Class:	CPE301L Digital Systems Architecture and Design 1001		Semester:	Spring 2025
Points		Document author:	Narek Kalikian	
		Author's email:	kalikn1@unlv.nevada.edu	
		Document topic:	Postlab 9	
Instructor's comments:				

1. Introduction / Theory of Operation

In this lab, we will be interfacing sensors and motors using the available peripherals on our ATmega328p microcontrollers. Specifically, our lab experiments will consist of implementing the distance sensor, temperature sensor, DC motor, and stepper motor. The distance sensor is good in applications that require measurements between moving or stationary objects. The (LM34) temperature sensor uses an output voltage that is linearly proportional to degrees Fahrenheit to accurately and easily measure temperature. The DC motor is used when high speed is preferred over torque and can be categorized as unidirectional or bidirectional. Finally, the stepper motor works directly opposite of the DC motor as it sacrifices speed for high torque levels.

2. Prelab Content

1. Vref for LM34 Temperature Sensor:

- Vout = (T)(10 mV) → 10 mV/°F scaling factor
- 2^n → 10 bit = 2^10 = 1024 levels
- Vref = (scaling factor)(n levels) → (10*10^-3)(1024) → **Vref = 10.24 V**
- The ATmega328p has a supply voltage range of 1.8 V - 5.5 V. The calculated Vref falls outside this range, so no, **it is not feasible.**

2. ADC Configuration:

- ADMUX = 0x45 → 01000101
- ADMUX[7:6] = 01 → **AVCC as reference voltage**
- ADMUX[5] = 0 → **Result is right-adjusted**
- ADMUX[4:0] = 00101 = 5 → **ADC5 input channel**
- ADCSRA = 0x8F → 10001111
- ADCSRA[7] = 1 → **ADC enabled**
- ADCSRA[3] = 1 → **ADC interrupts enabled**
- ADCSRA[2:0] = 111 = 7 → 2^7 → **Prescaler = 128**

3. L298N Pins:

- Pin 1 (OUT1): Output 1 for motor A

- Pin 2 (OUT2): Output 2 for motor A
- Pin 3 (VS): Supply voltage for motors
- Pin 4 (GND): Ground pin
- Pin 5 (IN1): Input 1 for controlling motor A
- Pin 6 (IN2): Input 2 for controlling motor A
- Pin 7 (ENABLEA): Enable for motor A, can control speed of motor
- Pin 8 (VSS): Logic supply voltage, powers internal logic circuit
- Pin 9 (OUT3): Output 3 for motor B
- Pin 10 (OUT4): Output 4 for motor B
- Pin 11 (IN3): Input 3 for controlling motor B
- Pin 12 (IN4): Input 5 for controlling motor B
- Pin 13 (ENABLEB): Enable for motor B, can control speed of motor
- Pin 14 (CS B): Current sensing output for motor B
- Pin 15 (CS A): Current sensing output for motor A

4. ULN2003 Pins:

- Pin 1 (IN1): Input 1 control signal for 1st Darlington pair
- Pin 2 (IN2): Input 2 control signal for 2nd Darlington pair
- Pin 3 (IN3): Input 3 control signal for 3rd Darlington pair
- Pin 4 (IN4): Input 4 control signal for 4th Darlington pair
- Pin 5 (IN5): Input 5 control signal for 5th Darlington pair
- Pin 6 (IN6): Input 6 control signal for 6th Darlington pair
- Pin 7 (IN7): Input 7 control signal for 7th Darlington pair
- Pin 8 (GND): Ground pin
- Pin 9 (VCC/COM): Common pin connected to positive supply of load
- Pin 10 (OUT7): Output 7 for 7th Darlington pair
- Pin 11 (OUT6): Output 6 for 6th Darlington pair
- Pin 12 (OUT5): Output 5 for 5th Darlington pair
- Pin 13 (OUT4): Output 4 for 4th Darlington pair
- Pin 14 (OUT3): Output 3 for 3rd Darlington pair
- Pin 15 (OUT2): Output 2 for 2nd Darlington pair
- Pin 16 (OUT1): Output 1 for 1st Darlington pair

5. HC-SR05 Ultrasonic Sonar Distance Sensor: The HC-SR04 is an ultrasonic sensor that can measure distances from 2 centimeters to 4 meters. It emits ultrasonic sound waves and measures the time it takes for the waves to be reflected back to the sensor after coming in contact with an object to determine the distance between the sensor and object. The sensor consists of a transmitter that emits the waves and a receiver that detects the waves once reflected. The calculation of the distance is done by multiplying the time it takes for the reflected waves to return multiplied by the speed of sound and then divided by two because the time accounts for the distance to the object and back to the sensor.

