



University of Sri Jayewardenepura

Faculty of Technology


Bachelor of Engineering Technology Honors

Department of Materials and Mechanical Technology

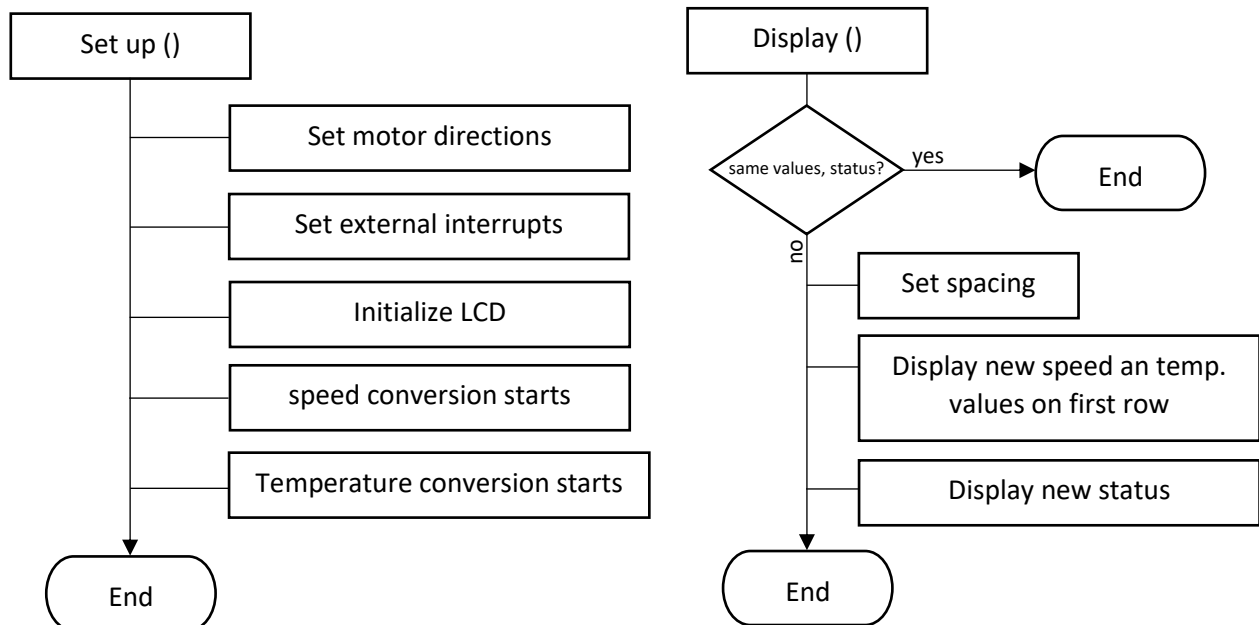
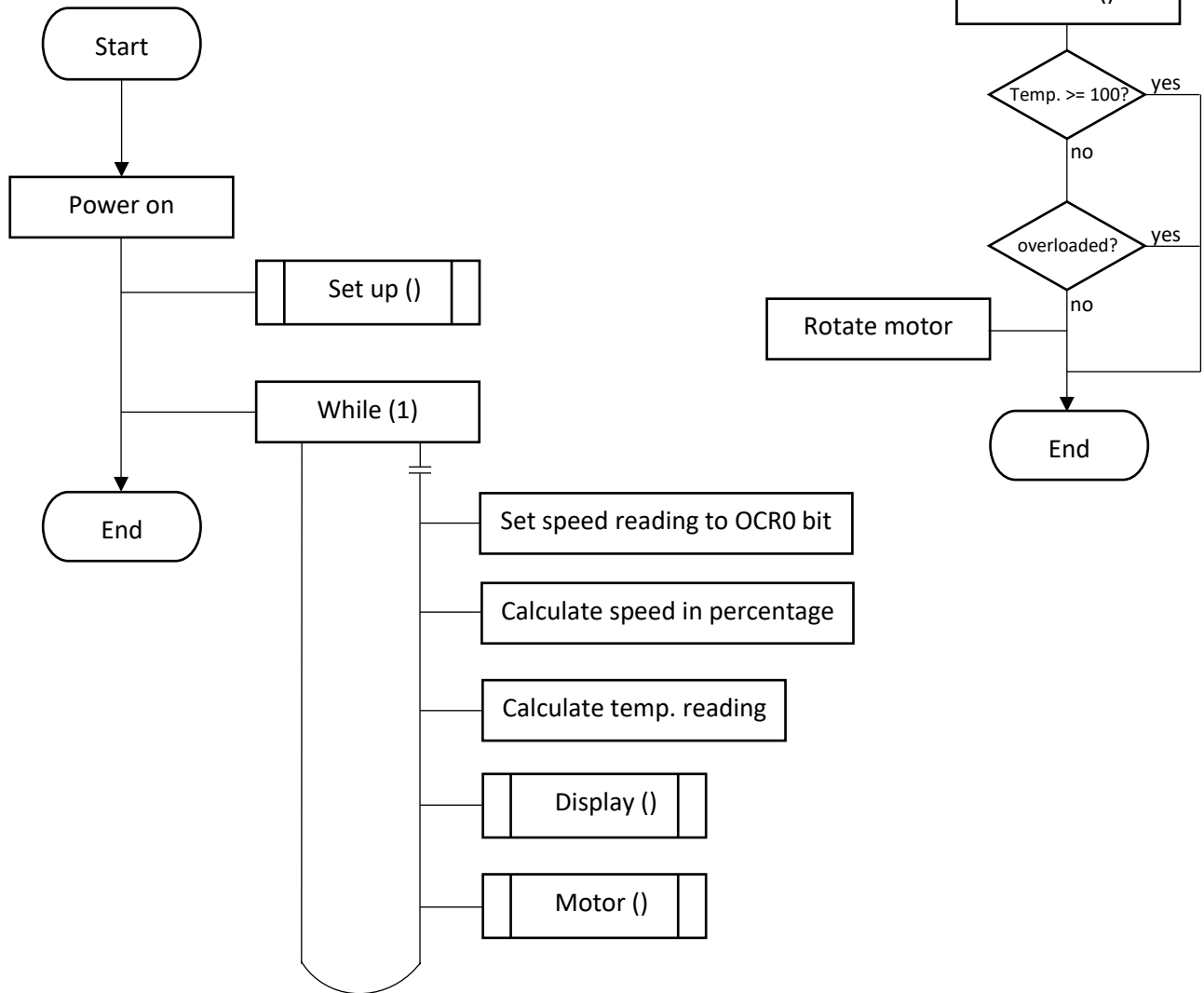
ETM 2082 –Embedded Systems and Applications

