

**Proiect la disciplina Microcontrolere**

**Termometru digital cu display LCD și ventilator**

Student: Ignat Codrina-Victoria

Electronică Aplicată

Anul IV, sgr. 342/1

1. **Obiectivul proiectului**

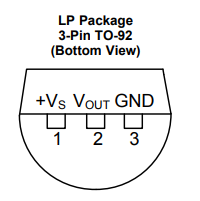
Proiectul are ca scop măsurarea temperaturii dintr-o încăpere folosind senzorul analogic LM35, afișarea temperaturii pe un display LCD, afișarea valorilor maxime și minime la apăsarea butoanelor asociate, dar și resetarea acestor valori la apăsarea unui buton de reset.

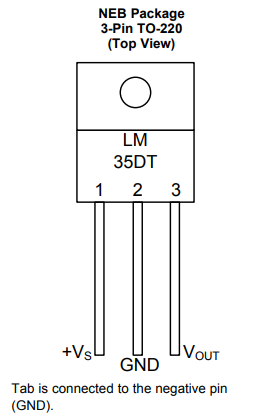
De asemenea, la o temperatură de peste 23 de grade, va porni un ventilator.

Legatura dintre intrari (senzorul de temperatură, butoane) și ieșiri (display, ventilator) se realizează prin intermediul microcontroller-ului produs de Microchip, dsPIC33FJ32MC202.

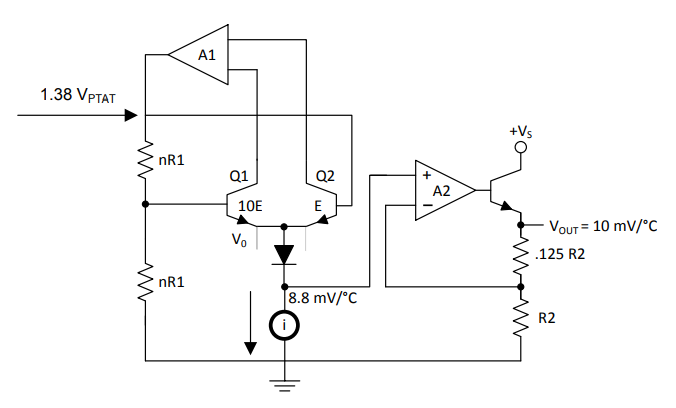
1. **Componente utilizate**
2. **Senzorul de temperatură LM35**

* Producător: Texas Instruments
* Calibrat direct în grade Celsius
* 10mV/C
* Domeniu: -55°C-> 150°C
* Low-Cost
* Package:





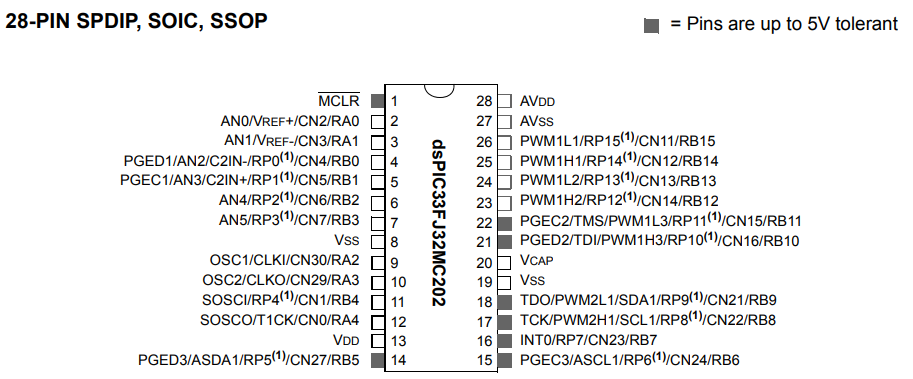
* Operează între 4V si 30V
* Aplicații: surse de alimentare, battery management, HVAC, aparate electrocasnice
* Nu necesită calibrare externă
* Diagrama bloc funcțională:



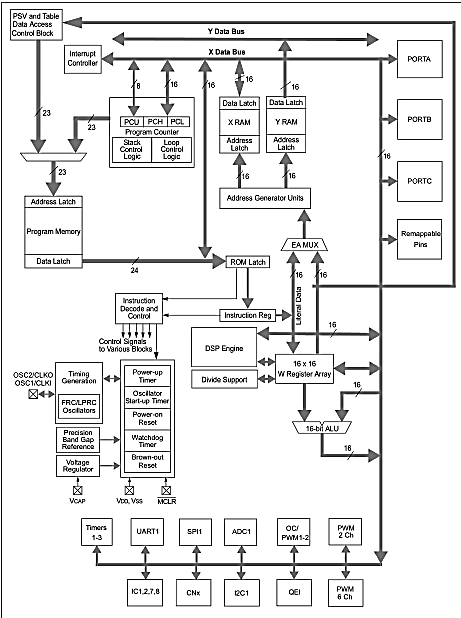
Vout=10mV/C \* T

1. **Microcontroller-ul dsPIC33FJ32MC202**

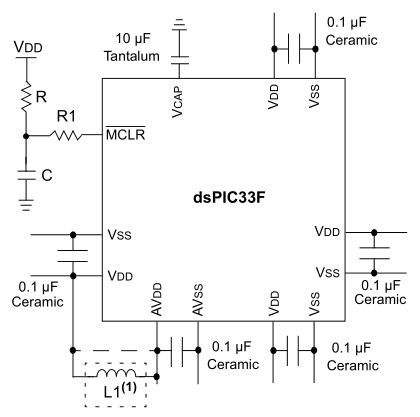
* Condiții de operare: 3V % 3.6V, -40°C ->150°C, DC la 20MIPS, 3V % 3.6V, -40°C ->125°C, DC la 40MIPS
* 16-bit Digital Signal Controller
* Up to 32KB Flash
* 2KB SRAM
* Arhitectură code-efficient: C și Asamblare
* Circuite PLL programabile și surse interne de clock
* Fail-Safe Clock Monitor
* Watchdog Timer
* PWM de mare viteză: până la 4 perechi PWM cu timp independent
* Modul ADC configurabil pentru 10 sau 12 biți
* 6 intrări analogice pe dispozitivele cu 28 de pini
* 3 timere/countere de 16 biti
* Modul UART (10Mbps)
* Suport pentru LIN 2.0
* Un modul I2C și un modul SPI
* Pin diagram:



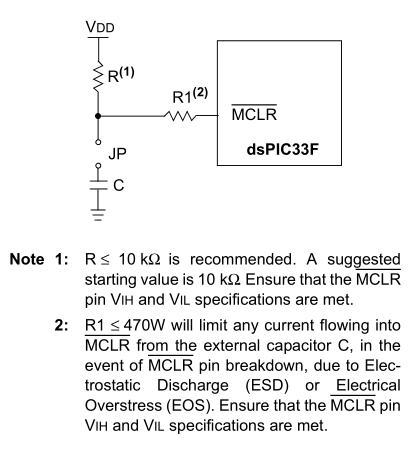
* Diagrama bloc:



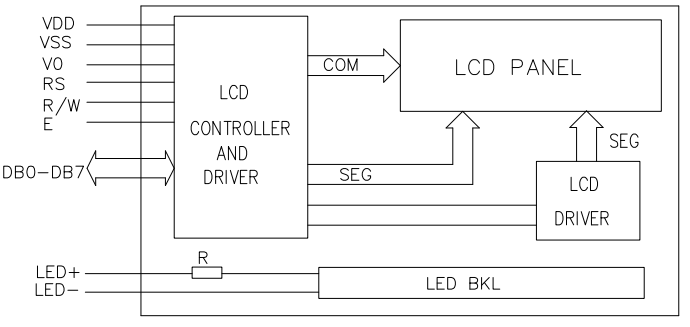
* Conexiuni minime recomandate:



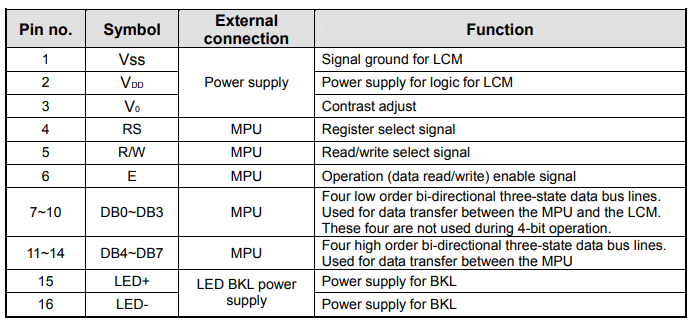
* Exemplu de conectare a pinului MCLR:



1. **Display LCD LM016**
   * 16 caractere \* 2 linii
   * 4-bit or 8-bit mode
   * Tensiune de alimentare: 0-7 V, tens. tipica: 3V
   * Temperaturi între care operează: 0-50°C
   * Diagrama bloc:



* Descrierea pinilor:



* Tabelul standard de caractere:



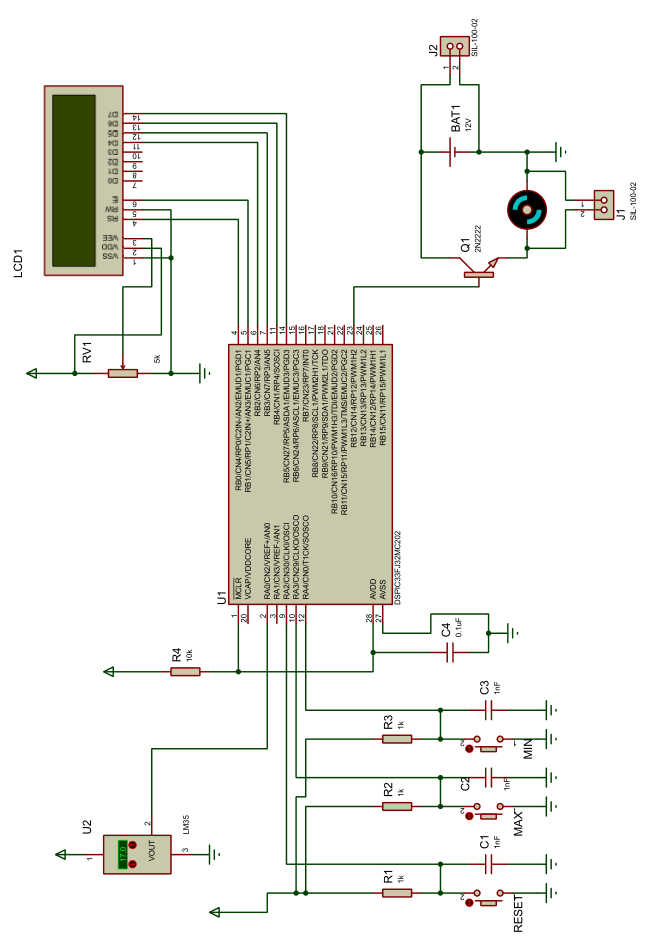
1. **Push button**
2. **DC Motor**
3. **Tranzistorul 2N2222**
4. **Funcționare**

