Microprocessors Course Project Report
2rd Year Computer Engineering

Project Title: Smart Fan

Team ID: T_11

Team Members

SN	Student Name	Section Number
1	Mohamed Hossam El-Din Osman	3
2	Omar El-Sayed Ahmed Hussien	2
3	Abdullah Eid El-Sayed Ammar	2
4	Abdulrahman Helal Saleh	2
5	Omar Hamdi Fathi	2

Table of Contents

1. Project Objective

The project is called Smart Fan, which operates in two different modes.

The first mode is manual, where users have the ability to choose from four different fan speeds. By activating this mode, users can personally adjust the speed according to their preference.

The second mode is automatic, in which the fan speed is determined by the temperature using a temperature sensor. When this mode is activated, the fan behaves as follows: if the temperature is below 20°C, the fan remains off; if the temperature is below 40°C, the fan operates at the first speed; if the temperature is below 60°C, the fan operates at the second speed; if the temperature is below 80°C, the fan operates at the third speed; and if the temperature is below 100°C, the fan operates at the fourth speed.

Additionally, there is a timer mode that can be activated in any of the above-mentioned modes. When this mode is activated, the initial timer duration is set to 30 seconds, which can be extended by the user. Once the timer expires, the fan automatically turns off (switches to the "OFF" mode).

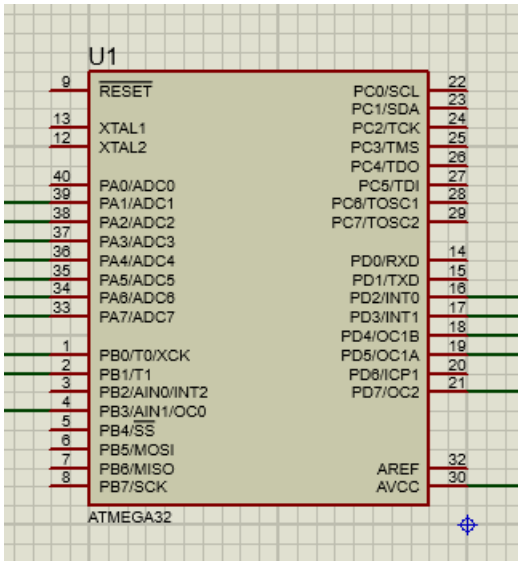
And motion sensor is used to detect the motion in front of the fan if there is motion the led will on, else the led will off.

2. System Block Diagram

2.1. Block Diagram

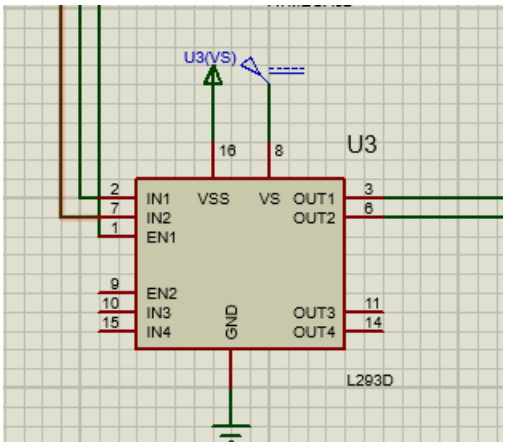
2.2. Block Diagram Description

1- Avr



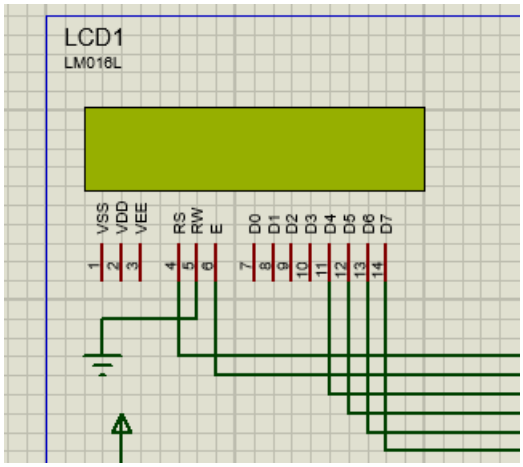
-The Microcontroller.

2- H-Bridge (connects between avr & DC motor)



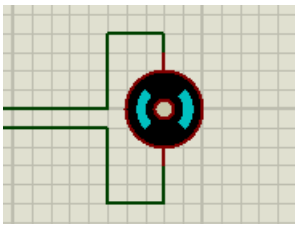
-The DC motor driver.

3- LCD(connect with Avr)



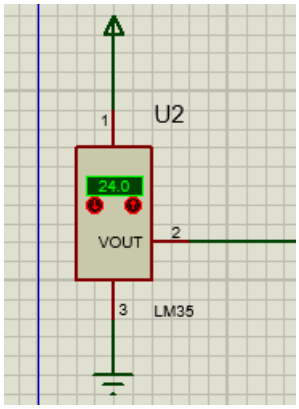
- Show the Temperature degree & the mode of Fun & if Timer on or off.

4-DC Motor(connect with avr by H-Bridge)



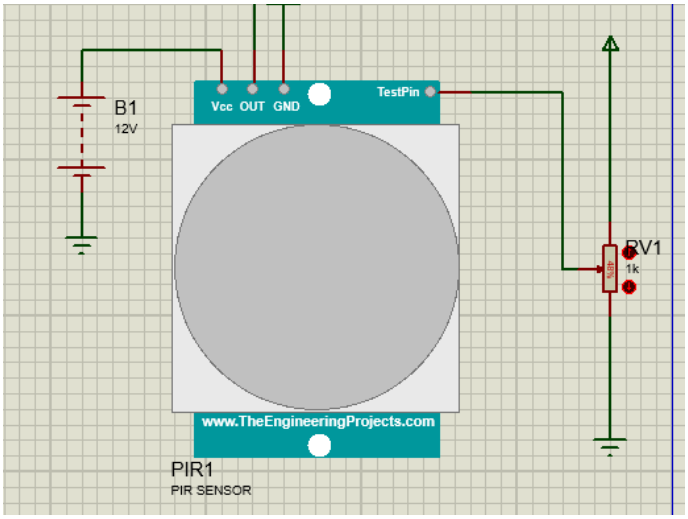
-The motor of the fan.

5-Temperature sensor (connect with Avr)



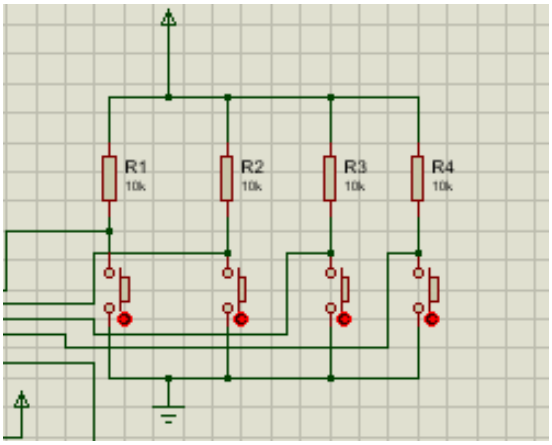
-It determine the motor speed according to the temperature.

6- Motion sensor(connect with Avr)



-Detect the motion if there is, the led will on, else the led will off.

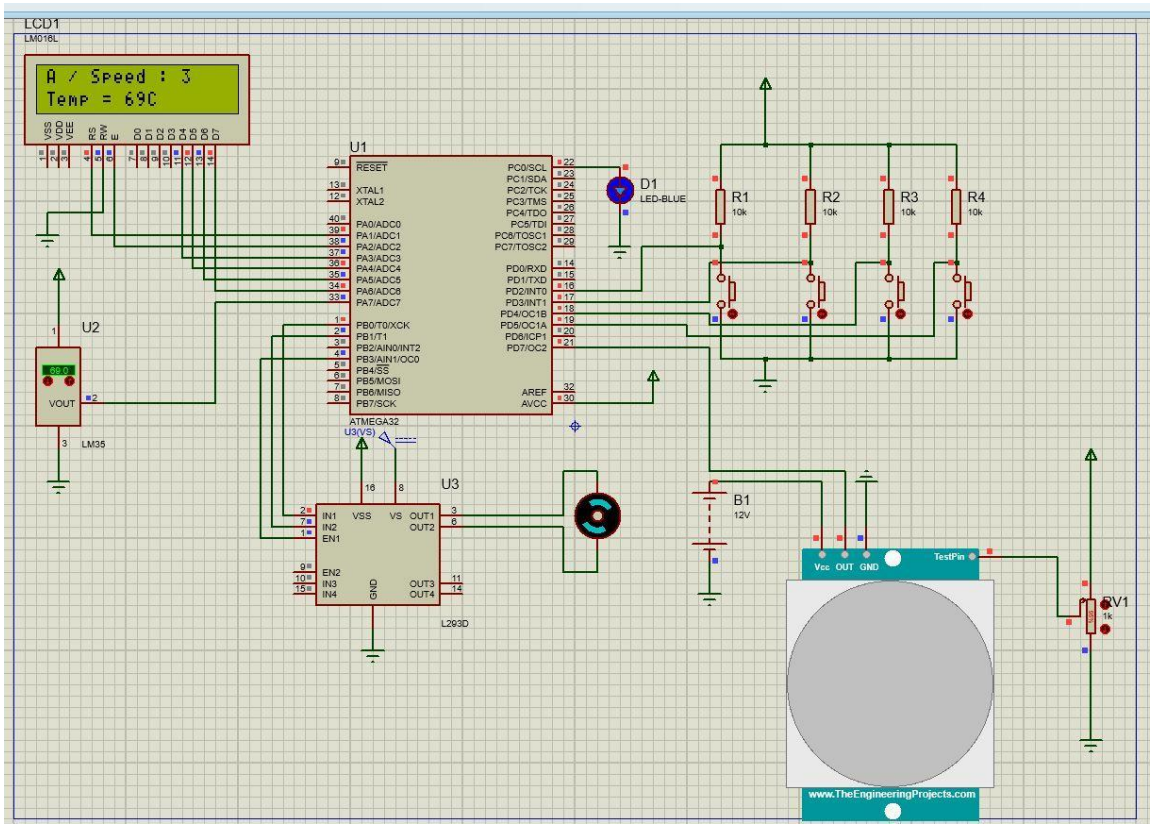
7- Buttons(connect with Avr)

