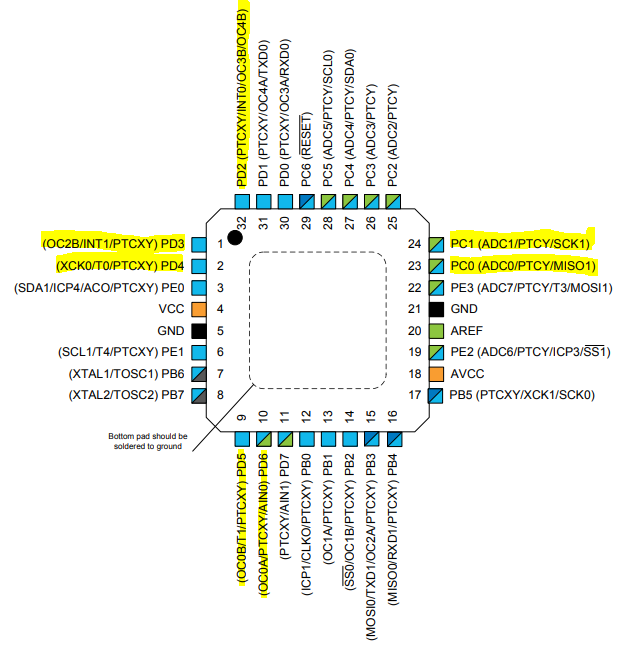
CPE301 – SPRING 2020

Midterm 2

Write, simulate, and demonstrate using Atmel Studio 7 a C code for the AVR ATmega328p/pb microcontroller that performs the following functions:

1. Use the motor driver, program the ATmega to drive the geared DC motor in CW and CCW direction for a given PWM.
2. Using the Potentiometer connected to ADC0, translate the ADC value to PWM value/speed of the motor. Verify the operation.
3. Using the CCP capture of PWM1, in mode 1X and 2X determine the speed of the DC Motor for a set ADC pot value/position
4. Using CCP capture and interrupt (mode 4x), determine the speed of the DC Motor for a set ADC Pot value/position.
5. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**



ATmega328PB Xplained mini

Data Visualizer

TB6612FNG

USB cable

DC motor

3 AA battery

Atmel Studio 7

BreadBoard

Notes:

* I was able to make encoder B work sometimes, but signal is sometimes unstable.
* I used INT0/INT1 interrupt to read the number of ticks manually
* I picked CCW for direction of motor.
* 1 pushbutton to turn on/off the motor.

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

/\*

\* midterm2.c

\*

\* Created: 4/13/2020 4:06:33 PM

\* Author : John Paulo Lumbres

\*/

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdio.h>

#include <stdlib.h>

#define MTR\_1 5

#define MTR\_2 4

#define SW (PINC&(1<<1))

#define UBRR\_9600 103

//declaring variable for controlling and storing values

volatile unsigned int power, revCtr0=0, revCtr1=0, escape=0, valcap0=100, valcap1=100;

volatile unsigned int rise0\_0, fall0\_0, rise0\_1, fall0\_1;

volatile unsigned int rise1\_0, fall1\_0, rise1\_1, fall1\_1;

volatile float t1ovf=0, t2ovf=0, done0=0, done1=0;

volatile float res1=0, res2=0, res4=0;

volatile float rpm1, rpm2, rpm4;

volatile unsigned int adc\_val;

char outs[20];

void adc\_init(void)

{

ADMUX =(1<<REFS0)|(1<<ADLAR); //AVcc reference, left adjust, ADC0 = PC0 = potentiometer,ADC enable

ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1); //prescaler 64

}

void read\_adc(void)

{

unsigned char i=4;

adc\_val = 0; //initial value

while(i--){ //get 4 values of ADC

ADCSRA |= (1<<ADSC);

while((ADCSRA & (1<<ADIF))==0);

ADCSRA |= (1<<ADIF);

adc\_val+=ADCH;

*\_delay\_ms*(50);

}

adc\_val = adc\_val/4; //have the average

}

void USART\_init(unsigned int ubrr) { //initializes USART

UBRR0H = (unsigned char)(ubrr>>8); //seting UBBR0

UBRR0L = (unsigned char)ubrr;

