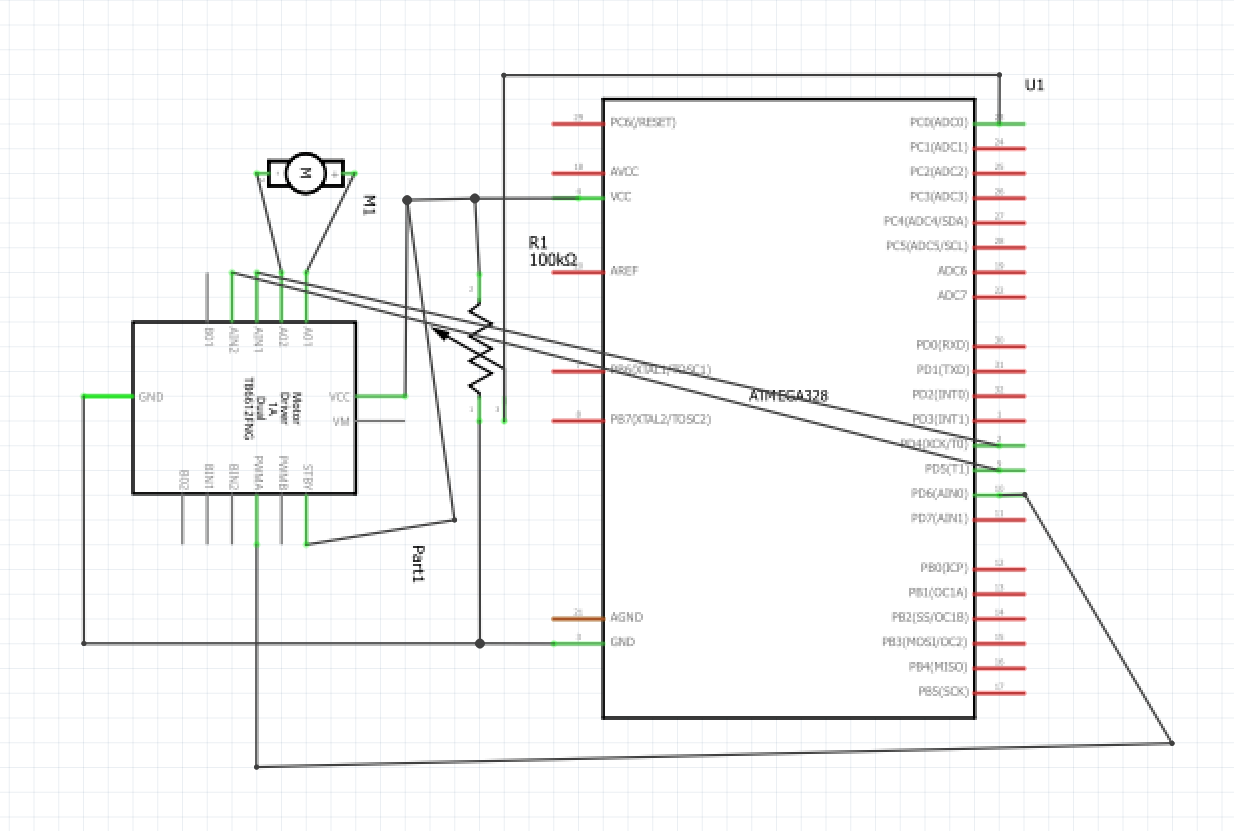
ISR(TIMER1\_OVF\_vect)