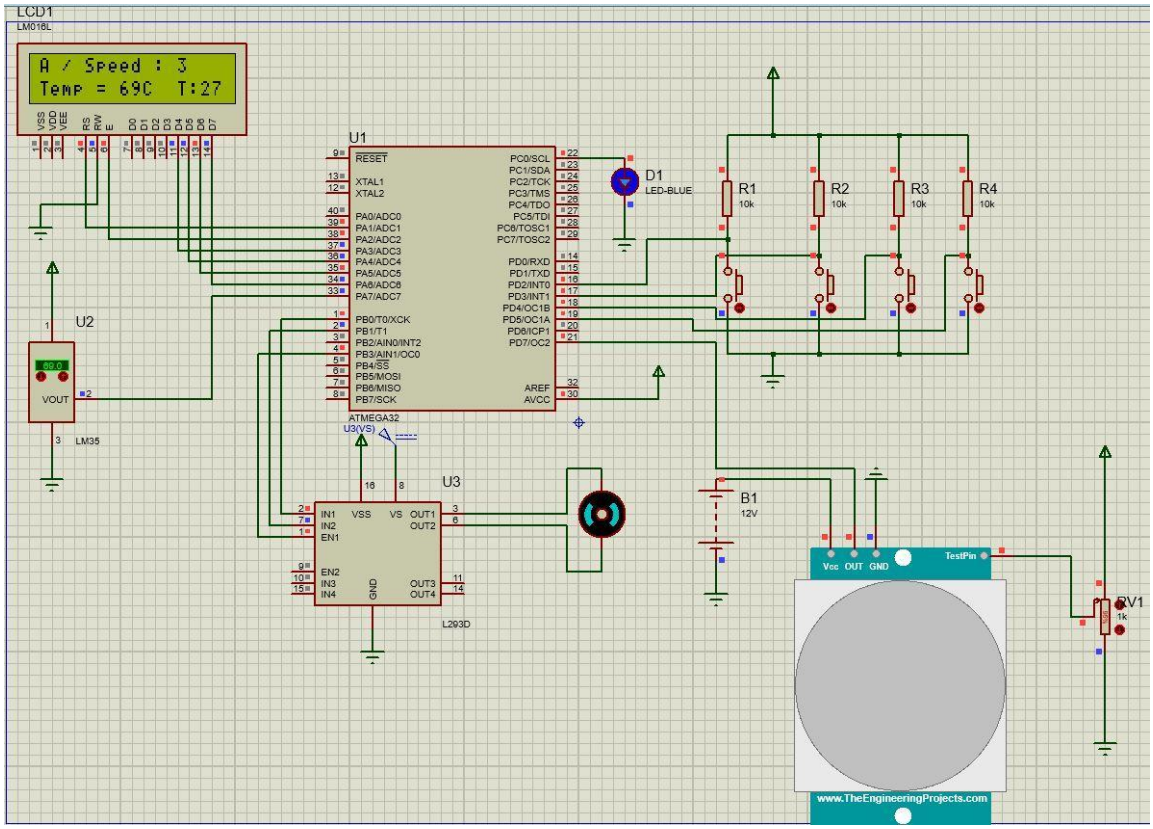


- button 1 : to togel between mode automatic or manual.
- button 2 : fan speed control.
- button 3 : to run timer.
- button 4 : to add 30 sec to timer counter.

(the button from left to right)

3. Schematic Diagram (Circuit Diagram)



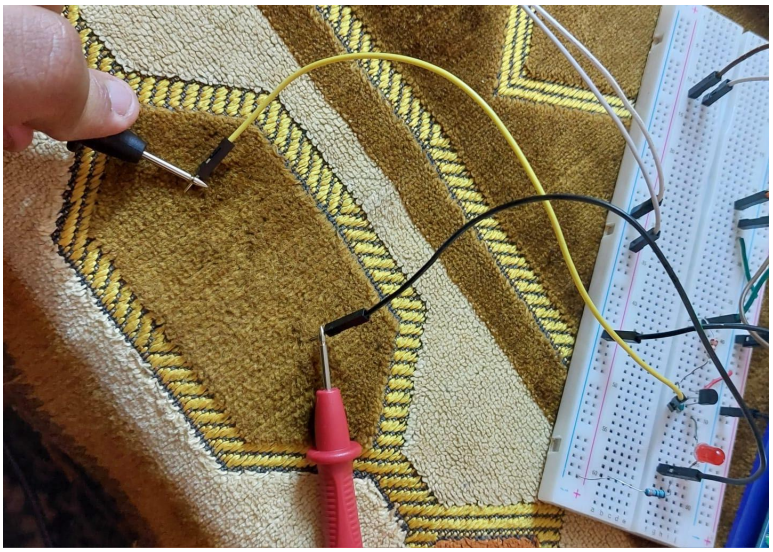


4. List Of Components

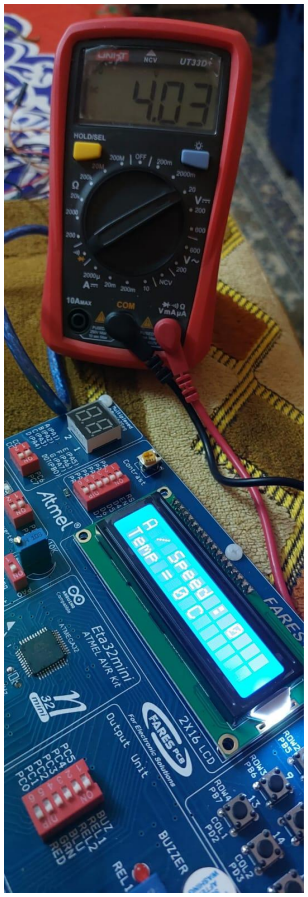
SN	Item Type	Item Code Name	Purpose	Quantity
1	Temperature sensor	LM35	To determine the temperature in Auto mode	1
2	Motion Sensor	PIR	To detect the motion and trun on/off the led	1
3	DC Motor	—	motor for fan	1
4	Buttons	—	To control motor	4
5	led	—		2-3
6	H-Bridge	L293d	Motor driver	1
7				
8				
9				
10				

5. Real-Time Hardware Photo

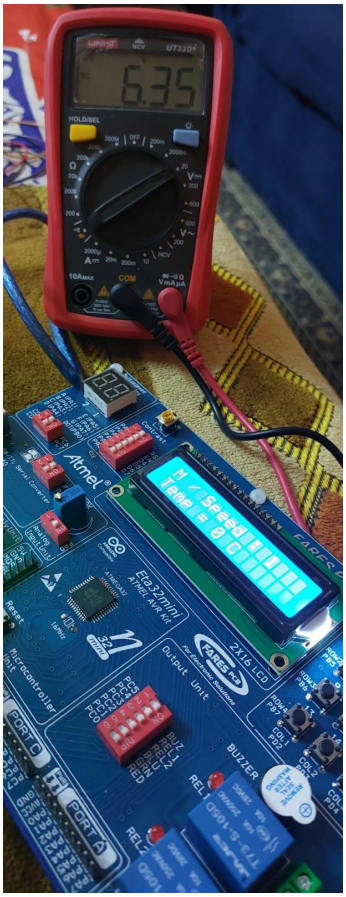
1-Motor wire:



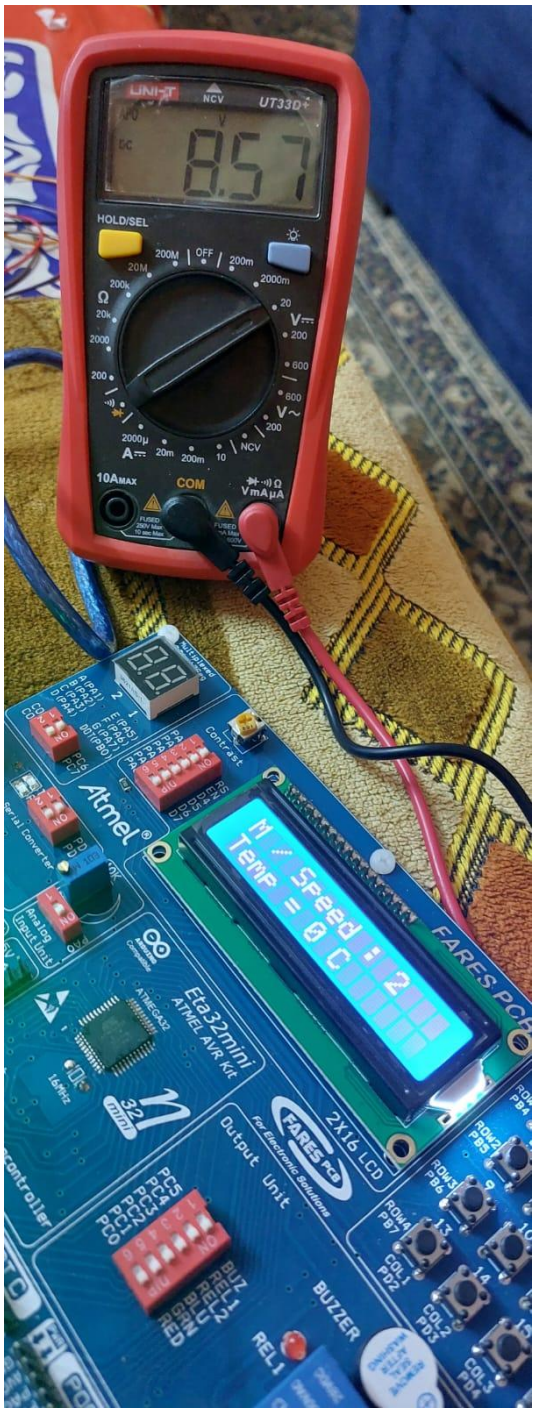
2- Automatic Mode (Speed 0):



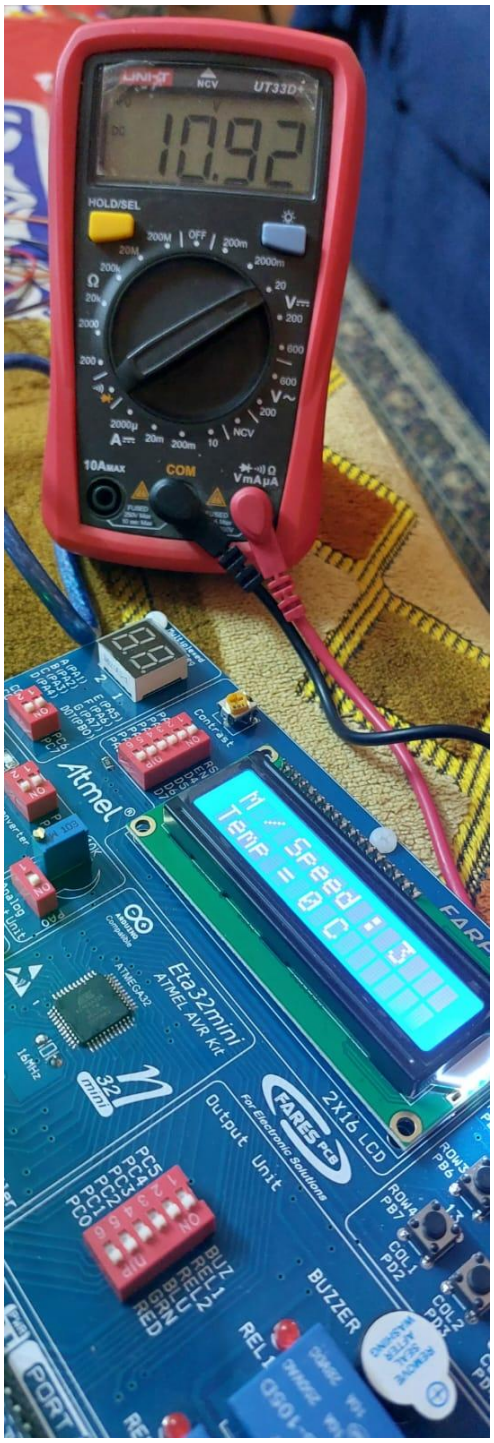
3- Manual Mode (Speed 0):



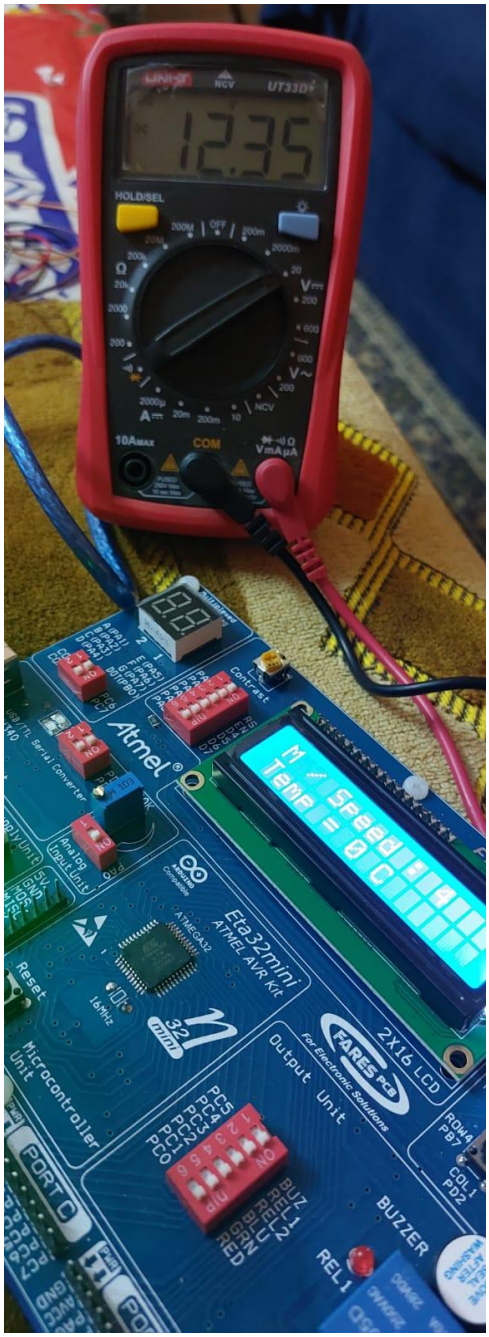
4- Manual Mode (Speed 2):



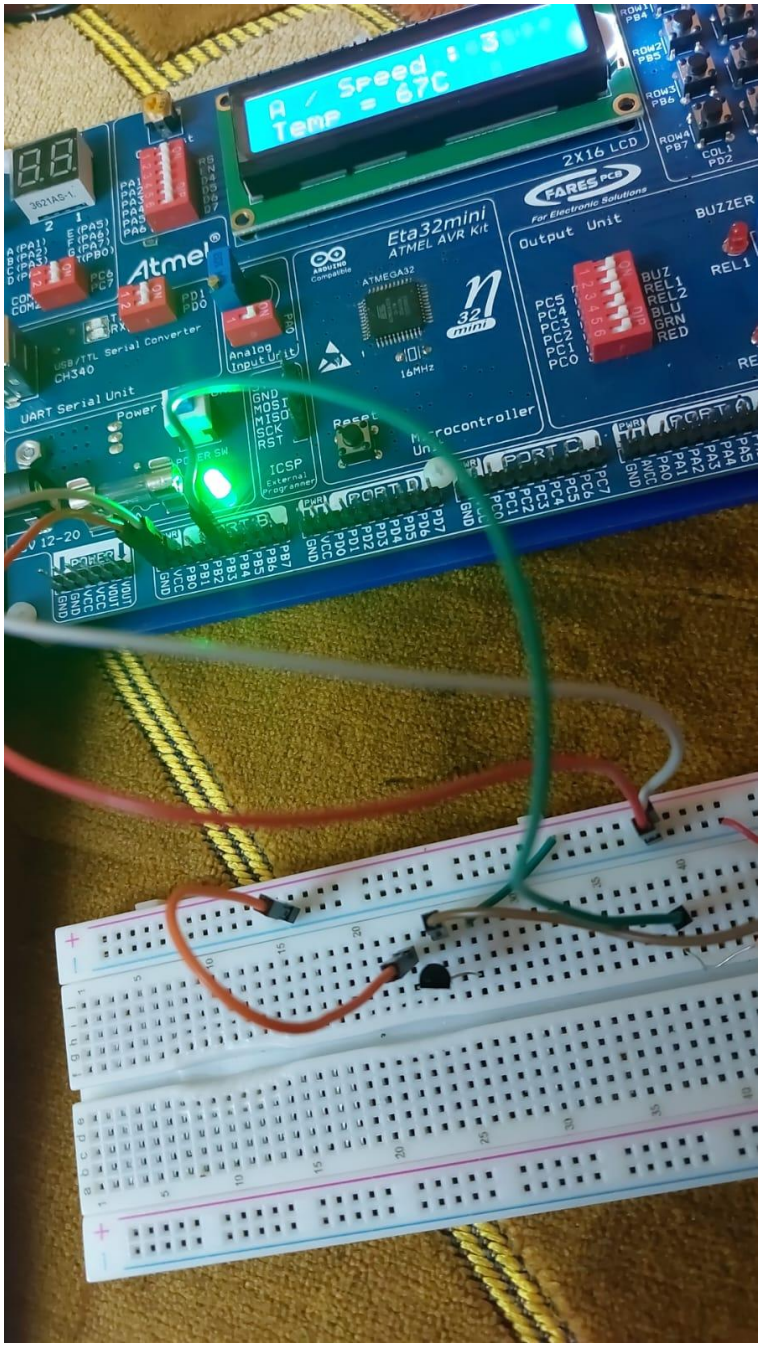
5- Manual Mode (Speed 3):