UCSR0B = (1<<RXEN0) | (1<<TXEN0); //enable receiver, transmitter & RX interrupt

UCSR0C = (1<<UCSZ01) | (1<<UCSZ00); //asynchronous 8 N 1

}

void USART\_tx\_string(char \*data) { //void to display on terminal

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

while(!(UCSR0A &(1<<UDRE0))); //while not done reading

UDR0 = \*data;

data++; //keep moving to next bit

}

}

unsigned char uart\_receive(void){

while(!(UCSR0A &(1<<7))); //while rxc0 is not set, keep reading bits

return UDR0;

}

void timer0\_init(){

TCCR0A = (1<<COM0A1)|(1<<WGM01)|(1<<WGM00); //fast PWM

TCCR0B = (1<<CS01); //prescaler 8

}

void timer1\_init(){

TCCR1A = 0; //normal mode

TCCR1B = (1<<CS11); //prescaler 8

TIMSK1 = 1; //overflow interrupt enabled

}

void timer2\_init(){

TCCR2A = 0; //normal mode

TCCR2B = (1<<CS21); //prescaler 8

TIMSK2 = 1; //overflow interrupt enabled

}

void pc\_int(){

PCMSK1 = (1<<PCINT9); //INT9 (PC1) enabled

PCICR = (1<<PCIE1); //Pin change interrupt enabled

}

void int\_interrupt(){

EIMSK = 0x03; //INT0 and INT1 external interrupt

EICRA = 0x0f; //rising edge for both

}

void calculate(){

//factor to multiply ticks by to get rsm

//(1920\*8\*2)/16000000 = 0.00192

//rpm = 60/rms

*\_delay\_ms*(100);

//get average of the 20 values

res1= res1/100;

res2= res2/100;

res4= res4/100;

//using the value and formula calculated above

rpm1 = 60/((float)res1 \* 0.00192);

rpm2 = 60/((float)res2 \* 0.00192);

rpm4 = 60/((float)res4 \* 0.00192);

}

void initial() {

rpm1=rpm2=rpm4=0; //initialize values

escape=1;

valcap0=20;

valcap1=20;

}

int main(void)

{

int spd\_control; //declare a variable

adc\_init(); //initialize adc

USART\_init(UBRR\_9600); //initialize usart

timer0\_init(); //initialize timers

timer1\_init();

timer2\_init();

pc\_int(); //initialize interrupts

int\_interrupt();

DDRC &= ~((1<<PC1)|(1<<PC0)); //PC1 and PC0 are inputs

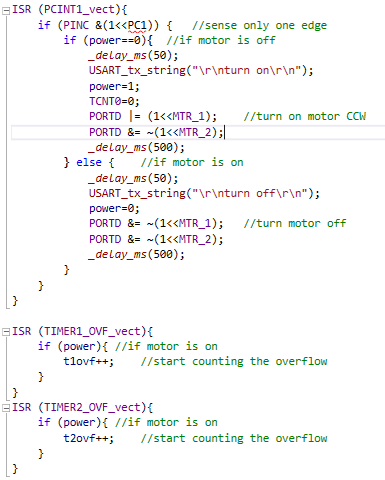
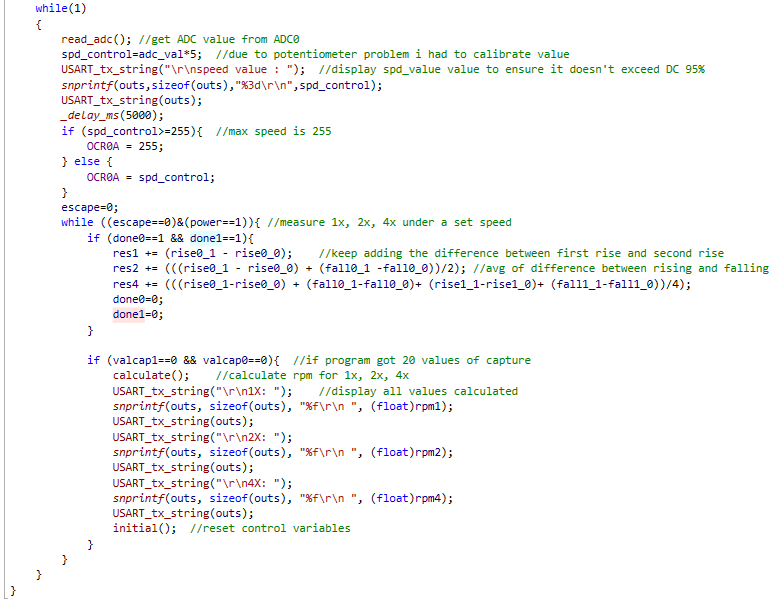
PORTC = (1<<PC1); //pull-up enabled