{

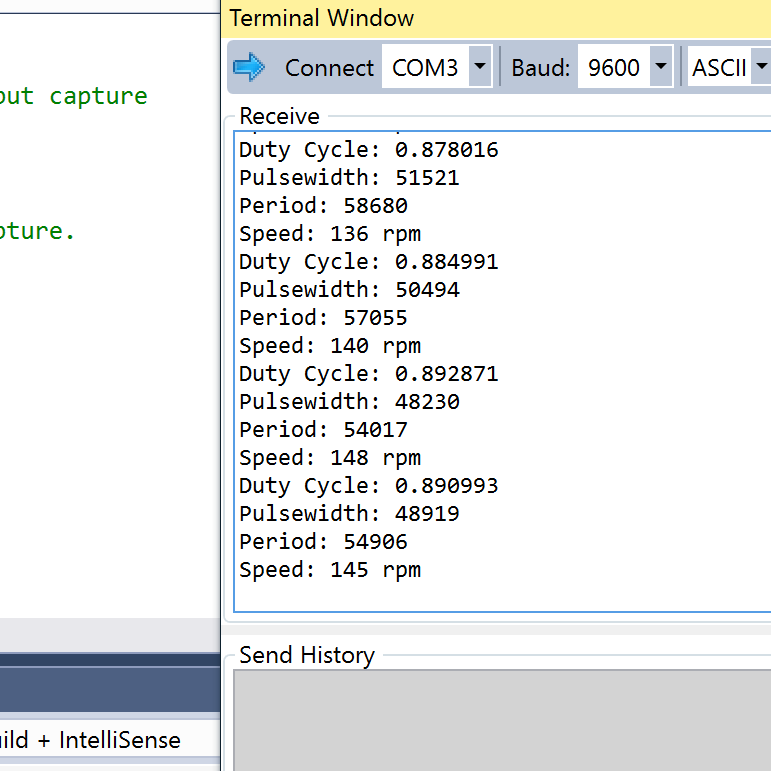
overflow\_cnt++;