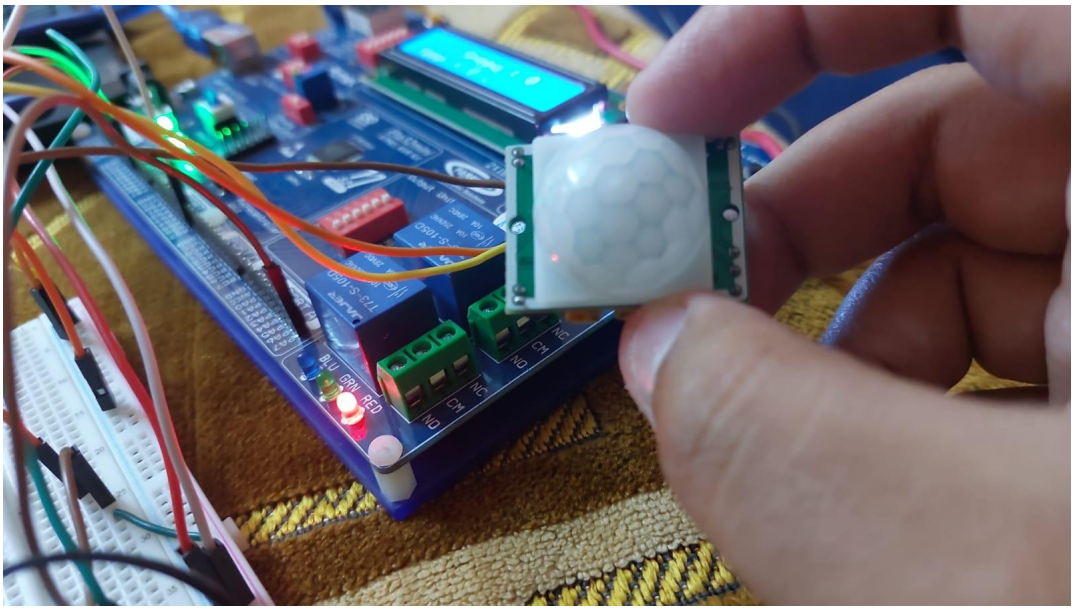
6- Manual Mode (Speed 4):



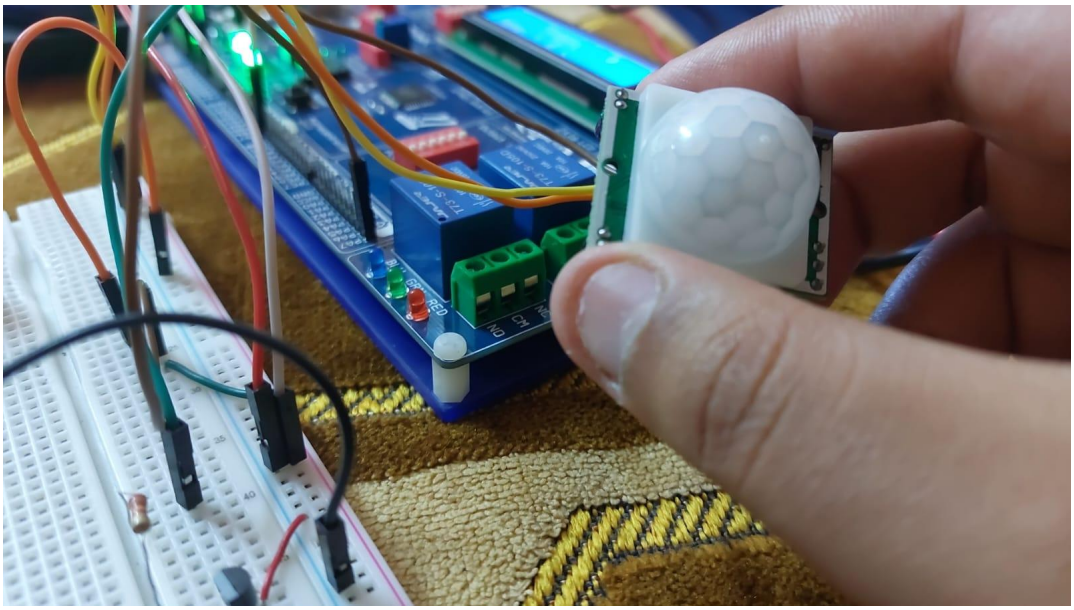
7- Temperature sensor:



8- Motion sensor detects motion (Led on):



9- Motion sensor doesn't detects motion (Led off):



6. Source Code

6.1. Hardware-side source code

(std_types.h)

```
#ifndef STD_TYPES_H_
#define STD_TYPES_H_

/* Boolean Data Type */
typedef unsigned char boolean;

/* Boolean Values */
#ifndef FALSE
#define FALSE       (0u)
#endif
#ifndef TRUE
#define TRUE        (1u)
#endif

#define LOGIC_HIGH      (1u)
#define LOGIC_LOW       (0u)

#define NULL_PTR    ((void*)0)

typedef unsigned char       uint8;          /*           0 .. 255             */
typedef signed char         sint8;          /*        -128 .. +127            */
typedef unsigned short      uint16;         /*           0 .. 65535           */
typedef signed short        sint16;         /*      -32768 .. +32767          */
typedef unsigned long       uint32;         /*           0 .. 4294967295      */
typedef signed long         sint32;         /* -2147483648 .. +2147483647     */
typedef unsigned long long  uint64;         /*       0 .. 18446744073709551615  */
typedef signed long long    sint64;         /* -9223372036854775808 ..
9223372036854775807 */
typedef float               float32;
typedef double              float64;

#endif /* STD_TYPE_H_ */
```

(adc.h)

```
#ifndef ADC_H_
#define ADC_H_
#include"std_types.h"

/*******************************************************************************
 *                                Definitions                                  *
 *******************************************************************************/
#define ADC_REFERENCE_VOLTAGE 2.56
#define ADC_MAXIMUM_VALUE     1023
/*******************************************************************************
 *                              Types Declarations                             *
 *******************************************************************************/
typedef enum{
    AREF=0, AVCC=1, INTERNAL=3
}ADC_ReferenceVolatge;

typedef enum{
    F_CPU_2=1, F_CPU_4, F_CPU_8, F_CPU_16, F_CPU_32, F_CPU_64,
    F_CPU_128
}ADC_Prescaler;

typedef struct{
    ADC_ReferenceVolatge ref_volt;
    ADC_Prescaler prescaler;
}ADC_ConfigType;
/*******************************************************************************
 *                          Functions Prototypes                               *
 *******************************************************************************/
/*
 * Description :
 * initialize the ADC with no interrupts
 */
void ADC_init(const ADC_ConfigType* config);

/*
 * Description :
 *       read analog value from specified channel and convert it to digital
 */
uint16 ADC_readChannel(uint8 ch_num);

#endif /* ADC_H_ */
```