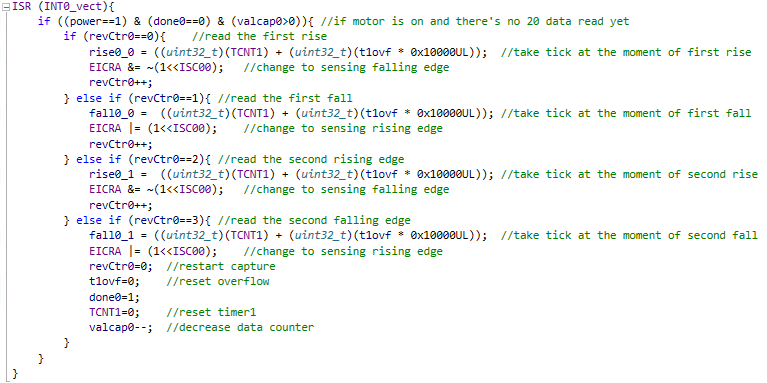
DDRD |= (1<<PD6)|(1<<PD5)|(1<<PD4); //PD6, PD5, PD4 outputs

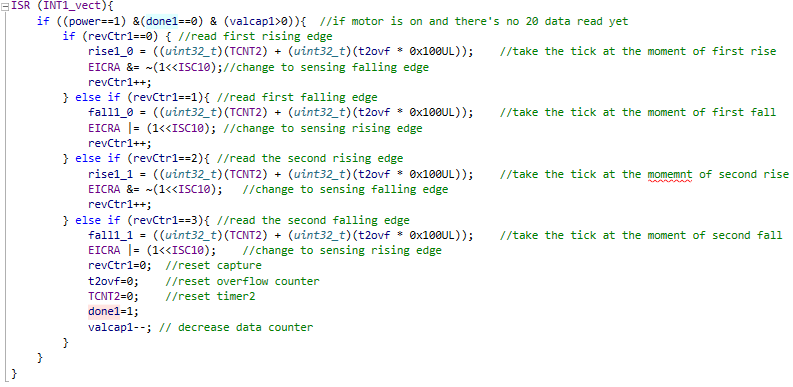
DDRD &= ~((1<<PD3)|(1<<PD2)); //PD2, PD3 inputs

PORTD =0;

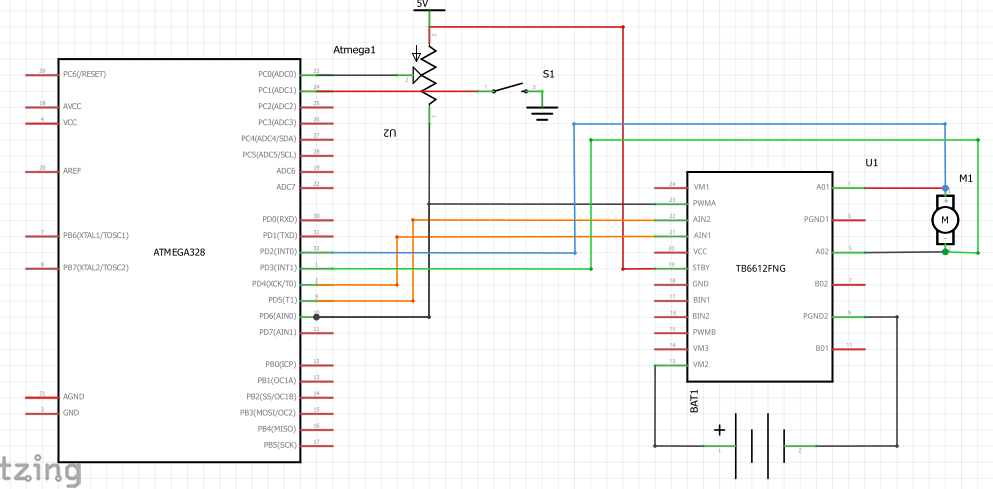
sei(); //global interrupt enabled



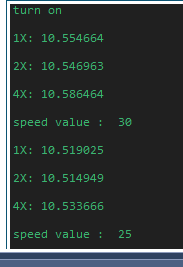
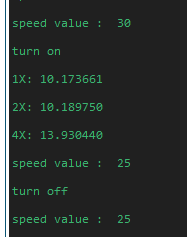
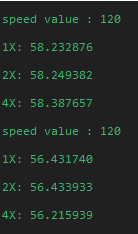


