Thermostat system

Student: Veliciu Vlad

Group: 2031

Coordinator: Șl. dr. ing. Pătărău Toma

Faculty: Electronics, Telecommunications, and Information Technology

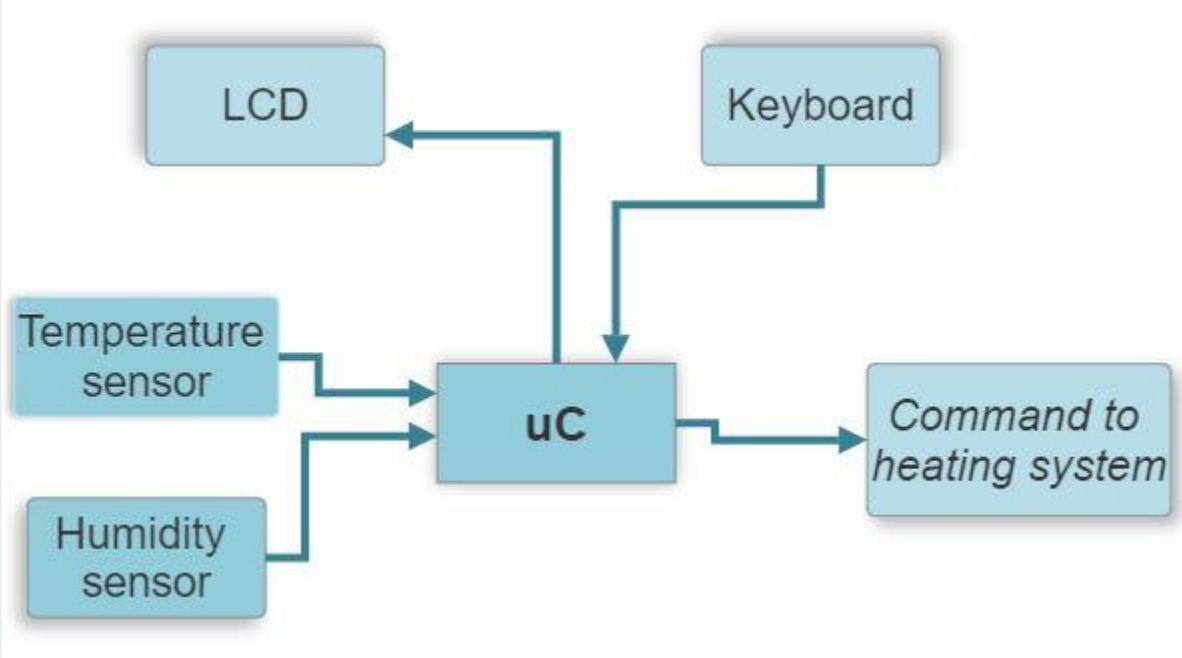
Contents

1. Introduction

The aim of this project is to simulate a thermostat system based on a microcontroller, having a temperature sensor which has an analog output, a humidity sensor having a digital output, an LCD where the information about the current temperature and the desired temperature is displayed. Also, an analog-to-digital converter is present in order to transform the output voltage from the analog temperature sensor into a digital format that can be used by the microcontroller. The microcontroller will further compare this temperature with the desired one and will choose whether or not to give information to the heating system to turn on.

The simulations are done in Proteus 8.13 and the code for the microcontroller is written in Keil µVision5, in assembly as well as in C language.

2. Block Diagram



(fig 2.1: block diagram)

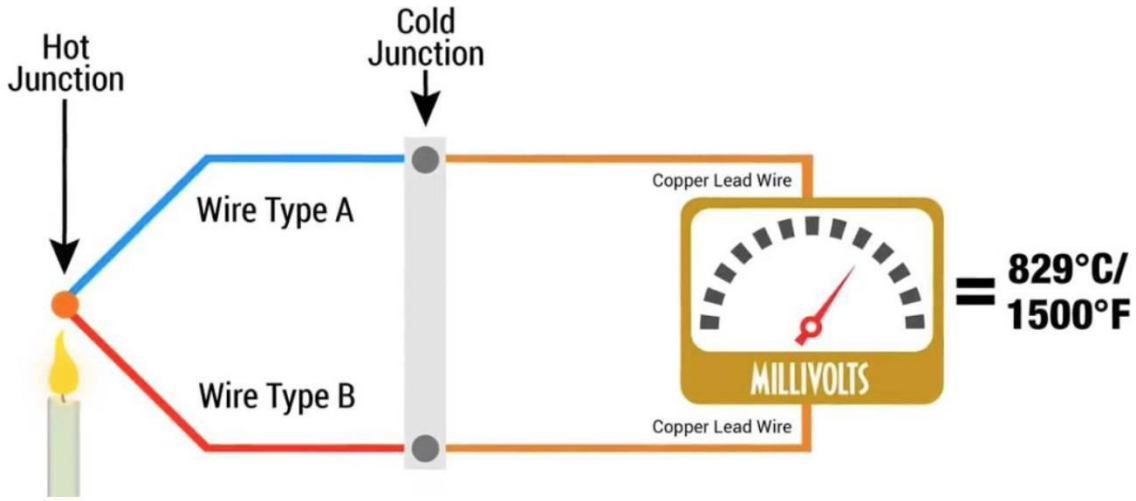
The block diagram shows the connections that are made with the microcontroller. As there can be seen, the microcontroller gets information from the temperature sensor, humidity sensor and the keyboard. The temperature sensor informs about the current room temperature and the keyboard represents the user input (e.g., the desired temperature). As outputs of the microcontroller there are the LCD for displaying information about the real and desired temperature, the humidity and the command to the heating system (activated when the real temperature is below the desired one).

3. The temperature sensor

3.1. Main types of temperature sensors

A. Thermocouples

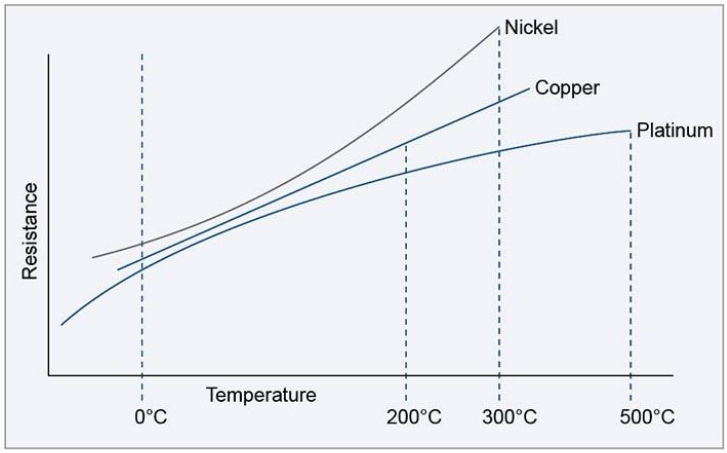
These sensors have a wide temperature range (-180°C - 2320°C) and quick response times. They are made by joining 2 dissimilar conducting metal wires, causing a Seebeck effect (a temperature difference of 2 different type of wires causes a voltage difference that is used to calculate the temperature). A disadvantage is the small output voltage which can cause problems in measuring the temperature. This means that precise amplification is required and that external noise over the wires can affect the result. Moreover, cold junction appears between the thermocouple wires and the wires of the external circuit that causes another Seebeck effect that must be compensated. There are integrated solutions (with precise ADC, low noise, and compensation of the cold junction). [2]



(fig 3.1.1: functioning of thermocouple)

B. Resistance temperature detector

This sensor is based on the fact that the resistance increases as temperature increases and the resistance vs temperature relationship is known for any material. Platinum is the most material used for building such a sensor because this material offers an almost linear response to temperature changes, is stable and accurate and offers a wide range. This sensor is suitable for low ranges (-200°C - 500°C). There are 2,3 or 4 wires configurations available, depending on the accuracy needed. [2]

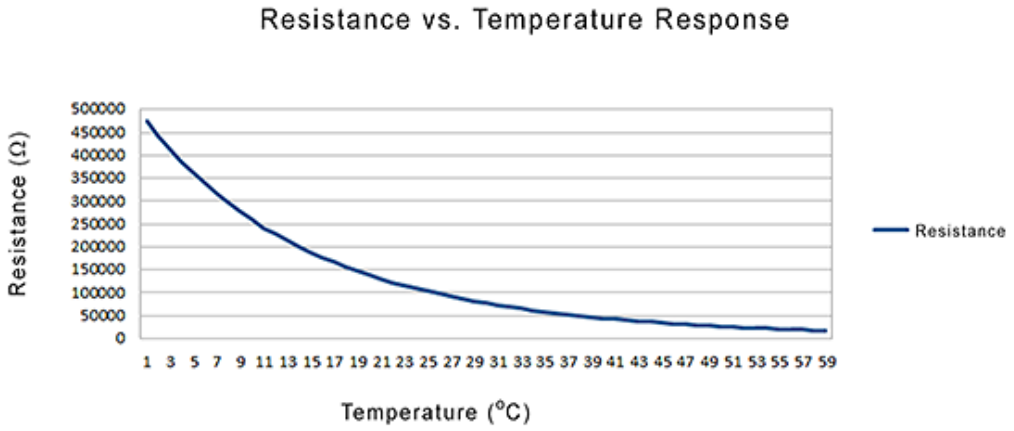


RTD Resistance versus Temperature

(fig 3.1.2: Resistance vs Temperature characteristic for different materials for RTD)

C. Thermistors

This type of sensor is like the previous type (dependency much stronger, a change in temperature causes change in resistance that is measurable, but most used thermistors are that whose resistances that increase when temperature decays -> negative temperature coefficient thermistor). They are cheaper but less accurate than resistance temperature detectors and have a non-linear resistance vs temperature characteristic. It is commonly used in a voltage divider configuration with an ADC. [2]



(fig 3.1.3: Resistance vs Temperature characteristic for thermistors)

D. Semiconductor based ICs

This sensor works with dual ICs. They contain 2 similar diodes (or transistors) with temperature-sensitive voltage and current characteristics to measure the temperature changes effectively. They give a linear output, are sensitive, but are less accurate at 1°C-5°C and have slow responsiveness (5s - 60s). Range is between -70°C - 150°C. [2]

3.2. *Choosing the temperature sensor: Semiconductor based ICs temperature sensor*

The chosen type of temperature sensor is semiconductor-based IC temperature sensor because of its advantageous size (regular very small size), enough range of temperature for this application (0°C-50°C) and generally gives a digital output.

Types of semiconductor-based ICs sensors with analog output: [1]

No.	Name	Temp range (°C)	Gain (mV/°C)	Supply voltage (V)	Supply current when measuring (µA)	Accuracy (°C)	Price (RON)
1.	TMP236AQDBZRQ1	-10 +125	19.5	3.1…5.5	14.5	+/-2.5	6.53
2.	MAX6611AUT+T	-40 +125	16	4.5…5.5	150	+/-1	23.51
3.	MAX6605MXK+T	-55 +125	11.9	2.7…5.5		+/-0.75	10.00
4.	TMP235A2DBZR	-40 +150	10	2.3…5.5	15	+/-0.5	6.93
5.	LMT70YFQT	-55 +150	5.194	2.2…5.5	12	+/-0.2°C	12.73

Considering these five sensors, there can be made some observations regarding which sensor would fit best for this application. Firstly, the required supply voltage is for all sensors equal or below 5V and also the supply current has acceptable values, which is suitable for this application. There can be observed that the parameter "Temperature resolution" would be suitable for the desired temperature range of 0°C-50°C in the case of all sensors. The accuracy though varies from one sensor to another. Knowing that the application refers to a thermostat that regulates the temperature in a room, there is desired that the variance from the real temperature that is measured by the sensor is as low as possible. Consequently, the last one would be more suitable. Moreover, it has to be taken into consideration the gain, so the variation in voltage that would sense of the temperature has changed or not. The gain should be high, so that a difference in voltage can be corrected interpreted as a difference in temperature. However, if the accuracy and gain are taken as decisional criteria between the sensors, the last sensor would be more suitable, although it has a low gain of only 5mV/°C, but its accuracy is high. If the 5mV gain can be successfully measured, this sensor could be used as a temperature sensor in this application, If the accuracy of the measurement of the voltage difference is not so high, the 4th sensor could be used as well.

