CPE301 – SPRING 2019

Design Assignment DA4b

Student Name: Daniel Senda
Student #: 5002362633
Student Email: sendad1@unlv.nevada.edu
Primary Github address: https://github.com/dsenda/Smiles
Directory: DA4b

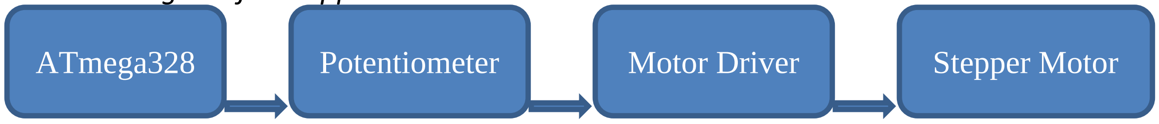
Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

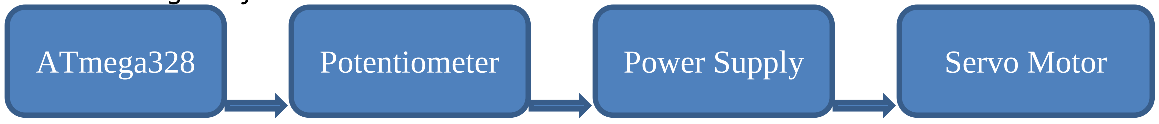
1. COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS

List of Components used:
Atmel Studio 7
ATmega328P Xplained Mini
Motor driver module (TB6612FNG)
Stepper motor
Servo motor
Multi-function shield
Potentiometer (Multi-function shield)
Breadboard
Power Supply

Block diagram about the transmission connections:
1st block diagram for stepper motor



2nd block diagram for servo motor



2. INITIAL/MODIFIED CODE OF TASK DA4a

The following is the code is the initial code of DA4a. This code was used as a starting point for the codes developed for DA4b.

```
// c_code_dc_motor_on_off_duty_cycle.c
// Daniel Senda
// Turn DC motor on and off with pushbutton and
// also changes speed with change of potentiometer.

#define       F_CPU 16000000UL
#include      <avr/io.h>
#include      <util/delay.h>
#include      <avr/interrupt.h>

// Declares global variables.
int motor = 0;       // On_Off variable for motor.
int poten = 0;       // Potentiometer variable.

// Declares function.
void adc_init(void);

int main(void)
{
       DDRD = 0b00100000;   // Sets PD5 as an output.
```

```
        DDRC = 0b00000000;   // Sets all pins in PORTC as inputs.
        TCCR2A = 0x00;       // Sets normal mode on timer0.
        TCCR2B = 0x01;       // Sets no pre-scaling.
        TIMSK2 = 0x01;       // Enables timer0 overflow interrupt.
        sei ();              // Enables global interrupts.

        TCCR0A = 0b10100011;//Sets Fast PWM.
        TCCR0B = 0x01;       // Sets no pre-scaling.
        OCR0B = 0xCC;        // Sets 80% or 20% duty cycle.
        adc_init();          // Initializes ADC.

        while (1)
        {
                ADCSRA|=(1<<ADSC);                   // Starts conversion.
                while((ADCSRA&(1<<ADIF))==0);        // Waits for conversion to finish.
                ADCSRA |= (1<<ADIF);                 // Resets conversion finished flag.
                poten = ADCL;                        // Records potentiometer data.
                poten = poten | (ADCH<<8);           // Data calculations.
                poten = (poten/1024.0) * 5000/10;
                OCR0B = poten;                       // Sets OCR0B to poten value.
        }
}

ISR (TIMER2_OVF_vect)                // timer0 overflow interrupt vector
{
        if ((PINC & 0b10) == 0)      // If pushbutton is pressed.
        {
                motor ^= 1;                  // Toggles motor variable.
                if (motor == 1)              // Turns motor on.
                {
                        TCCR0A = 0b00100011;
                }
                else                         // Turns motor off.
                {
                        TCCR0A = 0b00000011;
                }
                _delay_ms(500);              // Delay for de-bouncing.
        }
}

void adc_init (void) // Sets up and enables 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).
        (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 Pre-scaler Select Bits.
        (0<<ADPS1)|
        (1<<ADPS0);
}
```

