# Temperature Display and Controller-1

**GRP 39** - SHRINIVAS PRASAD KULKARNI, C NIKHIL, SHAAN UL HAQUE

*180070057, 180070016, 180070053*

**COURSE**: EE344

**FACULTY MENTOR**: PROF. KUSHAL TUCKLEY

**OBJECTIVE**:
Measure the room temperature and display it on LCD screen using PT-51 microcontroller. Given a reference temperature, control the the temperature of the room via an on-off controller installed in AC.

## 1 Complete Circuit Schematic

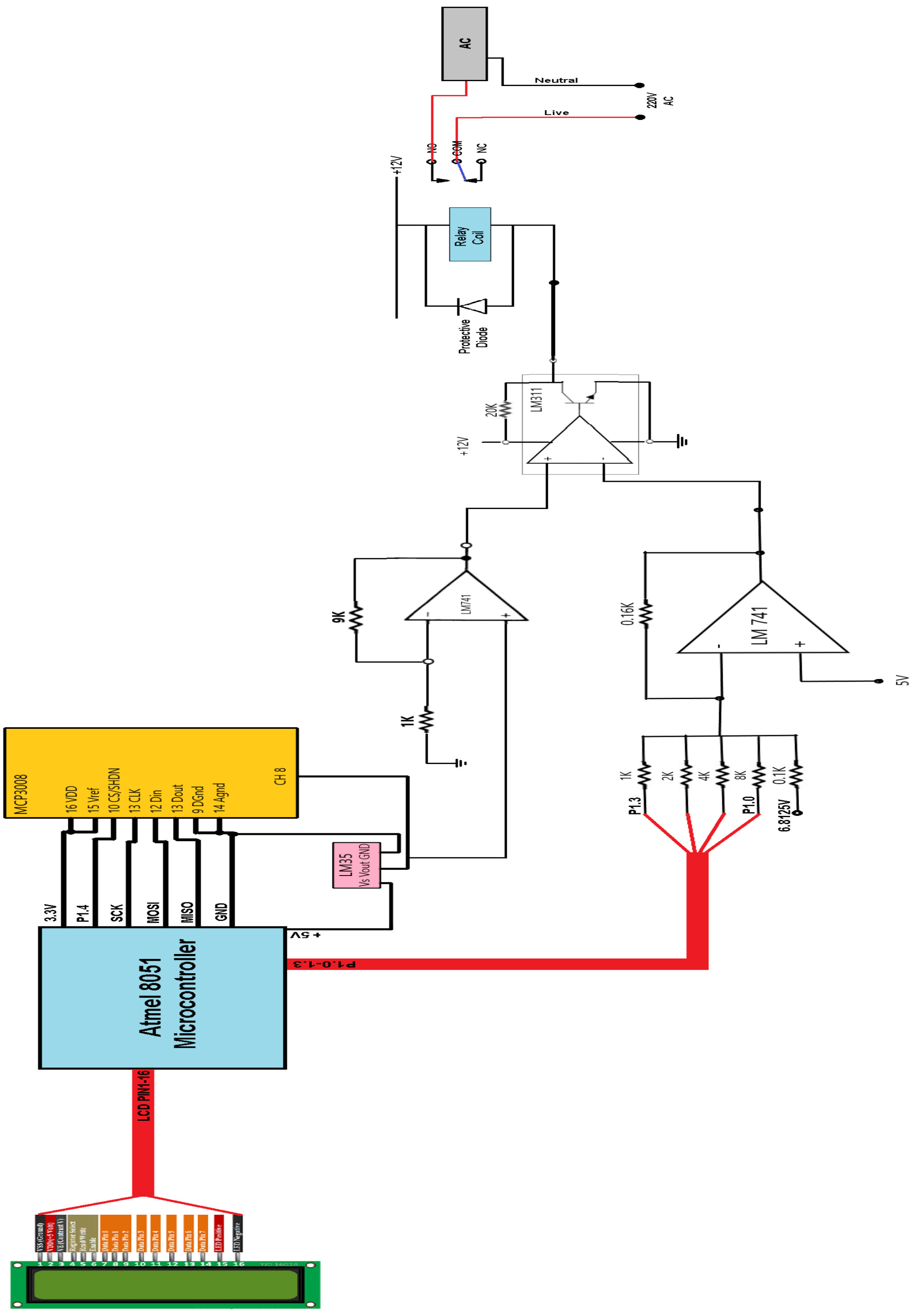


**Figure 1:** Circuit Schematic

## 2 Air Conditioning Model

For designing the controller we first need a model of our system. The following figure illustrates our model well.

