Temperature Monitoring System using Python

By
Nithin K Shine
(20BLC1002)

Submitted to:
FOSSEE club, VIT Chennai

29-August 2021

—

Under the guidance of:

Dr. Chanthini Baskar

—

Vellore Institute of Technology
Chennai

1. ABSTRACT

With connected home hubs, smart thermostats, remote door locks, and all the various app-controlled appliances, IoT is growing in importance, both for industrial use and everyday use. AI and IoT are inseparable. The entire idea of artificial intelligence is to capture more actionable data from IoT devices. The Internet of Things is already disrupting various industries impacting human lives in several ways.

The ability to program IOT devices using Python will make it more powerful and easy to apply Artificial Intelligence. This project is a Temperature monitoring system that can measure the room temperature and display the temperature in an LCD display and also in a webpage by using ESP32 as a webserver. A LM335 temperature sensor is used to measure the temperature. The output voltage from LM335 is analyzed by Python program and it calculates the room temperature. This temperature data is sent to an LCD display and an ESP32, which in turn sends the temperature data to a webpage.

UART communication protocol is used in this project. A firmware code is uploaded into the ESP32 and Arduino. Arduino responds to various commands send through the Serial port via a USB-B cable from the computer. The commands are sent to the Serial port using Python program.

Another part of this project includes an Analog to Digital converter, which converts the analog input received from a potentiometer wiper and converts it to digital data. A LED connected to the Arduino turns ON and OFF according to the converted digital data.

2. Design/Implementation

2.1 LM335:

The LM335 temperature sensor is an easy to use, cost-effective sensor with decent accuracy (around +/- 3 degrees C calibrated). The sensor is essentially a zener diode whose reverse breakdown voltage is proportional to absolute temperature.

Since the sensor is a zener diode, a bias current must be established in order to use the device. The spec sheet states that the diode should be biased between 400 uA and 5 mA; we'll bias it at 1 mA. It is important to note that self-heating can be a significant factor, which is why we are not choosing a higher bias current. The bias circuit is as follows:

The temperature sensor's voltage output is related to absolute temperature by the following equation: Vout = VoutT0 * T / T0, where T0 is the known reference temperature where VoutT0 is the corresponding output voltage. The nominal VoutT0 is equal to T0 * 10 mV/K. So, at 25 C, VoutT0 is nominally 298 K * 10 mV/K = 2.98 V (to be really accurate, we'd need a reference temperature and a voltmeter, but nominal values are OK for our purposes). Thus, the voltage dropped between +5 and the diode is 5V - 2.98V = 2.02V. In order to get 1 mA bias current, we need a 2.2 K resistor for R.

A pinout of the sensor is provided below:

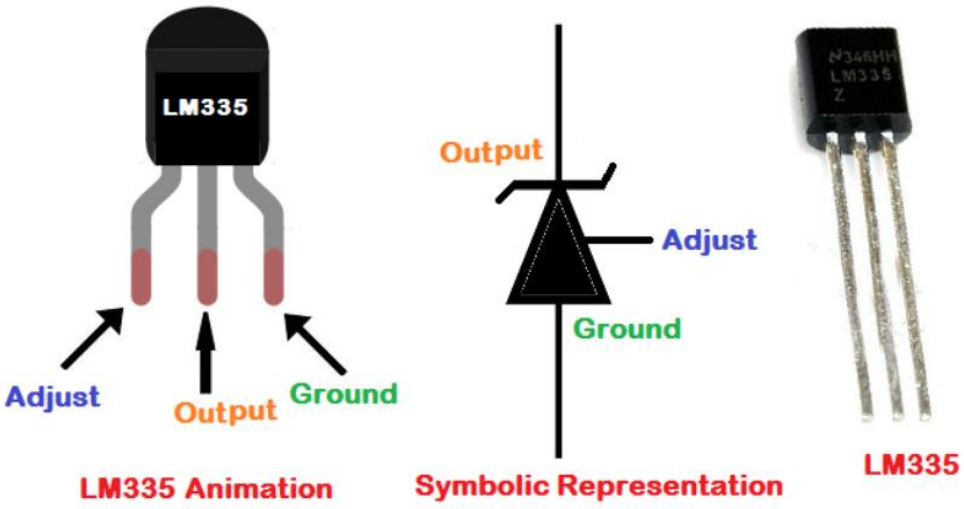

Figure 1:LM335 pinout

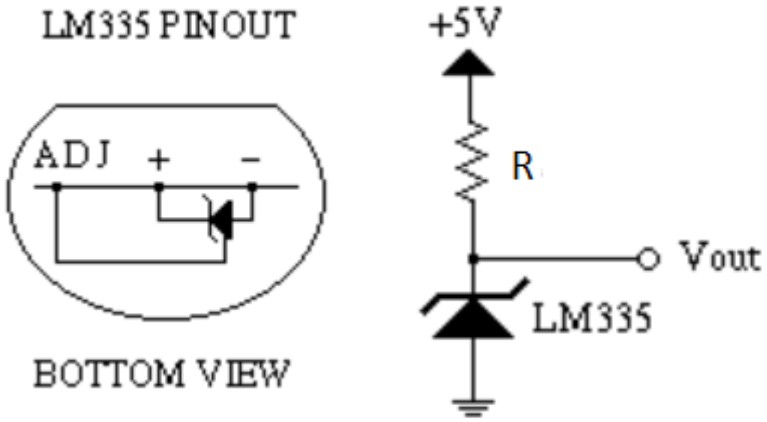

Figure 2:LM335 circuit schematic

Note that the adj pin is unconnected. The adj pin is used to trim the diode to be more accurate. A typical LM335 sensor and its symbolic representation are shown in the above figure. This LM335 is connected to the analog pin 5 of the Arduino Uno board. The analog voltage, corresponding to the voltage drop, across the terminals of LM335 needs to be digitized before being sent to the computer. This is taken care of by an onboard Analog to Digital Converter (ADC) of ATmega328 microcontroller on the Arduino Uno board. ATmega328 has a 6-channel, 0 through 5, 10-bit ADC. Analog pin 5 of the Arduino Uno board, to which the LM335 is connected, corresponds to channel 5 of the ADC. As there are 10 bits, 0-5V readings from LM335 are mapped to the ADC values from 0 to 1023. LM335 is a commonly available sensor in the market. It costs about Rs. 70.

2.1.1 Interfacing the LM335 through the Arduino IDE:

In this section, we shall describe how to read the voltage values from a LM335 connected to the analog pin 5 of the Arduino Uno board. The LM335 has to be connected to the Arduino Uno board before doing these experiments and the Arduino Uno needs to be connected to the computer with a USB cable, as shown in Figure. 10. Then run the program in Arduino IDE.

1. As discussed earlier, the 0-5V LM335 readings are mapped to 0-1023 through an ADC. The Arduino IDE based command for the analog read functionality is given below. where A5 represents the analog pin 5 to be read and the read LDR values are stored in the variable val.

```
val = analogRead(A5);//value of LM335
```

2. The output voltage from LM335 increases by 10mV per degree Celsius increase in temperature. Hence the conversion of the Analog input to temperature in Celsius is given below.

```
val = val*(500/1023)-273;
```

3. The read values are then displayed using,

```
Serial.println(val);//for display
```

4. The delay in the code is added so that the readings do not scroll away very fast.

```
delay(500);
```

2.1.2 Interfacing LM335 using Python:

In this section, we shall describe how to read the voltage values from a LM335 connected to the analog pin 5 of the Arduino Uno board. The firmware code given in appendix has to be uploaded to the Arduino board and the Arduino.py code has to be in the same folder in which the LM335.py code is present. The LM335 has to be connected to the Arduino Uno board before doing these experiments and the Arduino Uno needs to be connected to the computer with a USB cable, as shown in Figure.10. After doing all the above, run the LM335.py code.

1. Declare the pins used for the LM335 using a variable.

```
self.lm335 = 5
```

2. The Python based command for the analog read functionality is given by,

```
val = self.obj_arduino.cmd_analog_in(1, self.lm335)
```

3. The input values are converted as shown in the previous section and printed

```
val=val*0.489-273
print(val)
sleep(2)
```

The entire reading and display operation is carried out as long as the Python code is running. To observe the values, one has to open the Python IDLE. The values displayed are the room temperature in degree Celsius.