Assignment-1

(Design of an embedded system for a DC motor control application)

Name	P. D. KOSHALA CHATHURANGA
Index No	EGT - 19418
Focus Area	MECHATRONICS TECHNOLOGY
Date of Submission	10 TH JUNE 2022
Student's (electronic) Signature (I declare that this assignment is my own work)	

1. FLOW CHART



2. PIN SELECTION

- PORTC – LCD DATA INPUTS
- PD5, PD6, PD7 – LCD COMMAND PINS (RS, RW and E respectively)
- PA0/ADC0 – SPEED ADC VALUE INPUT
- PA1/ADC1 – TEMPERATURE SENSOR ADC VALUE INPUT
- PB0 – H-BRIDGE INPUT 1
- PB1 – H-BRIDGE INPUT 2
- PB3/OC0 – H-BRIDGE ENABLE INPUT / PWM PIN
- PB2/INT2 – EXTERNAL INTERRUPT

3. CODE

```
#define F_CPU 1000000UL /*CPU Frequency 1MHz */
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define LCD_Data_Dir DDRC          /* Define LCD data port direction to PORTB */
#define LCD_Command_Dir DDRD      /* Define LCD command port direction register to PORTC*/
#define LCD_Data_Port PORTC       /* Define LCD data port */
#define LCD_Command_Port PORTD    /* Define LCD data port */
#define RS PD5                    /* Define Register Select pin */
#define RW PD6                    /* Define Read/Write signal pin */
#define EN PD7                    /* Define Enable signal pin */

#define M_INT1 PB0                // Define INT1 pin of H-BRIDGE
#define M_INT2 PB1                // Define INT2 pin of H-BRIDGE
#define M_ENA PB3                 // Define ENA pin of H-BRIDGE

/*-----Display control - START -----*/

void LCD_Command(unsigned char cmd)
{
    LCD_Data_Port = cmd;
    LCD_Command_Port &= ~(1<<RS); /* RS=0 command reg. */
    LCD_Command_Port &= ~(1<<RW); /* RW=0 Write operation */
    LCD_Command_Port |= (1<<EN); /* Enable pulse */
    _delay_us(1);
    LCD_Command_Port &= ~(1<<EN);
    _delay_ms(3);
}
```

```

}

void LCD_Char (unsigned char char_data) /* LCD data write function */
{
    LCD_Data_Port= char_data;
    LCD_Command_Port |= (1<<RS); /* RS=1 Data reg. */
    LCD_Command_Port &= ~(1<<RW); /* RW=0 write operation */
    LCD_Command_Port |= (1<<EN); /* Enable Pulse */
    _delay_us(1);
    LCD_Command_Port &= ~(1<<EN);
    _delay_ms(1);
}

void LCD_Init (void) /* LCD Initialize function */
{
    LCD_Command_Dir = 0xFF; /* Make LCD command port direction as o/p */
    LCD_Data_Dir = 0xFF; /* Make LCD data port direction as o/p */
    _delay_ms(20); /* LCD Power ON delay always >15ms */
    LCD_Command (0x38); /* Initialization of 16X2 LCD in 8bit mode */
    LCD_Command (0x0C); /* Display ON Cursor OFF */
    LCD_Command (0x06); /* Auto Increment cursor */
    LCD_Command (0x01); /* Clear display */
    LCD_Command (0x80); /* Cursor at home position */
}

void LCD_String (char *str) /* Send string to LCD function */
{
    int i;
    for(i=0;str[i]!=0;i++) /* Send each char of string till the NULL */
    {
        LCD_Char (str[i]);
    }
}

void LCD_String_xy (char row, char pos, char *str)/* Send string to LCD with xy position */
{
    if (row == 0 && pos<16)
        LCD_Command((pos & 0x0F)|0x80); /* Command of first row and required position<16 */
    else if (row == 1 && pos<16)
        LCD_Command((pos & 0x0F)|0xC0); /* Command of first row and required position<16 */
    LCD_String(str); /* Call LCD string function */
}

void LCD_Clear()
{
    LCD_Command (0x01); /* clear display */
    LCD_Command (0x80); /* cursor at home position */
}

//-----

int pre_speed;
int pre_temp;
int pre_load;
char STATUS[20];

```

```

char space[5];

void Display(int line1, int line2, int line3) {

    if( (pre_speed != line1) || (pre_temp != line2) || (pre_load != line3)){

        if (line1 < 10){//this is to put a space in between speed and temp values
            char spc[5]= " ";
            strcpy(space,spc);
        }else if (line1 < 100 ){
            char spc[5]= " ";
            strcpy(space,spc);
        } else {
            char spc[5]= " ";
            strcpy(space,spc);
        }

        char STSA[20]= "STS: RUNNING";
        char STSB[20]= "STS: OVERLOAD";
        char STSC[20]= "STS: OVERTEMP";
        char str0[10]= "SPD:";
        char line1_text[16];

        char strA[16]= "TMP:";
        char strB[5]= "C";
        char line1_text2[16];

        char final_speed[20];
        char final_temp[20];

        sprintf(line1_text, "%d", line1);
        sprintf(line1_text2, "%d", line2);

        strcat(line1_text,space);
        strcat(str0,line1_text);
        strcpy(final_speed,str0);

        strcat(line1_text2,strB);
        strcat(strA,line1_text2);
        strcpy(final_temp,strA);

        strcat(final_speed,final_temp);

        if( pre_temp != line2 ){                // sets status to over-temperature..
            if (line2 >= 100 ){
                strcpy(STATUS,STSC);
            }else{
                strcpy(STATUS,STSA);
            }
        }

        if ( pre_load != line3 ){                // sets status to over-load..
            if (line3 == 1 ){
                strcpy(STATUS,STSB);
            }else{

