Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Catedra Microelectronica și Inginerie Biomedicala

Raport

Lucrare de Laborator nr.3

La Disciplina:Programarea Microprocesoarelor
Tema: Senzor de temperatură.LM20

A efectuat: st. gr.ISBM-141 Idricean Dionisie_____

A verificat: prof.univ. Eugeniu Lazari _____

Chișinău 2017

1.Scopul lucrării

1.Conexiunea senzorului LM20 la MCU.

2.Activarea/configurarea ADC-ului.

3.Determinearea temperaturii.

4.Afișarea rezultatelor obținute la LCD.

2.Sarcina

Realizarea unei aplicații MCU,care primește semnal de la senzorul de temperatură LM20,prelucrează aceste semnal,utilizînd tabelul din datasheet-ul senzorului se determină temperatura.

3.Noțiuni Teoretice

Modulul (**ADC**) Convertor Analog-Digital este utilizat pentru conversia tensiunii analogice într-o valoare digitală (în AVR pe 10 biţi). De exemplu, poate fi utilizat pentru perceptarea ieşirii de la un senzor (de temperatură, presiune, etc.) în anumite interval, sau pentru a executa anumite acţiuni în dependenţă de valoarea primită. Există mai multe tipuri de convertoare, dar cele utilizate în microcontrollerele AVR sunt convertoarele cu aproximare succesivă. Mai jos urmează o schemă simplificată a convertorului analog-digital prezent pe AVR:

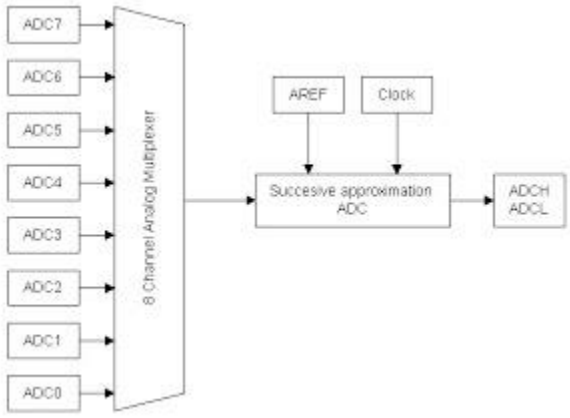

Figura 1 : Schemă simplificată a convertorului analog-digital

La intrare avem un multiplexor analog-digital, care este utilizat la selectarea între 8 intrări analogice diferite. Aceasta înseamnă că este posibilă conversia a opt semnale (bineînţeles că nu în acelaşi timp). La ieşire, valoarea convertită este înscrisă în regiştrii ADCL şi ADCH, deoarece regiştrii AVR au lăţime de 8 biţi este nevoie de o pereche p-u a stoca o valoare pe 10 biţi.

În modelele ATmega16x, ATmega32x, ATmega64x, ATmega128x pinii ADC-ului pot fi uniţi în perechi pentru a obţine în total pînă la 13 canale cu intrare diferenţială. Două canale în aşa caz au posibiliatatea de amplificare preventivă de 20 şi 200 ori a semnalului de intrare. În cazul coeficienţilor de amplificare 1x şi 20x rezoluţia efectivă a conversiei este de 8 biţi, pe cînd la coeficientul 200x – 7 biţi.

În calitate de tensiune de referinţă pentru ADC poate fi utilizată atît tensiunea microcontrollerului cît şi sursa internă sau externă de tensiune de referinţă. ADC-ul poate funcţiona în două regime:

- regimul conversiei unice, cînd startul fiecărei conversii este iniţializată de utilizator.
- regimul conversiei continue, cînd startul conversiei are loc continuu după anumite intervale de timp.

Tensiunea analogică la intrare trebuie să fie mai mare decît 0V, şi mai mică decît tensiunea de referinţă a convertorului – AREF. Tensiunea de referinţă este o tensiune externă care trebuie aplicată la pinul Aref a microcontrollerului. Valoarea convertită poate fi calculată folosind următorea formulă:

ADCH:L= ((Vin/Vref)*1023)

Deoarece convertorul are o rezoluţie de 10 biţi, avem 1024 valori posibile, adică dacă tensiunea de intrare Vin este egală cu 0, atunci valoarea convertită va fi zero, dacă Vin este egală cu Vref atunci valoarea convertită v-a fi 1023, iar dacă Vin este ½ din Vref atunci valoarea ADCH:L este 512. Acest proces de conversie se mai numeşte cuantificare, bineînţeles ce produce o eroare, numită eroare de cuantificare.