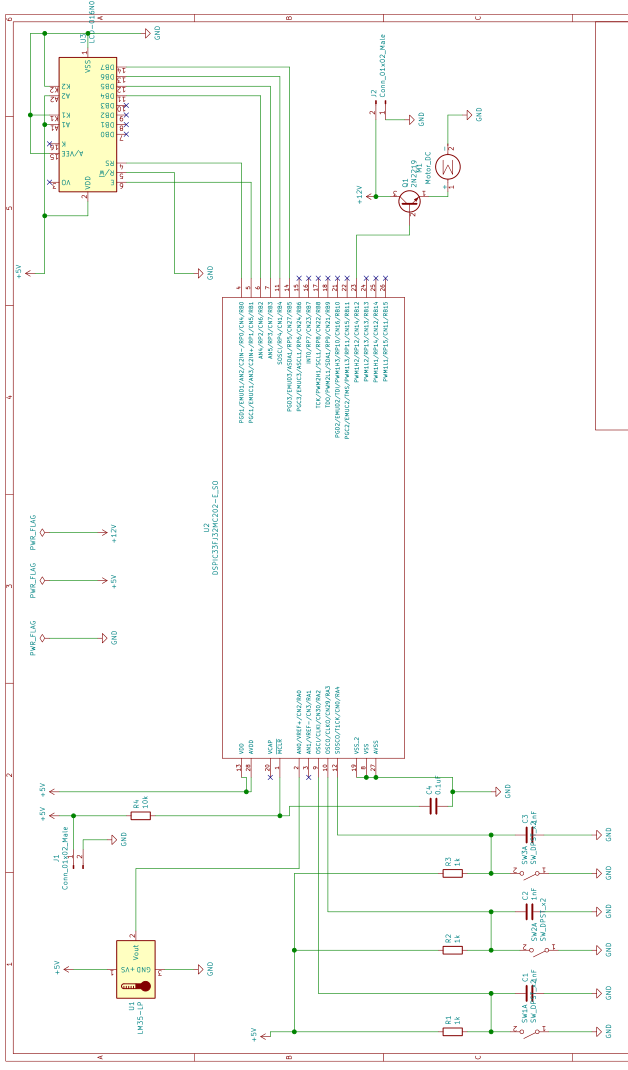
Senzorul de temperatură “citește valori” din mediul inconjurător pe care le transmite mai departe sub formă de nivele de tensiune: odată cu creșterea cu câte un grad Celsius, tensiunea de ieșire crește cu 10mV.

Această tensiune de la ieșire este trimisă mai departe la un pin analogic al microcontroller-ului, în vederea conversiei acestei valori într-o valoare înțeleasă de microcontroller (biți).

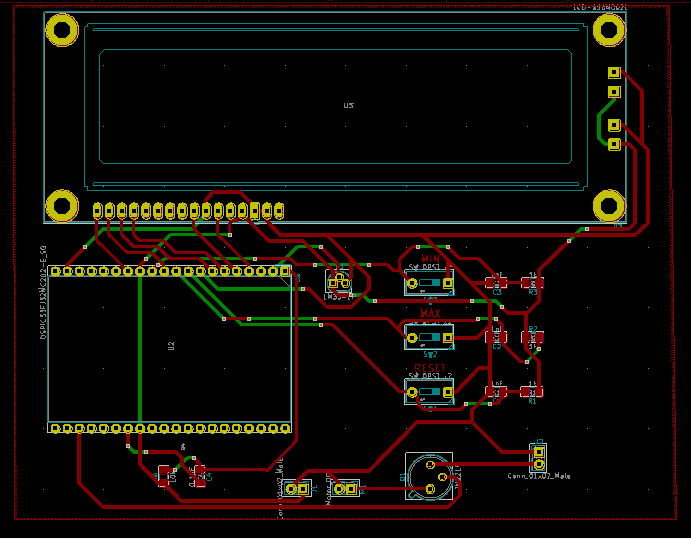
În continuare, valoarea în grade Celsius este afișată pe display-ul LCD. Dacă temperatura este mai mare de 23 de grade, un ventilator (reprezentat în schemă printr-un motor de curent continuu legat la un tranzistor și la o baterie de 12V) pornește. De asemenea, dacă butonul MAX este apăsat, pe display va apărea valoarea maximă a temperaturii înregistrate de la pornirea sistemului. La fel se va întampla la apăsarea butonului MIN, valoarea minimă a temperaturii va fi afișată pe display. La apăsarea butonului RESET, atât valoarea minimă cât și valoarea maximă vor fi resetate.

