Monty Sourjah

CPE301 – SPRING 2018

Design Assignment 4

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **SUBMISSION ITEM** | **COMPLETED (Y/N)** | **MARKS**  **(/MAX)** |
| 1 | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS | Y |  |
| 2. | INITIAL CODE OF TASK 1/A | Y |  |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 2/B | Y |  |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 3/C | Y |  |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 4/D | N/A |  |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 5/E | N/A |  |
| 4. | SCHEMATICS | Y |  |
| 5. | SCREENSHOTS OF EACH TASK OUTPUT | N/A |  |
| 5. | SCREENSHOT OF EACH DEMO | Y |  |
| 6. | VIDEO LINKS OF EACH DEMO | Y |  |
| 7. | GOOGLECODE LINK OF THE DA | Y |  |
|  |  |  |  |
|  |  |  |  |

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

List of Components used

Block diagram with pins used in the Atmega328P

Part (a).

Flow Chart:



Atmega 328P Xplained Mini, ULN 2003A,DC Motor, Potentiometer, Push Button, External Power Supply(5V).

PB1 PIN1

PIN16 (-) (+) 5V

ULN 2003A

Atmega 328P

Power Supply5Volts

Push Button

PIN9

PC1 GND PIN8 GND

PC0 Center pin

Left pin Right pin

Potentiometer

Part(b).

Flow Chart:



Atmega 328P Xplained Mini, ULN 2003A, Stepper Motor, Potentiometer, Push Button, External Power Supply(5V,12V),

Push Button Start

PB0 PB1

PB2 PB3 Pin1 Pin2 Pin3 Pin4

Pin16 BLACK Wire

Stepper Motor

ULN2003A

Atmega328P

Pin15 GREEN Wire

Pin14 RED Wire

Pin13 BLUE Wire

YELLW Wire WHITE Wire

PC1 PC2 GND Pin8

VCC

PC0

External Power Supply 5 Volts

Potentiometer 100K

Push Button For Change Direction Of The Stepper Motor

Part ( c ).

Flow Chart:



Components: Atmega 328P Xplained mini,10Kohms potentiometer, Servo motor HB 3001,5V power supply, USB cable.

Vcc VCC

PB1 GND GND

Signal pin

Power Supply 5V

Potentiometer 10Kohms

Servo Motor HB3001

Atmega328P

PC0 PIN2

GND PIN3

VCC(5V) PIN 1

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

(a)

/\*

\* CPE301\_Assignment04\_parta.c

\*

\*/

#define ISDEBUGGING

#ifdef ISDEBUGGING

#include <string.h>

#include <stdio.h>

#define FOSC 16000000 // Clock Speed

#define BAUD 9600

#define MYUBRR FOSC/16/BAUD - 1

volatile char ReceivedChar;

int USART0SendByte(char u8Data);

void USARTSendStr(char\* \_str);

void usart\_init(void);

volatile *uint8\_t* chartoSend;

float temp;

#endif

#define *F\_CPU* 16000000 //16MHz

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdlib.h>

#include <util/delay.h>

#define PWM\_PRESCALLER 64

#define TOP (double)*F\_CPU*/(PWM\_PRESCALLER\*50)-1 // 4999. (or Period\*F\_CPU/PWM\_PRESCALLER - 1, where period is 20ms at 50 Hz)

volatile *uint16\_t* debouncing =0;

volatile *uint16\_t* switched\_on = 0;

volatile *uint16\_t* adc\_value=0;

volatile *uint16\_t* old\_adc\_value=0;

volatile *uint16\_t* interrupt\_counter=0; // used for debouncing puposes;

void enable\_external\_interrupt(void)

{

PCICR = 1<<PCIE1; // Enables PCMSK0 scan

PCMSK1 = 1<<PCINT9; // Triggers interrupt whenever pin 19 changes state

}

void adc\_init(void)

{

ADMUX = 0; // use ADC0

ADMUX |= (1 << REFS0); // use AVcc as the reference. Input will be right justified

ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // 128 pre-scale for 16Mhz

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

ADCSRB = 0; // 0 for free running mode

}

void bounce\_polling(void)

{

unsigned long temp\_interrupt\_counter;

do

{

temp\_interrupt\_counter = interrupt\_counter;

*\_delay\_ms*(500);