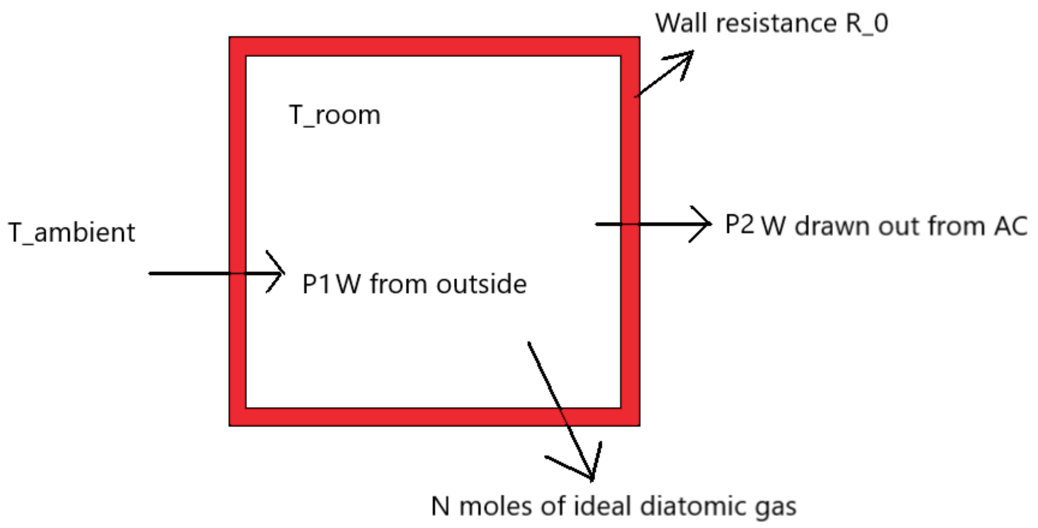


**Figure 2:** Model for our AC and controller.

Writing the heat equations for the model when AC is on, we get:

$$P_2 - (T_1 - T_2)/R_0 = -\frac{5}{2} N R dT_2/dt$$

where, P2 is the heat per unit time pulled out by AC, $T_1$ is the ambient temperature, $T_2$ is the room's temperature, $R_0$ is the combined heat resistance of the wall, roof and floor and R is the Universal Gas Constant. While when AC is off, we get:

$$-(T_1 - T_2)/R_0 = -\frac{5}{2} N R dT_2/dt$$

## 3 Temperature Display on PT-51 Microcontroller

The ADC MCP3008 we are using has in total 8 channels and can be easily interfaced with a Micro-controller. We program the microcontroller to sample the temperature every 5sec. The pin diagram shown below explains our implementation clearly.

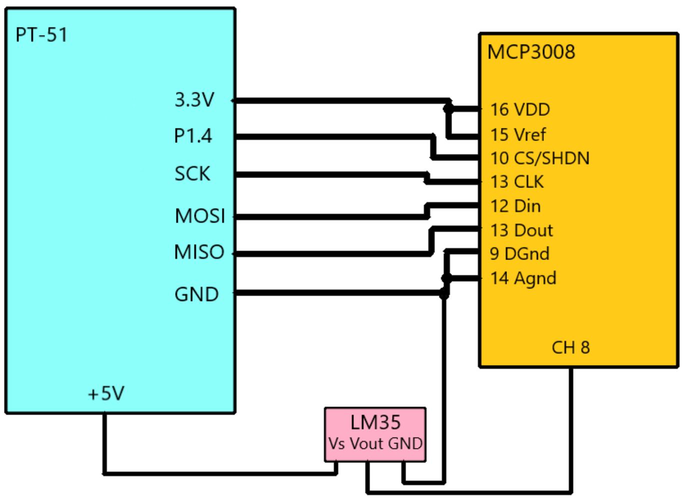


**Figure 3:** Pin Diagram for ADC interfacing.

## 3.1 LM35[3]

This is a temperature measuring device with working range between -55°$C$ to 150°$C$. At typical room temperature the error in measurement is about ±0.25°$C$ but not above ±0.5°$C$. The ambient temperature is given by the following relation:

$$Voltage(V_{out}) = 10mV \times T(in^{\circ}C)$$

## 3.2 MCP3008

This is a 10 bit ADC which send data to the master serially. The ADC divides the reference voltage($V_{DD}$) levels into 1024 bins and then with respect to $V_{ref}$ outputs the digital 10 bit code for $V_{in}$. Thus the maximum error in temperature displaying is about $\approx \pm V_{DD}/2048$ which is of the order of $10^{-3}$ when $V_{DD}$ is 3.3V. The error in displaying the temperature due this is ±0.16°$C$. We remark here that this error would not creep in controlling the temperature of the room since ADC is only used for displaying purpose while for controlling, direct voltage measured from LM35 is taken.
**NOTE:** We have also included the code in the appendix section that will be used to read ADC data using the microprocessor. It was coded on keil $\mu$vision.