```

```

        strcpy(STATUS,STSA);
    }
}

LCD_Command (0x01);
LCD_String(final_speed);           //display line one.
LCD_String_xy(1,0,STATUS);         //display line two.
}

    pre_speed = line1;
    pre_temp = line2;
    pre_load = line3;
}

/*----- Display control - END -----*/

/*----- ADC control - START -----*/

int setupADC_speed(void)           //A0 pin analog reading
{
    ADMUX = ((1 << REFS0) | (1 << REFS1) | (1 << ADLAR));
    ADCSRA = (1 << ADEN) | (1 << ADSC);
    while(ADCSRA & (1 << ADSC));
    ADCSRA|= (1<<ADIF);
    return ADCH;
}

int setupADC_temp(void)           //A1 pin analog reading
{
    ADMUX = ((1 << REFS0) | (1 << REFS1) | (1 << MUX0) | (1 << ADLAR));
    ADCSRA = (1 << ADEN) | (1 << ADSC);
    while(ADCSRA & (1 << ADSC));
    ADCSRA|= (1<<ADIF);
    return ADCH;
}

/*----- ADC control - END -----*/

/*----- MOTOR control - START -----*/

void motor(int mode_value, int temp_value, int direction_mode) {
    if (mode_value == 1)
    {
        if (temp_value >= 100){
            PORTB = ((1 << M_INT1) | (1 << M_INT2) | (1 << M_ENA));
        }else if (direction_mode == 1){
            PORTB = ((1 << M_INT1) | (1 << M_ENA));
        }else if(direction_mode == 2){
            PORTB = ((1 << M_INT2) | (1 << M_ENA));
        }
    }else if (mode_value == 2){
        PORTB = ((1 << M_INT1) | (1 << M_INT2) | (1 << M_ENA));
    }
}

```

```

    }
}

/*----- MOTOR control - END -----*/

/*----- MAIN FUNCTION - START -----*/

void setup(){

    DDRB = ((1 << M_INT1) | (1 << M_INT2) | (1 << M_ENA)); // set direction to H-BRIDGE

    TCCR0 = (1<<WGM00) | (1<<COM01) | (1<<CS00);
    DDRB |= (0 << PB2);
    PORTB |= (1 << PB2);
    GICR = (1 << INT2);
    MCUCSR &= ~(1 << ISC2);
    sei();

    LCD_Init(); // Initialize LCD */
    LCD_String("ASSIGNMENT ONE"); // write string on 1st line of LCD*/
    LCD_Command(0xC0); // Go to 2nd line*/
    LCD_String("EGT19418"); // Write string on 2nd line*/
    _delay_ms(1000);
    LCD_Clear();

    setupADC_speed(); // speed conversion starts*/
    setupADC_temp(); // Temperature conversion starts*/

}

int overload = 0;
volatile uint8_t Direct = 0;
int direction = 0;
int temperature= 0;

int main(void)
{
    setup(); // Initiating the setup conditions..

    int SPEED_result= 0;
    int speed = 0;

    while(1){

        if (setupADC_speed()>=250) // this makes the ADC error reduced..
        {
            speed = 255;
        }else{
            speed = setupADC_speed();
        }

        OCR0 = speed;

        direction = 1; // motor is rotating in clockwise direction.

        SPEED_result = setupADC_speed()/2.48;
        temperature= setupADC_temp();
    }
}