} while (temp\_interrupt\_counter !=interrupt\_counter); // If still bouncing the interrupt\_counter will change during the delay

};

int main(void)

{

#ifdef ISDEBUGGING

char printBuffer[128];

#endif

DDRC = 0x0; // Clear all pins;

PORTC |= (1 << PORTC1) ; // Enables the pull-up

DDRB |= (1<<DDB1); // Set pin PB1 328P for output

#ifdef ISDEBUGGING

usart\_init();

#endif

enable\_external\_interrupt();

adc\_init();

// Timer1 setup:

//COM1A1/COM1B1 = None-inverted mode (HIGH at bottom, LOW on match), WGM11=Fast PWM

TCCR1A|=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);

// WGM12 & WGM13=Fast PWM, Prescaler=64

TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS11)|(1<<CS10);

// Set ICR1 register = PWM period

ICR1 = TOP;

sei();

while (1)

{

ADCSRA |= (1 << ADSC); // Start the ADC conversion

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

if(switched\_on)

{

adc\_value=ADCL;

adc\_value = (ADCH<<8) + adc\_value;

if (*abs*(adc\_value-old\_adc\_value)>1)

{

old\_adc\_value=adc\_value;

*uint16\_t* ocr\_value = 0.95\*(double)adc\_value\*((double)TOP)/1024;

OCR1A = ocr\_value;

#ifdef ISDEBUGGING

*sprintf*(printBuffer, "adc\_value %d ocr\_value is %d\n",(int)adc\_value, (int)ocr\_value );

USARTSendStr(printBuffer);

#endif

}

}

else

{

if(old\_adc\_value)

{

OCR1A=0;

old\_adc\_value=0;

}

}

if (debouncing) // wait a little if debouncing

{

bounce\_polling();

debouncing=0;

}

}

}

ISR (PCINT1\_vect)

{

if (bit\_is\_clear(PINC, 1))

{

#ifdef ISDEBUGGING

USARTSendStr("interrupt\n");

#endif

interrupt\_counter++;

if (!debouncing)

{

debouncing=1;

switched\_on = !switched\_on;

}

}

}

#ifdef ISDEBUGGING

void usart\_init()

{

UBRR0H = (MYUBRR) >> 8;

UBRR0L = MYUBRR;

UCSR0B |= (1 << RXEN0) | (1 << TXEN0); // Enable receiver and transmitter

UCSR0B |= (1 << RXCIE0); // Enable receiver interrupt

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop

}

ISR (USART\_RX\_vect)

{

ReceivedChar = UDR0; // Read data from the RX buffer

UDR0 = ReceivedChar; // Write the data to the TX buffer

}

int USART0SendByte(char u8Data)

{

//wait while previous byte is completed

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

// Transmit data

UDR0=u8Data;

return 0;

}

void USARTSendStr(char\* \_str)

{

int thesize=*strlen*(\_str);

for (*uint8\_t* i=0; i<thesize;i++)

{

USART0SendByte(\_str[i]);

}

}

#endif

(b)

/\*

\* CPE301\_Assignment04\_partb.c

\*AVR C program to control the speed of the Stepper Motor using a

\*potentiometer connected to any of the analog-in port. Using a timer in CTC mode to

\*control the delay.

\*/

#define ISDEBUGGING

#ifdef ISDEBUGGING

#include <string.h>

#include <stdio.h>

#define FOSC 16000000 // Clock Speed

#define BAUD 9600

#define MYUBRR FOSC/16/BAUD - 1

volatile char ReceivedChar;

int USART0SendByte(char u8Data);

void USARTSendStr(char\* \_str);

void usart\_init(void);

volatile *uint8\_t* chartoSend;

float temp;

#endif

#define *F\_CPU* 16000000 //16MHz

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdlib.h>

#include <util/delay.h>

#define FORWARD 1

#define REVERSE -1

/\*

For motor PK243-02AA, Stepper Motor:

Step Angle = 1.8º

Number of step per 360º revolution:

TOTALSTEPS = 360 / 1.8 = 200

\*/

#define TOTALSTEPS 200

volatile *uint16\_t* adc\_value=0;

volatile *uint16\_t* old\_adc\_value=0;

volatile *uint8\_t* interrupted =0;

volatile *uint16\_t* steps\_togo = TOTALSTEPS - 1;

volatile *uint16\_t* step = 0;

volatile *uint8\_t* direction = FORWARD;

volatile *uint8\_t* running = 0;