## 3.3 Displaying on LCD

The measured voltage has to be further displayed on a LCD screen. The schematic for the same is shown below.

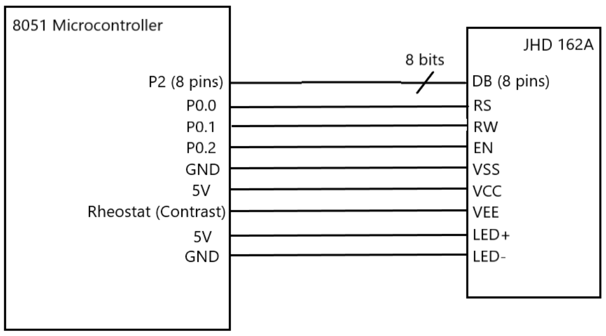


**Figure 4:** Pin Diagram for LCD interfacing.

**NOTE:** We have also included the code in the appendix section that will be used to display data on LCD using the microprocessor. It was coded on keil $\mu$vision.

## 4 Converting $T_{ref}$ to appropriate voltage level

After modelling the system, we need to convert the reference temperature provided by the user to appropriate voltage levels so as to compare with the current temperature measured by LM35. The following DAC was made by us to get voltages in the range of 2.1V-3V. The encoding for different temperatures by the microprocessor is given below:

| Temperature(C) | Encoding | Output Voltage(V) |
|---|---|---|
| 21 | 1111 | 2.1 |
| 22 | 1110 | 2.2 |
| 23 | 1101 | 2.3 |
| 24 | 1100 | 2.4 |
| 25 | 1011 | 2.5 |
| 26 | 1010 | 2.6 |
| 27 | 1001 | 2.7 |
| 28 | 1000 | 2.8 |
| 29 | 0111 | 2.9 |
| 30 | 0110 | 3.0 |

**Table 1:** Temperature Encoding

Shown below is the model for our DAC. The values of the resistor and the voltages are so chosen to get initial voltage value of 2.1V(21°$C$ while increasing with a value of 0.1V for every 1°$C$ increase in reference temperature. The op-amp voltage supply is as usual.
**Note:** The voltage 6.8125V is applied to provide a bias to the output voltage because we start from 2.1V and increase by 0.1V for every 1°$C$ rise. This voltage can be easily made available using the same voltage supply as other ic's and the using a voltage divider.

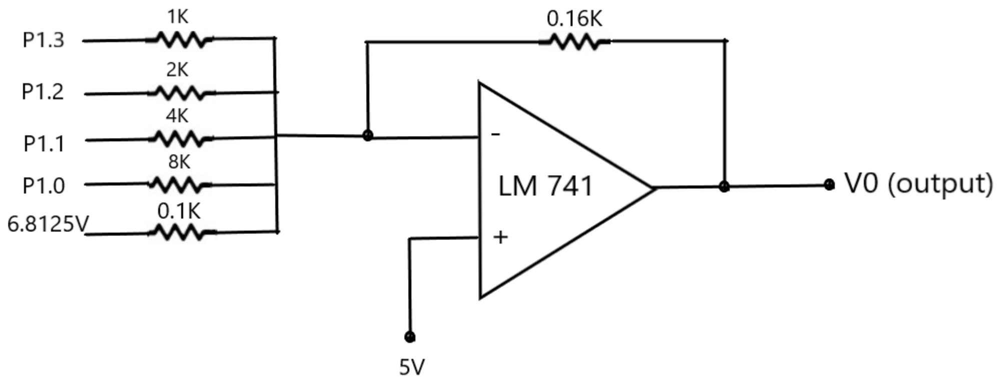


**Figure 5:** DAC model.

Now this is the reference temperature according to which we need to control the AC. Temperature from LM35 is scaled by 10 using op-amp as shown below and put to comparator:

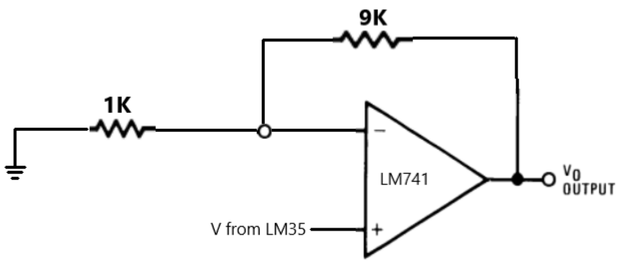


**Figure 6:** Voltage Scaled from LM35.

For the sake of completeness, we have shown the pin diagram of the LM741 as well:

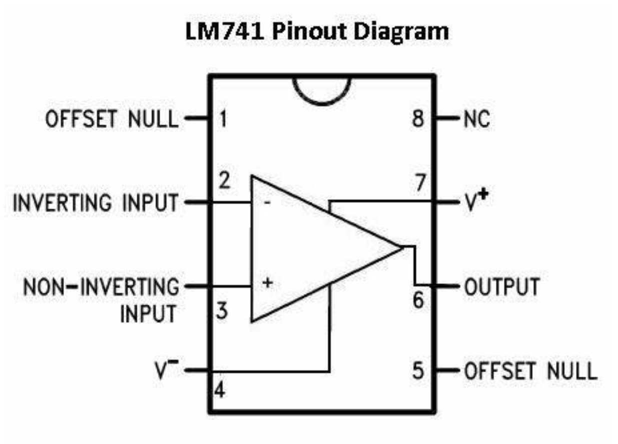


**Figure 7:** LM741 Pin Diagram[1].

## 5 Comparator Circuit

We have used LM311 as comparator because of it's high response time and easier interfacing with relay. The circuit shown below is called as a zero crossing detector(in this case $V_ref$ is considered as zero). We would be using a 12V relay hence the voltage supply is as shown in fig. When input voltage is greater than $V_{ref}$ output voltage is +12V driving the transistor(in built in the LM311 IC) in saturation(ON mode) and thus the voltage of the collector is ≈ 0.2V or almost 0V. When input voltage is less than $V_{ref}$ then output voltage is 0V thus driving the transistor into cut-off(Off mode). Thus, voltage of the collector is +12V. This is further connected to a 12V relay(next section) which is used for interfacing our controller with AC.

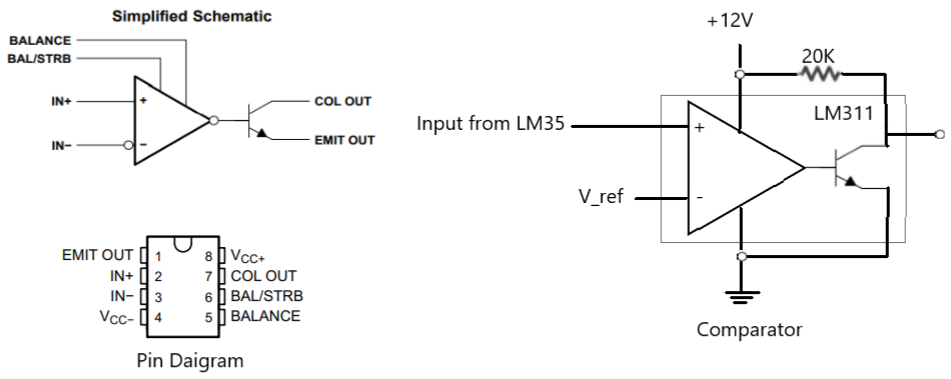


**Figure 8:** LM311 Comparator Circuit[2].

## 6 Relay Circuit to Control The AC

The output of the comparator will vary between 0V and 12V. Also, the comparator cannot handle large amounts of currents. Therefore, to drive the AC using this signal, we require the use of a 5-pin Normally Open 12V relay.
From the discussion above, we know that the comparator output will be 0V when input voltage is greater than $V_{ref}$ and therefore, the relay coil will be energized. In such a case, we also have $T > T_{ref}$ and therefore the AC must be ON. We can easily conclude that the relay must therefore be **Normally Open**. As a result, when $T < T_{ref}$, the comparator output is +12V and the relay coil gets deactivated, turning off the AC.
The schematic diagram for the relay circuit is given below. The relay coil will be controlled by the comparator output. The AC will be connected to 220V AC supply via the NO and COM/POLE ports of the relay.