2.2 LCD display:

The Liquid Crystal Display has a parallel interface. It means that the microcontroller operates several pins at once to control the LCD display.

The 16-pins present on the LCD display are discussed below:

- *Contrast Select (VEE)*: Pin3 will connect with power and ground through 3 pin potentiometers. It will help to control the contrast of PIXELS according to the 16X2 LCD light. 10K ohm variable resistor is

connected with the VEE pin to adjust light contrast. Variable resistor one side is connected with 5 volts and the other side is connected with ground. The third terminal is connected with the VEE pin.

- *RS*: The Register Select (RS) pin controls the memory of the LCD in which we write the data. We can select either the data register or the instruction register. The LCD looks for the upcoming instruction, which is present in the instruction register.
- *R/W*: The Read/Write pin selects the reading or writing mode.
- *E*: The Enable (E) mode is used to enable the writing to the registers. It sends the data to the data pins when the mode is HIGH.
- *D0 to D7*: These are eight data pins numbered as D0, D1, D3, D3, D4, D5, D6, and D7. We can set the state of the data pin either HIGH or LOW.
- *VSS*: It's a ground pin for common grounds.
- *VDD*: The power pin will use for voltage input to the 16X2 LCD.
- *Backlight Anode and Backlight Cathode*: These pins are used to power the backlight LED. They are connected to the 5V through a 220-ohm resistor and to the ground respectively.

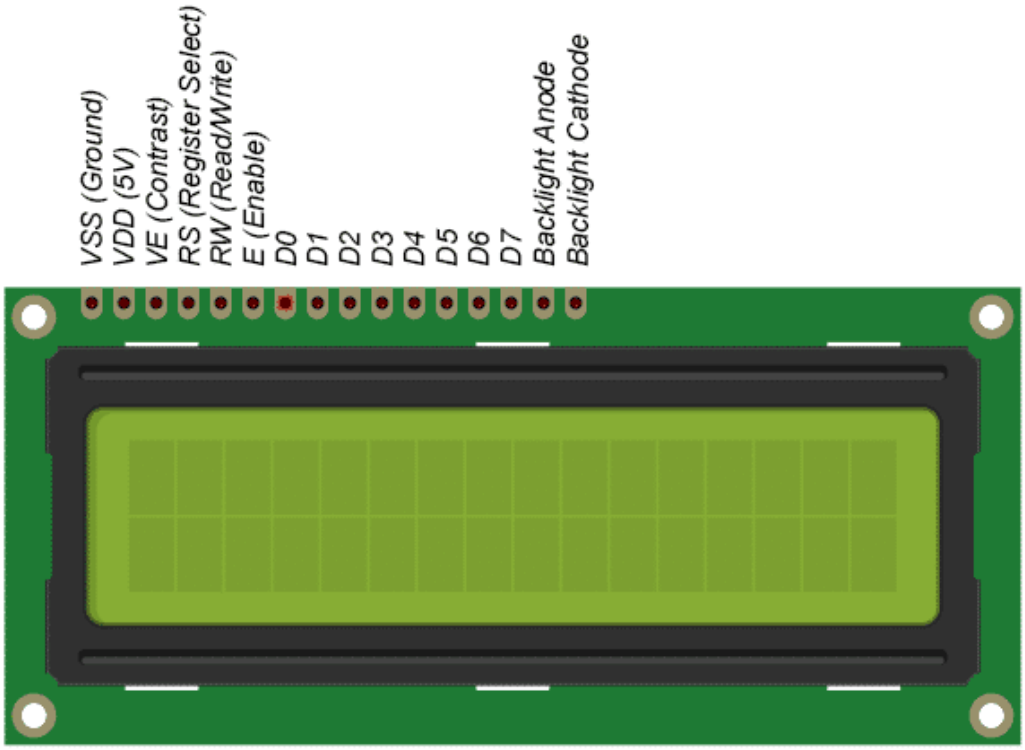

Figure 3: LCD pinout

There are two modes of operation for LCD:4-bit mode and 8-bit mode

- 8-bit mode: 8 bits of a byte are sent at the same time in pin DO to D7.
- 4-bit mode: 8 bits of a byte is sent two times, each time 4 bits in pin D4 to D7.