#define STARTSTOPBTN 0

#define REVERSEBTN 1

void adc\_init(void)

{

ADMUX = 0; // use ADC0

ADMUX |= (1 << REFS0); // use AVcc as the reference. Input will be right justified

ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // 128 pre-scale for 16Mhz

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

ADCSRB = 0; // 0 for free running mode

}

void timer1\_init()

{

TCCR1A = 0;

// set up timer with CTC mode and prescaling = 64

TCCR1B |= (1 << WGM12)|(1 << CS11)|(1 << CS10);

// initialize counter

TCNT1 = 0;

/\*

Initialize compare value

With prescalar = 64, the frequency 16,000,000 Hz / 64 -> period = 0.000004 s

TimerCount=Requireddelay/period -1

For example for 0.0020 s delay -> TimerCount = 0.002/0.000004 -1 = 499

Let's consider this as our temporary start value.

\*/

OCR1A = 499; // value for 20 millisecond delay

// enable compare interrupt

TIMSK1 |= (1 << OCIE1A);

}

void debounce\_polling(*uint8\_t* btn)

{

if (btn==STARTSTOPBTN)

{

while (bit\_is\_clear(PINC, 2))

*\_delay\_ms*(20);

}

else if(btn==REVERSEBTN)

{

while (bit\_is\_clear(PINC, 1))

*\_delay\_ms*(20);

}

}

int main(void)

{

DDRC = 0x0; // Clear all pins;

PORTC |= (1 << PORTC1) ; // Enables the pull-up of the reverse direction button

PORTC |= (1 << PORTC2) ; // Enables the pull-up of start stop button

DDRB |= (1<<DDB0) | (1<<DDB1) | (1<<DDB2) | (1<<DDB3); // Set PB0 to PB3 for output

#ifdef ISDEBUGGING

char printBuffer[128];

usart\_init();

#endif

timer1\_init();

adc\_init();

sei();

while (1)

{

ADCSRA |= (1 << ADSC); // Start the ADC conversion

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

if (bit\_is\_clear(PINC,2))

{

debounce\_polling(STARTSTOPBTN);

running=!running;

}

if (bit\_is\_clear(PINC,1))

{

debounce\_polling(REVERSEBTN);

direction = direction == FORWARD ? REVERSE : FORWARD;

steps\_togo = TOTALSTEPS - steps\_togo - 1;

step=0;

}

adc\_value=ADCL;

adc\_value = (ADCH<<8) + adc\_value;

if (*abs*(adc\_value-old\_adc\_value)>1)

{

old\_adc\_value=adc\_value;

/\*

We establish a range between 2 milliseconds and 20 milliseconds delay.

This corresponds to OCR1A values between 499 and 4999 (see above our reasoning)

\*/

*uint16\_t* ocr\_value = 499+(double)adc\_value\*((double)(4999-499)/1024);

TCNT1 = 0;

OCR1A = ocr\_value;

#ifdef ISDEBUGGING

*sprintf*(printBuffer, "adc\_value %d ocr\_value is %d\n",(int)adc\_value, (int)ocr\_value );

USARTSendStr(printBuffer);

#endif

}

}

}

ISR (TIMER1\_COMPA\_vect)

{

#ifdef ISDEBUGGING

//USARTSendStr("interrupt\n"); Not to be used normally because will disturb the established delay.

#endif

if (running && (steps\_togo>0))

{

PORTB &= 0xF0;

switch(step)

{

case 0:

PORTB |= (1 << PORTB0) | (1 << PORTB2);

break;

case 1:

PORTB |= (1 << PORTB1) | (1 << PORTB2);

break;

case 2:

PORTB |= (1 << PORTB1) | (1 << PORTB3);

break;

case 3:

PORTB |= (1 << PORTB3) | (1 << PORTB0) ;

break;

default:

break;

}

if (direction==FORWARD) step++;

else step--;

step = step % 4;

steps\_togo--;

}

}

#ifdef ISDEBUGGING

void usart\_init()

{

UBRR0H = (MYUBRR) >> 8;

UBRR0L = MYUBRR;

UCSR0B |= (1 << RXEN0) | (1 << TXEN0); // Enable receiver and transmitter

UCSR0B |= (1 << RXCIE0); // Enable receiver interrupt

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop

}

ISR (USART\_RX\_vect)