In conclusion, the sensor used in this application would be **LMT70YFQT.**

(fig 3.2.1: the LM70YFQT sensor)

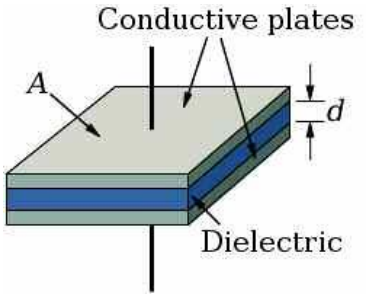
4. The humidity sensor

4.1. Main types of humidity sensors

A. Capacitive

This sensor measures the relative humidity. It places a thin strip of hygroscopic polymer film between 2 electrodes. The capacity of the metal oxide changes with the atmosphere's relative humidity, this material being hygroscopic. So, the capacitance where this material is used as a dielectric will change because of the dependence of the dielectric to the relative humidity (a function of the ambient temperature and the water vapors pressure). [3]

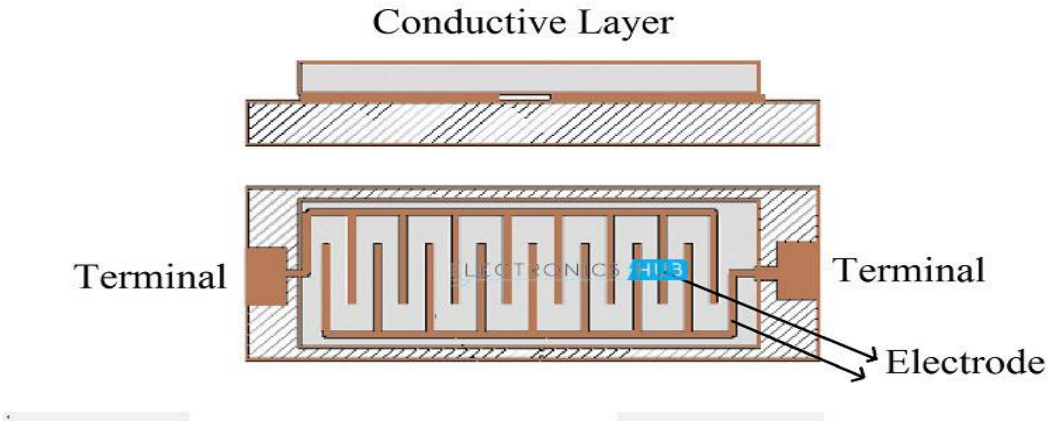
$$C = \varepsilon * \frac{A}{d};\ \varepsilon = dielectric\ constant$$



(fig 4.1.1: structure of a capacitor)

B. Resistive

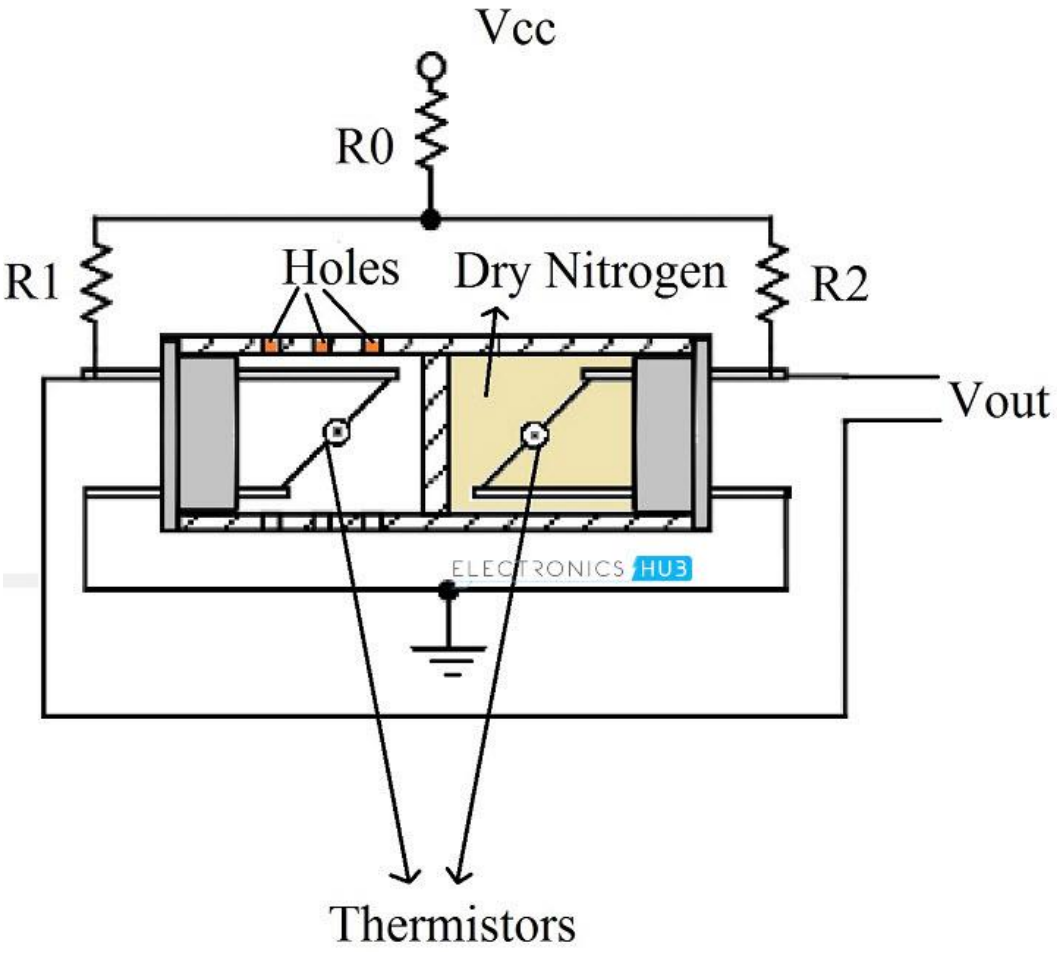
This type of sensor is made up of materials with relatively low resistivity and the resistivity varies with humidity in an inverse proportional way. The low resistivity material is placed on top of the two electrodes. The contact area between the electrodes and the material has to be increased as much as possible and the resistivity of the material changes when the top layer absorbs water. Commonly used materials are salt, conductive polymers.



(fig 4.1.2: structure of a resistive humidity sensor)

C. Thermal conductivity

This type of sensor is used to measure the absolute humidity. It measures the thermal conductivity of dry air and air with water vapors and the difference is referred to as absolute humidity. Two thermistors with negative temperature coefficients are used. One is isolated in a room filled with dry Nitrogen and the other one Is exposed to open air through small venting holes. When powering on the circuit, the resistance of the two are measured and the difference is direct proportional to the absolute humidity. [4]



(fig 4.1.3: structure of a humidity sensor based on thermal conductivity)

4.2. Choosing the humidity sensor: Capacitive humidity sensor

This type of sensor measures the relative humidity by measuring the changing capacitance.

Types of capacitive humidity sensors with digital output: [1]

No.	Name	Range (% RH)	Supply voltage (V)	Supply current when measuring (µA)	Accuracy (% RH)	Price (RON)
1.	SHT40-AD1B-R3	0 100	1.08…3.6	320	+/-1.8	12.57
2.	DHT11	20 90	3…5	500	+/-5	10.85
3.	SHT31-DIS-B2.5kS	0 100	2.4…5.5	600	+/-2	24.21
4.	HS3004	0 100	3.3…5.5	7	+/-3.5	12.23
5.	HS3001	0 100	3.3…5.5	24.4	+/-1.5	32.92

From these sensors, it can be seen that all would provide a good power consumption (V*I) for this application. The range also is wide enough for displaying the humidity in a room (the DHT11 has a slightly narrower range, but acceptable though). The accuracy although is different from one sensor to another and if the price is also taken into account, the conclusion would be that the first sensor is more suitable.

Consequently, the humidity sensor used would be **SHT40-AD1B-R3**.

(fig 4.2.1: the SHT40-AD1B-R3 sensor)

5. Connecting the analog temperature sensor

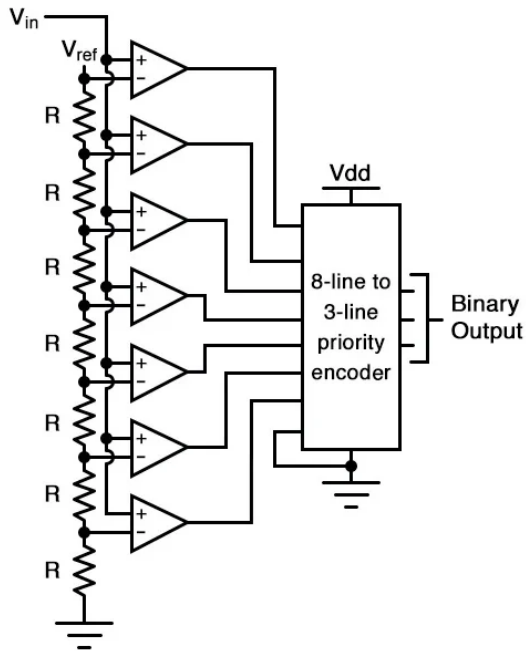
For connecting the temperature sensor which has an analog output to the microcontroller, between the sensor and the microcontroller there has to be placed an analog-to-digital converter (ADC) and, if necessary, also an amplifier. The amplification of the amplifier depends on the gain of the sensor (given in mV/°C) and the voltage corresponding to a LSB of the ADC, so that the analog-to-digital converter is able to detect any change noticed by the temperature sensor.

First, the analog-to-digital converter on 8 bits has to be chosen. This means that the converter is able to give at the output 256 (2^8) numbers.

5.1. Types of ADC

A. Flash:

It uses a number of n-1 (n=number of bits) comparators, comparing the input analog voltage to a fraction of Vref. This converter has a high conversion time (depending on the propagation time), but a low resolution that is affected by the non-idealities of the components (e.g. offset of the comparators) [5].

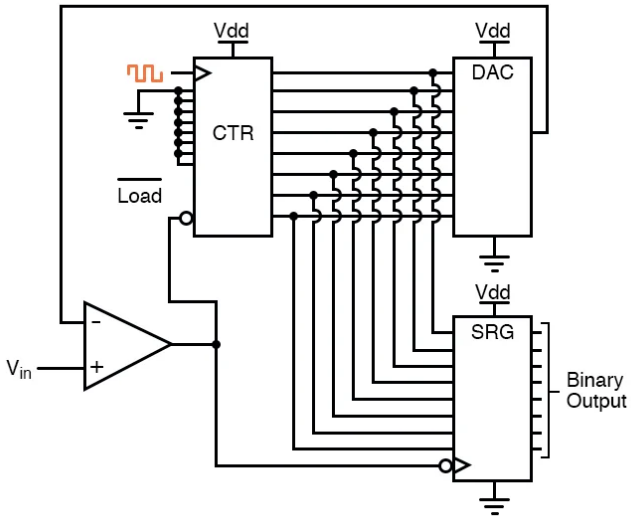


(fig 5.1.1: structure of flash ADC)

B. ADC implemented with counter:

This type of converter uses a counter that can only count up, together with a DAC and a comparator. The counter practically will count up until the corresponding voltage given by the

DAC reaches the analog input voltage (comparison made with the comparator between the output voltage of the DAC and the Vin) [5]. The main drawback is that after every conversion the counter is reset and the duration of the conversions vary, maximum being $2^n * T_{Clk}$, n=number of bits.



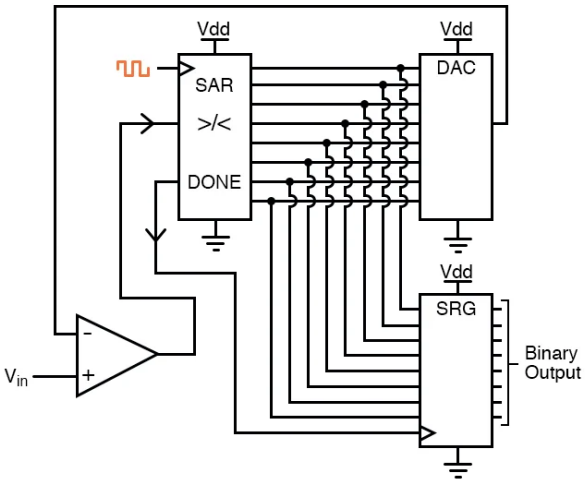
(fig 5.1.2: structure of ADC w counter)

C. ADC implemented with up/down counter:

It is similar to the previous version, but it is implemented with a counter that can count up and down. This means that the voltage corresponding to every conversion will try to stick to the analog voltage and the time will be the same for all conversions [5], having one at every T_{Clk}. The drawback is that the binary output is never stable, it changes at every clock pulse.