(ADC.c)

```
#include"adc.h"
#include<avr/io.h>
#include"common_macros.h"

/*
 * Description :
 * initialize the ADC with no interrupts
 */
void ADC_init(const ADC_ConfigType* Config_Ptr)
{
    /* choose voltage reference voltage */
    ADMUX=(Config_Ptr->ref_volt<<6);
    /* enable ADC and set prescaler */
    ADCSRA=(1<<ADEN)|(Config_Ptr->prescaler);
}

/*
 * Description :
 *       read analog value from specified channel and convert it to digital
 */
uint16 ADC_readChannel(uint8 ch_num)
{
    /* choosing the channel */
    ADMUX=(ADMUX & 0xE0) | (0x1F & ch_num);
    /* start ADC conversion */
    SET_BIT(ADCSRA, ADSC);
    /* waiting for conversion to finish */
    while(BIT_IS_CLEAR(ADCSRA, ADIF));
    /* clear interrupt flag */
    SET_BIT(ADCSRA, ADIF);
    /* return converted data */
    return ADC;
}
```

(common_macros)

```
#ifndef COMMON_MACROS
#define COMMON_MACROS

/* Set a certain bit in any register */
#define SET_BIT(REG,BIT) (REG|=(1<<BIT))

/* Clear a certain bit in any register */
#define CLEAR_BIT(REG,BIT) (REG&=(~(1<<BIT)))

/* Toggle a certain bit in any register */
#define TOGGLE_BIT(REG,BIT) (REG^=(1<<BIT))

/* Rotate right the register value with specific number of rotates */
#define ROR(REG,num) ( REG= (REG>>num) | (REG<<(8-num)) )

/* Rotate left the register value with specific number of rotates */
#define ROL(REG,num) ( REG= (REG<<num) | (REG>>(8-num)) )

/* Check if a specific bit is set in any register and return true if yes */
#define BIT_IS_SET(REG,BIT) ( REG & (1<<BIT) )

/* Check if a specific bit is cleared in any register and return true if yes */
#define BIT_IS_CLEAR(REG,BIT) ( !(REG & (1<<BIT)) )

#define GET_BIT(REG,BIT) ((REG & (1<<BIT)) >> BIT)
#endif
```

(dc_motor.c)

```
#include"dc_motor.h"
#include"gpio.h"
#include"pwm.h"

/*
 * Description :
 * setup the direction for the two motor pins through the GPIO driver
 */
void DcMotor_init(void)
{
    /* set motor pins as output */
```

```
        ○ GPIO_setupPinDirection(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID, PIN_OUTPUT);
        ○ GPIO_setupPinDirection(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID+1, PIN_OUTPUT);
        ○ /* stop the motor at the beginning */
        ○ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID, LOGIC_LOW);
        ○ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID+1, LOGIC_LOW);
    • }
    •
    • /*
    •  * Description :
    •  * a function to choose motor state(ON/OFF, clock wise / anti clock wise)
    •  * and control speed
    •  */
    • void DcMotor_Rotate(DcMotor_State state,uint8 speed)
    • {
        ○ switch(state)
        ○ {
        ○ case STOP :
            ■ /* stop motor */
            ■ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID, LOGIC_LOW);
            ■ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID+1, LOGIC_LOW);
            ■ break;
        ○ case CW :
            ■ /* rotate clock wise */
            ■ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID, LOGIC_HIGH);
            ■ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID+1, LOGIC_LOW);
            ■ break;
        ○ case A_CW :
            ■ /* rotate anti clock wise */
            ■ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID, LOGIC_LOW);
            ■ GPIO_writePin(DC_MOTOR_PORT_ID, DC_MOTOR_PIN_ID+1, LOGIC_HIGH);
            ■ break;
        ○ }
        ○ PWM_Timer0_Start(speed);
    • }
```

(dc_motor.h)

```
#ifndef DC_MOTOR_H_
#define DC_MOTOR_H_
#include"std_types.h"
/*******************************************************************************
 *                                                                      Definitions
 *******************************************************************************/
#define DC_MOTOR_PORT_ID PORTB_ID
#define DC_MOTOR_PIN_ID PIN0_ID
/*******************************************************************************
 *                                                                      Types Declarations
 *******************************************************************************/
typedef enum{
    STOP, CW, A_CW
}DcMotor_State;

/*******************************************************************************
 *                                                              Functions Prototypes
 *******************************************************************************/
/*
 * Description :
 * setup the direction for the two motor pins through the GPIO driver
 */
void DcMotor_init(void);

/*
 * Description :
 * a function to choose motor state(ON/OFF, clock wise / anti clock wise)
 * and control speed
 */
void DcMotor_Rotate(DcMotor_State state,uint8 speed);
#endif /* DC_MOTOR_H_ */
```

(gpio.c)

```
#include "gpio.h"
#include "common_macros.h" /* To use the macros like SET_BIT */
#include "avr/io.h" /* To use the IO Ports Registers */

/*
 * Description :
 * Setup the direction of the required pin input/output.
 * If the input port number or pin number are not correct, The function will not handle the request.
 */
void GPIO_setupPinDirection(uint8 port_num, uint8 pin_num, GPIO_PinDirectionType direction)
{
    /*
     * Check if the input port number is greater than NUM_OF_PINS_PER_PORT value.
     * Or if the input pin number is greater than NUM_OF_PINS_PER_PORT value.
     * In this case the input is not valid port/pin number
     */
    if((pin_num >= NUM_OF_PINS_PER_PORT) || (port_num >= NUM_OF_PORTS))
    {
        /* Do Nothing */
    }
    else
    {
        /* Setup the pin direction as required */
        switch(port_num)
        {
        case PORTA_ID:
            if(direction == PIN_OUTPUT)
            {
                SET_BIT(DDRA,pin_num);
            }
            else
            {
                CLEAR_BIT(DDRA,pin_num);
            }
            break;
        case PORTB_ID:
            if(direction == PIN_OUTPUT)
            {
```

- SET_BIT(DDRB,pin_num);
- }
- else
- {
- CLEAR_BIT(DDRB,pin_num);
- }
- break;

- case PORTC_ID:
- if(direction == PIN_OUTPUT)
- {
- SET_BIT(DDRC,pin_num);
- }
- else
- {
- CLEAR_BIT(DDRC,pin_num);
- }
- break;

- case PORTD_ID:
- if(direction == PIN_OUTPUT)
- {
- SET_BIT(DDRD,pin_num);
- }
- else
- {
- CLEAR_BIT(DDRD,pin_num);
- }
- break;