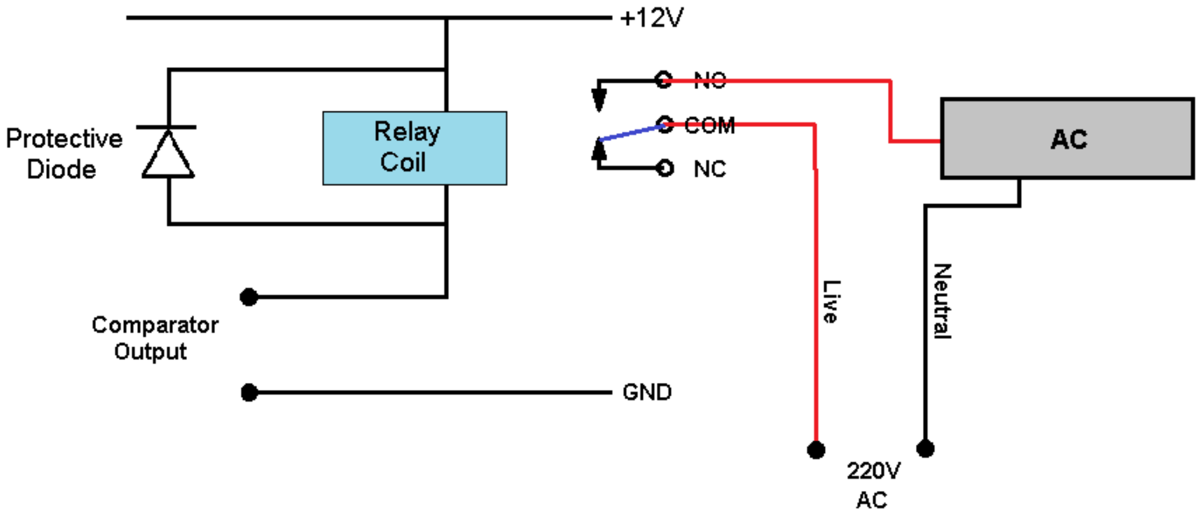


**Figure 9:** Relay Circuit Diagram

## 7 Simulation

Let us study the model by assuming some values for our variables and simulate the behaviour of our controller on MATLAB. We assume a 2 ton AC with working temperature between $21°C - 30°C$, ambient temperature to be 32°, dimensions of the room be $4m \times 4m \times 4m$ and a 20cm thick concrete(heat conductivity K = 1.26 $Wm^{-1}C^{-1}$ roof and floor with brick walls(heat conductivity K = 0.65 $Wm^{-1}C^{-1}$. After calculating the relevant values, we get: $R_0$ = 1.686x$10^{-3}CW^{-1}$, $P_2$ = 7.04$KW$, N = 2565.953(calculated by assuming ideal gas at 300K and 1 atm pressure), $T_1$ = 32°$C$ and R = 8.314 $JK^{-1}mole^{-1}$. Plugging in the differential equation we get:

$$7.04 \times 10^3 - (32 - T_2) \times 10^3/1.686 = 53333dT_2/dt$$

Now when AC is off P2 = 0, in that case our differential equation becomes:

$$-(32 - T_2) \times 10^3/1.686 = 53333dT_2/dt$$

Solving the above two equations gives us:

$$T_f = 20.13(1 - exp(-t/89.92)) + T_i exp(-t/89.92)$$

where, $T_f$ and $T_i$ are the initial and final temperature of the room after t sec for the case when AC is on.

$$T_f = 32(1 - exp(-t/89.92)) + T_i exp(-t/89.92)$$

where, $T_f$ and $T_i$ are the initial and final temperature of the room after t sec for the case when AC is off. Simulating the above behaviour on MATLAB for the case when $T_{ref}$ = 25°$C$, i.e, the reference temperature that user wants to keep by using the on-off controller gives the following result.