D. ADC with Successive Approximation Registers (SAR)

This converter uses a successive approximation register instead of the counter. The SAR will get the result by dividing in every clock pulse the analog interval in two and setting bit by bit the result. So, at every clock pulse it will know if it has been below or above the analog voltage and will set the bit accordingly [5] (1 if below, 0 if above). It facilitates thus a time that is the same for every conversion, $n * T_{Clk}$, n=number of bits. This represents an advantage because the digital device that will use the result will know the time interval in which a result is ready. It needs though a Sample and Hold circuit so that the analog signal will not vary during the conversion to achieve a reliable result.



(fig 5.1.3: structure of ADC w SAR)

Consequently, the type of ADC chosen will be ADC with SAR. The device is **ADC0808**. It has a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The ADC0808 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy, and repeatability, and consumes minimal power.[6] The main characteristics are:

Power: 15 mW (low)

Conversion time: 100 µs

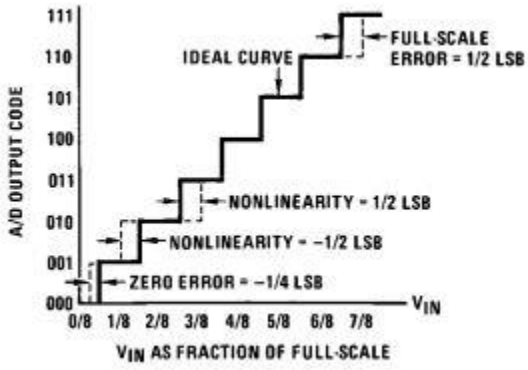
Voltage Supply: 4.5V … 6V (typical 5V) (VREF + = 5V and VREF - = 0V)

Total error: +/- 0.5 LSB

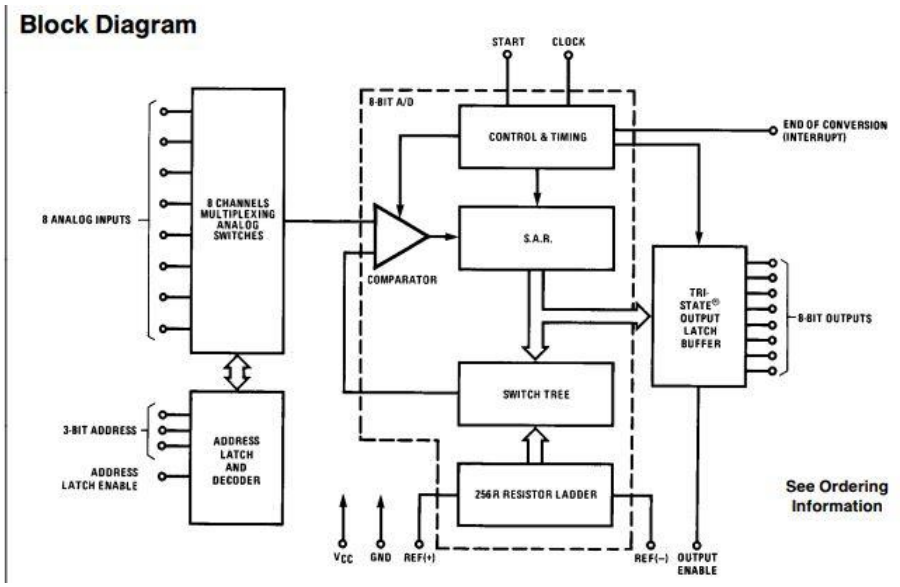
Input current: 0.5 µA

Output characteristics: the switch in the binary output is made by averaging, not by truncating.

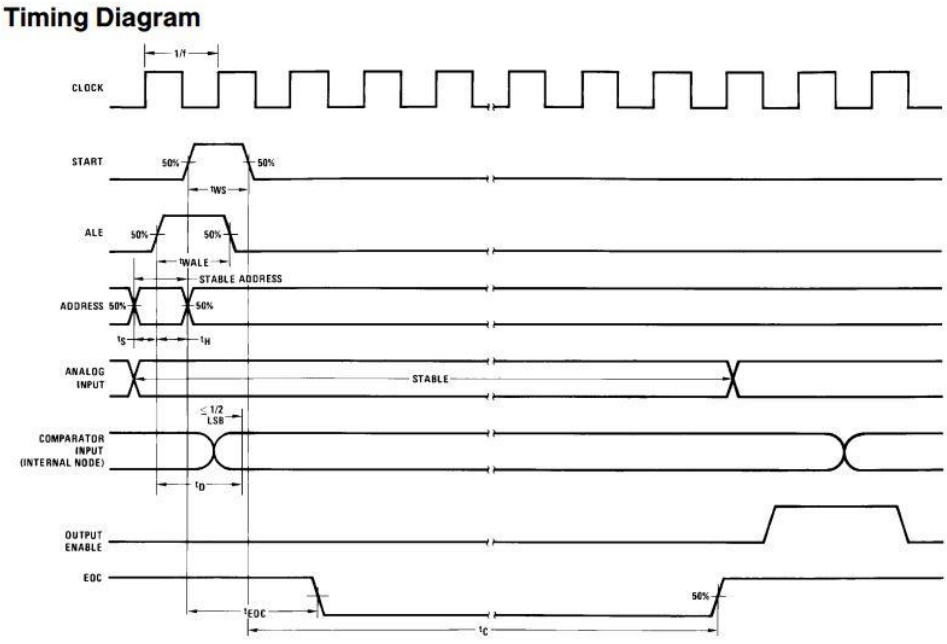
Clock frequency: 640 kHz [6]



(fig 5.1.4: transfer characteristics)



(fig 5.1.5: internal structure)



(fig 5.1.6: timing for ADC)

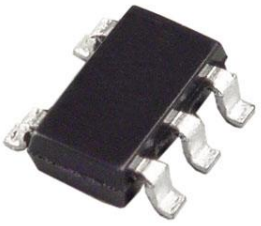
From *fig 5.1.6* there can be seen that, in order to start the conversion on the *START* pin, there have to be a pulse of width equal to the period of the clock signal and the signal *EOC* will signalize the end of conversion only when it gets from LOW level to HIGH level. The *OE (Output enable)* signal specifies when it is high that the result can be read from the output lines.

Knowing that the voltage corresponding to the least significant bit (input analog voltage that senses a change) is $V_{LSB} = \frac{V_{FS}}{2^N} = \frac{V_{Ref}}{2^8} = 20mV\ (0.0195V)$ and that the sensor has a gain of approximately 10 mV/°C [7] (sensor chosen for simulating in Proteus is **LM45**, as it has characteristics close to the one chosen previously). Also, the voltage corresponding to 0°C is 0V and that corresponding to 50°C gives 0.5V at the output of the sensor [7]. Hence in order for the

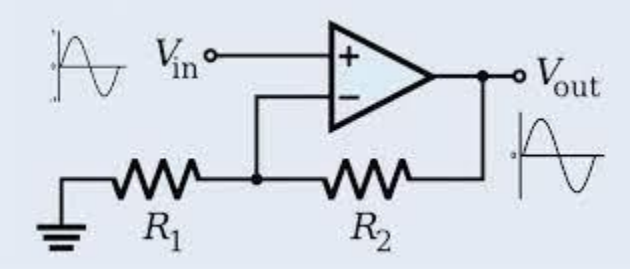
ADC to give a result that covers the interval of interest for this application, namely 0°C -> 00h and 50°C -> FFh (255), the output voltage from the sensor needs to be amplified before it enters the ADC. The voltage that gives FFh at the output of the ADC is 255*VLSB = 4.99V. This means that the output voltage from the sensor (its maximum value of 0.5V) needs to be amplified 4.99/0.5=9.98 times (approximatively). The next step is to choose an OpAmp that will be put in a non-inverting configuration to achieve this gain.

The most important aspect when choosing an operational amplifier to be used as an amplifier in this configuration is related to the offset voltage and bias currents which can cause an error at the output, especially when there are small voltages such as it is the case in this application. The effect of the biasing currents can be reduced by having equal resistances seen at both inverting and non-inverting input of the OpAmp. Consequently, the offset voltage represents a problem, such that the main criteria when choosing the OpAmp would be that it should have low offset voltage. The response time of the sensor is about 30µs, which is sufficient for the application, as the temperature in a room doesn't vary so fast. This means also that the slew rate of the OpAmp doesn't affect the relation voltage-temperature. Also, the OpAmp should be rail-to-rail (Vout can reach as high as the power supplies) and single-power one.

The chosen OpAmp is **AD8603** [1]. It has a very low offset voltage (50µV), which would give at the output an error of approx. 500µV (at an amplification of 10). This is far smaller than VLSB, meaning that the correct conversion of the ADC is not affected. Furthermore, it has single-supply operation (1.8V-5V), is a precision amplifier (can amplify small voltages, as it is the case for the voltages at the output of the sensor) and has a high CMRR of 100dB (it can reduce very effectively the noise that can appear between the sensor and the amplifier). Also, it is a rail-to-rail amplifier. [8]



(fig 5.1.7: AD8603)



(fig 5.1.8: schematic of a non-inverting amplifier)

The configuration from above will be used around the AD8603 amplifier. Knowing that the desired gain is about 9.98 and the formula for the gain for the non-inverting amplifier:

$$A = 1 + \frac{R_2}{R_1} = 9.98 \Rightarrow\ R_2 = 8.98 * R_1 \Rightarrow take\ R_1 = 1k\Omega\ and\ R_2 = 8.98k\Omega$$

5.2. *Simulating in Proteus*

Connecting the pins of the ADC0808 for simulation in Proteus:

IN0…IN7 are the inputs where the analog signal is applied.

ADD A, ADD B, ADD C are the addresses that select the corresponding input (e.g. if A=B=C=LOW, then IN0 is selected and so on)

ALE is required to load the selected lines into the ADC (put to 0 as only one line is used)

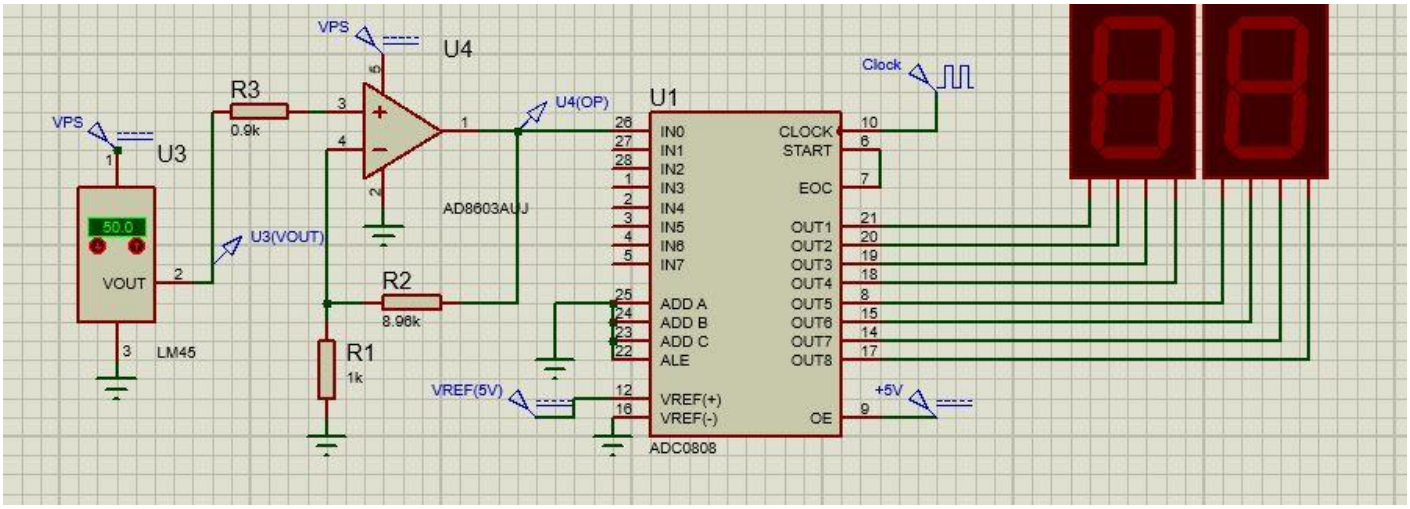
VREF+, VREF- are the supplies (+5V/0V)

Clock: clock signal at a frequency of 640kHz

START and EOC pins are connected together so that continuous conversion is allowed.

OUT1…OUT8 (MSB…LSB) are connected to 2 7-segment display.