8-bit mode is faster than the 4-bit mode, but use more pins than 4-bit mode. The mode selection is performed at the initialization process by sending a command to LCD.
This project uses 4-bit mode, which is the most common-used.

In this mode, LCD's pins:

- 6 pins (RS, EN, D4, D5, D6, and D7) are connected to Arduino's pin.
- 4 pins (DO, DI, D2, and D3) are NOT connected.
- 6 remaining pins are connected to GND/VCC or potentiometer.

LCD Coordinate: LCD 16x2 includes 16 columns and 2 rows. the columns and rows are indexed from 0.

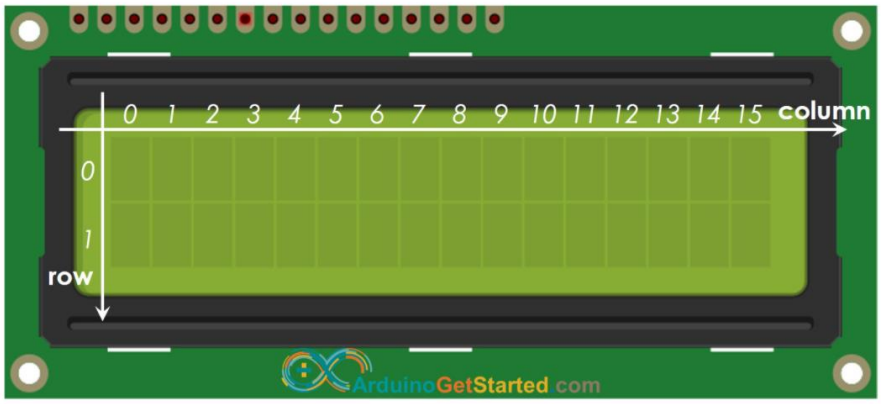

Figure 4:LCD position

The process of sending data (to be displayed) to LCD:

1. Arduino sets RS pin to HIGH (to select data register)
2. Arduino writes data to D4 - D7 pins (data bus).
3. LCD receives data on the data bus.
4. LCD stores the received data in the data resistor since the RS pin is HIGH. Then, LCD displays the data on the screen.

2.2.1 Interfacing the LCD display using Arduino IDE:

Controlling LCD is a quite complicated task. Fortunately, thanks to the LiquidCrystal library, this library simplifies the process of controlling LCD for you so we don't need to know the low-level instructions. We just need to connect Arduino to LCD and use the functions of the library.

1. Include the library:

```
#include<LiquidCrystal.h>
```

2. Define which Arduino's pin connected to six LCD's pins: RS, EN, D4, D4, D6, D7

```
const int RS=11, EN=12, D4=2, D5=3, D6=4, D7=5;
```

3. Declare a LiquidCrystal object:

```
LiquidCrystal lcd(RS,EN,D4,D5,D6,D7);
```

4. Set up the LCD's number of columns and rows.

```
lcd.begin(16,2);
```

5. Move cursor to the desired position (column_index, row_index)

```
lcd.setCursor(column_index, row_index);
```

6. Print a message to the LCD.

```
lcd.print("Hello World!");
```

2.2.2 Interfacing LCD display using Python

In this section, we discuss how to carry out the experiments of the previous section from Python. We will list the same experiment. The LCD should be attached to the Arduino Uno board before doing these experiments and the Arduino Uno needs to be connected to the computer with a USB cable, as shown in Fig.11. The firmware code given in appendix has to be uploaded to the Arduino board and the Arduino.py code has to be in the same folder in which the LM335.py code is present. After doing all the above, run the LM335.py code.

1. Declare the pins used for the LCD using a variable.

```
self.rs=12
self.en=11
self.d4=7
self.d5=6
self.d6=5
self.d7=4
```

2. The python-based command to print the string in LCD is:

```
self.obj_arduino.lcd_out(1,st,self.rs,self.en,self.d4,self.d5,self.d6,self.d7 )
```

2.3 ESP32:

ESP32 is a low-cost System on Chip (SoC) Microcontroller from Espressif Systems, the developers of the famous ESP8266 SoC. It is a successor to ESP8266 SoC and comes in both single-core and dual-core variations of the Tensilica's 32-bit Xtensa LX6 Microprocessor with integrated Wi-Fi and Bluetooth.