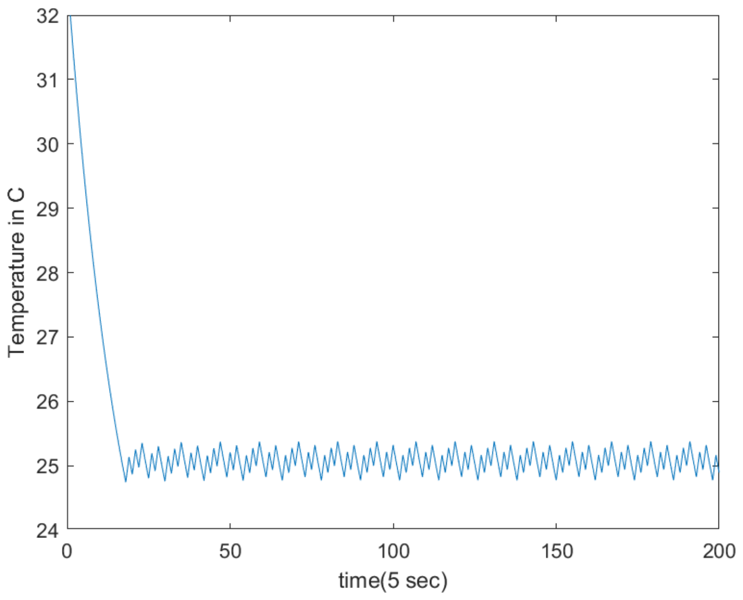
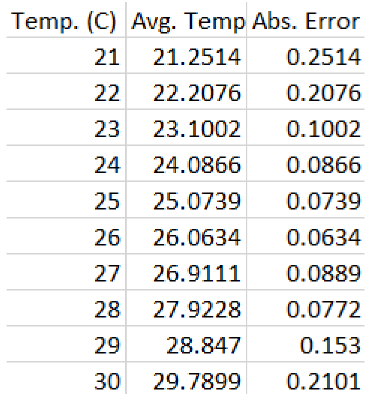


**Figure 10:** Temperature variation with time.

Here we remark that the temperature sensor is assumed to be sensing the temperature every 5sec and accordingly sending the signal to switch on or off the AC, thus giving oscillatory behaviour.

## 7.1 Error Calculation

When the temperature became stabilized, we calculated the average temperature of the room and then calculated the deviation of room temperature from $T_{ref}$. The following table summarizes our findings for the case when the ambient temperature is constant at 32°.



| Temp. (C) | Avg. Temp | Abs. Error |
|---|---|---|
| 21 | 21.2514 | 0.2514 |
| 22 | 22.2076 | 0.2076 |
| 23 | 23.1002 | 0.1002 |
| 24 | 24.0866 | 0.0866 |
| 25 | 25.0739 | 0.0739 |
| 26 | 26.0634 | 0.0634 |
| 27 | 26.9111 | 0.0889 |
| 28 | 27.9228 | 0.0772 |
| 29 | 28.847 | 0.153 |
| 30 | 29.7899 | 0.2101 |

**Figure 11:** Error in room temperature.

Average error in room temperature due to oscillatory behaviour is equal to $0.131°C$ while average temperature measurement error by LM35 is $\pm 0.25°C$. Thus total average error in room temperature(from $T_{ref}$) is $\pm 0.381°C$.
Maximum error in measurement by LM35 is $\pm 0.5°C$ while due to oscillatory behaviour is $\pm 0.2514°C$ giving net error to be $\pm 0.7414°C$. Thus, maximum deviation of room temperature from $T_{ref}$ is $\pm 0.7414°C$.

## 7.2 Real time Simulation using GNURadio

The MATLAB simulation described above illustrates the actual behavior of the system, when the temperature is sampled every 5 seconds, and the output characteristic obtained exactly illustrates how will the actual system evolve with time.
Even though the MATLAB simulation gives us an exact idea of how will the system operate, we still would not get the feel of how the system behaves in *real time*, how does the applied control affect the system, and how fast does the box temperature converge to the reference temperature as we make changes to it.
**GNURadio** provides various signal processing blocks, like a comparator, amplifier, modulator, additive noise, various scopes etc which look like actual devices. Almost any kind of processing can be handles by GNURadio, and hence we used it to simulate our system. In Figure 12, the block diagram used for the implementation is shown.
In the simulation, the reference temperature is set as a slider (a knob), whose value can be varied as we want, and we can see the effect of this on the box temperature in real time. The constants in the differential equations above are set conveniently, and can be ignored. The differential equation is updated in discrete steps, and the temperature sampling time is assumed to be high (and not 0.2 samples/sec as happening in the actual system). The updated temperature is fed back in a loop to the comparator, which decides whether to switch the AC ON/OFF. The simulation would be demonstrated in the presentation.