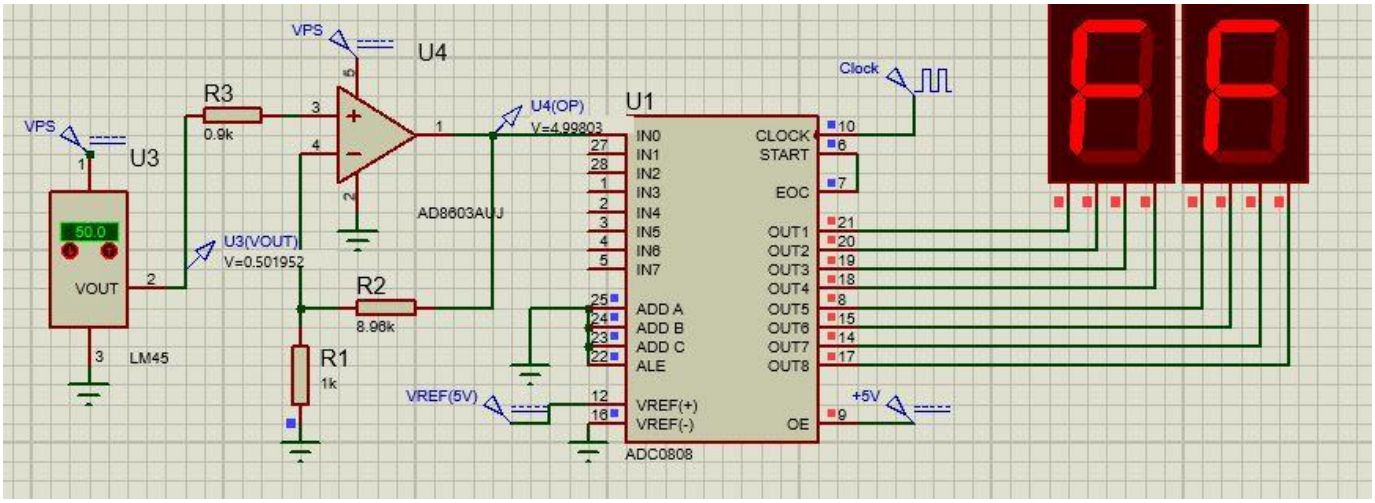
OE connected to 5V to ensure that every conversion result is given at the output every time.



(fig. 5.2.1: schematic in Proteus)

As there can be seen on the schematic, the value of the resistance R2 is slightly different then the value theoretically computed because of other non-idealities. Also, a resistance R3 has been added (whose value is the value of R1||R2) in order to reduce the effect of the bias currents.

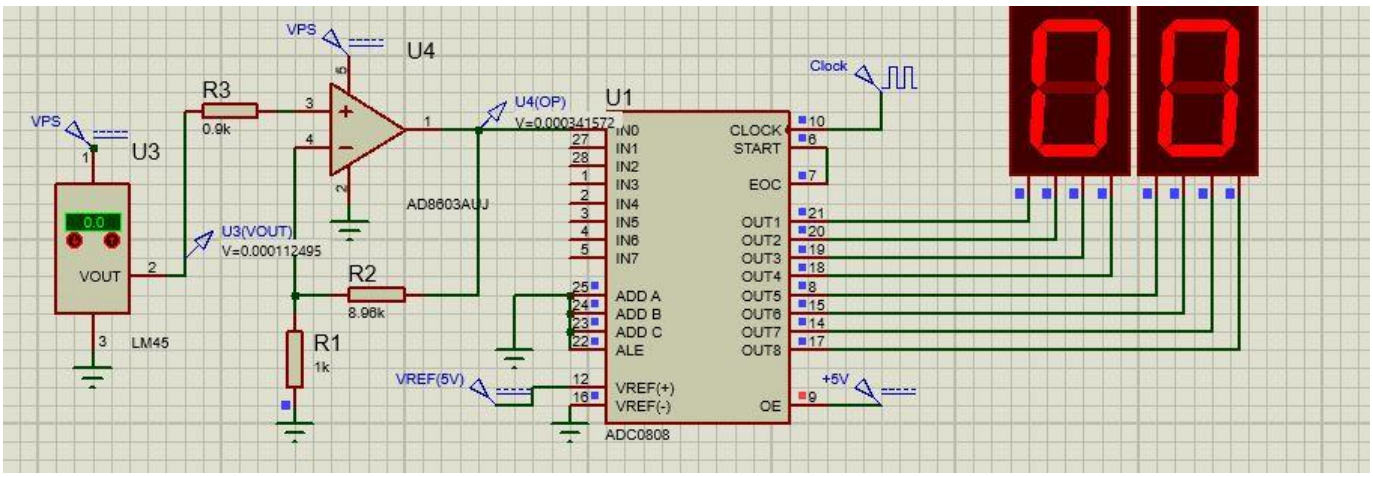
1st case:



For 50°C => Vout_sensor=0.5V => Vout_opamp=4.99803V => for ADC:

$$Vout_{opamp} = N * V_{LSB} => N = \frac{Vout_{opamp}}{V_{LSB}} = \frac{4.99803V}{0.01953V} = 255$$

2nd case:



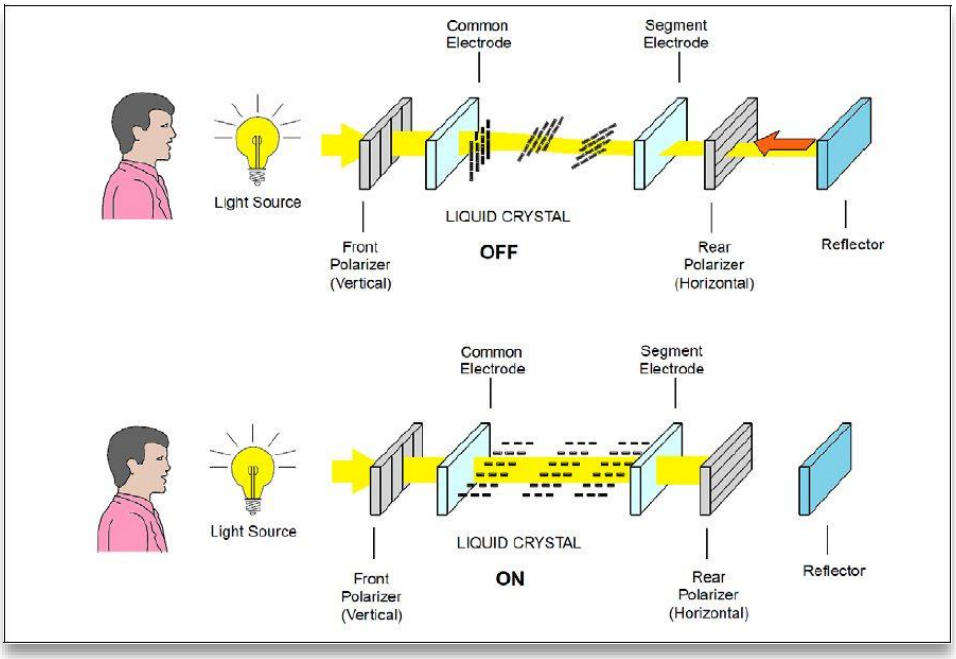
For 0°C => Vout_sensor=0V => Vout_opamp=0V => for ADC:

$$Vout_{opamp} = N * V_{LSB} => N = \frac{Vout_{opamp}}{V_{LSB}} = 0$$

6. Functioning of the LCD

6.1. How does the LCD work?

LCDs (Liquid Crystal Displays) work on the principle of blocking light. The molecules inside the crystal tend to untwist when electrical current is applied to the liquid crystal. The angle of the top polarizing filter changes, as well as the angle of the light that passes through the polarizing glass. Because at the back of the LCD there is a mirror (reflector), when there is no current the light can pass through the liquid crystal almost at the same angle and through the reflection of the mirror, there is a bright spot created on the LCD. When there is current through the electrodes, the angle of the light changes in such a way that it doesn't reach the mirror (the horizontal polarizer will block it) and thus a dark spot is created on the LCD. [9]



(fig 6.1.1: functioning of the LCD)

For displaying the information to the user, an LCD is needed. Most LCDs have 16x2 or 20x2 displays (16 characters on 2 lines or 20 characters on 2 lines). The standard for communication for the most LCDs is HD44780U which is the controller chip that receives data and sends it to the LCD. It can typically have 4 or 8 lines for data bus and 3 lines for control.

The chosen LCD for this application is one with 16x2 display, **LM016L**, which also has the communication standard HD44780U built in it. The signals (which are transmitted on pins) involved in the writing on the LCD as well as their function are explained next (in the order they appear on the LCD) [10] :

- *RS (Register Select)*: when it is *low (0)*, the data is interpreted as a command or instruction for the LCD (position cursor, clear screen, increment cursor and others) and when it is *high (1)* the data is interpreted as text which is to be displayed on the LCD. There are two 8-bit registers in the HD44780U: instruction register which is selected when RS=0 (commands) and data register, selected when RS=1 (data to display).
- *RW (Read/Write)*: when it is *low (0)*, the information on the data bus is written to the LCD and when it is *high (1)*, it means that the information from the LCD is read.
- *EN (Enable)*: when sending data to the LCD, this line should be *low (0)*, then the other control lines and the data lines can be set. As the lines have the information on them, EN can be set *high (1)* for a minimum period of time (said by the datasheet, in this case, for writing, 230 ns is the minimum time for the width of the signal in high level and the rise/fall time 20 ns). After that period, EN has to be brought in *low (0)* state again.
- 8 data lines denoted D0…D7 (there can be also used only D0…D3 lines, meaning 4-bit mode)

The *RW* and *RS* signals need to be set accordingly (depending on the application), before the *EN* signal is set. Also, the data hold time refers to the minimum time the data lines should be kept unchanged (with the desired data on them) after the *EN* signal is switched to LOW and the data set-up time represents the minimum time the data lines should be kept unchanged before the *EN* signal switches to LOW.

Beside these lines which are pins on the LCD, there are 3 pins more that are actually the first 3 pins. These pins are explained below (in the order they appear on the LCD):

- *Vss* represents the ground connection (0V)
- *Vdd* represents the supply connection (typically 5V)
- *Vo (Vee)* represents a pin that controls the contrast of the LCD (usually connected through a potentiometer that can vary the voltage seen by Vo between 0V and 5V). This pin is not mandatory to be connected [11].

The next table shows the pin connection and their function in short (from datasheet of LM016L):

INTERNAL PIN CONNECTION

Pin No.	Symbol	Level	Function	
1	V_{SS}	–	0V	Power supply
2	V_{DD}	–	+5V	
3	V_O	–	–	
4	RS	H/L	L: Instruction code input H: Data input	
5	R/W	H/L	H: Data read (LCD module→MPU) L: Data write (LCD module←MPU)	
6	E	H, H→L	Enable signal	
7	DB0	H/L	Data bus line Note (1), (2)	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L		
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

(fig. 6.1.2: pin configuration of LM016L)

6.2. Commands for the LCD

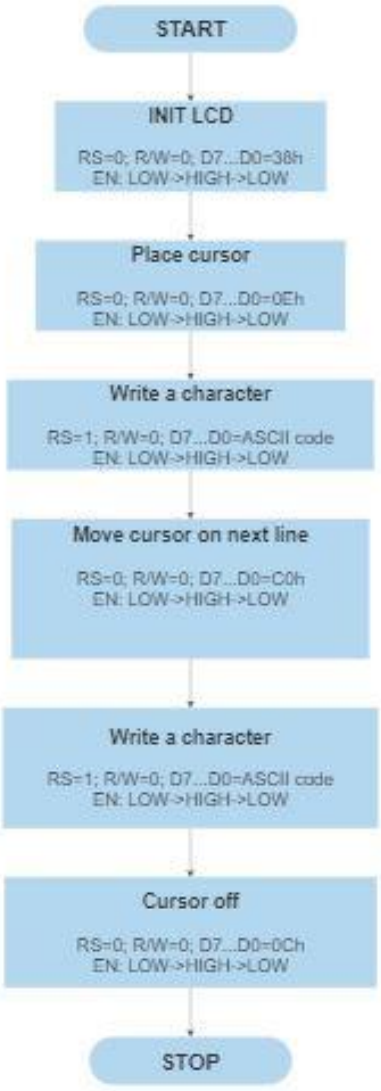
There are many commands that can be given to the LCD, but there are also commands that need to be given before using the LCD (initialization). The table below shows how the signals should be set on the corresponding pins in order to execute a certain command (LCD is used in 8-bit mode, writing on 2 lines):