Regimurile de operare a convertorului analog-digital (CAD)

Convertorul Analog-Digital are două regimuri fundamentale de funcţionare: Conversie Unară (singulară) şi Conversie Continuă. În regim de conversie unară, este necesar de a iniţializa fiecare conversie, cînd este gata rezultatul este plasat în perechea de regiştri ADCH:L, şi nu se porneşte o altă conversie. În regim continuu, este necesar de a porni doar o dată conversia şi CAD v-a porni automat următoarea conversie în dată ce este finisată cea actuală.

Conversia analog-digitală nu este infăptuită instantaneu, este necesar de un anumit interval de timp, care depinde de frecvenţa de clock folosită de CAD şi este proporţională cu frecvenţa de clock şi poate fi între 50-200 kHz.

Dacă este de ajuns o rezoluţie de conversie mai mică de 10 biţi, frecvenţa CAD poate fi mărită prin ajustarea unui prescaler prezent în CAD, care divide frecvenţa de clock la un anumit coeficient, setat prin intermediul biţilor ADPS2:0 descrişi mai jos.

Pentru a afla intervalul de timp necesar unei conversii se divide numărul de cicluri necesari unei conversii la frecvenţa DAC. Normal, o conversie este îndeplinită în 13 cicluri CAD, dar prima conversie (deodată după pornirea CAD) se îndeplineşte în 25 cicluri de clock, şi mai este numită “Conversie Extinsă”. De exemplu dacă folosim un CAD la frecvenţa de 200kHz, o conversie normală va fi îndeplinită în 65 µs, iar una extinsă în 125 µs.

Regiştrii de stare şi control ai CAD

CAD conţine 4 regiştri de I/O:

- ADMUX – ADC Multiplexer Select Register
- ADCSR – ADC Control and Status Register
- ADCH:L – ADC Data Register(High, Low)

ADCSR	0x06(0x26)	♦	–	♦	♦	–	–	–	Registrul de control şi stare
ADCSRA	0x06(0x26)	–	♦	–	–	♦	♦	♦	Registrul de control şi stare A
ADCSRB	(0x8E)	–	–	–	–	–	♦		Registru de control şi stare B
ADMUX	0x07(0x27)	♦	♦	♦	♦	♦	♦	♦	Registrul de control cu multiplexorul
SFIOR	0x30(0x50)	♦	♦	–	–	♦	–	–	Registrul funcţiilor speciale
	0x20(0x40)	–	–	–	–	–	♦	♦	

Figura 2 : Tabel 1. Regiştrii de control a ADC-ului

ADMUX

Acest registru este utilizat pentru selectarea unuia din cele 8 canale care va fi convertit. Tabelul de mai jos arată setările posibile ale acestui registru:

MUX2	MUX1	MUX0	Intrarea Selectată
0	0	0	ADC0
0	0	1	ADC1
0	1	0	ADC2
0	1	1	ADC3
1	0	0	ADC4
1	0	1	ADC5
1	1	0	ADC6
1	1	1	ADC7

Figura 3 : Tab. 3 Setările posibile ale ADMUX

	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS 2	ADPS 1	ADPS 0	ATmega8 x ATmega1 63x ATmega3 23x ATmega1 28x
Citire(R)/ Scriere(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valoarea inițială	0	0	0	0	0	0	0	0	
	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADAT E	ADIF	ADIE	ADPS 2	ADPS 1	ADPS 0	ATmega1 6x ATmega3 2x ATmega6 4x
Citire(R)/ Scriere(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valoarea inițială	0	0	0	0	0	0	0	0	

Figura 4 : Formatul registrului ADCSRA(ADCSR)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Figura 5 : Tab. 4 Registrul ADCSR

ADEN – ADC Enable, setînd bitul dat în "1" este pornit CAD, iar în "0" CAD este oprit. Oprind CAD în timpul unei conversii va duce la anularea conversiei date.