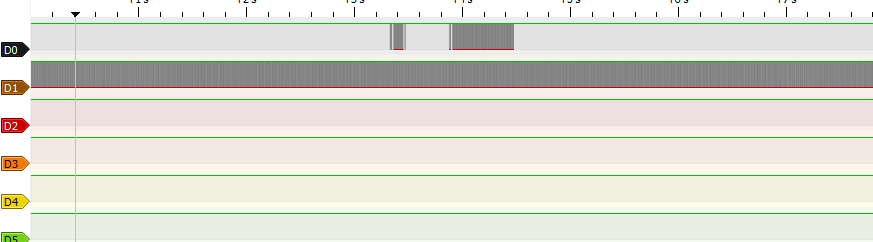


1. **SCHEMATICS**



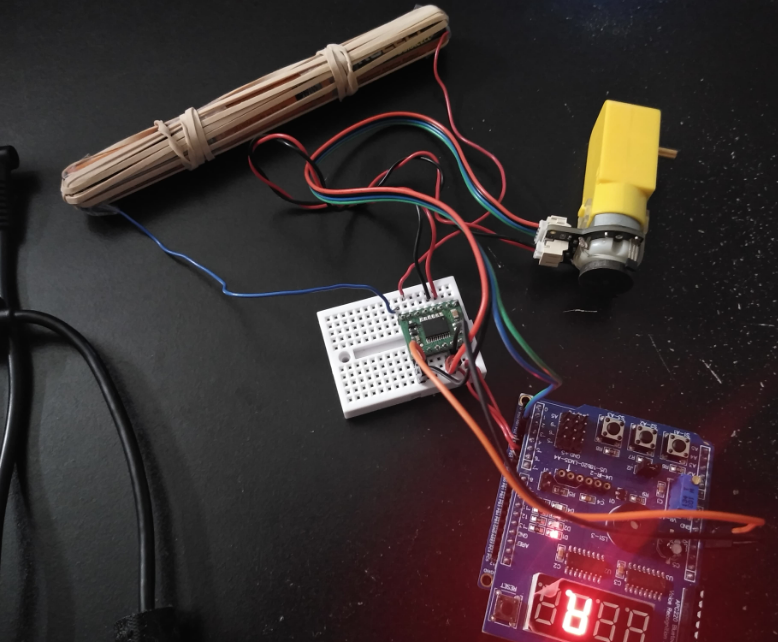
1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

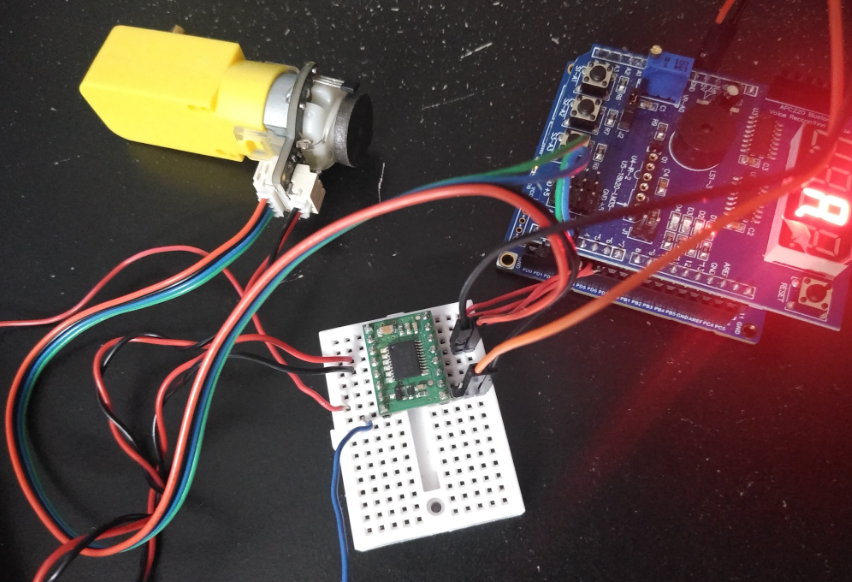




D0 is connected to encoder B, D1 is connected to encoder A. As you can see, encoder B only sometimes work which is why there’s inconsistencies on 4x results. The results given above are the few instances where encoder was working for a while.

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





1. **VIDEO LINKS OF EACH DEMO**

<https://youtu.be/ZLLbIZ0vyJM>

1. **GITHUB LINK OF THIS DA**

<https://github.com/lumbrj1/submission/tree/master/Midterms>

**Student Academic Misconduct Policy**

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

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

John Paulo Lumbres