Instruction	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Hex code (D7…D0)
16x2 LCD (init)	0	0	0	0	1	1	1	0	0	0	38h
Place cursor	0	0	0	0	0	0	1	1	1	0	0Eh
Clear display	0	0	0	0	0	0	0	0	0	1	01h
Write a character	1	0	ASCII code for the character (D0=LSB and D7=MSB)								
Move cursor on line 2	0	0	1	1	0	0	0	0	0	0	C0h
Move cursor on line 1	0	0	1	0	0	0	0	0	0	0	80h
Shift cursor to the left	0	0	0	0	0	1	0	0	0	0	10h
Cursor off	0	0	0	0	0	0	1	1	0	0	0Ch
Display off, cursor off	0	0	0	0	0	0	1	0	0	0	8h

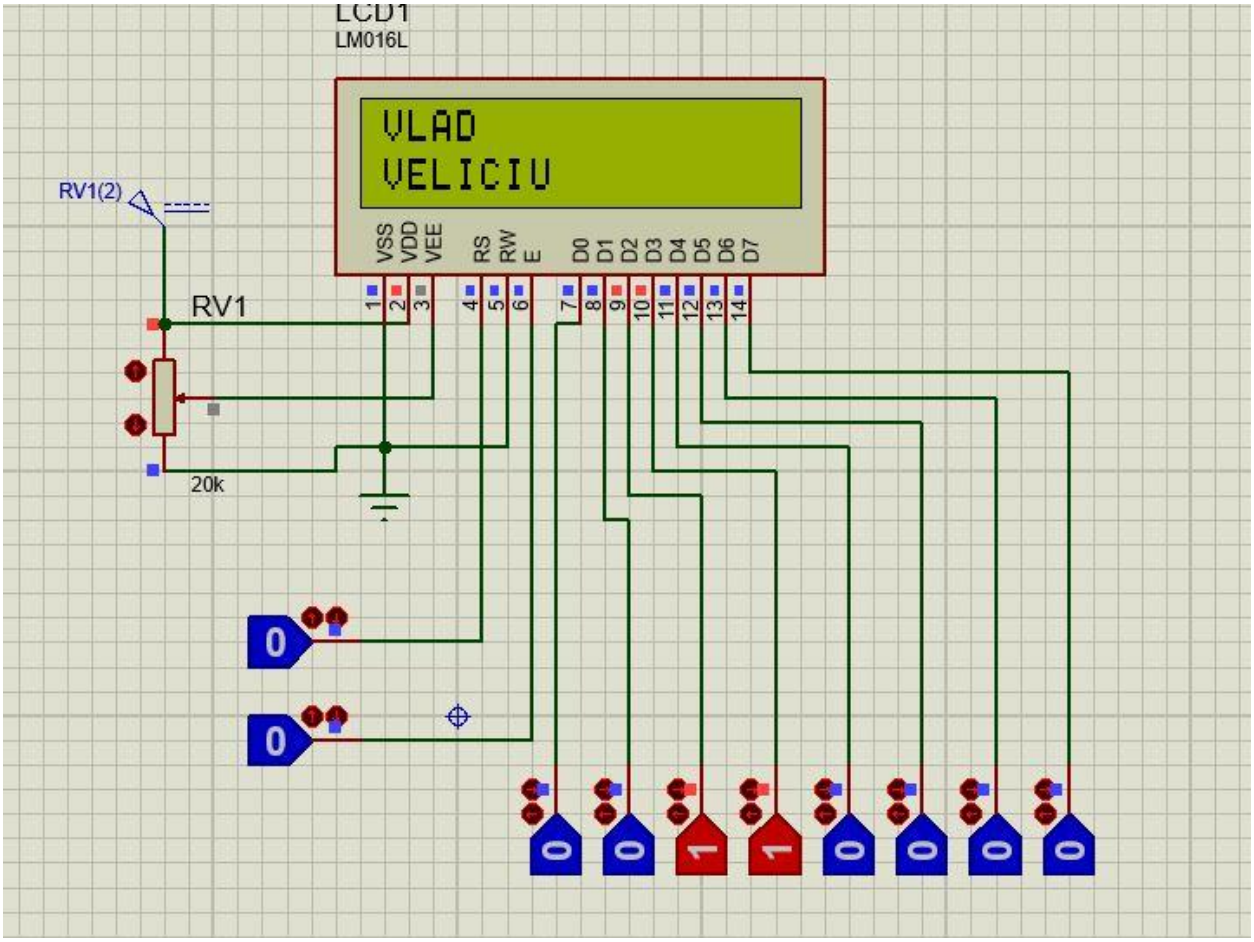
[11]

After setting the signals for every instruction, the EN signal needs to be put from LOW to HIGH, then after the minimum period again in LOW.

6.3. Implementation in Proteus



(fig. 6.3.1: flowchart for functioning of the LCD)

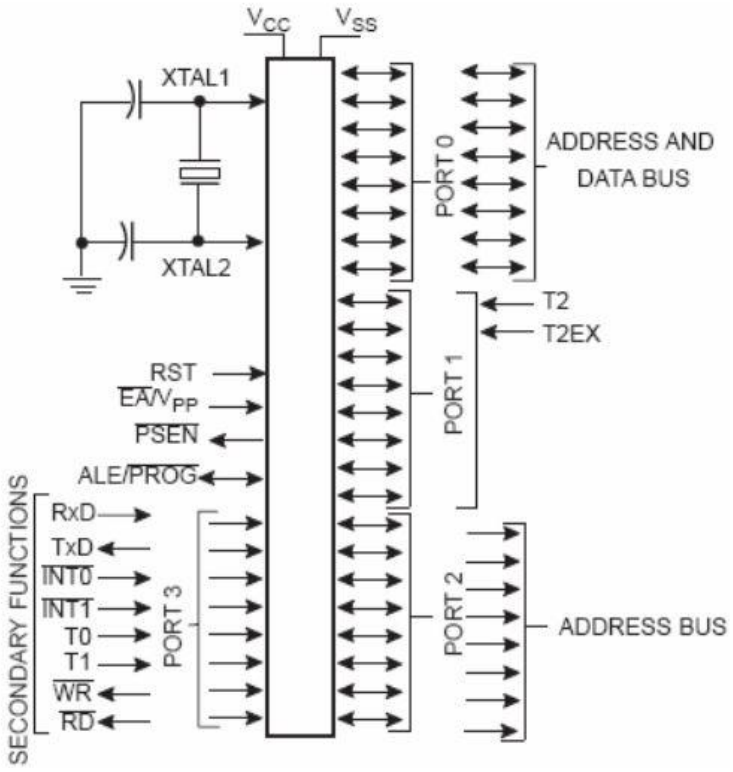


(fig. 6.3.2: implementation of LCD in Proteus)

It can be seen that Vdd (power supply) was connected to 5V, typical value from the datasheet, and Vss to ground. Also, the RW pin is connected to ground, as the LCD is used only for writing on it, not for reading from it. RS, RW and the data pins are connected to logic states in order to initialize the LCD and write a text on it. Vee pin is connected to a 20 kΩ potentiometer (typical value taken from datasheet of LM016L) and is used to adjust the brightness.

7. Choosing the microcontroller

The family of microcontrollers chosen for this application is 80C51. These microcontrollers usually have 4x8-bit I/O ports, full-duplex UART (Universal Asynchronous Receiver-Transmitter), 4kx8 internal ROM (program memory), 128x8 internal RAM (data memory), 3x16-bit counters/timers.



(fig. 7.1: ports of 8051)

The function of each pin is explained briefly below:

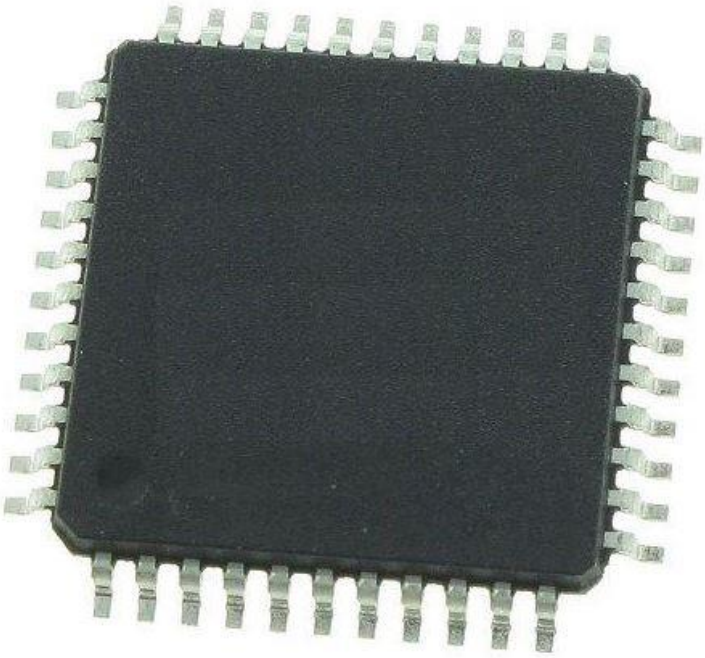
- Vss: ground
- Vcc: power supply
- P0.0…P0.7: I/O ports for address and data bus (low order). If it is used for simple data I/O operations, the pins need to be connected to externally pull-up resistors of 10kΩ when they are used as output pins (because P0 is open drain; without these resistors, the logic level cannot be predicted). If it is used for address/data multiplexing, there is no need for pull-up resistors, as a 74LS373 is connected to latch the addresses.
- P1.0…P1.7: on P1.0 is timer T2 and on P1.1 timer/counter 2 with capture function. The rest are I/O ports.
- P2.0…P2.7: connected to the address bus (higher order addresses).

- P3.0…P3.7: special function registers: RxD (serial input port), TxD (serial output port), INT0 (external interrupt), INT1 (external interrupt), T0 (timer 0), T1 (timer 1), WR (external memory write), RD (external memory read)
- RST: it can reset the microcontroller when it is HIGH.
- ALE/PROG (Address Latch Enable/Program Pulse): ALE is used for latching lower order addresses during access to external memory and PROG is used for programming the EPROM.
- PSEN (Program Strobe Enable): external program memory read strobe. Used if memory is accessed for code (external ROM memory). If it is low, a byte pf program code gets on data bus.
- EA/Vpp (External Access Enable/Programming Supply Voltage): if EA is LOW, the microcontroller executes the code from external program memory. If EA is HIGH, the code from internal ROM memory is executed (from addresses 0000h…0FFFh). Vpp receives the program supply voltage during programming the EPROM.
- XTAL1: input to clock generator circuit and inverting oscillating amplifier.
- XTAL2: output from inverting oscillating amplifier. [12]

Several microcontrollers from this family are presented next in comparison, in order to choose one suitable for the application of thermostat system: [1]

No.	Name	RAM size	ROM size	Voltage range (V)	Clock frequency (MHz)	I/O ports	ADC intern	Price (RON)
1.	**C8051F986-C-GU**	512x8	8kx8	1.8…3.6	25	17	yes	16.53
2.	**AT89LP828-20AU**	768x8	8kx8	2.4…5.5	20	30	no	15.15
3.	**AT89LP51RB2-20JU**	256x8	24kx8	2.4…5.5	20	40	yes	19.85
4.	**AT89LP51-20JU**	256x8	4kx8	2.4…5.5	20	36	no	17.08
5.	**AT89LP51RB2-20AU**	256x8	24kx8	2.4…5.5	20	40	yes	18.96