- }

- }

- }
-
- /*
- * Description :
- * Write the value Logic High or Logic Low on the required pin.
- * If the input port number or pin number are not correct, The function will not handle the request.
- * If the pin is input, this function will enable/disable the internal pull-up resistor.
- */
- void GPIO_writePin(uint8 port_num, uint8 pin_num, uint8 value)
- {
- /*
- * Check if the input port number is greater than NUM_OF_PINS_PER_PORT value.
- * Or if the input pin number is greater than NUM_OF_PINS_PER_PORT value.

```
 * In this case the input is not valid port/pin number
 */
if((pin_num >= NUM_OF_PINS_PER_PORT) || (port_num >= NUM_OF_PORTS))
{
    /* Do Nothing */
}
else
{
    /* Write the pin value as required */
    switch(port_num)
    {
    case PORTA_ID:
        if(value == LOGIC_HIGH)
        {
            SET_BIT(PORTA,pin_num);
        }
        else
        {
            CLEAR_BIT(PORTA,pin_num);
        }
        break;
    case PORTB_ID:
        if(value == LOGIC_HIGH)
        {
            SET_BIT(PORTB,pin_num);
        }
        else
        {
            CLEAR_BIT(PORTB,pin_num);
        }
        break;
    case PORTC_ID:
        if(value == LOGIC_HIGH)
        {
            SET_BIT(PORTC,pin_num);
        }
        else
        {
            CLEAR_BIT(PORTC,pin_num);
        }
        break;
    case PORTD_ID:
        if(value == LOGIC_HIGH)
```

```
                    {
                        SET_BIT(PORTD,pin_num);
                    }
                    else
                    {
                        CLEAR_BIT(PORTD,pin_num);
                    }
                    break;
                }
            }
        }

        /*
         * Description :
         * Read and return the value for the required pin, it should be Logic High or Logic Low.
         * If the input port number or pin number are not correct, The function will return Logic Low.
         */
        uint8 GPIO_readPin(uint8 port_num, uint8 pin_num)
        {
            uint8 pin_value = LOGIC_LOW;

            /*
             * Check if the input port number is greater than NUM_OF_PINS_PER_PORT value.
             * Or if the input pin number is greater than NUM_OF_PINS_PER_PORT value.
             * In this case the input is not valid port/pin number
             */
            if((pin_num >= NUM_OF_PINS_PER_PORT) || (port_num >= NUM_OF_PORTS))
            {
                /* Do Nothing */
            }
            else
            {
                /* Read the pin value as required */
                switch(port_num)
                {
                case PORTA_ID:
                    if(BIT_IS_SET(PINA,pin_num))
                    {
                        pin_value = LOGIC_HIGH;
                    }
                    else
```

```
                    {
                        pin_value = LOGIC_LOW;
                    }
                    break;
                case PORTB_ID:
                    if(BIT_IS_SET(PINB,pin_num))
                    {
                        pin_value = LOGIC_HIGH;
                    }
                    else
                    {
                        pin_value = LOGIC_LOW;
                    }
                    break;
                case PORTC_ID:
                    if(BIT_IS_SET(PINC,pin_num))
                    {
                        pin_value = LOGIC_HIGH;
                    }
                    else
                    {
                        pin_value = LOGIC_LOW;
                    }
                    break;
                case PORTD_ID:
                    if(BIT_IS_SET(PIND,pin_num))
                    {
                        pin_value = LOGIC_HIGH;
                    }
                    else
                    {
                        pin_value = LOGIC_LOW;
                    }
                    break;
                }
            }

            return pin_value;
}

/*
 * Description :
 * Setup the direction of the required port all pins input/output.
 * If the direction value is PORT_INPUT all pins in this port should be input pins.
```

```
 * If the direction value is PORT_OUTPUT all pins in this port should be output pins.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_setupPortDirection(uint8 port_num, GPIO_PortDirectionType direction)
{
    /*
     * Check if the input number is greater than NUM_OF_PORTS value.
     * In this case the input is not valid port number
     */
    if(port_num >= NUM_OF_PORTS)
    {
        /* Do Nothing */
    }
    else
    {
        /* Setup the port direction as required */
        switch(port_num)
        {
        case PORTA_ID:
            DDRA = direction;
            break;
        case PORTB_ID:
            DDRB = direction;
            break;
        case PORTC_ID:
            DDRC = direction;
            break;
        case PORTD_ID:
            DDRD = direction;
            break;
        }
    }
}

/*
 * Description :
 * Write the value on the required port.
 * If any pin in the port is output pin the value will be written.
 * If any pin in the port is input pin this will activate/deactivate the internal pull-up resistor.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_writePort(uint8 port_num, uint8 value)
{
    /*
```

```
    * Check if the input number is greater than NUM_OF_PORTS value.
    * In this case the input is not valid port number
    */
    if(port_num >= NUM_OF_PORTS)
    {
        /* Do Nothing */
    }
    else
    {
        /* Write the port value as required */
        switch(port_num)
        {
        case PORTA_ID:
            PORTA = value;
            break;
        case PORTB_ID:
            PORTB = value;
            break;
        case PORTC_ID:
            PORTC = value;
            break;
        case PORTD_ID:
            PORTD = value;
            break;
        }
    }
}

/*
 * Description :
 * Read and return the value of the required port.
 * If the input port number is not correct, The function will return ZERO value.
 */
uint8 GPIO_readPort(uint8 port_num)
{
    uint8 value = LOGIC_LOW;

    /*
     * Check if the input number is greater than NUM_OF_PORTS value.
     * In this case the input is not valid port number
     */
    if(port_num >= NUM_OF_PORTS)
    {
        /* Do Nothing */
```

```
        }
        else
        {
            /* Read the port value as required */
            switch(port_num)
            {
            case PORTA_ID:
                value = PINA;
                break;
            case PORTB_ID:
                value = PINB;
                break;
            case PORTC_ID:
                value = PINC;
                break;
            case PORTD_ID:
                value = PIND;
                break;
            }
        }

        return value;
}
```

(gpio.h)

```
#ifndef GPIO_H_
#define GPIO_H_

#include "std_types.h"

/*******************************************************************************
 *                      Definitions                                            *
 *******************************************************************************/
#define NUM_OF_PORTS         4
#define NUM_OF_PINS_PER_PORT   8

#define PORTA_ID             0
#define PORTB_ID             1
#define PORTC_ID             2
#define PORTD_ID             3
```

- #define PIN0_ID 0
- #define PIN1_ID 1
- #define PIN2_ID 2
- #define PIN3_ID 3
- #define PIN4_ID 4
- #define PIN5_ID 5
- #define PIN6_ID 6
- #define PIN7_ID 7
-
- /***
- * Types Declaration *
- ***/
- typedef enum
- {
  - PIN_INPUT,PIN_OUTPUT
- }GPIO_PinDirectionType;
-
- typedef enum
- {
  - PORT_INPUT,PORT_OUTPUT=0xFF
- }GPIO_PortDirectionType;
-
- /***
- * Functions Prototypes *
- ***/
-
- /*
- * Description :
- * Setup the direction of the required pin input/output.
- * If the input port number or pin number are not correct, The function will not handle the request.
- */
- void GPIO_setupPinDirection(uint8 port_num, uint8 pin_num, GPIO_PinDirectionType direction);
-
- /*
- * Description :
- * Write the value Logic High or Logic Low on the required pin.
- * If the input port number or pin number are not correct, The function will not handle the request.
- * If the pin is input, this function will enable/disable the internal pull-up resistor.
- */
- void GPIO_writePin(uint8 port_num, uint8 pin_num, uint8 value);
-

```
/*
 * Description :
 * Read and return the value for the required pin, it should be Logic High or Logic Low.
 * If the input port number or pin number are not correct, The function will return Logic Low.
 */
uint8 GPIO_readPin(uint8 port_num, uint8 pin_num);

/*
 * Description :
 * Setup the direction of the required port all pins input/output.
 * If the direction value is PORT_INPUT all pins in this port should be input pins.
 * If the direction value is PORT_OUTPUT all pins in this port should be output pins.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_setupPortDirection(uint8 port_num, uint8 direction);

/*
 * Description :
 * Write the value on the required port.
 * If any pin in the port is output pin the value will be written.
 * If any pin in the port is input pin this will activate/deactivate the internal pull-up resistor.
 * If the input port number is not correct, The function will not handle the request.
 */
void GPIO_writePort(uint8 port_num, uint8 value);

/*
 * Description :
 * Read and return the value of the required port.
 * If the input port number is not correct, The function will return ZERO value.
 */
uint8 GPIO_readPort(uint8 port_num);

#endif /* GPIO_H_ */
```