ADSC – ADC Start Convertion, în regim de conversie continuă va fi necesar de setat acest bit doar la prima conversie, conversiile următoare vor fi pornite automat. În regim de conversie unară acest bit trebuie setat la fiecare conversie necesară. Acest bit se setează în "0" automat la sfîrşitul oricărei conversii.

ADFR – ADC Free Running Mode, se setează în "1" la conversie continuă.

ADIF – ADC Interrupt Flag, acest bit este setat automat în 1 la finisarea unei conversii, sau este setat automat în "0" doar la executarea ISR corespunzător vectorului de întreruperi, alternativ poate fi setat în "0" înscriind "1" în acest bit.

ADIE – ADC Interrupt Enable, cînd este setat în "1" şi SREG(I)=1 la apariţia ADIF se cheamă ISR corespunzătoare din vectorul de întreruperi.

ADPS2:0 – ADC Prescaler Select, aceşti biţi determină factorul de dividere a frecvenţei de clock pentru CAD, tabela de mai jos arată valorile posibile:

ADPS2	ADPS1	ADPS0	Factorul de Diviziune
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Figura 6 : ADC Prescaler Select

Exactitatea maximă a conversiei se obţine, dacă frecvenţa de tact a modulului ADC e cuprinsă între 50...200kHz. respectiv coeficientul scalării trebuie de ales în aşa mod, ca frecvenţa ADC-ului să se afle în acest diapazon. Dacă însă exactitatea conversiei e mai mică de 10 biţi e destul, se poate de utilizat o frecvenţă mai mare, mărind astfel frecvenţa măsurării. În modelele ATmega8x, ATmega16x, ATmega64x şi ATmega128x pentru acest scop se utilizează bitul ASCHM din registrul SFIOR. La setarea lui în ”1” viteza de conversie se măreşte. Însă concomitent creşte şi consumul de energie a microcontrollerului.

	7	6	5	4	3	2	1	0	
	–	–	–	–	–	ADTS 2	ADTS 1	ADTS 0	ATmega64x
Citire(R)/ Scriere(W)	R	R	R	R	R	R/W	R/W	R/W	
Valoarea inițială	0	0	0	0	0	0	0	0	

Figura 7 : Formatul registrului ADCSRB

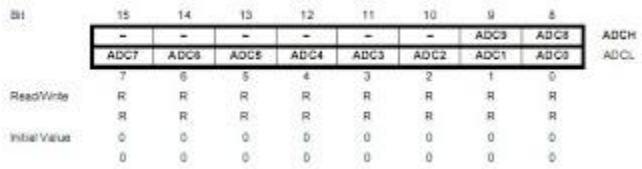

Figura 8 : Formatul registrelor ADCH şi ADCL

Aceşti regiştri conţin ultima valoare convertită, ADCH – biţii ADC9:8, ADCL – biţii ADC7:0. Cînd ADCL este citit datele din ADCH nu sunt modificate pînă cind nu se citeşte şi ADCH, adică este esenţial ca ambii regiştri să fie citiţi în acelaţi timp adică ADCL se citeşte înaintea ADCH.

Seria ATMEGA conţin un CAD mai complex, care are aceleaşi funcţii de bază dar şi anumite posibilităţi mai avansate ca:

- Şapte Canale de intrare diferenţiale;
- Două Canale de intrare diferenţiale cu amplificare 10x şi 200x;
- Ajustarea la stînga a regiştrilor ADCH:L (p-u rezoluţie de 8 biţi);
- Tensiune de referinţă selectabilă de 2.56V;

LM20

LM20 este un sensor, un circuit integrat de tip CMOS cu o precizie înaltă de măsurare a temperaturii cuprinse între -55°C și 130°C.Tensiune de alimentare a senzorului este cuprinsă între 2.4V și 5.5V.Funcția de transfer este aproape liniară,însă are o mica înclinare parabolică.Acuratețea la LM20 este de ±1.5°C la temperatura de 30°C.Eroarea crește linear cu temperature și ajunge la maximum ±4°C.