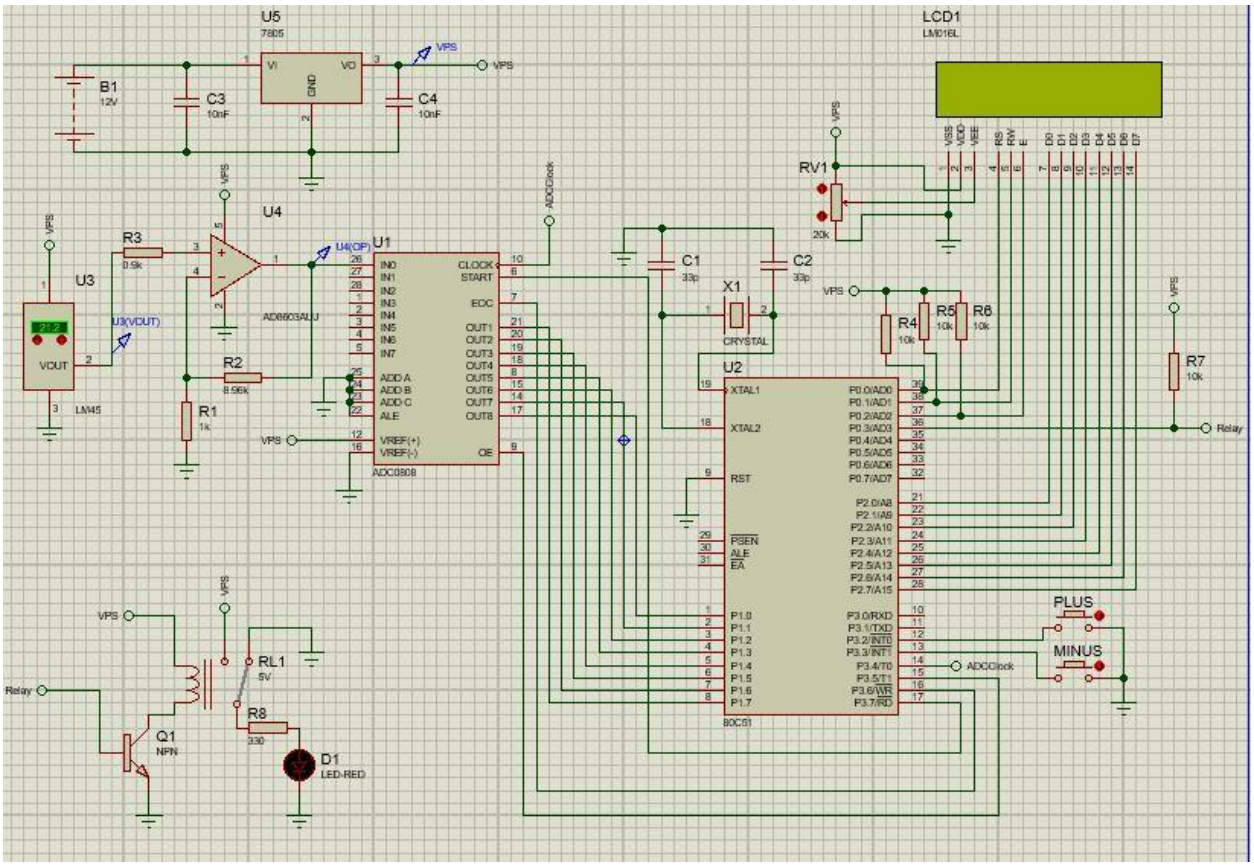
Taking into account all these 5 microcontrollers, there can be seen that all have similar voltage ranges and similar clock frequencies. The difference from the others represents the 1st one, having a smaller voltage range, but no sufficient input/output ports for this application (which requires 26 ports (8 for ADC data, 8 for LCD data, 3 for control ADC, 3 for control LCD, 3 for buttons, 1 for command to heating system)). Taking this aspect into consideration, there can be seen that only the last three ones would be suitable for this application. The 4th one, however, doesn't have an internal ADC and if there is a need to get rid of the external ADC placed in the schematic, this microcontroller wouldn't be suitable. If not, it is the best choice. If the price is taken into consideration and the need for an internal ADC, the **AT89LP51RB2-20AU** would be the best option. The data memory (RAM) and the code memory (ROM) are not factors that influence the decision, as the code and data needed for the application are of small size.



(fig. 7.2: the AT89LP51RB2-20AU)

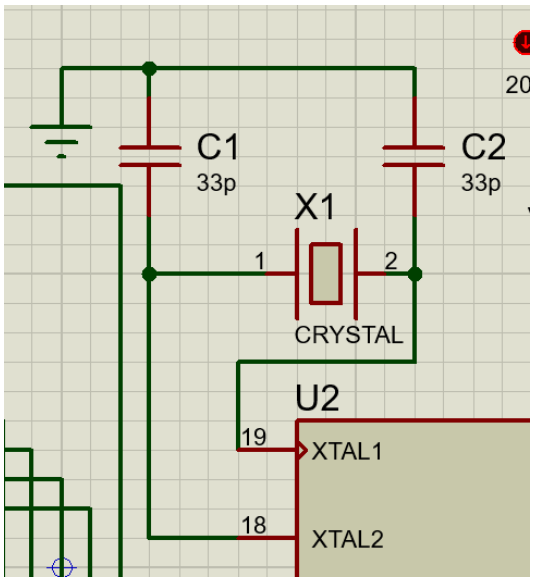
8. Implementing the schematic

8.1. Complete electrical schematic in Proteus



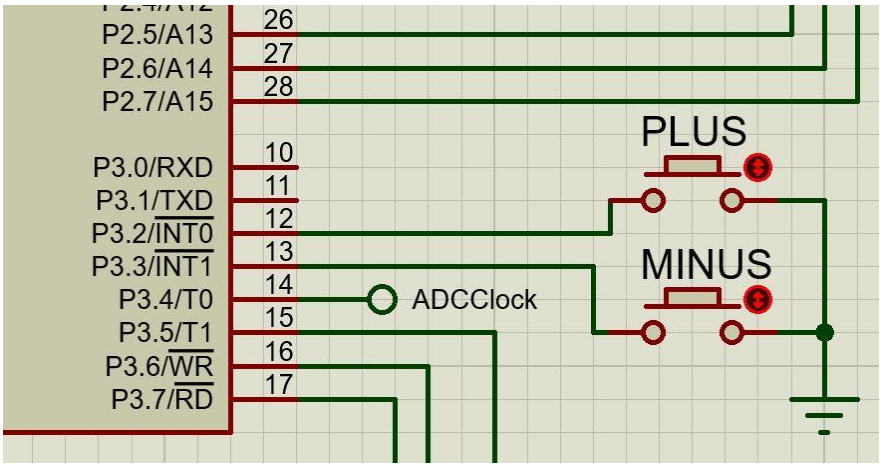
(fig 8.1.1: complete schematic)

The data coming from the ADC are connected on P1 (MSB: OUT1 from ADC in P1.7 and LSB OUT8 from ADC in P1.0). Thus, this port needs to be configured from code as being an input port. The command lines for the ADC (Start, EOC, OE) are connected on P3.7, P3.6, P3.5. Port P3.6 is an input port, as from the code the status of the pin EOC needs to be read to know when the conversion finishes. The data lines of the LCD are connected on port P2 (MSB: D7 at LCD in P2.7 and LSB: D0 at LCD in P2.0). The control signals of the LCD (RS, RW, E) are connected on P0.0, P0.1, P0.2. As these pins of the port P0 are used as output pins, pull-up resistors are needed.



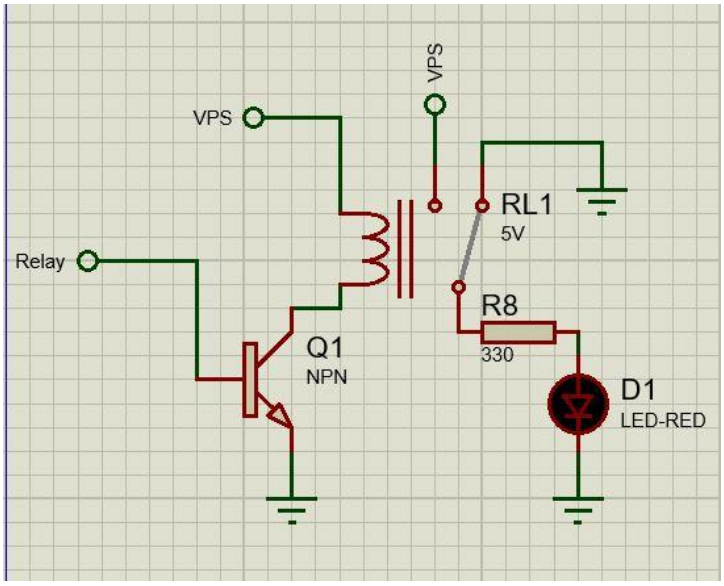
(fig 8.1.2: crystal oscillator)

There can be seen that at the pins XTAL1 and XTAL 2 a crystal oscillator of frequency 12MHz is connected. The crystal oscillator is very precise and suitable to give the clock frequency for the microcontroller. At XTAL1 there is the input of an inverting oscillating amplifier (internal) and at XTAL2 its output. Consequently, the two capacitors which are connected between the edges of the crystal oscillator and ground have the role to keep the phase of the signal unchanged, because the internal inverting amplifier realizes a phase shift of 180 degrees, the capacitor at the output of XTAL2 together with the crystal oscillator and the other capacitor realize a phase shift of another 180 degrees which means that the clock signal will not suffer any phase shifts. Moreover, the crystal oscillator can also generate beside the fundamental frequency other frequencies which can be filtered out by the capacitors, ensuring a stability in frequency of the oscillator. [13]



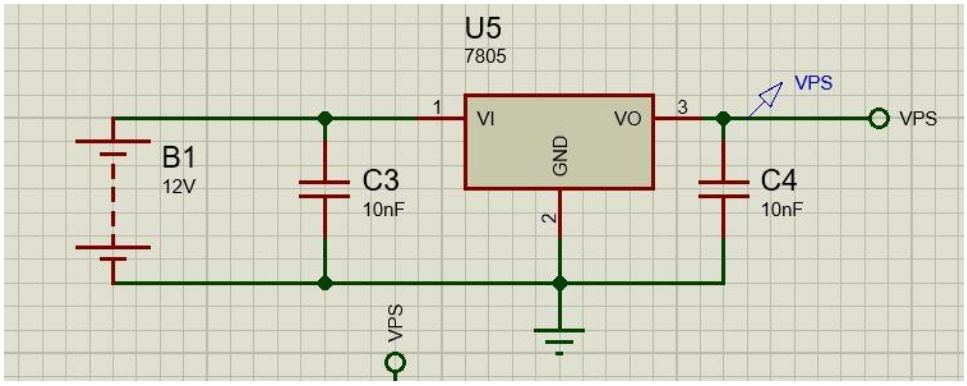
(fig 8.1.3: buttons)

The buttons to set the desired temperature are connected on P3.2, P3.3, as on these pins external hardware interrupts can be interpreted. There are 2 buttons: PLUS (for increasing the desired temperature), MINUS (for decreasing it). From the schematic, when the buttons are off, at the pins logic '1' is seen. When the buttons get pushed, the corresponding pin gets pushed to GND, so at that pin logic '0' is seen. In this way, the code can be configured in order to know whether a button is pushed or not.



(fig 8.1.3: relay)

On pin P0.3, the command to the heating system will be given in form of logic '1' to turn on and '0' to turn off (also pull-up resistor needed because this pin is used as output pin). This command will be given to a transistor that controls the operation of the relay. If the transistor is on (logic '1'), current will flow through the coil that has a magnetic core inside, pushing the metallic arch towards it. In this way, on the anode of the LED will be 5V, so that the LED gets on (and also the heating system that can be connected here, ensuring though a higher voltage for the left circuit part). When the transistor is off, the arch will be pushed away from the coil, turning off the LED.



(fig 8.1.4: voltage regulator)

To supply accordingly the devices (LCD, ADC, amplifier, temperature sensor, relay) from the system (with 5V), a voltage regulator that gives at the output 5V is used (it is supplied with a battery of 12V, having decoupling capacitors of 10nF at the input and output).

8.2. Code implementation

First, for displaying a message on the LCD, the settings of the signals that were previously presented need to be made (explanations are in the comments in the code below). To give commands or to display data to the LCD, the busy flag is used. For this, *D7* needs to be read (also *RS* LOW and *RW* HIGH, as there is a read from the LCD and *E* switched from HIGH to LOW). If *D7* is HIGH, it means that the LCD is still busy and if it is LOW, the next operation can be performed. Further, to display the temperature measured by the sensor and converted in digital form by the ADC, the following steps are made:

- First check if the conversion is ready. To do this, on START line a pulse needs to be created, then check if EOC goes from LOW to HIGH (this means end of conversion) and then OE is set HIGH. After that, the data from the output pins can be read and then OE is brought LOW again.
- To find the number: we know we can have from 00h…FFh possible hex values and the range of numbers from 0 to 50.
- To find digit of tens: the numbers 00h…32h correspond to temperatures between 0.00 and 9.08 (ideally is 31h, but experimentally it has been shown that 9.08 is represented by 32h). So, a division by 33h is performed and the quotient represents the digit of tens.
- To find the digit of units: the rest of the previous division has to be taken into consideration. The rest will then be divided by 5h (because there are 5 possible hex values for each unit: e.g.: 0.0, 0.2, 0.4, 0.6, 0.8) and the quotient represents the digit of units.
- To find the digit after the comma: take into consideration the rest of the previous division by 5h and multiply by 2 (as these digits are even numbers) and so the digits after the comma are obtained.