(lcd.c)

```
#include"lcd.h"
#include"gpio.h"
#include<util/delay.h>
#include"common_macros.h"
#include<stdlib.h>

/*
 * Description :
 * initialize the LCD ports
 * configure 2 lines 8 bit mode
 * clear the screen and disable cursor
 */
void LCD_init(void)
{
    /* configure RS pin output */
    GPIO_setupPinDirection(LCD_RS_PORT_ID, LCD_RS_PIN_ID, PIN_OUTPUT);
    /* configure enable pin output */
    GPIO_setupPinDirection(LCD_E_PORT_ID, LCD_E_PIN_ID, PIN_OUTPUT);
#if (LCD_NUM_OF_BITS_MODE == 8)
    /* configure data port output for 8 bit operations */
    GPIO_setupPortDirection(LCD_DATA_PORT_ID, PORT_OUTPUT);
    /* choose 2 lines 8 bits mode */
    LCD_sendCommand(LCD_2LINES_8BITS_MODE);
#elif (LCD_NUM_OF_BITS_MODE == 4)
    /* configure data pins output for 8 bit operations */
    GPIO_setupPinDirection(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID,
    PIN_OUTPUT);
    GPIO_setupPinDirection(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+1,
    PIN_OUTPUT);
    GPIO_setupPinDirection(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+2,
    PIN_OUTPUT);
    GPIO_setupPinDirection(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+3,
    PIN_OUTPUT);

    /* send 4 bits initialization */
    LCD_sendCommand(LCD_2LINES_4BITS_MODE_INIT1);
    LCD_sendCommand(LCD_2LINES_4BITS_MODE_INIT2);
    /* choose 2 lines 4 bits mode */
    LCD_sendCommand(LCD_2LINES_4BITS_MODE);
#endif
    _delay_ms(20);
    LCD_sendCommand(LCD_DISABLE_CURSOR);
```

```
        LCD_sendCommand(LCD_CLEAR_SCREEN);
}

/*
 * Description :
 * send a command to the LCD
 */
void LCD_sendCommand(uint8 command)
{
    /* set RS to low to send a command */
    GPIO_writePin(LCD_RS_PORT_ID, LCD_RS_PIN_ID, LOGIC_LOW);
    _delay_ms(1);
    /* set enable bit to high */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_HIGH);
    _delay_ms(1);
    /* send the command */
#if (LCD_NUM_OF_BITS_MODE == 8)
    GPIO_writePort(LCD_DATA_PORT_ID, command);
    _delay_ms(1);

    /* set enable bit to low */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_LOW);
    _delay_ms(1);
#elif (LCD_NUM_OF_BITS_MODE == 4)
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID,
    GET_BIT(command, 4));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+1,
    GET_BIT(command, 5));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+2,
    GET_BIT(command, 6));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+3,
    GET_BIT(command, 7));
    _delay_ms(1);

    /* set enable bit to low*/
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_LOW);
    _delay_ms(1);

    /* set enable bit to high */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_HIGH);
    _delay_ms(1);

    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID,
    GET_BIT(command, 0));
```

```
        GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+1, GET_BIT(command, 1));
        GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+2, GET_BIT(command, 2));
        GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+3, GET_BIT(command, 3));
        _delay_ms(1);

        /* set enable bit to low */
        GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_LOW);
        _delay_ms(1);
#endif
}

/*
 * Description :
 * display a character on the LCD
 */
void LCD_displayCharacter(uint8 character)
{
    /* set RS to high to send a character */
    GPIO_writePin(LCD_RS_PORT_ID, LCD_RS_PIN_ID, LOGIC_HIGH);
    _delay_ms(1);
    /* set enable bit to high */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_HIGH);
    _delay_ms(1);
    /* send the character */
#if (LCD_NUM_OF_BITS_MODE == 8)
    GPIO_writePort(LCD_DATA_PORT_ID, character);
    _delay_ms(1);

    /* set enable bit to low */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_LOW);
    _delay_ms(1);
#elif (LCD_NUM_OF_BITS_MODE == 4)
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID, GET_BIT(character, 4));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+1, GET_BIT(character, 5));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+2, GET_BIT(character, 6));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+3, GET_BIT(character, 7));
    _delay_ms(1);
```

```
    /* set enable bit to low*/
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_LOW);
    _delay_ms(1);

    /* set enable bit to high */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_HIGH);
    _delay_ms(1);

    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID, GET_BIT(character, 0));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+1, GET_BIT(character, 1));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+2, GET_BIT(character, 2));
    GPIO_writePin(LCD_DATA_PORT_ID, LCD_DATA_PIN_ID+3, GET_BIT(character, 3));
    _delay_ms(1);

    /* set enable bit to low */
    GPIO_writePin(LCD_E_PORT_ID, LCD_E_PIN_ID, LOGIC_LOW);
    _delay_ms(1);
#endif
}

/*
 * Description :
 * display a string on the LCD;
 */
void LCD_displayString(const sint8* string)
{
    for(; *string!='\0'; string++)
    {
        LCD_displayCharacter(*string);
    }
}

/*
 * Description :
 * display a string in a specific location
 */
void LCD_displayStringRowColumn(uint8 row, uint8 column,const sint8* string)
{
    LCD_moveCursor(row, column);
```

```
        LCD_displayString(string);
}

/*
 * Description :
 * convert integers to characters then display it
 */
void LCD_integerToString(int num)
{
    char buff[16];
    itoa(num, buff, 10);
    LCD_displayString((const uint8*)buff);
}

/*
 * Description :
 * clear the screen of the LCD
 */
void LCD_clearScreen(void)
{
    LCD_sendCommand(LCD_CLEAR_SCREEN);
}

/*
 * Description :
 * move the LCD cursor to the desired location
 */
void LCD_moveCursor(uint8 row, uint8 column)
{
    switch(row)
    {
    case 0:
        LCD_sendCommand(LCD_SET_CURSOR_LOCATION | column);
        break;
    case 1:
        LCD_sendCommand(LCD_SET_CURSOR_LOCATION | (column + 0x40));
        break;
    case 3:
        LCD_sendCommand(LCD_SET_CURSOR_LOCATION | (column + 0x10));
        break;
    case 4:
```

        - LCD_sendCommand(LCD_SET_CURSOR_LOCATION | (column + 0x50));
        - break;
    - }
- }

(lcd.h)

```
#ifndef LCD_H_
#define LCD_H_
#include"std_types.h"

/*******************************************************************************
 *                              Definitions                                    *
 *******************************************************************************/
/* configuration */
#define LCD_NUM_OF_BITS_MODE 4

#define LCD_RS_PORT_ID PORTA_ID
#define LCD_RS_PIN_ID PIN1_ID

#define LCD_E_PORT_ID PORTA_ID
#define LCD_E_PIN_ID PIN2_ID

#define LCD_DATA_PORT_ID PORTA_ID
#if(LCD_NUM_OF_BITS_MODE == 4)
#define LCD_DATA_PIN_ID PIN3_ID
#endif
/* LCD commands */
#define LCD_DISABLE_CURSOR                  0x0C
#define LCD_CLEAR_SCREEN                    0x01
#define LCD_SET_CURSOR_LOCATION             0x80
#define LCD_2LINES_8BITS_MODE               0x38
#define LCD_2LINES_4BITS_MODE       0x28
#define LCD_2LINES_4BITS_MODE_INIT1  0x33
#define LCD_2LINES_4BITS_MODE_INIT2  0x32

/*******************************************************************************
 *                          Functions Prototypes                               *
 *******************************************************************************/
/*
 * Description :
 * initialize the LCD ports
```

```
 * configure 2 lines 8 bit mode
 * clear the screen and disable cursor
 */
void LCD_init(void);

/*
 * Description :
 * send a command to the LCD
 */
void LCD_sendCommand(uint8 command);

/*
 * Description :
 * display a character on the LCD
 */
void LCD_displayCharacter(uint8 character);

/*
 * Description :
 * display a string on the LCD;
 */
void LCD_displayString(const sint8* string);

/*
 * Description :
 * display a string in a specific location
 */
void LCD_displayStringRowColumn(uint8 row, uint8 column,const sint8* string);


/*
 * Description :
 * convert integers to characters then display it
 */
void LCD_integerToString(int num);

/*
 * Description :
 * clear the screen of the LCD
 */
void LCD_clearScreen(void);

/*
 * Description :
```

```
 * move the LCD cursor to the desired location
 */
void LCD_moveCursor(uint8 row, uint8 column);
#endif /* LCD_H_ */
```

(lm35_sensor.c)

```
#include "lm35_sensor.h"
#include "adc.h"

/*
 * Description :
 * Function responsible for calculate the temperature from the ADC digital value.
 */
uint8 LM35_getTemperature(void)
{
    uint8 temp_value = 0;

    uint16 adc_value = 0;

    /* Read ADC channel where the temperature sensor is connected */
    adc_value = ADC_readChannel(SENSOR_CHANNEL_ID);

    /* Calculate the temperature from the ADC value*/
    temp_value = (uint8)(((uint32)adc_value*SENSOR_MAX_TEMPERATURE*ADC_REFERENCE_VOLTAGE)/(ADC_MAXIMUM_VALUE*SENSOR_MAX_VOLT_VALUE));

    return temp_value;
}
```

(lm35_sensor.h)

```
#ifndef LM35_SENSOR_H_
#define LM35_SENSOR_H_

#include "std_types.h"

/*******************************************************************************
 *                                Definitions                                  *
 *******************************************************************************/

#define SENSOR_CHANNEL_ID        7
#define SENSOR_MAX_VOLT_VALUE    1.5
#define SENSOR_MAX_TEMPERATURE   150

/*******************************************************************************
 *                          Functions Prototypes                               *
 *******************************************************************************/

/*
 * Description :
 * Function responsible for calculate the temperature from the ADC digital value.
 */
uint8 LM35_getTemperature(void);

#endif /* LM35_SENSOR_H_ */
```

(pwm.h)

```
#include"pwm.h"
#include"gpio.h"
#include<avr/io.h>

/*
 * Description :
 * initialize timer0 with non-inverting PWM mode
 * setup duty cycle
 * generate 500hz frequency
 */
void PWM_Timer0_Start(uint8 duty_cycle)
{
    /* setup OC0 pin as output */
```

```
        GPIO_setupPinDirection(PORTB_ID, PIN3_ID, PIN_OUTPUT);
        /* select non-inverting fast PWM mode with prescaler F_CPU/8 */
        TCCR0=(1<<WGM00)|(1<<WGM01)|(1<<COM01)|(1<<CS02);
        TCNT0=0;
        OCR0=((uint16)duty_cycle * 255) / 100;
}
```

(pwm.h)

```
#ifndef PWM_H_
#define PWM_H_
#include"std_types.h"
/*******************************************************************************
 *                                                      Functions Prototypes
 *******************************************************************************/
/*
 * Description :
 * initialize timer0 with non-inverting PWM mode
 * setup duty cycle
 * generate 500hz frequency
 */
void PWM_Timer0_Start(uint8 duty_cycle);


#endif /* PWM_H_ */
```