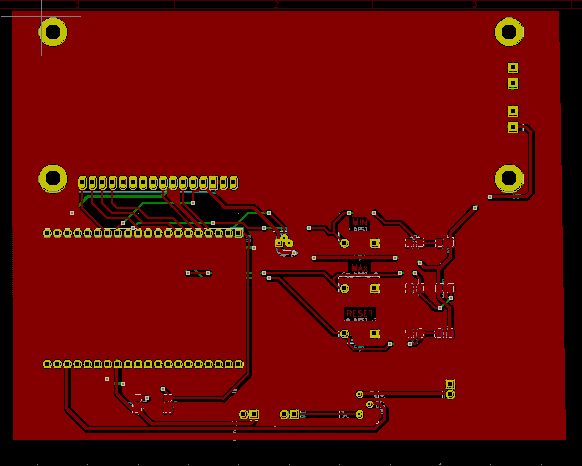
1. **Schema electrică**
2. **Proteus**

****

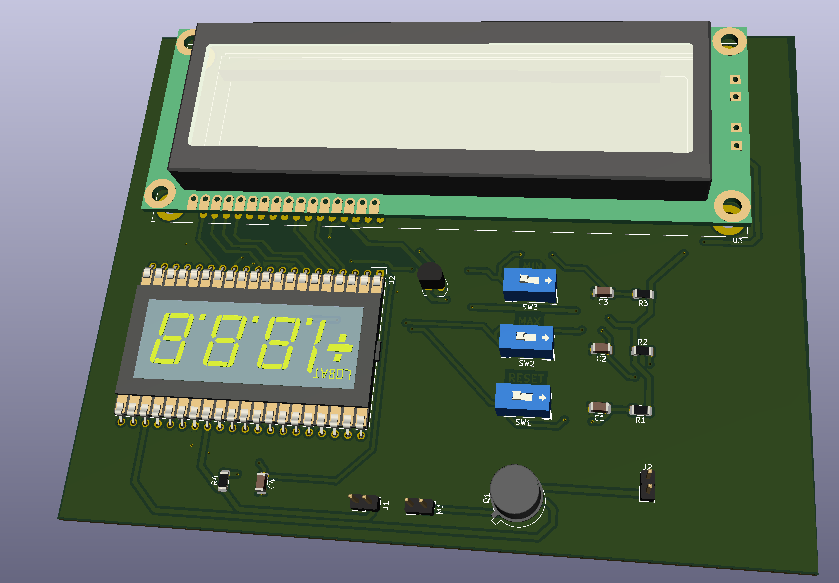
****

1. **PCB Layout**





**Vedere 3D:**

****

1. **Codul în limbajul C**

**“newfile.h”**

#ifndef NEWFILE\_H

#define NEWFILE\_H

#ifdef \_\_cplusplus

#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)

#pragma config PWRTE = ON // Power-up Timer Enable bit (PWRT enabled)

#pragma config BWRP = WRPROTECT\_OFF // Boot Segment Write Protect (Boot Segment may be written)

#pragma config BSS = NO\_FLASH // Boot Segment Program Flash Code Protection (No Boot program Flash segment)

#pragma config GWRP = OFF // General Code Segment Write Protect (User program memory is not write-protected)

#pragma config GSS = OFF // General Segment Code Protection (User program memory is not code-protected)

#pragma config FNOSC = LPRCDIVN // Oscillator Mode (Internal Fast RC (FRC) with divide by N)

#pragma config IESO = ON // Internal External Switch Over Mode (Start-up device with FRC, then automatically switch to user-selected oscillator source when ready)

#pragma config POSCMD = NONE // Primary Oscillator Source (Primary Oscillator Disabled)

#pragma config OSCIOFNC = OFF // OSC2 Pin Function (OSC2 pin has clock out function)

#pragma config IOL1WAY = ON // Peripheral Pin Select Configuration (Allow Only One Re-configuration)

#pragma config FCKSM = CSDCMD // Clock Switching and Monitor (Both Clock Switching and Fail-Safe Clock Monitor are disabled)