{

ReceivedChar = UDR0; // Read data from the RX buffer

UDR0 = ReceivedChar; // Write the data to the TX buffer

}

int USART0SendByte(char u8Data)

{

//wait while previous byte is completed

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

// Transmit data

UDR0=u8Data;

return 0;

}

void USARTSendStr(char\* \_str)

{

int thesize=*strlen*(\_str);

for (*uint8\_t* i=0; i<thesize;i++)

{

USART0SendByte(\_str[i]);

}

}

#endif

(c)

/\*

\* CPE301\_Assignment04\_partc.c

\*C program to control the position of the Servo Motor using a

\*potentiometer connected to PC0 the analog-in port. When pot value is 0 the servo is

\*at position 0 deg. and when pot value is max (approx. 5V) the servo is at position 180

\*degrees

\*/

#define ISDEBUGGING

#ifdef ISDEBUGGING

#include <string.h>

#include <avr/interrupt.h>

#include <stdio.h>

#endif

#define *F\_CPU* 16000000 //16MHz

#include <avr/io.h>

#include <util/delay.h>

#include <stdlib.h>

#define PWM\_PRESCALLER 64

#define TOP (double)*F\_CPU*/(PWM\_PRESCALLER\*50)-1; // 4999. (or Period\*F\_CPU/PWM\_PRESCALLER - 1, where period is 20ms at 50 Hz)

//#define PULSE\_MIN 800

//#define PULSE\_MAX 2200

#define PULSE\_MIN 544 // According to data sheet minimum pulse width is 800

#define PULSE\_MAX 2200 // According to data sheet maximum pulse with is 2200

volatile unsigned long adc\_value=0;

volatile unsigned long old\_ocr\_value=0;

#ifdef ISDEBUGGING

#define FOSC 16000000 // Clock Speed

#define BAUD 9600

#define MYUBRR FOSC/16/BAUD - 1

volatile char ReceivedChar;

int USART0SendByte(char u8Data);

void usart\_init(void);

char printBuffer[128];

volatile *uint8\_t* chartoSend;

float temp;

#endif

void adc\_init(void)

{

ADMUX = 0; // use ADC0

ADMUX |= (1 << REFS0); // use AVcc as the reference. Input will be right justified

ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // 128 pre-scale for 16Mhz

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

ADCSRB = 0; // 0 for free running mode

}

int main(void)

{

adc\_init();

#ifdef ISDEBUGGING

usart\_init();

#endif

DDRB |= (1<<DDB1); // Set pin PB1 on Atmega328P Xplainedmini as output

// Timer1 setup:

//COM1A1/COM1B1 = None-inverted mode (HIGH at bottom, LOW on match), WGM11=Fast PWM

TCCR1A|=(1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);

// WGM12 & WGM13=Fast PWM, Prescaler=64

TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS11)|(1<<CS10);

// Set ICR1 register = PWM period

ICR1 = TOP; //Period = 20 ms

while (1)

{

ADCSRA |= (1 << ADSC); // Start the ADC conversion

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

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

adc\_value=ADCL;

adc\_value = (ADCH<<8) + adc\_value;

unsigned long pulse = PULSE\_MIN+(double)adc\_value\*(PULSE\_MAX-PULSE\_MIN)/1024;

unsigned long ocr\_value=(double)pulse \* 16/64 -1;

#ifdef ISDEBUGGING

*sprintf*(printBuffer, "adc\_value=%d ocr\_value is %d\n", (int)adc\_value, (int)ocr\_value);

int thesize=*strlen*(printBuffer);

for (*uint8\_t* i=0; i<thesize;i++)

{

USART0SendByte(printBuffer[i]);

}

#endif

if (*abs*(ocr\_value-old\_ocr\_value)>1)

{

old\_ocr\_value=ocr\_value;

OCR1A = ocr\_value;

}

*\_delay\_ms*(1000);

}

}

#ifdef ISDEBUGGING

void usart\_init()

{

UBRR0H = (MYUBRR) >> 8;

UBRR0L = MYUBRR;

UCSR0B |= (1 << RXEN0) | (1 << TXEN0); // Enable receiver and transmitter

UCSR0B |= (1 << RXCIE0); // Enable receiver interrupt

UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00); // Set frame: 8data, 1 stop

sei();

}

ISR (USART\_RX\_vect)