```

```

Display(SPEED_result,temperature,overload);
// calling the Display_line_one() to display speed, temp. and status in LCD.

if (Direct !=0){
    motor(2,temperature, direction);
    overload = 1;
    _delay_ms(3000);
}else{
    motor(1,temperature, direction);
    overload = 0;
}

}

}

ISR(INT2_vect){

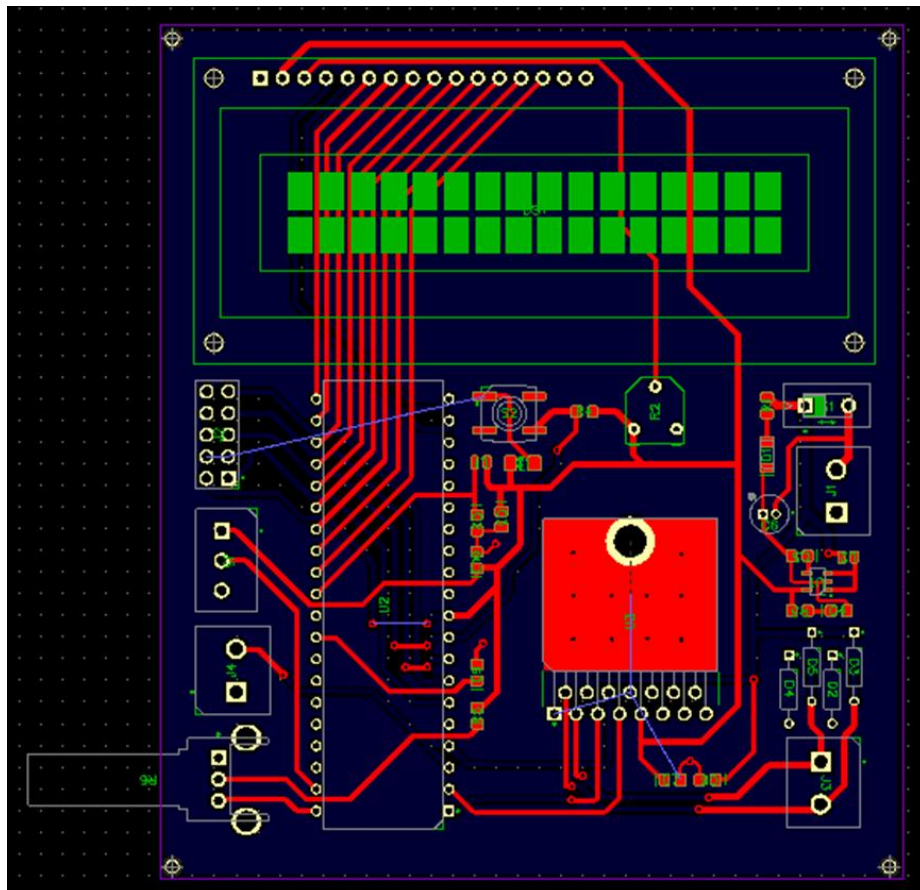
    Direct = ~Direct;

}

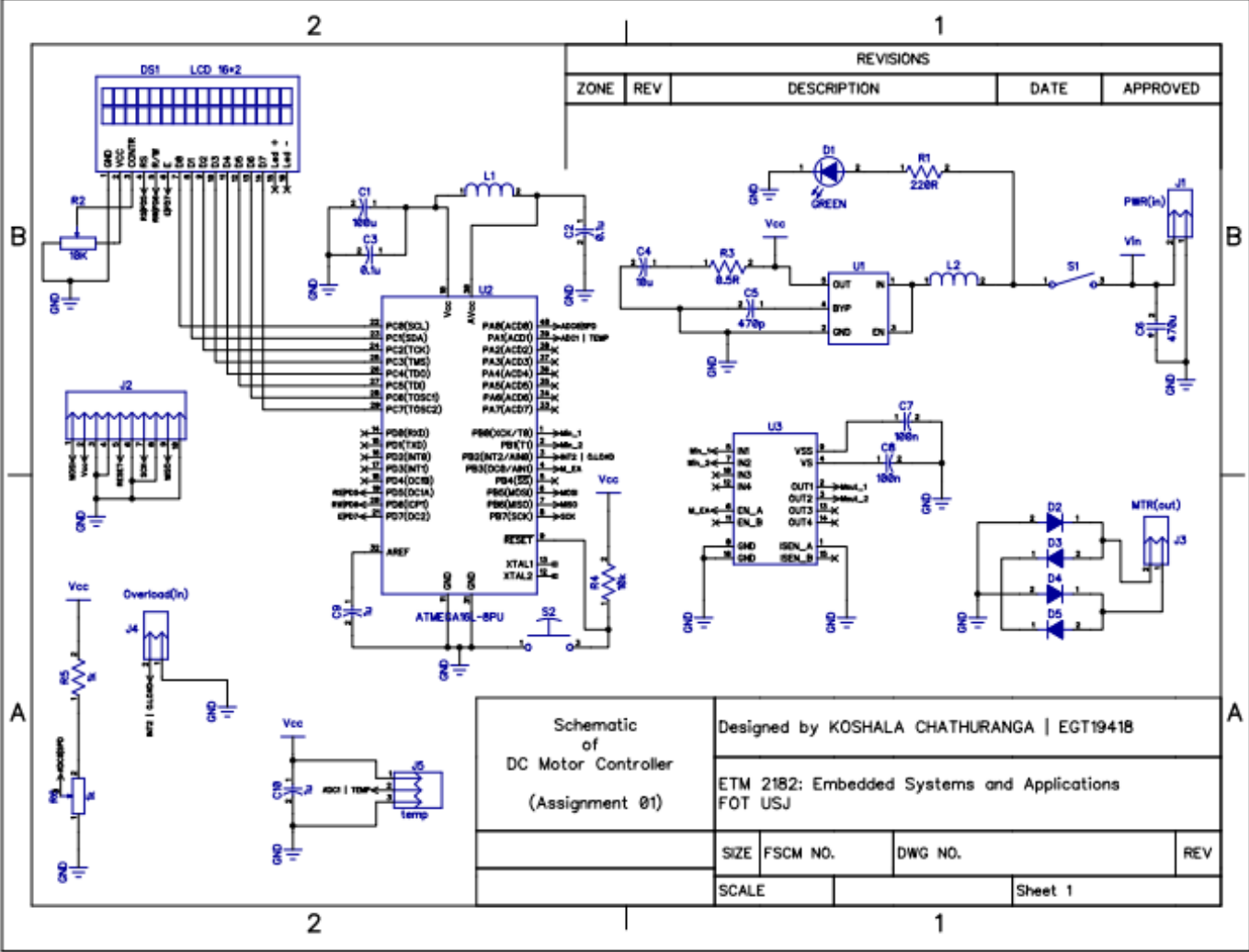
/*----- MAIN FUNCTION - END -----*/