#pragma config WDTPOST = PS32768 // Watchdog Timer Postscaler (1:32,768)

#pragma config WDTPRE = PR128 // WDT Prescaler (1:128)

#pragma config WINDIS = OFF // Watchdog Timer Window (Watchdog Timer in Non-Window mode)

#pragma config FWDTEN = OFF // Watchdog Timer Enable (Watchdog timer enabled/disabled by user software)

#pragma config FPWRT = PWR128 // POR Timer Value (128ms)

#pragma config ALTI2C = ON // Alternate I2C pins (I2C mapped to ASDA1/ASCL1 pins)

#pragma config LPOL = OFF // Motor Control PWM Low Side Polarity bit (PWM module low side output pins have active-low output polarity)

#pragma config HPOL = OFF // Motor Control PWM High Side Polarity bit (PWM module high side output pins have active-low output polarity)

#pragma config PWMPIN = OFF // Motor Control PWM Module Pin Mode bit (PWM module pins controlled by PWM module at device Reset)

#pragma config ICS = PGD1 // Comm Channel Select (Communicate on PGC1/EMUC1 and PGD1/EMUD1)

#pragma config JTAGEN = OFF // JTAG Port Enable (JTAG is Disabled)

#endif

#endif

**main.c**

#include <xc.h>

#include <time.h>

#include <stdio.h>

#include "newfile.h"

#include <adc.h>

#include <string.h>

#include <stdlib.h>

#define BUTON\_MIN PORTAbits.RA4

#define BUTON\_MAX PORTAbits.RA3

#define BUTON\_RESET PORTAbits.RA2

#define MOTOR PORTBbits.RB12

#define RS PORTBbits.RB0

#define EN PORTBbits.RB1

#define D4 PORTBbits.RB2

#define D5 PORTBbits.RB3

#define D6 PORTBbits.RB4

#define D7 PORTBbits.RB5

int max = 0;

int min = 1000;

float ADCValue;

float tempinC;

int tempinC1;

char Buffer[2];

char Buffer\_max[2];

char Buffer\_min[2];

int i;

void delay(int number\_of\_seconds) {

int milli\_seconds = 1000 \* number\_of\_seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + milli\_seconds);

}

void Lcd\_SetBit(char data\_bit) {

if (data\_bit & 1)

D4 = 1;

else

D4 = 0;

if (data\_bit & 2)

D5 = 1;

else

D5 = 0;

if (data\_bit & 4)

D6 = 1;

else

D6 = 0;

if (data\_bit & 8)

D7 = 1;

else

D7 = 0;

}

void Lcd\_Cmd(char a) {

RS = 0;

Lcd\_SetBit(a); //Incoming Hex value

EN = 1;

delay(4);

EN = 0;

}

void Lcd\_Clear() {

Lcd\_Cmd(0); //Clear the LCD

Lcd\_Cmd(1); //Move the cursor to first position

}

void Lcd\_Set\_Cursor(char a, char b) {

char temp;

char z = 0;

char y = 0;

if (a == 1) {

temp = 0x80 + b - 1; //80H is used to move the cursor

z = temp >> 4; //Lower 8-bits

y = temp & 0x0F; //Upper 8-bits

Lcd\_Cmd(z); //Set Row

Lcd\_Cmd(y); //Set Column

} else if (a == 2) {

temp = 0xC0 + b - 1;

z = temp >> 4; //Lower 8-bits

y = temp & 0x0F; //Upper 8-bits

Lcd\_Cmd(z); //Set Row

Lcd\_Cmd(y); //Set Column

}

}

void Lcd\_Start() {

Lcd\_SetBit(0x00);

int i;

for (i = 32767; i <= 0; i--) Nop();

Lcd\_Cmd(0x03);

delay(5);

Lcd\_Cmd(0x03);

delay(11);

Lcd\_Cmd(0x03);

Lcd\_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes the LCD

Lcd\_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes the LCD

Lcd\_Cmd(0x08); //Select Row 1

Lcd\_Cmd(0x00); //Clear Row 1 Display

Lcd\_Cmd(0x0C); //Select Row 2

Lcd\_Cmd(0x00); //Clear Row 2 Display

Lcd\_Cmd(0x06);