Also, the heater (transistor that drives the relay) is commanded by comparing continuously the real temperature with the desired temperature (digit by digit in ASM and by forming numbers of 3 digits in C). In ASM: the comparison is made by starting from the digit of tens till digit after comma, comparison which is made by dividing the real digits by the desired ones and then the quotient is compared with 0 (meaning that if the quotient is 0, the real digit is smaller than the desired one and if so, the heater is turned on). If the digits are equal, the comparison continues cascaded till digit after comma.

There is also necessary to implement 3 interrupts: one timer interrupt to generate the clock for the ADC (T0) and 2 external hardware interrupts for the two buttons. First, the interrupt enable register needs to be set accordingly to enable the 3 interrupts (0,1,2). The clock for the ADC is designed at a frequency of 100kHz and the timer 0 is used in mode 2 (8-bit timer, auto-charge).

Interrupt 1 is used here. For the buttons, the increment/decrement of temperature is made with a 0.2 step, starting from digit after comma till digit of tens. PLUS button is connected to INT0 and MINUS to INT1. For the buttons, the interrupts will be enabled on edge (for this the TCON.0 and TCON.2 bits need to be set).

The display of the real temperature is made continuously on the first row in an infinite loop and the display of the desired temperature is on the second row and is changed only when a button is pressed and after that the cursor is put back to the first row.

A. Code in ASM

```
    org 0000h
    ljmp main
    org 0003h;INT0 interrupt service routine
    ljmp buttonPlus
    org 000Bh;address of interrupt timer 0
    cpl P3.4;pin where the ADC Clock is connected
    reti
    org 0013h;INT1 interreupt service routine
    ljmp buttonMinus

    org 30h

main:
    mov IE,#10000111b;enable INT1, INT0, T0 interrupts
    acall clockADC
    setb TCON.2;enable INT1 when a transition from HIGH in LOW
    setb TCON.0;enable INT0 when a transition from HIGH in LOW
    mov P1,#0FFh;make P1 input port
    ;initial value for desired temperature
    mov r3,#2h;digit of tens of desired temp
    mov r4,#1h;digit of units of desired temp
    mov r5,#4h;digit after comma
    mov A,#38h;init LCD
    acall command
    mov A,#0Eh;place cursor
    acall command
    mov A,#01h;clear display
    acall command
    mov dptr,#1000h;read a string
    label:
        mov a,#0
        movc a,@a+dptr;read letter by letter
        jz finish
        acall display
        inc dptr
        sjmp label
```

(fig 8.2.1)

```
28      acall command
29      mov dptr,#1000h;read a string
30      label:
31          mov a,#0
32          movc a,@a+dptr;read letter by letter
33          jz finish
34          acall display
35          inc dptr
36          sjmp label
37      finish:
38          mov a,#0C0h;force cursor to 2nd line
39          acall command
40          mov dptr,#2000h;read the string for desired temperature
41          labell:
42              mov a,#0h
43              movc a,@a+dptr;
44              jz finishl
45              acall display
46              inc dptr
47              sjmp labell
48      finishl:
49          acall shift2ndLineRight
50          acall displayDesired;display desired temp
51      convert:
52          acall shiftlstLineRight;to display real temp
53          acall findTemp
54          mov a,#10h;shift cursor to the left to display a new value of the temp
55          acall command
56          mov a,#10h
57          acall command
58          mov a,#10h
59          acall command
60          mov a,#10h
61          acall command
62
63          sjmp convert
```

(fig 8.2.2)

```
65  command:
66      acall ready;see if the LCD is ready
67      mov P2,A;put command to data lines
68      clr P0.0;put RS in 0 for command
69      clr P0.1;put R/W in 0 for command
70      setb P0.2;put E in 1
71      nop
72      nop
73      clr P0.2;make E H->L
74      ret
75
76  display:
77      acall ready
78      mov P2,A;put data on data lines
79      setb P0.0;set RS in 1 for writing
80      clr P0.1;put R/W in 0 for writing
81      setb P0.2
82      nop
83      nop
84      clr P0.2
85      ret
86
87  ready:
88      clr P0.2;put E LOW
89      nop
90      nop
91      setb P2.7;make P2.7 (D7) input port
92      clr P0.0;put RS in 0 for command register
93      setb P0.1;set R/W in 1 for reading
94      ;read command register and check busy flag
95      back:
96          clr P0.2
97          nop
98          nop
99          setb P0.2;switch E L->H and wait for command to be executed
100         jb P2.7,back;until busy flag is 0
```

(fig 8.2.3)

```
 87  ready:
 88      clr P0.2;put E LOW
 89      nop
 90      nop
 91      setb P2.7;make P2.7 (D7) input port
 92      clr P0.0;put RS in 0 for command register
 93      setb P0.1;set R/W in 1 for reading
 94      ;read command register and check busy flag
 95      back:
 96          clr P0.2
 97          nop
 98          nop
 99          setb P0.2;switch E L->H and wait for command to be executed
100          jb P2.7,back;until busy flag is 0
101      clr P0.2
102      ret
103
104  readyConversion:
105      setb P3.7;start the conversion (H->L)
106      nop
107      nop
108      clr P3.7
109      setb P3.6;make pin 6 of port 3 input port to read EOC signal
110      ;check if EOC gets low to high
111      t:
112          jb P3.6,t
113      nop
114      nop
115      r:
116          jnb P3.6,r
117      setb P3.5;set OE High to have the rsult on the output lines
118      ret
119
120  findTemp:
121      acall readyConversion;see if conversion is ready
122      mov a,P1;take value from ADC
```

(fig 8.2.4)

```
119
120  findTemp:
121      acall readyConversion;see if conversion is ready
122      mov a,P1;take value from ADC
123      clr P3.5;set OE Low
124
125      mov b,#33h
126      div ab;divide a by 33h (51 in decimal = 50 possible hex numbers
127  ;that have same tens number + 1 to make the division correct
128  ;e.g. 32h = 50d and is 09.8 degrees, so tens 0) and store in a the quotient
129  ;the quotient is the tens of the temperature number
130      mov r0,a;save digit of tens
131      add a,#30h;convert to ASCII
132      acall display
133
134      mov a,b;get b
135      mov b,#5h;divide by 5 beacuse for each unit, there are 5
136  ;possible hex values
137      div ab
138      mov r1,a;save digit of units
139      add a,#30h
140      acall display
141
142      mov a,#'.'
143      acall display
144
145      mov a,b;get the remainders of the division
146      mov b,#2h
147      mul ab;multiply by 2
148  ;beacuse the no after comma are multiples of 2
149  ;store lower 8-bits in a
150      mov r2,a;save digit after comma
151      add a,#30h
152      acall display
153      acall heater
154      ret
```

(fig 8.2.5)

```
156  clockADC:
157      ;ADC clock frequency = 100kHz
158      ;T=10us => Ton=Toff=5us
159      ;5/1.085 = 5
160      ;load in TH0=-5d
161      mov TMOD,#02h;mode 2 for timer 0
162      mov TH0,#-5d
163      setb TR0;start timer
164  ret
165
166
167  displayDesired:
168      mov a,r3
169      add a,#30h
170      acall display
171      mov a,r4
172      add a,#30h
173      acall display
174      mov a,#'.'
175      acall display
176      mov a,r5
177      add a,#30h
178      acall display
179  ret
180
181  buttonPlus:
182  plus:
183      acall shift2ndLineRight
184      cjne r3,#5h,below50
185      acall displayDesired
186      reti
187      below50:
188      cjne r5,#8h,commaplus
189      cjne r4,#9h,unitsplus
190      cjne r3,#5h,tensplus
191      acall displayDesired
```

(fig 8.2.6)

```
180
181  buttonPlus:
182  plus:
183      acall shift2ndLineRight
184      cjne r3,#5h,below50
185      acall displayDesired
186      reti
187      below50:
188      cjne r5,#8h,commaplus
189      cjne r4,#9h,unitsplus
190      cjne r3,#5h,tensplus
191      acall displayDesired
192      reti
193      commaplus:
194          inc r5
195          inc r5
196          sjmp ok1
197      unitsplus:
198          inc r4
199          mov r5,#0h
200          sjmp ok1
201      tensplus:
202          inc r3
203          mov r4,#0h
204          mov r5,#0h
205          sjmp ok1
206      ok1:
207
208      acall displayDesired
209
210  reti
211
212  buttonMinus:
213  minus:
214      acall shift2ndLineRight
215      cjne r5,#0h,commaminus
```

(fig 8.2.7)

```
buttonMinus:
minus:
    acall shift2ndLineRight
    cjne r5,#0h,commaminus
    cjne r4,#0h,unitsminus
    cjne r3,#0h,tensminus
    acall displayDesired
    reti
    commaminus:
        dec r5
        dec r5
        sjmp ok
    unitsminus:
        dec r4
        mov r5,#8h
        sjmp ok
    tensminus:
        dec r3
        mov r4,#9h
        mov r5,#8h
        sjmp ok
    ok:

    acall displayDesired

reti

shiftToRightLine:
    mov b,#0Ch;jump over the already written message (12 characters)
    rightShift:
        mov a,#14h
        acall command
        dec b
        mov a,b
    cjne a,#0h,rightShift
    ret
```

(fig 8.2.7)

```
249  shift1stLineRight:
250      mov a,#80h;force cursor to 1st line
251      acall command
252      acall shiftToRightLine
253
254  ret
255
256  shift2ndLineRight:
257      mov a,#0C0h
258      acall command
259      acall shiftToRightLine
260
261  ret
262
263  heater:
264      ;divide real temp by desired temp
265      mov b,r3
266      mov a,b
267      jz verifytens;verify divison by 0
268      mov a,r0
269      div ab
270      jnz verifytens;if quotient is 0
271      ;real temp<desired temp
272
273      acall turnOnHeater
274      ret
275
276      verifytens:
277      mov a,r0
278      subb a,r3
279      jnz goodtemp;if tens are equal, comapare
280      ;the units. If not, real temp>desired temp
281
282      units:
283      ;mov a,r1
284      mov b,r4
```

(fig 8.2.8)

```
282     units:
283     ;mov a,r1
284     mov b,r4
285     mov a,b
286     jz verifyunits;verify divison by 0
287     mov a,r1
288     div ab
289     jnz verifyunits;if quotient 0
290     ;real temp<desired temp
291
292     acall turnOnHeater
293     ret
294
295     verifyunits:
296     mov a,r1
297     subb a,r4
298     jnz goodtemp;if units are equal
299     ;verify digit after comma
300
301     comma:
302     ;mov a,r2
303     mov b,r5
304     mov a,b
305     jz verifycomma;verify divison by 0
306     mov a,r2
307     div ab
308     jnz verifycomma;if quotient 0
309     ;real temp<desired temp
310
311     acall turnOnHeater
312     ret
313
314     verifycomma:
315     mov a,r2
316     subb a,r5
317     jnz goodtemp;if digits after
```

(fig 8.2.9)

```
305     jz verifycomma;verify divison by 0
306     mov a,r2
307     div ab
308     jnz verifycomma;if quotient 0
309     ;real temp<desired temp
310
311     acall turnOnHeater
312     ret
313
314     verifycomma:
315     mov a,r2
316     subb a,r5
317     jnz goodtemp;if digits after
318     ;comma are equal => same temp
319
320     goodtemp:
321     acall turnOffHeater
322
323 ret
324
325 turnOnHeater:
326     setb P0.3
327 ret
328
329 turnOffHeater:
330     clr P0.3
331 ret
332
333
334 org 1000h
335     string: db 'Temperature:',0
336
337 org 2000h
338     s: db 'Desired: ',0
339
340 end
```

(fig 8.2.10)

B. Code in C

```
#include <reg51.h>
sfr lcddata=0xA0; //P2 = LCD D7...D0 pins (A0=address of P2)
sbit RS=P0^0;
sbit RW=P0^1;
sbit EN=P0^2;
sbit busy=P2^7;
sfr adcdata=0x90; //P1 = ADC pins OUT1...OUT8 (90=address of P1)
sbit Start=P3^7;
sbit EOC=P3^6;
sbit OE=P3^5;
sbit Heater=P0^3;
sbit ADCClock=P3^4;
sbit EINT1=TCON^2;
sbit EINT0=TCON^0;

code unsigned char string[] = {"Temperature:"};
code unsigned char desstring[] = {"Desired:"};
unsigned int a=2,b=1,c=4;//init values for desired temp

void plusButton();
void minusButton();

void ADCClk () interrupt 1 {//interr for timer 0
  ADCClock = ~ADCClock;
}

void ButtonMinus() interrupt 2 {//interr for INT1 external
  minusButton();
}

void ButtonPlus() interrupt 0 {//interr for INT0 external
  plusButton();
}

void clockADC();
void displayMessage(unsigned char code *dchar);
```