{

ReceivedChar = UDR0; // Read data from the RX buffer

UDR0 = ReceivedChar; // Write the data to the TX buffer

}

int USART0SendByte(char u8Data)

{

//wait while previous byte is completed

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

// Transmit data

//UDR0 = u8Data;

UDR0=u8Data;

return 0;

}

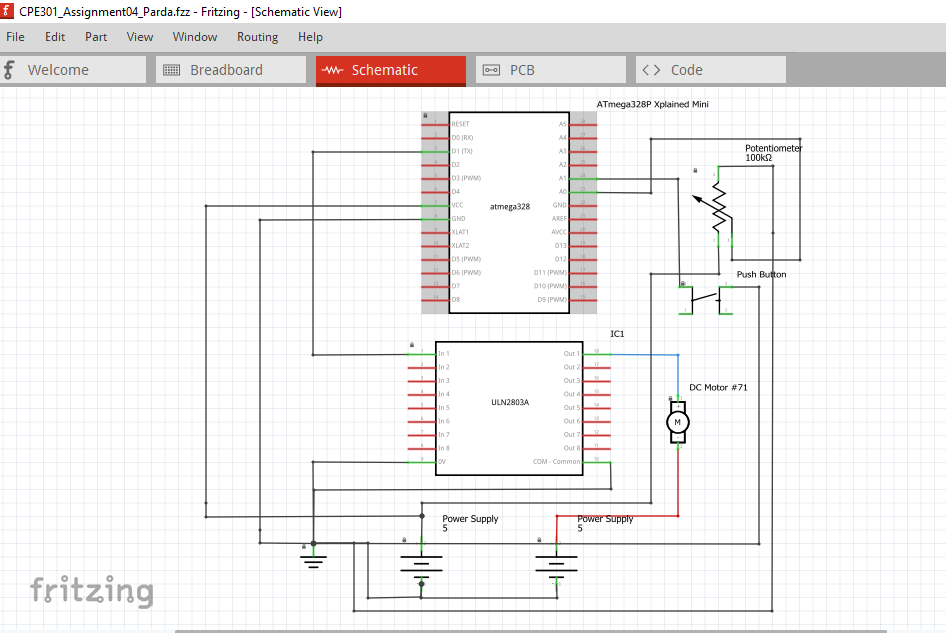
#endif

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

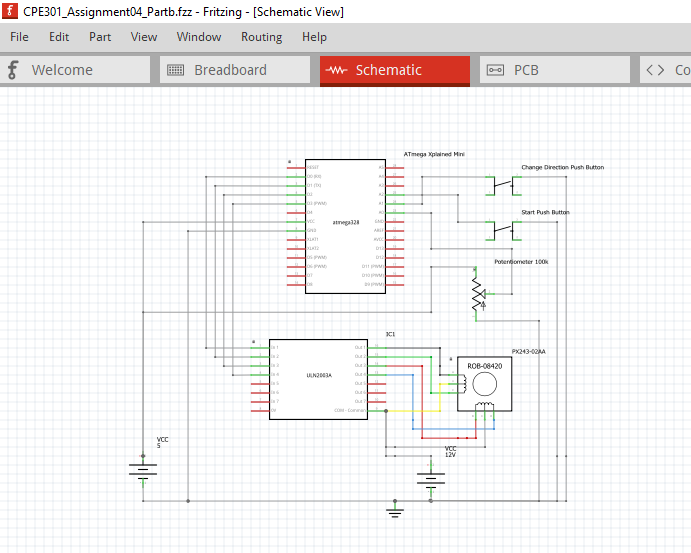
Insert only the modified sections here. Use more sections if needed

1. **SCHEMATICS**

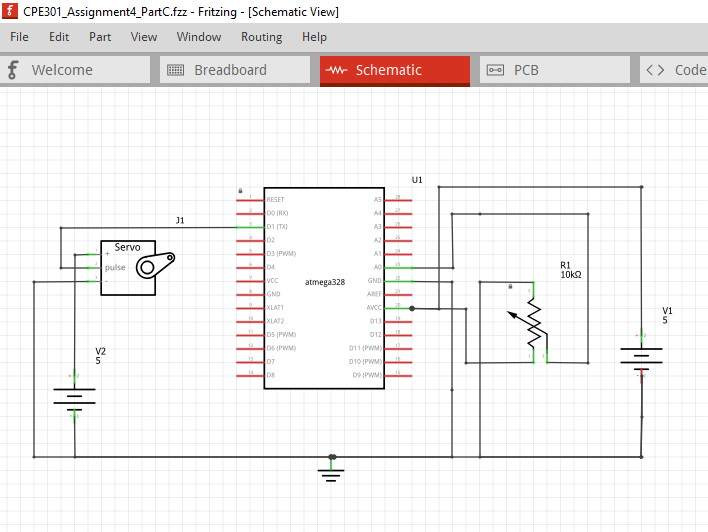
Part (a)