}

void Lcd\_Print\_Char(char data) //Send 8-bits through 4-bit mode

{

char Lower\_Nibble;

char Upper\_Nibble;

Lower\_Nibble = data & 0x0F;

Upper\_Nibble = data & 0xF0;

RS = 1; //

Lcd\_SetBit(Upper\_Nibble >> 4); //Send upper half by shifting by 4

EN = 1;

int i;

for (i = 65535; i <= 0; i--) Nop();

EN = 0;

Lcd\_SetBit(Lower\_Nibble); //Send Lower half

EN = 1;

for (i = 65535; i <= 0; i--) Nop();

EN = 0;

}

void Lcd\_Print\_String(char \*a) {

int i;

for (i = 0; a[i] != '\0'; i++)

Lcd\_Print\_Char(a[i]); //Split the string using pointers and call the Char function

}

char\* itoa(int i, char b[]) {

char const digit[] = "0123456789";

char\* p = b;

if (i < 0) {

\*p++ = '-';

i \*= -1;

}

int shifter = i;

do { //Move to where representation ends

++p;

shifter = shifter / 10;

} while (shifter);

\*p = '\0';

do { //Move back, inserting digits as u go

\*--p = digit[i % 10];

i = i / 10;

} while (i);

return b;

}

int main() {

TRISA = 0;

TRISB = 0;

TRISAbits.TRISA0 = 1;

TRISAbits.TRISA2 = 1;

TRISAbits.TRISA3 = 1;

TRISAbits.TRISA4 = 1;

int temp;

AD1CON1 = 0; // ASAM bit = 1 implies sampling ..

AD1CHS0 = 0; // set channel 0 for sampling

AD1PCFGL = 0xFFFE; // all pins set as digital except A0

AD1CSSL = 0;

AD1CON1 = 0x0E4;

AD1CON2 = 0;

AD1CON3 = 0x1F05; // Sample time manual, Tad = internal 2 Tcy

AD1CON1bits.ADON = 1; // turn ADC ON

Lcd\_Start();

while (1) {

AD1CON1bits.SAMP = 0; // start Converting

while (AD1CON1bits.DONE) // conversion done

{

ADCValue = ADC1BUF0; // get ADC value

ADCValue = ADCValue \* 4.88281; //convert it into voltage

tempinC = ADCValue / 10; //getting the temperature values

itoa(tempinC, Buffer);

}

temp = atoi(Buffer);

if (temp < 23) {

MOTOR = 0;

}

else {

MOTOR = 1;

}

if (temp > max) {

max = temp;

itoa(max, Buffer\_max);

}

if ((temp < min)&(temp < max)){

min = temp;

itoa(min, Buffer\_min);

}

if (BUTON\_RESET == 0) {

max = 0;

min = 1000;

Lcd\_Clear();

Lcd\_Set\_Cursor(1, 2);

Lcd\_Print\_String("Max resetat");

}

if (BUTON\_MAX == 0) {

Lcd\_Clear();

Lcd\_Set\_Cursor(1, 2);

Lcd\_Print\_String("Temp max: ");

Lcd\_Set\_Cursor(2, 4);

for (i = 0; i <= 1; i++) {

Lcd\_Print\_Char(Buffer\_max[i]);

}

Lcd\_Print\_Char(0xDF);

}

if (BUTON\_MIN == 0){

Lcd\_Clear();

Lcd\_Set\_Cursor(1, 2);

Lcd\_Print\_String("Temp min: ");

Lcd\_Set\_Cursor(2, 4);

for (i = 0; i <= 1; i++) {

Lcd\_Print\_Char(Buffer\_min[i]);

}

Lcd\_Print\_Char(0xDF);

}

Lcd\_Clear();

Lcd\_Set\_Cursor(1, 2);

Lcd\_Print\_String("Temperatura");