```

4. PCB LAYOUT



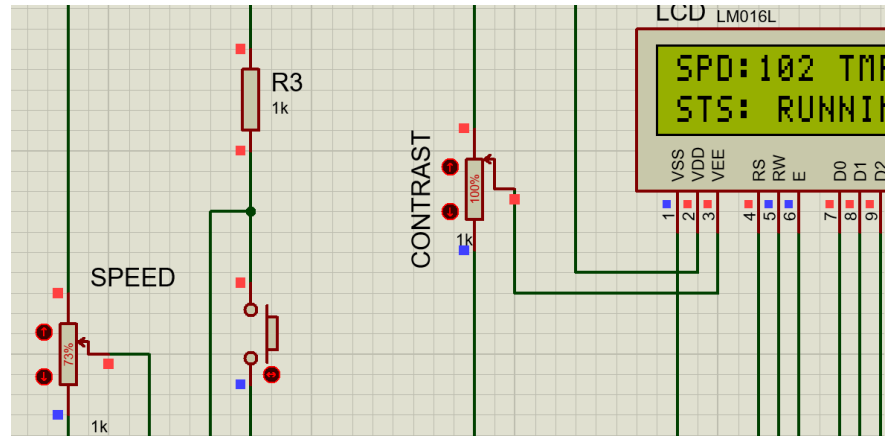
5. SCHEMATIC DIAGRAM



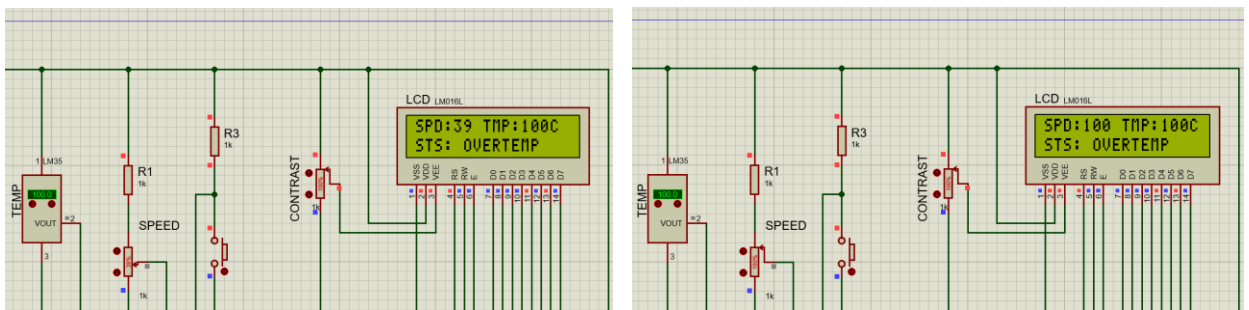
6. ERRORS MET WITH.

1. ADC reading of the speed-controller is not the same with what appears on display.

- RES-POT was connected in series with 1k resistor and the formula was adjusted accordingly.