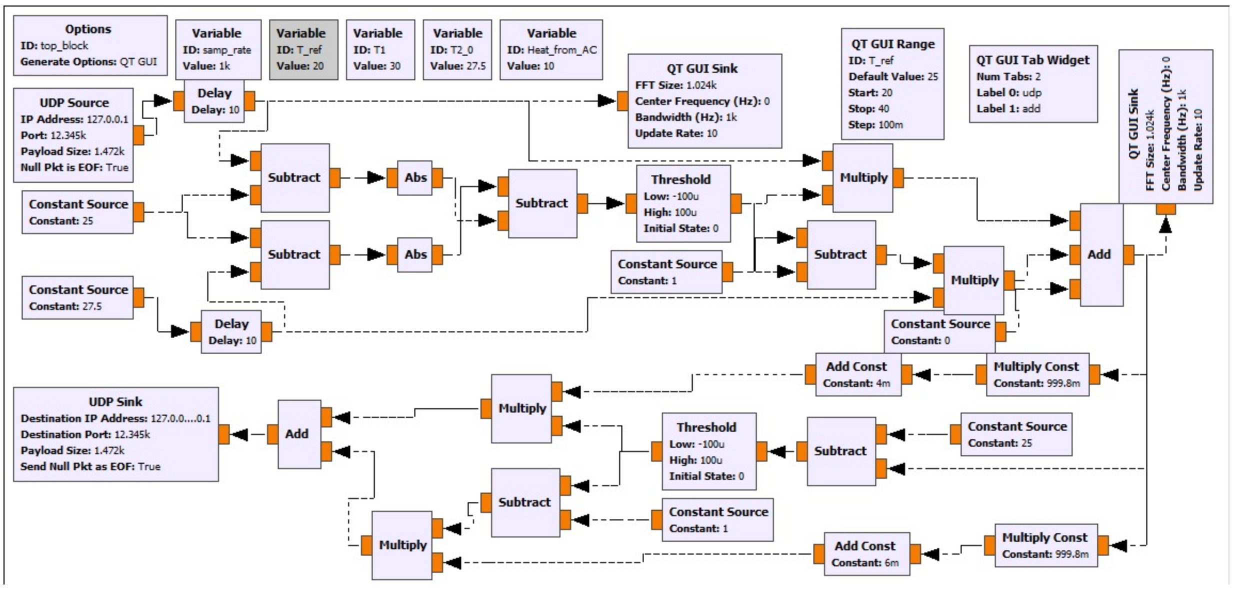


**Figure 12:** Simulator Flowgraph

In this simulation, we can change the the $T_{ref}$ and observe the instantaneous effect on the box temperature on run-time.

## 8 Bill of Components

- Atmel Micro-controller(8051) - Rs.477/- (×1)
- JHD LCD Screen - Rs. 118/- (×1)
- Temperature Sensor LM35 - Rs. 102/- (×1)

- OPAMP LM741 - Rs. 40/- (×3)
- Comparator LM311 - Rs. 15/- (×1)
- 12 V Relay - Rs. 30/- (×1)
- Resistors(0.16k, 0.1K, 1K, 2K, 4K, 8K, 9K, 20K) - Rs. 3/- (×1)

Total amount of the items is almost Rs. 865/-. Maximum, if pcb design, wires, etc. are also taken into account the whole price would be about Rs. 1000/-.

## 9 Comments based on Feedback

- We updated the circuit diagram for LM311 comparator.
- The coil resistance is for 12V 15A relay is about 400 ohm. Thus the maximum current through it is 30mA when the collector voltage is low-level. Since the IC can handle current upto 50mA(verified from datasheet), we can safely use 12V relay with LM311.
- The protective diode is actually to protect the LM311 IC from sudden large jump in voltage which limits the collector voltage to 12.7V when a transition from 12V to 0V occurs.

## 10 Appendix

### 10.1 lcd.h

```
void msdelay(unsigned int);
void lcd_init(void);
void lcd_cmd(unsigned int i);
void lcd_char(unsigned char ch);
void lcd_write_string(unsigned char *s);
void int_to_string(unsigned int,unsigned char *temp_string);

sbit RS=P0^0;
sbit RW=P0^1;
sbit EN=P0^2;

void lcd_init(void)
{
P2=0x00;
EN=0;
RS=0;
RW=0;

lcd_cmd(0x38);
msdelay(4);
lcd_cmd(0x06);
msdelay(4);
lcd_cmd(0x0C);
msdelay(4);
```

```
lcd_cmd(0x01);
msdelay(4);
lcd_cmd(0x80);
}

void msdelay(unsigned int time)
{
int i,j;
for(i=0;i<time;i++)
{
for(j=0;j<382;j++);
}
}

void int_to_string(unsigned int val,unsigned char *temp_str_data)
{
   // char str_data[4]=0;
temp_str_data[0]=48+(val/10000);
  temp_str_data[1]=48+(val%10000/1000);
  temp_str_data[2]=48+((val%1000)/100);
  temp_str_data[3]=48+((val%100)/10);
  temp_str_data[4]=48+(val%10);
   // return str_data;
}

void lcd_cmd(unsigned int i)
{
RS=0;
RW=0;
EN=1;
P2=i;
msdelay(10);
EN=0;
}

void lcd_write_char(unsigned char ch)
{
RS=1;
RW=0;
EN=1;
P2=ch;
msdelay(10);
EN=0;
}

void lcd_write_string(unsigned char *s)
{
while(*s!='\0')
{
```

```
lcd_write_char(*s++);
}
}
```