}

1. **SCHEMATICS**

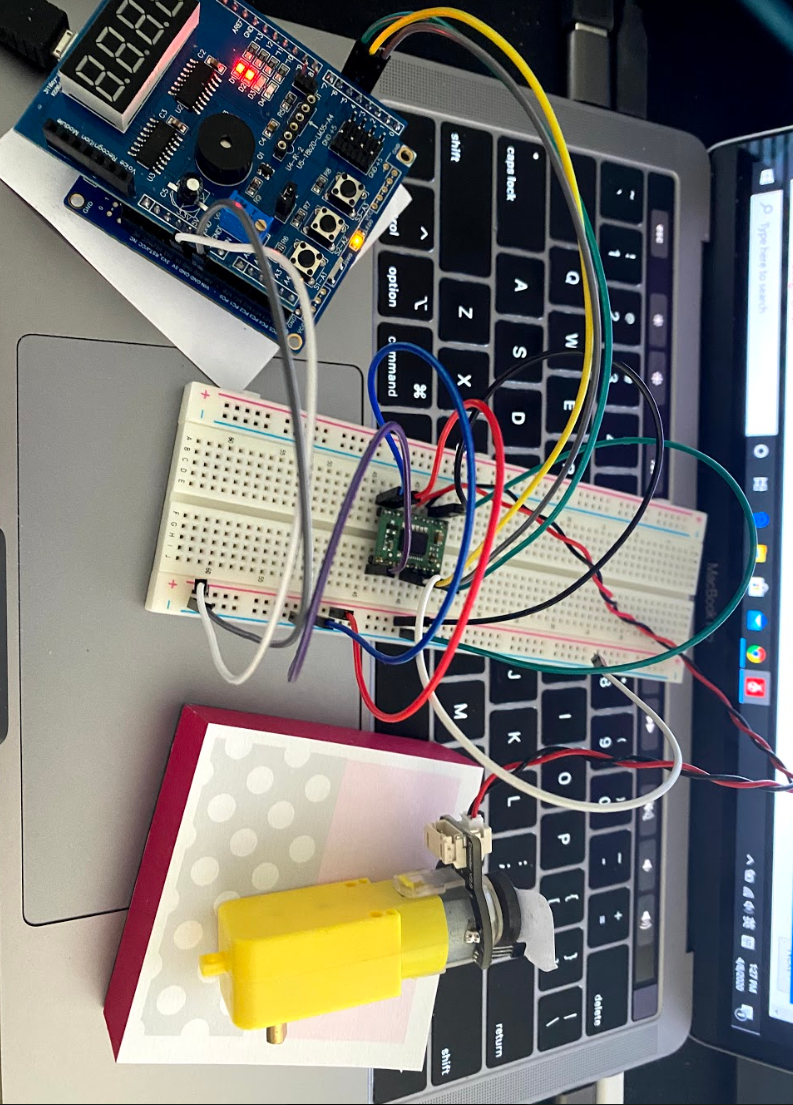


1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



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

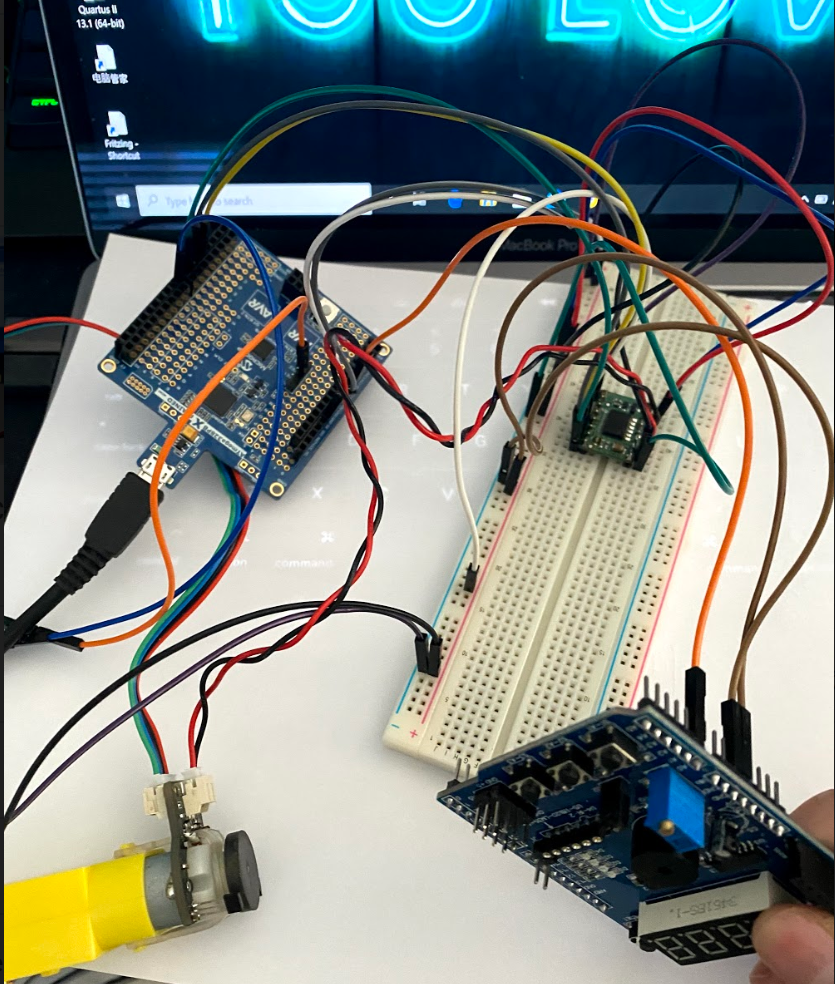
Task 1 and Task 2:



Task 3:



Task 4:



1. **VIDEO LINKS OF EACH DEMO**

Task 1: <https://youtu.be/Ng6VMq2XRT4>

Task 2: <https://youtu.be/dxA80agWcLc>

Task 3: <https://youtu.be/ekZuzCWtv3A>

Task 4: <https://youtu.be/4X7eyJI9g_A>

1. **GITHUB LINK OF THIS DA**

<https://github.com/c1029324620/Mocha/tree/master/Midterm2>

**Student Academic Misconduct Policy**

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

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

Xianjie Cao