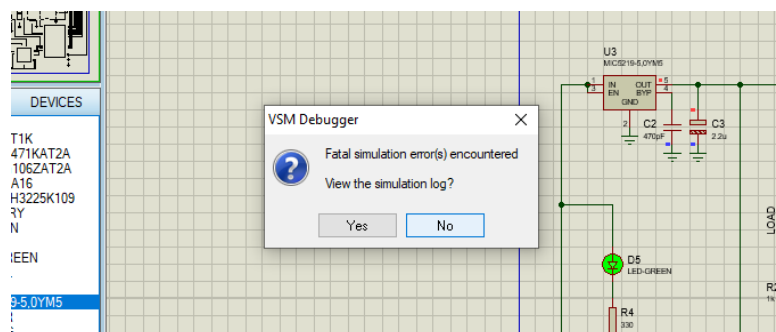
2. Once speed value hits 100, or changes from one digit to three digits, the temperature readings were changing its positioning or get removed from the row.



- Code was changed to vary the space in between speed value and temp values, as the first speed value character's quantity changes.

3. There was a simulation error keep getting.

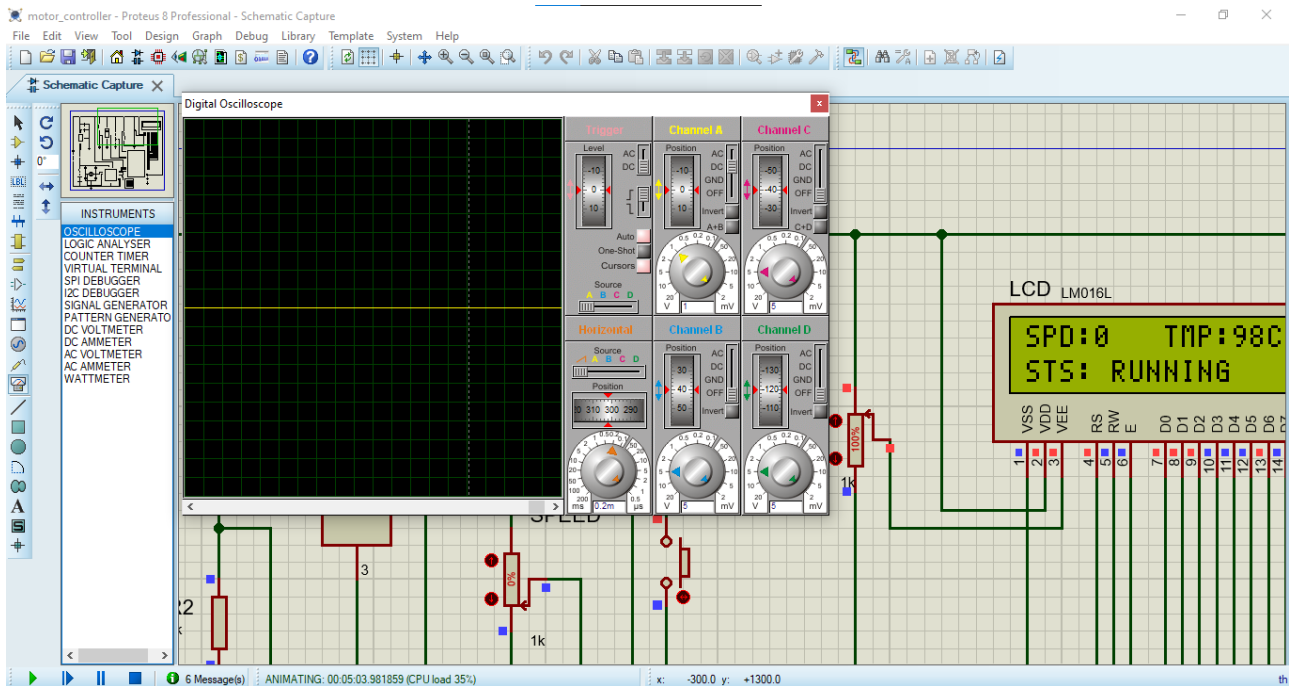
- Toggled the settings of the voltage regulator.



4. External interrupt for the overload did not work.
 - Correct registers were used as mentioned in the datasheet of the ATmega16L-8PU.
 - Toggling switch was used as an externally pulled low.
 - A new variable was used in the ISR to toggle the overload status and volatile data type was selected.
5. Even the code was fully functioned, slight change in proteus made all the readings wrong.
 - Re-drawn the entire proteus simulation layout and reduced the wiring as much as I noticed.
6. Schematic drawn in Diptrace showed a verification error.
 - All the joints and wires re-labeled.
7. PCB drawing in the Diptrace also showed so many errors.
 - Copper pouring was applied to the bottom layer and connected all the GNDs to copper pour with through holes.