3. DEVELOPED MODIFIED CODE OF TASK DA4a

The following code controls the speed of the stepper motor using a potentiometer.

```c
// c_code_stepper_motor_pot_speed.c
// Daniel Senda
// This code rotates the stepper motor clockwise.
// The speed of the motor is determined by a potentiometer.

#include      <avr/io.h>

// Declares global variables.
int speed = 0;              // Speed variable for stepper motor.
int poten = 0;              // Potentiometer variable.

// Declares function.
void adc_init(void);        // Initializes the ADC.
void speed_calc(void);      // Calculates the speed of stepper motor.

int main(void)
{
       DDRB = 0b00111100;   // Sets PB2-PB5 as outputs.
       DDRC = 0b00000000;   // Sets all pins in PORTC as inputs.
       TCCR1A = 0x00;       // Sets normal mode on timer1.
       TCCR1B = 0b011;      // Sets pre-scaler of 64.
       adc_init();          // Initializes ADC.
       while (1)
       {
              PORTB = 0b101000;           // Sequence that turns the stepper
              TCNT1 = 0;                  // motor in the clockwise direction.
              while(TCNT1 < speed);
              PORTB = 0b011000;
              TCNT1 = 0;                  // Resets timer1.
              while(TCNT1 < speed);       // Waits for timer to reach speed time.
              speed_calc();               // Recalculates speed halfway through.
              PORTB = 0b010100;
              TCNT1 = 0;
              while(TCNT1 < speed);
              PORTB = 0b100100;
              TCNT1 = 0;
              while(TCNT1 < speed);
              speed_calc();               // Recalculates speed at end of cycle.
       }
}

void adc_init (void) // Sets up and enables 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).
       (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 Pre-scaler Select Bits.
```

```
        (0<<ADPS1)|
        (1<<ADPS0);
}

void speed_calc(void)
{
        ADCSRA|=(1<<ADSC);                  // Starts conversion.
        while((ADCSRA&(1<<ADIF))==0);       // Waits for conversion to finish.
        ADCSRA |= (1<<ADIF);                // Resets conversion finished flag.
        poten = ADCL;                       // Records potentiometer data.
        poten = poten | (ADCH<<8);          // Data calculations.
        poten = (poten/1024.0) * 5000/10;
        if (poten >= 255)
        {
                speed = 1249;         // 5ms
        }
        else if (poten >= 200)
        {
                speed = 2499;         // 10ms
        }
        else if (poten >= 150)
        {
                speed = 9999;         // 40ms
        }
        else if (poten >= 100)
        {
                speed = 17499;        // 70ms
        }
        else if (poten >= 50)
        {
                speed = 24999;        // 100ms
        }
        else if (poten >= 0)
        {
                speed = 32499;        // 130ms
        }
}
```

The following code controls the position of the servo motor using a potentiometer.

```
// c_code_servo_motor_pot_0_180.c
// Daniel Senda
// This codes rotates the servo motor to 180 degrees when the
// potentiometer is at high value, and rotates to 0 degrees when
// the potentiometer is at low value.

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

// Declares global variables.
int poten = 0;              // Potentiometer variable.

// Declares function.
void adc_init(void);        // Initializes the ADC.
void poten_calc(void);      // Calculates the value of potentiometer.

int main(void)
```

```
{
       DDRC = 0b00000000;           // Sets all pins in PORTC as inputs. (Poten. Input)
       DDRD = 0b01100000;           // Sets PB5 and PB6 as outputs. (PWM output)
       TCCR0A = 0b10000011;         // Sets Compare1 on non-inverting and Fast PWM.
       TCCR0B = 0b101;              // Sets pre-scaler of 1024.
       adc_init();                  // Initializes ADC.
       while (1)
       {
              poten_calc();         // Recalculates the value of potentiometer.
       // If statements that determine position of servo motor based on pot. value.
              if (poten >= 255)     // If pot. value high do this:
              {
                     OCR0A = 34;           // Sets pulse width for 180 degree position.
              }
              else if (poten <= 200)       // If pot. value low do this:
              {
                     OCR0A = 8;            // Sets pulse width for 0 degree position.
              }
       }
}

void adc_init (void)        // Sets up and enables 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).
       (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 Pre-scaler Select Bits.
       (0<<ADPS1)|
       (1<<ADPS0);
}

void poten_calc(void)
{
       ADCSRA|=(1<<ADSC);                  // Starts conversion.
       while((ADCSRA&(1<<ADIF))==0);       // Waits for conversion to finish.
       ADCSRA |= (1<<ADIF);                // Resets conversion finished flag.
       poten = ADCL;                       // Records potentiometer data.
       poten = poten | (ADCH<<8);          // Data calculations.
       poten = (poten/1024.0) * 5000/10;
}
```

4. SCHEMATICS

The following schematics from the slides describe the connections that are made relatively well.