6. LM34 Temperature Sensor: The LM34 is a standard temperature sensor that measures temperature in Fahrenheit. It outputs an analog voltage value that is proportional to degrees in Fahrenheit with a scale factor of 10 mV per degree. The LM34 has 3 pins, including one for output voltage, one for supply voltage, and one for ground. This sensor is an easy and effective way to measure temperatures quickly, especially when being used with microcontrollers.

3. Description of Experiments

Experiment 1 - Code:

```
#include <avr/io.h>
#include <util/delay.h>

#define F_CPU 16000000UL // Define CPU frequency

// Pin definitions
#define SIG_PIN PB0      // SIG pin connected to PB0
#define BUZZER_PIN PB1   // Buzzer connected to PB1

void setup() {
    // Set SIG pin as output initially
    DDRB |= (1 << SIG_PIN); // SIG pin as output
    DDRB |= (1 << BUZZER_PIN); // Buzzer pin as output
}

void trigger_sensor() {
    // Trigger sensor with a 5 µs HIGH pulse
    PORTB &= ~(1 << SIG_PIN); // Set SIG pin LOW
    _delay_us(2);
    PORTB |= (1 << SIG_PIN);  // Set SIG pin HIGH
    _delay_us(5);
    PORTB &= ~(1 << SIG_PIN); // Set SIG pin LOW
}

uint16_t measure_echo_time() {
    uint16_t echo_time;

    DDRB &= ~(1 << SIG_PIN); // Set SIG pin as input to read the echo pulse

    while (!(PINB & (1 << SIG_PIN))); // Wait until SIG pin goes HIGH

    // Start Timer1
    TCCR1A = 0x00; // Normal mode
    TCCR1B = 0x02; // Prescaler = 8
    TCNT1 = 0;     // Reset counter

    while (PINB & (1 << SIG_PIN)); // Wait until SIG pin goes LOW

    TCCR1B = 0x00; // Disable timer
    echo_time = TCNT1; // Read counter value

    return echo_time;
}
```

```
float calculate_distance(uint16_t echo_time) {
    // Calculate distance in cm
    return (float)echo_time / 58.0;
}

int main(void) {
    setup();

    while (1) {
        // Trigger the sensor
        trigger_sensor();

        // Measure the echo time
        uint16_t echo_time = measure_echo_time();

        // Calculate the distance
        float distance = calculate_distance(echo_time);

        // Check if distance is less than 7 inches (17.78 cm)
        if (distance < 17.78) {
            // Turn on the buzzer
            PORTB |= (1 << BUZZER_PIN);
            } else {
            // Turn off the buzzer
            PORTB &= ~(1 << BUZZER_PIN);
        }

        // Wait for 1 millisecond before next iteration
        _delay_ms(1);
    }
```

```
Output
Show output from: Build
    Task "RunCompilerTask"
        Shell Utils Path C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils
        C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils\make.exe all --jobs 16 --output-sync
        make: Nothing to be done for 'all'.
    Done executing task "RunCompilerTask".
    Task "RunOutputFileVerifyTask"
                Program Memory Usage     :   898 bytes   2.7 % Full
                Data Memory Usage        :   0 bytes   0.0 % Full
                Warning: Memory Usage estimation may not be accurate if there are sections other than .text sections in ELF file
    Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "Lab9.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "C:\Users\Narek\Documents\Atmel Studio\7.0\Lab9\Lab9\Lab9.cproj" (entry point):
Done building target "Build" in project "Lab9.cproj".
Done building project "Lab9.cproj".

Build succeeded.
```

Experiment 2 - Code:

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define F_CPU 16000000UL

// Port definitions
#define LCD_DATA_PORT PORTD
#define LCD_CTRL_PORT PORTB
#define RS PB0
#define EN PB1

void lcd_cmd(unsigned char cmd);
void lcd_data(unsigned char data);
void lcd_init();
void lcd_set_cursor(uint8_t row, uint8_t col);
void lcd_clear();
void lcd_string(char *str);

void view(uint16_t temp) {
    // Extract tens and units digits of the temperature
    lcd_data((0x30 + ((temp / 10) % 10))); // Tens place
    lcd_data((0x30 + (temp % 10)));       // Units place
}

ISR(ADC_vect) {
    uint16_t adc_value = ADC; // Read ADC value
    float vref = 5.0;         // Reference voltage
    uint16_t temperature;

    // Calculate temperature in °F
    temperature = (uint16_t)((adc_value * vref * 100) / 1024);

    // Set cursor to beginning of line 1
    lcd_set_cursor(0, 0);

    // Display "T:"
    lcd_string("T: ");

    // Display temperature
    view(temperature);

    // Display "F"
    lcd_string(" F");

    // Restart ADC conversion
    ADCSRA |= (1 << ADSC);
}
```

```c
void setup() {
    // Configure ADC
    ADMUX = (1 << REFS0); // Use AVCC as reference voltage
    ADCSRA = (1 << ADEN) | (1 << ADIE) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Enable ADC, interrupt, and prescaler of 128

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

    // Initialize LCD
    lcd_init();

    // Enable global interrupts
    sei();
}

int main(void) {
    setup();

    while (1) {
        // Main loop does nothing; everything is handled in the ISR
    }
}

// LCD Functions
void lcd_cmd(unsigned char cmd) {
    LCD_DATA_PORT = cmd;
    LCD_CTRL_PORT &= ~(1 << RS); // RS = 0 for command
    LCD_CTRL_PORT |= (1 << EN);  // EN = 1
    _delay_us(1);
    LCD_CTRL_PORT &= ~(1 << EN); // EN = 0
    _delay_us(100);
}

void lcd_data(unsigned char data) {
    LCD_DATA_PORT = data;
    LCD_CTRL_PORT |= (1 << RS);  // RS = 1 for data
    LCD_CTRL_PORT |= (1 << EN);  // EN = 1
    _delay_us(1);
    LCD_CTRL_PORT &= ~(1 << EN); // EN = 0
    _delay_us(100);
}

void lcd_init() {
    lcd_cmd(0x38); // 8-bit mode, 2 lines, 5x7 dots
    lcd_cmd(0x0C); // Display ON, Cursor OFF
    lcd_cmd(0x06); // Increment cursor
    lcd_cmd(0x01); // Clear display
    _delay_ms(2);
}
```

```
void lcd_set_cursor(uint8_t row, uint8_t col) {
    if (row == 0) {
        lcd_cmd(0x80 + col); // Line 1
        } else {
        lcd_cmd(0xC0 + col); // Line 2
    }
}

void lcd_clear() {
    lcd_cmd(0x01); // Clear display
    _delay_ms(2);
}

void lcd_string(char *str) {
    while (*str) {
        lcd_data(*str++);
    }
}
```

```
Output
Show output from: Build
    Task "RunCompilerTask"
        Shell Utils Path C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils
        C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils\make.exe all --jobs 16 --output-sync
        make: Nothing to be done for 'all'.
    Done executing task "RunCompilerTask".
    Task "RunOutputFileVerifyTask"
                Program Memory Usage     :   1290 bytes   3.9 % Full
                Data Memory Usage        :   8 bytes   0.4 % Full
                Warning: Memory Usage estimation may not be accurate if there are sections other than .text sections in ELF file
    Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "Lab9.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "C:\Users\Narek\Documents\Atmel Studio\7.0\Lab9\Lab9\Lab9.cproj" (entry point):
Done building target "Build" in project "Lab9.cproj".
Done building project "Lab9.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Experiment 3 - Code:

```
#include <avr/io.h>
#include <util/delay.h>

#define F_CPU 16000000UL

// Initialize ADC
void adc_init() {
    ADMUX = (1 << REFS0); // Use AVCC as ref voltage
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // Enable ADC and prescaler of 128
}

// Read ADC value
uint16_t adc_read(uint8_t channel) {
    ADMUX = (ADMUX & 0xF0) | (channel & 0x0F); // Select ADC channel
    ADCSRA |= (1 << ADSC); // Start conversion
    while (ADCSRA & (1 << ADSC)); // Wait for conversion to complete
    return ADC; // Return ADC value
}

// Initialize PWM with Timer0
void pwm_init() {
    TCCR0A = (1 << COM0A1) | (1 << WGM01) | (1 << WGM00); // Non-inverting mode
    TCCR0B = (1 << CS01) | (1 << CS00); // Prescaler = 64
    DDRD |= (1 << PORTD6); // Set OC0A (PD6) as output
}

int main(void) {
    uint16_t adc_value;
    uint8_t pwm_duty_cycle;

    // Initialize ADC and PWM
    adc_init();
    pwm_init();

    while (1) {
        // Read ADC value from potentiometer
        adc_value = adc_read(0);

        // Map ADC value to PWM duty cycle
        pwm_duty_cycle = (uint8_t)((adc_value * 255) / 1023);

        // Set duty cycle
        OCR0A = pwm_duty_cycle;
```

```
Output
Show output from: Build
Done building project "Lab9.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Experiment 5 - Code:

```c
#define F_CPU 16000000UL
// Initialize Timer1 for PWM
void pwm_init() {
    // Set Timer1 in Fast PWM mode with ICR1 as TOP
    TCCR1A = (1 << COM1A1) | (1 << WGM11); // Non-inverting mode
    TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11); // Prescaler = 8
    ICR1 = 19999; // Set TOP value for 50 Hz PWM
    DDRB |= (1 << PORTB1); //Set OC1A as output
}

// Move servo to a specific angle
void set_servo_angle(uint16_t angle) {
    if (angle > 180) angle = 180; // Limit angle to 180
    uint16_t ocr_value = ((angle * (4999 - 2499)) / 180) + 2499; // Calculate OCR1A value
    OCR1A = ocr_value; // Set OCR1A to control servo position
}

// Read DIP switch input
uint16_t read_dip_switch() {
    // PC0-PC3 as inputs
    DDRC &= ~((1 << PORTC0) | (1 << PORTC1) | (1 << PORTC2) | (1 << PORTC3));
    PORTC |= ((1 << PORTC0) | (1 << PORTC1) | (1 << PORTC2) | (1 << PORTC3)); // Enable pull-up resistors

    // Read input from DIP switch
    uint16_t dip_value = PINC & 0x0F; // Read lower 4 bits of PORTC
    return dip_value;
}

int main(void) {
    // Initialize PWM
    pwm_init();
    while (1) {
        // Read DIP switch hex input
        uint16_t hex_input = read_dip_switch();

        // Convert hex input to degrees
        uint16_t angle = hex_input * 30; // Map 0x0-0xF to 0°-180°

        // Set servo to the angle
        set_servo_angle(angle);

        // Small delay
        _delay_ms(500);
```

```
Output
Show output from: Build
Done building project "Lab9.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

4. Questions and Answers

1. Signal Conditioning with Sensors: Signal conditioning is very important when working with sensors because it prepares the sensor for the next stage of processing by manipulating one or multiple of its signals. This is also important for further measurement and control of the sensor's raw output.

2. LM34 vs LM35 Temperature Sensors: The two temperature sensors are very similar in look and functionality. However, the primary difference between the two is their output, specifically the voltage scale. LM35 temperature sensors operate with an output voltage that is proportional to degrees Celsius. LM34 temperature sensors, on the other hand, operate with an output voltage that is proportional to degrees Fahrenheit.

3. Linear Scale for a Sensor: Linear scale refers to the direct proportional relationship between the physical measurement (temperature, distance, light, etc.) and the sensor's electrical/digital output (voltage, current, etc.).

5. Conclusions

This lab taught us how to interface various peripherals, such as sensors and motors, with our ATmega328p microcontroller. This lab was interesting because we had previous experience interfacing these peripherals with an Arduino microcontroller. We were curious to see how the processes would differ with an AVR microcontroller instead. However, the experiments themselves were quite difficult, much more so than Arduino-based programming. We struggled to get outputs of our code on the microcontroller, LCD, etc., because our board was crashing every time we tried to run any of the experiment's code on it.