Part (b)



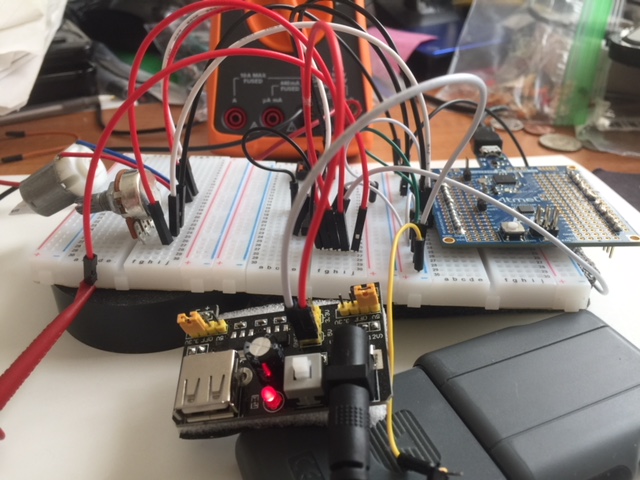
Part ( c )



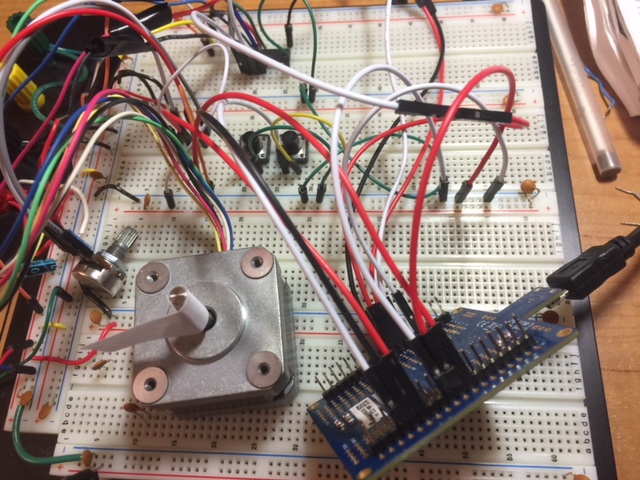
1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**

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

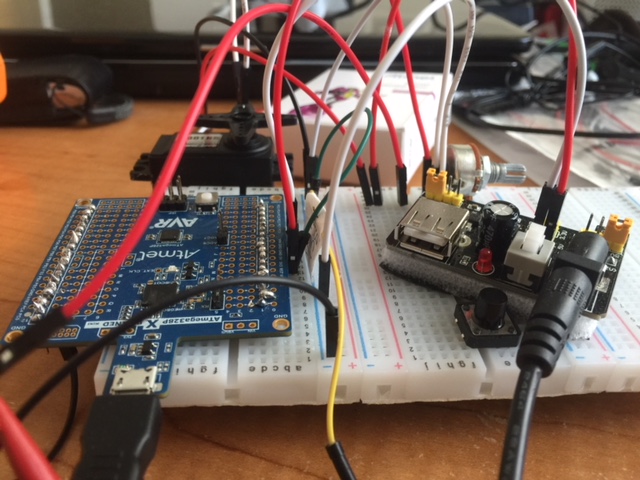
(a).



(b).



(c).



1. **VIDEO LINKS OF EACH DEMO**

(a). <http://www.youtube.com/watch?v=FQTe3TlKlUI>

(b). <http://www.youtube.com/watch?v=6k05rImSFt8>

(c). <http://www.youtube.com/watch?v=eVDuFqtkX0w>

1. **GITHUB LINK OF THIS DA**

<https://github.com/ballasl/CPE301_Assignment4>

**Student Academic Misconduct Policy**

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

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

Monty Sourjah.