(fig 8.2.11)

```
37  void command(unsigned char value);
38  void ready();
39  void Delay(unsigned int delaytime);
40  void writeToLCD(unsigned char value);
41  void readyConversion();
42  void readTemp();
43  void findNumber(unsigned int x);
44  void shiftLineToRight();
45  void shift2ndLineToRight();
46  void shift1stLineToRight();
47  void displayDesiredTemp();
48  void heater(unsigned int x, unsigned int y, unsigned int z);
49  unsigned int formDesiredTemp();
50
51  void main() {
52    IE=0x87;//enable interr for INT0, INT1, timer 0
53    EINT1=1;//enable INT1 when a transition from HIGH in LOW
54    EINT0=1;//enable INT0 when a transition from HIGH in LOW
55    clockADC();
56    command(0x38);//init LCD
57    command(0x0E);//place cursor
58    command(0x01);//clear display
59    displayMessage(&string);
60    command(0xC0);//move in 2nd row
61    displayMessage(&desstring);
62    shift2ndLineToRight();
63    displayDesiredTemp();
64    while(1){
65      shift1stLineToRight();
66      readTemp();
67      command(0x10);//shift the cursor to the left 4 positions to display the new value
68      command(0x10);
69      command(0x10);
70      command(0x10);
71    }
72  }
```

(fig 8.2.12)

```
73
74  void command(unsigned char value) {
75    ready();
76    lcddata=value;//put on P2 the value of the command
77    RS=0;
78    RW=0;//prepare to write command
79    EN=1;
80    Delay(1);
81    EN=0;
82  }
83
84  void ready() {
85    EN=0;
86    busy=1;//make P2.7 input port (D7 of LCD)
87    RS=0;
88    RW=1;//prepare to read from LCD
89    while(busy==1) {
90      EN=0;
91      Delay(1);
92      EN=1;
93    }
94    EN=0;
95  }
96
97  void Delay(unsigned int delaytime) {
98    unsigned int i,j;
99    for (i=0;i<delaytime;i++)
100   for (j=0;j<1275;j++);//delay of approx 12ms
101 }
102
103 void displayMessage(unsigned char code *string){
104   while(*string!=0){
105     writeToLCD(*string);
106     string++;
107   }
108 }
```

(fig 8.2.13)

```
109 
110 void writeToLCD(unsigned char value){
111   ready();
112   lcddata=value;
113   RS=1;
114   RW=0;//prepare to write
115   EN=1;
116   Delay(1);
117   EN=0;
118 }
119 
120 void readTemp(){
121   unsigned int x;
122   adcdata=0xFF;//input port
123   readyConversion();//see if cnversion is ready
124   x=adcdata;
125   OE=0;//bring OE back to 0
126   findNumber(x);
127 }
128 
129 void readyConversion(){
130   Delay(50);
131   Start=1;//start the conversion
132   Delay(1);
133   Start=0;
134   EOC=1;//make P3.6 input port
135   //check if EOC=LOW to HIGH -> ready conversion
136   while(EOC!=0){
137     }
138   while(EOC!=1){
139     }
140   OE=1;//set OE to 1 to put result on output lines
141   Delay(1);
142 }
143 
144 void findNumber(unsigned int x){
```

(fig 8.2.14)

```
142  }
143
144  void findNumber(unsigned int x){
145    unsigned int y;
146    unsigned int d,e,f;//to save real temp
147    d=x/51;//save digit of tens
148    y=x%51;
149    e=y/5;//save digit of units
150    writeToLCD(d+0x30);//digit of tens
151    writeToLCD(e+0x30);//digit of units
152    writeToLCD(0x2E);//put the dot
153    y=x%0x33;
154    y=y%0x05;
155    y=y*0x02;//digit after comma, only even
156    f=y;//save digit after comma
157    writeToLCD(y+0x30);
158    heater(d,e,f);
159  }
160
161  void clockADC() {
162    //ADC clock frequency = 100kHz
163    //T=10us => Ton=Toff=5us
164    //5/1.085 = 5
165    //load in TH0=-5d
166    TMOD=0x02;//mode 2 for timer 0
167    TH0=-5;
168    TR0=1;//start timer 0
169  }
170
171  void shiftLineToRight(){
172    unsigned int i;
173    for (i=0;i<12;i++){
174      command(0x14);//jump over 12 positions
175    }
176  }
177
```

(fig 8.2.15)

```
178 void shift2ndLineToRight(){
179   command(0xC0);//start of 2nd line
180   shiftLineToRight();
181 }
182
183 void shift1stLineToRight(){
184   command(0x80);
185   shiftLineToRight();
186 }
187
188
189 void displayDesiredTemp() {//write the desired temp
190   writeToLCD(a+0x30);
191   writeToLCD(b+0x30);
192   writeToLCD('.');
193   writeToLCD(c+0x30);
194 }
195
196 void heater(unsigned int d, unsigned int e, unsigned int f){
197   unsigned int x;
198   x=formDesiredTemp();//form number corresp to desired temp
199   d*=100;
200   e*=10;
201   d=d+e;
202   d=d+f;//form number corresp to real temp
203   if (d<x)
204     Heater=1;
205   else
206     Heater=0;
207 }
208
209 unsigned int formDesiredTemp(){
210   unsigned int x,y,z;
211   x=a;
212   y=b;
213   z=c;
```

(fig 8.2.16)

```
209 unsigned int formDesiredTemp(){
210   unsigned int x,y,z;
211   x=a;
212   y=b;
213   z=c;
214   x*=100;
215   y*=10;
216   x=x+y;
217   x=x+z;
218   return x;
219 }
220
221 void plusButton(){
222   shift2ndLineToRight();
223   if(a<5){//verify if max is not reached
224     if (c!=8)//if comma hasn't reached max value, increment
225       c+=2;
226     else//else increment unit
227       if (b!=9){//if units hasn't reached max value
228         c=0;//put comma in 0
229         b++;
230       }
231       else
232         if (a!=5){//if tens hasn't reached max value
233           b=0;//reset units
234           c=0;//reset comma
235           a++;
236         }
237   }
238   displayDesiredTemp();
239 }
240
241 void minusButton(){
242   shift2ndLineToRight();
243   if (c!=0)//if comma is not zero, decrement
244     c-=2;
```

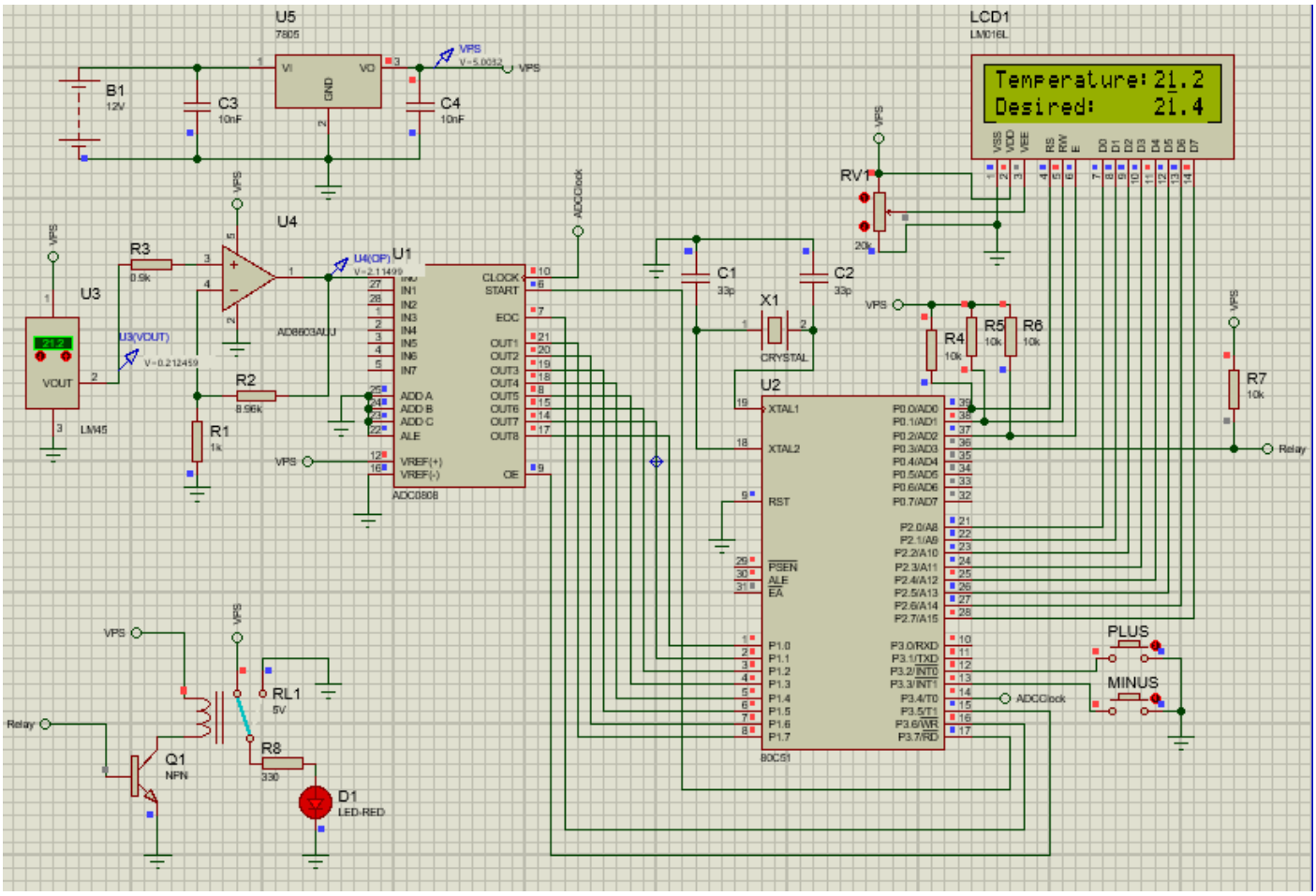
(fig 8.2.17)

```
223   if(a<5){//verify if max is not reached
224     if (c!=8)//if comma hasn't reached max value, increment
225       c+=2;
226     else//else increment unit
227       if (b!=9){//if units hasn't reached max value
228         c=0;//put comma in 0
229         b++;
230       }
231       else
232         if (a!=5){//if tens hasn't reached max value
233           b=0;//reset units
234           c=0;//reset comma
235           a++;
236         }
237   }
238   displayDesiredTemp();
239 }
240
241 void minusButton(){
242   shift2ndLineToRight();
243   if (c!=0)//if comma is not zero, decrement
244     c-=2;
245   else//else decr units
246     if (b!=0){//if units hasn't reached min value
247       c=8;//reset comma
248       b--;//decr units
249     }
250     else
251       if (a!=0){//if tens hasn't reached min value
252         c=8;//reset comma
253         b=9;//reset units
254         a--;//decr tens
255       }
256   displayDesiredTemp();
257 }
258
```

(fig 8.2.18)

8.3. Simulation

- At the temperature of 21.2°C



(fig 8.3.1: simulation)

9. References

1. https://ro.mouser.com/?gclid=EAIaIQobChMI_6HP5Orn_QIVh5XVCh2cFQnsEAAYASAAEgIjGPD_BwE
2. https://atlas-scientific.com/blog/how-do-temperature-sensors-work/
3. https://www.rotronic.com/en-us/humidity_measurement-feuchtemessung-mesure_de_l_humidite/capacitive-sensors-technical-notes-mr
4. https://www.electronicshub.org/humidity-sensor-types-working-principle/#Thermal_Conductivity_Humidity_Sensors
5. https://www.allaboutcircuits.com/textbook/digital/#chpt-13
6. http://www.edutecnica.it/elettronica/datasheets/DAC-ADC/ADC0808.pdf
7. https://www.ti.com/cn/lit/ds/symlink/lm45.pdf?ts=1681022156418&ref_url=https%253A%252F%252Fwww.google.com%252F
8. https://www.alldatasheet.com/view.jsp?Searchword=Ad8603&gclid=Cj0KCQjwxMmhBhDJARIsANFGOSt6W-
9. https://www.javatpoint.com/what-is-lcd
10. LCD 16×2 (LM016L) « Innovation of Engineers (wordpress.com)
11. HD44780 pdf, HD44780 Download, HD44780 Description, HD44780 Datasheet, HD44780 view ::: ALLDATASHEET :::
12. Microcontrollers courses
13. https://www.edaboard.com/threads/why-do-we-connect-capacitors-to-the-crystal-in-the-microcontroller.259887/