### 10.2 spi.h

```
#ifndef spi_h
#define spi_h 1
#endif

void spi_init(void);
unsigned long int spi_trx(unsigned long int spi_data_tx);
void spi_interrupt(void);

bit transmit_completed= 0;
unsigned char temp_spi_data;

void spi_init(void)
{
SPCON = 0x33;
IEN1|= 0x04;
EA = 1;
SPCON |= 0x40;

}

unsigned long int spi_trx(unsigned long int spi_data_tx)
{
//declaration of local variables
unsigned long int spi_data_rx;
unsigned char spi_data_3,spi_data_2,spi_data_1;

//Send 3rd byte
transmit_completed = 0;

SPDAT = (spi_data_tx>>16)%256;

while(!transmit_completed);
spi_data_3 = temp_spi_data ;

//Send 2nd byte
transmit_completed=0;
SPDAT = (spi_data_tx>>8)%256;

while(!transmit_completed);
spi_data_2 = temp_spi_data ;

//Send 1st byte
transmit_completed=0;
SPDAT = (spi_data_tx%256);

while(!transmit_completed);
spi_data_1 = temp_spi_data ;

spi_data_rx= (spi_data_3<<16) + (spi_data_2<<8) + spi_data_1;

return spi_data_rx;
}

void spi_interrupt(void) interrupt 9
{

switch ( SPSTA & 0xf0)
{
case 0x80:
temp_spi_data=SPDAT;
transmit_completed=1;
  break;

case 0x10:

break;

case 0x40:
break;
}
}
```

### 10.3 mcp3008.h

```
#ifndef spi_h
#include "spi.h"
#endif

void adc_init(void);
unsigned int adc(unsigned char);

sbit cs_bar_adc = P1^4;


void adc_init(void)
{
cs_bar_adc=1;
}


unsigned int adc(unsigned char channel)
{
```

```
unsigned long int temp_adc_data,address,start,ch_address;

start = (((unsigned long int) 0x01)<<16);
ch_address = (( (unsigned long int) (channel<<4 & 0x70)|0x80 )<<8);



address = start + ch_address;


cs_bar_adc = 0;
temp_adc_data = spi_trx(address);
cs_bar_adc = 1;


temp_adc_data=temp_adc_data & 0x03ff;
  return (unsigned int) temp_adc_data;
}
```

### 10.4 main.c

```
#include <at89c5131.h>
#include "lcd.h"
#include "mcp3008.h"
char adc_ip_data_ascii[6]={0,0,0,0,0,'\0'};
code unsigned char display_msg1[]="Volt.: ";
code unsigned char display_msg2[]=" mV";
unsigned char display_msg3[]={0,0,0,'.',0,' ',223,'C','\0'};
code unsigned char display_msg4[]="Temp.: ";


void main(void)
{
int j=0;
unsigned int adc_data=0;
//float temperature;

spi_init();
adc_init();
  lcd_init();



while(1)
{
  unsigned int x;
float temperature;
temperature = 30.0;
x = adc(0);
adc_data = (unsigned int) (x*3.2258);

lcd_cmd(0x80);
lcd_write_string(display_msg1);

int_to_string(adc_data,adc_ip_data_ascii);

lcd_write_string(adc_ip_data_ascii);
lcd_write_string(display_msg2);

x = adc(7);

temperature = (x*0.32258);



x = (unsigned int) (temperature*10.0);

int_to_string(x,adc_ip_data_ascii);
lcd_cmd(0xC0);
lcd_write_string(display_msg4);
display_msg3[0] = adc_ip_data_ascii[1];
display_msg3[1] = adc_ip_data_ascii[2];
display_msg3[2] = adc_ip_data_ascii[3];
display_msg3[4] = adc_ip_data_ascii[4];

lcd_write_string(display_msg3);

}
}
```

## 11 References

[1] Texas Instruments LM741 Datasheet.

[2] Texas Instruments LM311 Datasheet.

[3] Texas Instruments LM35 Datasheet.