Stepper Motor Schematic:

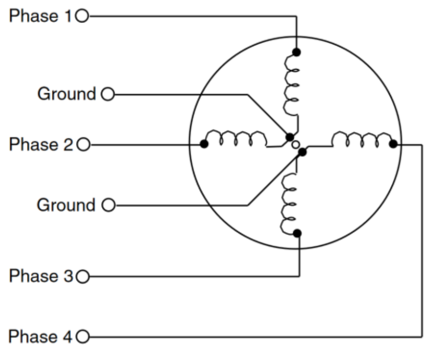
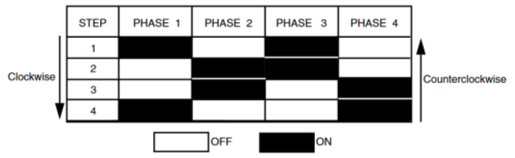


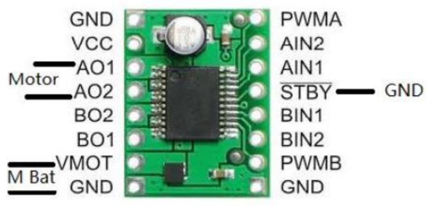
FIGURE 19.3 **The wiring diagram of the unipolar stepper. The common connections can be separate or combined.**



STEP	PHASE 1	PHASE 2	PHASE 3	PHASE 4
1	■	□	■	□
2	□	■	■	□
3	□	■	□	■
4	■	□	□	■

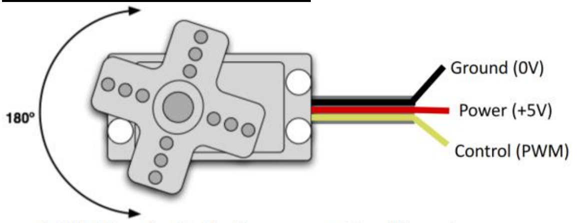
Clockwise
Counterclockwise

□ OFF ■ ON



- **Vmotor** to 12V (red wire)
- **Vcc** to 5V (orange wire)
- **GND** to ground
- **AIN2** to Digital 4
- **AIN1** to Digital 5
- **BIN1** to Digital 6
- **BIN2** to Digital 7
- **PWMA** and **PWMB** to Vcc (orange wire)

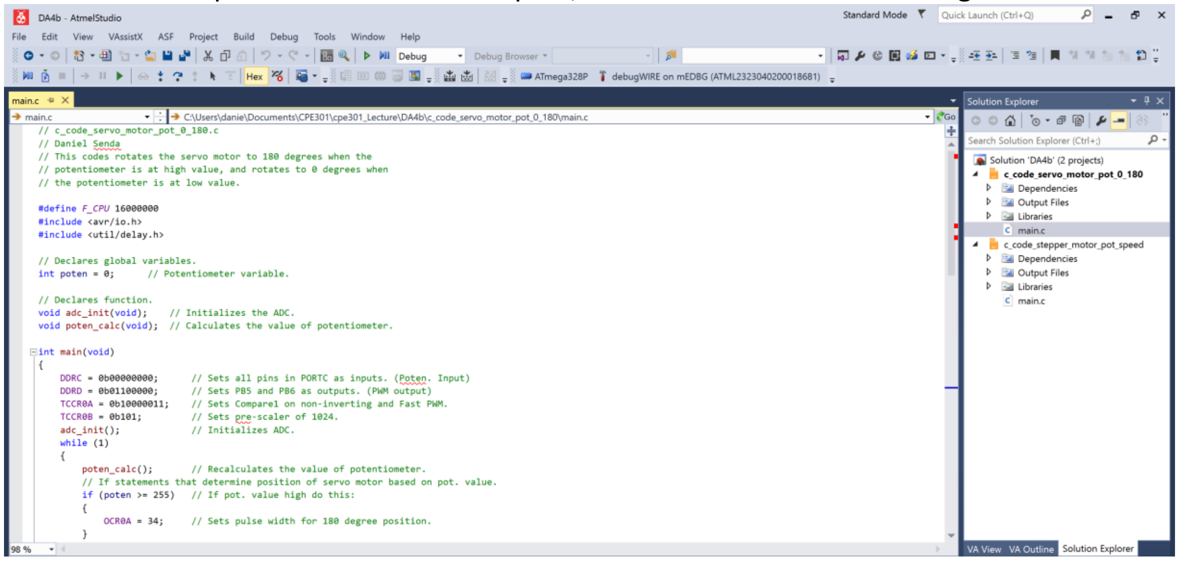
Servo motor schematic:



- PWM freq is 50 Hz (i.e. every 20 millisecs)
- Pulse width ranges from 1 to 2 millisecs
  - 1 millisec = full anti-clockwise position
  - 2 millisec = full clockwise position

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

There are no required Atmel studio outputs, but below is an included image of Atmel.