7. PWM SIGNALS

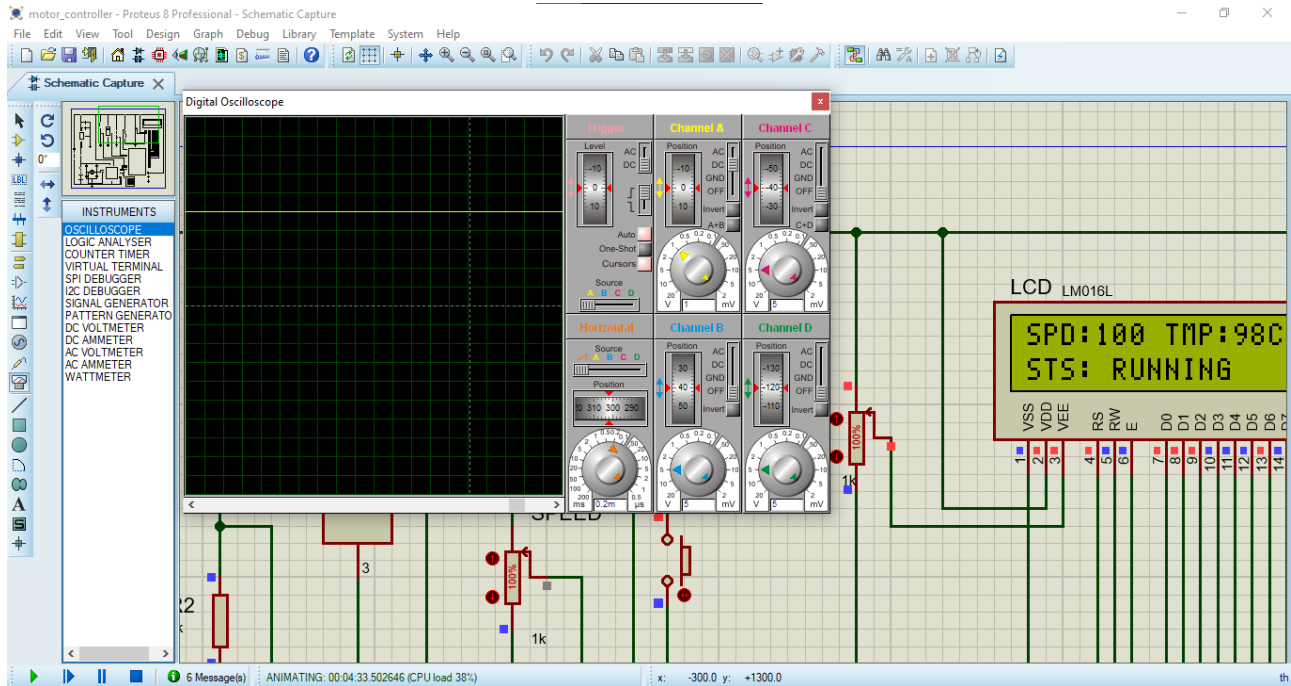
- 0 percent



-
- The screenshot shows the Proteus 8 Professional Schematic Capture window. The circuit includes a microcontroller (PIC16F877A), a 10k resistor, a 1k resistor, and an LCD LM016L. The digital oscilloscope is configured with four channels (A, B, C, D) and displays a square wave on Channel A. The LCD displays 'SPD:25 TMP:98C STS: RUNNING'.

-
- The screenshot displays the Proteus 8 Professional Schematic Capture interface. The main workspace shows a digital circuit schematic with a microcontroller, an LCD LM016L, and various peripheral components. A digital oscilloscope is overlaid on the schematic, displaying four channels of waveforms. The oscilloscope settings are visible, including Level, Position, AC/DC coupling, and Source. The LCD screen displays 'SPD:50 TMP:98C' and 'STS: RUNNING'. The status bar at the bottom indicates 'ANIMATING: 00:05:18.336279 (CPU load 66%)'.

- 100 percent



8. REFERENCES

- Lecture notes
- <https://www.electronicwings.com/>