4.1.Schema-bloc:

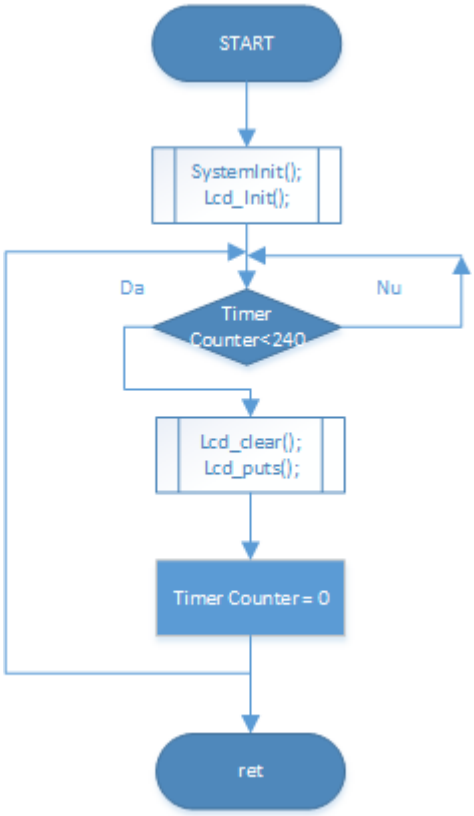

Figura 9 : Schema Bloc a funcției "main"

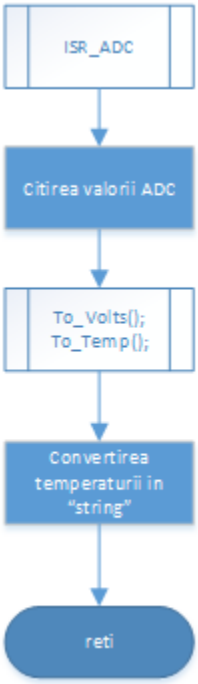

Figura 10 : Schema Bloc "ISR_ADC"

4.2.Schema Flux de Date:

Figura 1 : Flux de Date

4.3.Schema electrică:

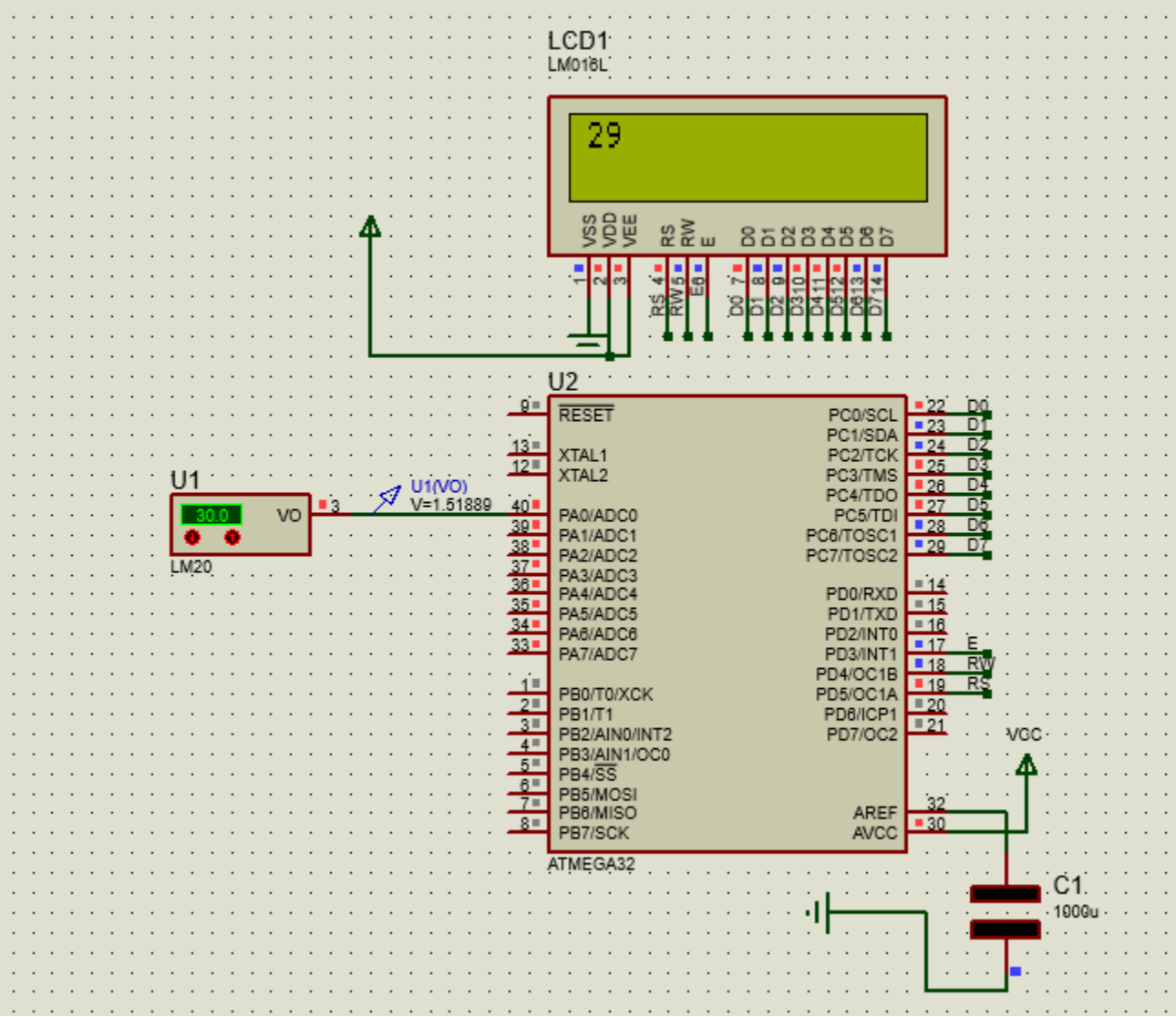

Figura 2 : Schema electrică în Proteus

5.Programul:

main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "lcd.h"
#include "lm20.h"

void SystemInit(void);
unsigned char the_low_ADC;
int ten_bit_value;
char str[10];

ISR(ADC_vect)
{
/*1.Citirea ADCL,are loc blocarea
reg ADC pentru scriere*/
	the_low_ADC = ADCL;
/*2.Citirea ADCH,are loc deblocarea
reg. ADC,pastrarea datelor(ADCH,ADCL)*/
	ten_bit_value = (ADCH << 2) | (the_low_ADC >> 6);
	ten_bit_value=To_Volts(ten_bit_value);

//get Temperature
	ten_bit_value=To_Temp(ten_bit_value,0);

	itoa(ten_bit_value,str,10);

	}

int main(void)
{
	SystemInit();
	Lcd_Init();
	sei();
    while (1)
    {
		if(TCNT0 > 240)
		{
			Lcd_clear();
			Lcd_puts(str);
			TCNT0 = 0;
		}

    }
}

void SystemInit()
{
/*---------TIMER0 INITIALIZATION----------*/
	TCCR0 = (1 << CS02) | (1 << CS00); //clk/1024
	TCNT0=0;
/*----------ADC INITIALIZATION------------*/
	//ADC PORT INIT
	DDRA=0x00;
```

```
	PORTA=0xff;

/*Alegerea canalului*/
	ADMUX = (0 << MUX0) | (0 << MUX1) | (0 << MUX2);
/*Select Ualim*/
	ADMUX |= (1 << REFS0) | (1 << REFS1);
/*Pozitionarea rezultatului*/
	ADMUX |= (1 << ADLAR);
/*Configurarea registrului ADCSRA*/
	ADCSRA |= (1 << ADIE) | (1 << ADIF) | (1 << ADATE) | (1 << ADEN) | (1 << ADPS2) |
(1 << ADSC);
	SFIOR = 0x00;
}

/*
```

Lm20.c

```
/*
 * lm20.c
 *
 * Created: 3/19/2017 1:16:36 PM
 *  Author: denis
 */

#include "lm20.h"

/*tabelul valorilor
(temperatura:voltaj)*/
int myLookUpTable[9][2]=
{
	{130,303},
	{100,675},
	{80,919},
	{30,1515},
	{25,1574},
	{0,1864},
	{-30,2205},
	{-40,2318},
	{-55,2485},
};