DA4b - AtmelStudio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Debug
ATmega328P
debugWIRE on mEDBG (ATML2323040200018681)
main.c
Solution Explorer
Solution 'DA4b' (2 projects)
c_code_servo_motor_pot_0_180
Dependencies
Output Files
Libraries
main.c
c_code_stepper_motor_pot_speed
Dependencies
Output Files
Libraries
main.c
98 %
VA View VA Outline Solution Explorer

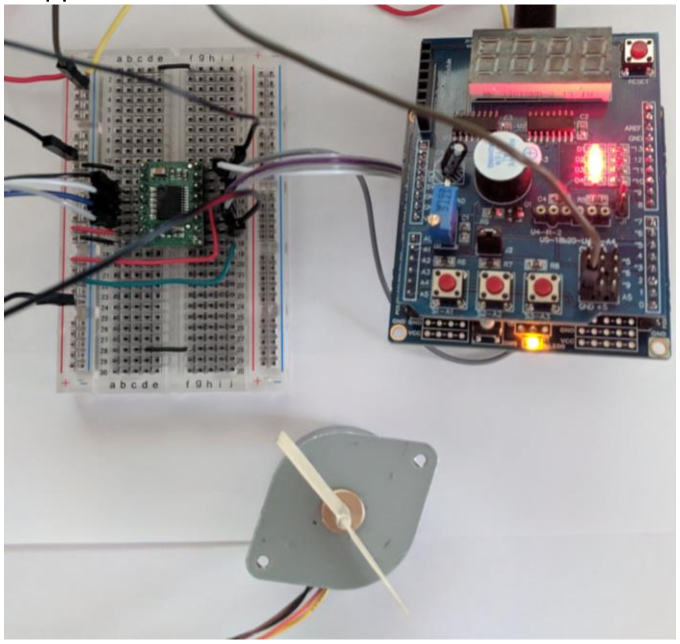
C:\Users\danie\Documents\CPE301\cpe301_Lecture\DA4b\c_code_servo_motor_pot_0_180\main.c

```c
// c_code_servo_motor_pot_0_180.c
// Daniel Senda
// This codes rotates the servo motor to 180 degrees when the
// potentiometer is at high value, and rotates to 0 degrees when
// the potentiometer is at low value.

#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

// Declares global variables.
int poten = 0;       // Potentiometer variable.

// Declares function.
void adc_init(void);    // Initializes the ADC.
void poten_calc(void);  // Calculates the value of potentiometer.

int main(void)
{
    DDRC = 0b00000000;      // Sets all pins in PORTC as inputs. (Poten. Input)
    DDRD = 0b01100000;      // Sets PB5 and PB6 as outputs. (PWM output)
    TCCR0A = 0b10000011;    // Sets Compare1 on non-inverting and Fast PWM.
    TCCR0B = 0b101;         // Sets pre-scaler of 1024.
    adc_init();             // Initializes ADC.
    while (1)
    {
        poten_calc();       // Recalculates the value of potentiometer.
        // If statements that determine position of servo motor based on pot. value.
        if (poten >= 255)   // If pot. value high do this:
        {
            OCR0A = 34;     // Sets pulse width for 180 degree position.
        }
```

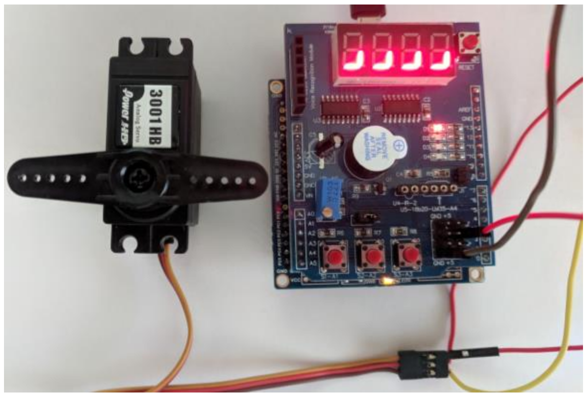
6. SCREENSHOT OF EACH DEMO (BOARD SETUP)

The following are pictures of the board setups:

Stepper Motor:



Servo Motor:



7. VIDEO LINKS OF EACH DEMO

Stepper Motor: https://youtu.be/_k2gOAOCpP4

Servo Motor: https://youtu.be/Pfu2lwGuyv0

8. GITHUB LINK OF THIS DA

https://github.com/dsenda/Smiles/tree/master/DA4b

Student Academic Misconduct Policy

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

"This assignment submission is my own, original work".

Daniel Senda