(timer1.c)

```
#include<avr/io.h>
#include<avr/interrupt.h>
#include"common_macros.h"
#include"timer1.h"
/*******************************************************************************
 *                                                  Interrupt Service Routines
 *******************************************************************************/
static void  (*g_callBack)(void);
/* ISR for compare match interrupt */
ISR(TIMER1_COMPA_vect)
{
    (*g_callBack)();
}
```

```
/* ISR for overflow interrupt */
ISR(TIMER1_OVF_vect)
{
    (*g_callBack)();
}
/*******************************************************************************
 *                                      Functions Definitions
 *******************************************************************************/
/*
 * Description :
 * a function to initiate timer1 with desired configuration(prescaler, mode, initial register values)
 */
void Timer1_init(const Timer1_ConfigType * Config_Ptr)
{
    /* initializing timer 1 counter register */
    TCNT1 = Config_Ptr->initial_value;
    /* initializing compare register if compare mode specified */
    if(Config_Ptr->mode == COMPARE)
    {
        OCR1A = Config_Ptr->compare_value;
        /* enable compare match interrupt */
        TIMSK|=(1<<OCIE1A);
    }
    else
    {
        /* enable overflow interrupt */
        TIMSK|=(1<<TOIE1);
    }
    /* set FOC1A to 1 for non_PWM mode */
    TCCR1A|=(1<<FOC1A);
    /* set prescaler and choose mode of operation */
    /* note : WGM!0 and WGM11 are set to 0 in both normal and compare mode */
    TCCR1B= Config_Ptr->prescaler | (Config_Ptr->mode<<WGM12);
}

/*
 * Description :
 * a function to deactivate timer1
 */
void Timer1_deInit(void)
{
    TCCR1A=0;
    TCCR1B=0;
```

```
}

/*
 * Description :
 * a function to set the call back function pointer
 */
void Timer1_setCallBack(void (*a_ptr)(void))
{
    g_callBack=a_ptr;
}
```

(timer1.h)

```
#ifndef TIMER1_H_
#define TIMER1_H_
#include"std_types.h"
/*******************************************************************************
 *                                      Types Declarations
 *******************************************************************************/
typedef enum{
    NO_CLK, CLK_1, CLK_8, CLK_64, CLK_256, CLK_1024,
    EXTERNAL_FALLING_EDGE, EXTERNAL_RISING_EDGE
}Timer1_Prescaler;

typedef enum{
    NORMAL, COMPARE
}Timer1_Mode;

typedef struct {
 uint16 initial_value;
 uint16 compare_value; // it will be used in compare mode only.
 Timer1_Prescaler prescaler;
 Timer1_Mode mode;
}Timer1_ConfigType;
/*******************************************************************************
 *                                      Functions Prototype
 *******************************************************************************/
/*
 * Description :
 * a function to initiate timer1 with desired configuration(prescaler, mode, initial register values)
 */
```

```
void Timer1_init(const Timer1_ConfigType * Config_Ptr);

/*
 * Description :
 * a function to deactivate timer1
 */
void Timer1_deInit(void);

/*
 * Description :
 * a function to set the call back function pointer
 */
void Timer1_setCallBack(void (*a_ptr)(void));
#endif /* TIMER1_H_ */
```

(project3.c)

```
#include<util/delay.h>
#include<avr/io.h>
#include"lm35_sensor.h"
#include"dc_motor.h"
#include"lcd.h"
#include"adc.h"
#include"gpio.h"
#include"timer1.h"

#define MANUAL         0
#define AUTOMATIC 1

void timerTick();

/* variable to set time */
uint16 g_time=30;

int main()
{
    /* variable to control mode */
    uint8 mode = AUTOMATIC;
    /* variable to control fan speed */
    uint8 speed = 0;
    /* variable to indicate whether timer1 is working or not */
    uint8 timerOn=FALSE;
```

- /* variable to store temperature of LM35 */
- uint8 T;
- /* variable to configure ADC */
- ADC_ConfigType ADC_config;
- /* variable to configure Timer1 */
- Timer1_ConfigType Timer1_config;
- /* ADC configuration */
- ADC_config.prescaler=F_CPU_8;
- ADC_config.ref_volt=INTERNAL;
- /* initialize ADC */
- ADC_init(&ADC_config);
- /* Timer1 configuration */
- Timer1_config.initial_value=0;
- Timer1_config.compare_value=15635;
- Timer1_config.prescaler=CLK_1024;
- Timer1_config.mode=COMPARE;
- /* call back function for Timer1 interrupts */
- Timer1_setCallBack(timerTick);
- /* initialize motor pins */
- DcMotor_init();
- /* initialize LCD */
- LCD_init();
- LCD_displayString("A / ");
- //LCD_displayStringRowColumn(1, 3, "Temp = C");
- /* set buttons pins input */
- GPIO_setupPinDirection(PORTD_ID, PIN2_ID, PIN_INPUT);
- GPIO_setupPinDirection(PORTD_ID, PIN3_ID, PIN_INPUT);
- GPIO_setupPinDirection(PORTD_ID, PIN4_ID, PIN_INPUT);
- GPIO_setupPinDirection(PORTD_ID, PIN5_ID, PIN_INPUT);

- GPIO_setupPinDirection(PORTC_ID, PIN0_ID, PIN_OUTPUT);
- /* set PIR pin input */
- GPIO_setupPinDirection(PORTD_ID, PIN7_ID, PIN_INPUT);
- /* global interrupt enable */
- SREG|=(1<<7);
- for(;;)
- {
    - /* Turn the lamp on if there is motion */
    - if(GPIO_readPin(PORTD_ID, PIN7_ID) == LOGIC_HIGH)
    - {
        - GPIO_writePin(PORTC_ID, PIN0_ID, LOGIC_HIGH);
    - }
    - else
    - {

```
                GPIO_writePin(PORTC_ID, PIN0_ID, LOGIC_LOW);
        }
        /* when D2 button pressed toggle mode */
        if(GPIO_readPin(PORTD_ID, PIN2_ID) == LOGIC_LOW)
        {
            _delay_ms(30);
            if(GPIO_readPin(PORTD_ID, PIN2_ID) == LOGIC_LOW)
            {
                if(mode == AUTOMATIC)
                {
                    mode = MANUAL;
                    LCD_displayStringRowColumn(0,0,"M / ");
                }
                else
                {
                    mode = AUTOMATIC;
                    LCD_displayStringRowColumn(0,0,"A / ");
                }
            }
            _delay_ms(10);
        }
    •
        /* when D3 button pressed and the mode is manual increase speed  */
        if(GPIO_readPin(PORTD_ID, PIN3_ID) == LOGIC_LOW && mode ==
        MANUAL)
        {
            _delay_ms(30);
            if(GPIO_readPin(PORTD_ID, PIN3_ID) == LOGIC_LOW && mode
            == MANUAL)
            {
                speed++;
                /* max speed is 4 and start from 0 when exceeding 4 */
                if(speed == 5)
                {
                    speed=0;
                }
            }
            _delay_ms(10);
        }
    •
        /* when D4 button pressed toggle Timer1 */
        if(GPIO_readPin(PORTD_ID, PIN4_ID) == LOGIC_LOW)
        {
            _delay_ms(30);
```

- if(GPIO_readPin(PORTD_ID, PIN4_ID) == LOGIC_LOW)
- {
    - if(!timerOn)
    - {
        - timerOn=TRUE;
        - g_time=30;
        - Timer1_init(&Timer1_config);
        - LCD_displayStringRowColumn(1, 12,"T:");
        - LCD_integerToString(g_time);
    - }
    - else
    - {
        - timerOn=FALSE;
        - g_time=30;
        - Timer1_deInit();
        - LCD_displayStringRowColumn(1,12," ");
    - }
- }
- _delay_ms(10);

■ }

•

■ if(timerOn)
■ {

- LCD_moveCursor(1,14);
- LCD_integerToString(g_time);
- /* when D5 button pressed and the Timer is on increase time */
- if(GPIO_readPin(PORTD_ID, PIN5_ID) == LOGIC_LOW && g_time!=0)
- {
    - _delay_ms(30);
    - if(GPIO_readPin(PORTD_ID, PIN5_ID) == LOGIC_LOW &g_time!=0)
    - {
        - g_time+=30;
        - if(g_time == 300)
        - {
            - g_time=30;
        - }
    - }
    - _delay_ms(10);
- }

•

- /* stop motor when time reach 0 */
- if(g_time == 0)

```
                    {
                        speed = 0;
                        Timer1_deInit();
                    }
                }

                /* get sensor temperature */
                T=LM35_getTemperature();
                /* display the temperature on the LCD */
                LCD_displayStringRowColumn(1,0,"Temp = ");
                if(T < 10)
                {
                    LCD_integerToString(T);
                    LCD_displayString(" C");
                }
                else if(T < 100)
                {
                    LCD_integerToString(T);
                    LCD_displayCharacter('C');
                }
                if(mode == AUTOMATIC && g_time != 0)
                {
                    /* control speed based on temperature */
                    /* speed increases every 20 degrees */
                    speed = T/20;
                }
                /* rotate motor with desired speed */
                DcMotor_Rotate(CW, speed * 25);
                /* display speed */
                LCD_displayStringRowColumn(0, 4, "Speed : ");
                LCD_displayCharacter(speed+'0');


            }
        }

        /*
         * Description :
         * call back function for Timer1 to act as the Interrupt Service Routine
         */
        void timerTick()
        {
            g_time--;
        }
```

6.2. PC-side source code **[if applicable]**