Lcd\_Set\_Cursor(2, 4);

for (i = 0; i <= 1; i++) {

Lcd\_Print\_Char(Buffer[i]);

}

Lcd\_Print\_Char(0xDF);

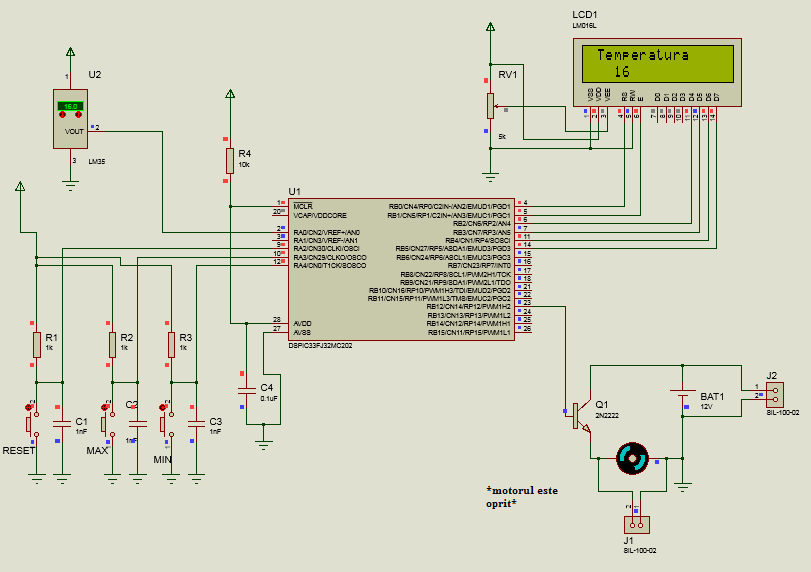
Lcd\_Print\_Char(0x43);

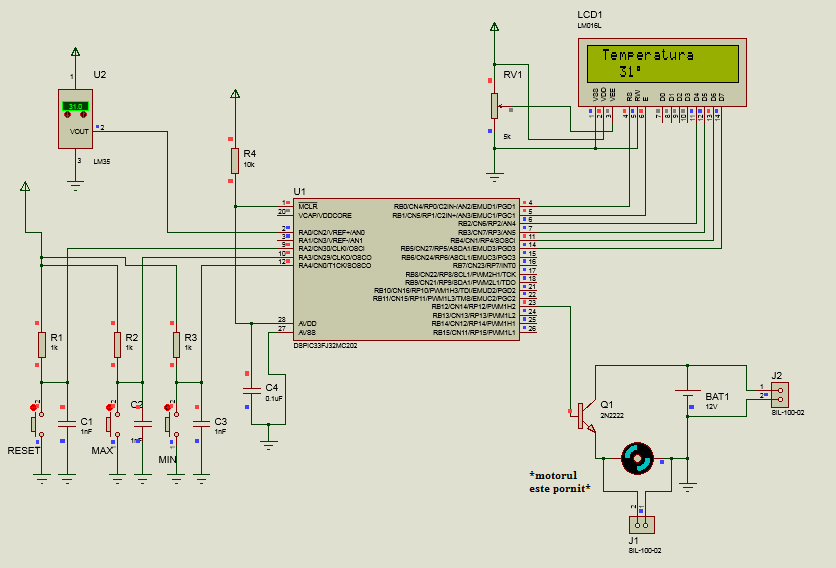
}

return 0;

}

1. **Simulări**

****

****

1. **Bibliografie**
   * **LM35 Precision Centigrade Temperature Sensors Datasheet**

<http://www.ti.com/lit/ds/symlink/lm35.pdf>

* + **dsPIC33FJ32MC202 Datasheet**

<http://ww1.microchip.com/downloads/en/devicedoc/70283k.pdf>

* + **Specifications of LCD Module**

<https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf>

* + **How to use intelligent LCDs – Part One- by Julyan Ilett**
  + **How to use intelligent LCDs – Part Two- by Julyan Ilett**