/*
convertarea valorii ADC
in Volataj.
Param.IN:ADC value;Type-int
Param.OUT:Voltage-value;Type-int
*/
int To_Volts(int val)
{
	return ((long int)val * ADC_VOLTAGE) / ADC_RESOLUTION;
}

/*Determinarea temperaturii
Param.IN-
```

```
1)Tensiunea
2)metoda utilizata(daca '1' efectueaza metoda
prin analiza intervalelor,daca orice alta val.
atunci efectueza calculul dupa expresia din datasheet
Param.OUT-Temperatura*/
int To_Temp(int voltageValue,char method)
{
	int tmp,i;
	int dTemp;
	float dVoltage;

	for(i=0; i < 8; i++)
	{
		if(i==0)
		tmp=myLookUpTable[i][0];


		if(voltageValue-1 < myLookUpTable[i][1])
		{
			tmp = myLookUpTable[i][0];
			//return tmp;
			break;
		}
	}
	if(i==8) tmp=myLookUpTable[i][0];

	//a2 parte,cercetam segmentul obtinut
	if(method == 1)
	tmp = BisectMethod(voltageValue,myLookUpTable[i-
1][1],myLookUpTable[i][1],myLookUpTable[i-1][0],myLookUpTable[i][0]);
	else
	tmp = FunctionalMethod(voltageValue);
	return tmp;
}

/*
Functia data cerceteaza intervalul de temperaturi
p/u gasirea temp. precise.
Param. IN-
1)volajul de la senzor
2),3)valorile voltajului la extremele intervalului pozitia(i-1,i)
4),5)valorile voltajului la extremele intervalului pozitia(i-1,i)
In caz de eroare returneaza '1000';
Param.OUT-Temperatura(int);
*/
int BisectMethod(int vV,int v1,int v2,int t1,int t2)
{
	float vMed,tMed;

	vMed = v1;
	tMed = t1;

	// |vMed-vV| > 10 - eroare 10mV
	while(abs(vMed-vV) > 10)
	{
		//Injumatatirea segmentului
		vMed = (v1 + v2) / 2;
		tMed = (t1 + t2) / 2;
```

```
			/*Gasirea extremelor a segm. injumatatit*/
			if(vV > vMed) {
				v1 = vMed;
				t1 = tMed;
			}
			else if(vV < vMed) {
				v2 = vMed;
				t2 = tMed;
			}
			else if(vV == vMed) {
				return tMed;
			}
			else return 1000;//error
		}
		return tMed;
}


/*Expresia din lm20 datasheet p/t
calcularea temperaturii
Param.In-Voltajul
Param.OUT-Temperatura*/
int FunctionalMethod(int vV)
{
		return (-1481.96f + sqrt(2.1962f * 1000000u + (1.8639f - (float)vV/1000)/(3.88f *
0.000001f)));
}
```

Lm20.h

```
/*
 * lm20.h
 *
 * Created: 3/19/2017 1:16:25 PM
 *  Author: denis
 */


#ifndef LM20_H_
#define LM20_H_

#define ADC_VOLTAGE 2560 /*in mV*/
#define ADC_RESOLUTION 1023 /*10bits*/
int To_Volts(int);
int BisectMethod(int,int,int,int,int);
int To_Temp(int,char);
int FunctionalMethod(int);




#endif /* LM20_H_ */
```

6.Soft-uri utilizate pentru sarcina dată:

- Proteus 8.5
- Atmel Studio 7.0
- Microsoft Visio

7.Concluzie

La efectuarea acestei lucrări am utilizat 2 metode pentru determinarea temperaturii.Prima metodă consta în utilizarea tabelului de valori a tensiunii și temperaturii din datasheet,apoi folosind metoda bisecției pentru intervalul respectiv determinam temperatura.A 2 metodă constă în utilizarea formulei din datasheet.Am observat că acuratețea la metoda 2 este un pic mai mare(aprox. ±1.°C), în plus implementarea este mai simplă.

8.Referințe

www.microlab.club

http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf

http://www.avrfreaks.net/

http://www.ti.com/product/LM20