ESP32 has an integrated RF components like Power Amplifier, Low-Noise Receive Amplifier, Antenna Switch, Filters and RF Balun. This makes designing hardware around ESP32 very easy as we require very few external components. Designing battery operated applications like wearables, audio equipment, baby monitors, smart watches, etc., using ESP32 is very easy.

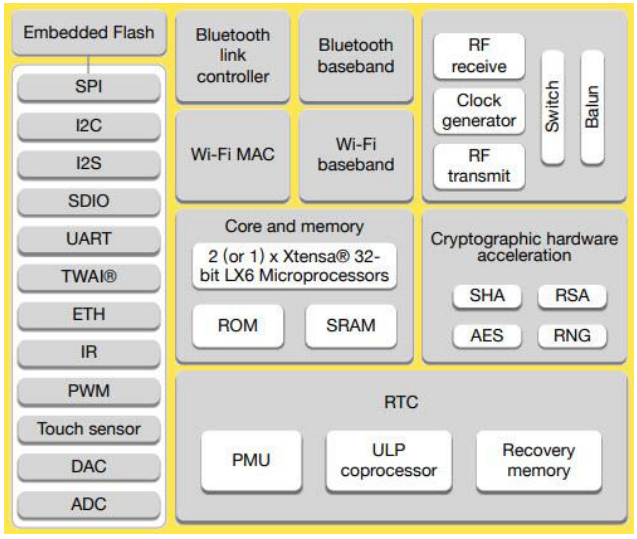

Figure 5:ESP32-Block-Diagram

Specifications:

- Single or Dual-Core 32-bit LX6 Microprocessor with clock frequency up to 240 MHz.
- 520 KB of SRAM, 448 KB of ROM and 16 KB of RTC SRAM.
- Supports 802.11 b/g/n Wi-Fi connectivity with speeds up to 150 Mbps.
- Support for both Classic Bluetooth v4.2 and BLE specifications.
- 34 Programmable GPIOs.
- Up to 18 channels of 12-bit SAR ADC and 2 channels of 8-bit DAC
- Serial Connectivity include 4 x SPI, 2 x I^2C, 2 x I^2S, 3 x UART.
- Ethernet MAC for physical LAN Communication (requires external PHY).
- 1 Host controller for SD/SDIO/MMC and 1 Slave controller for SDIO/SPI.
- Motor PWM and up to 16-channels of LED PWM.

- Secure Boot and Flash Encryption.
- Cryptographic Hardware Acceleration for AES, Hash (SHA-2), RSA, ECC and RNG.

Espressif Systems released several modules based on ESP32 and one of the popular options is the ESP-WROOM-32 Module. It consists of ESP32 SoC, a 40 MHz crystal oscillator, 4 MB Flash IC and some passive components.

The ESP32 DevKit Board contains the ESP-WROOM-32 as the main module and also some additional hardware to easily program ESP32 and make connections with the GPIO Pins.

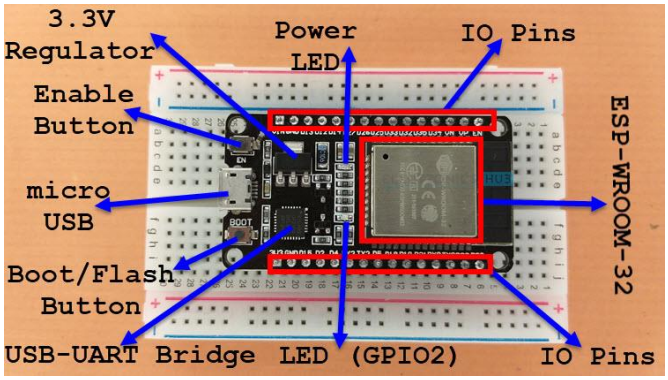

Figure 6:ESP32-DevKit-Board-Layout

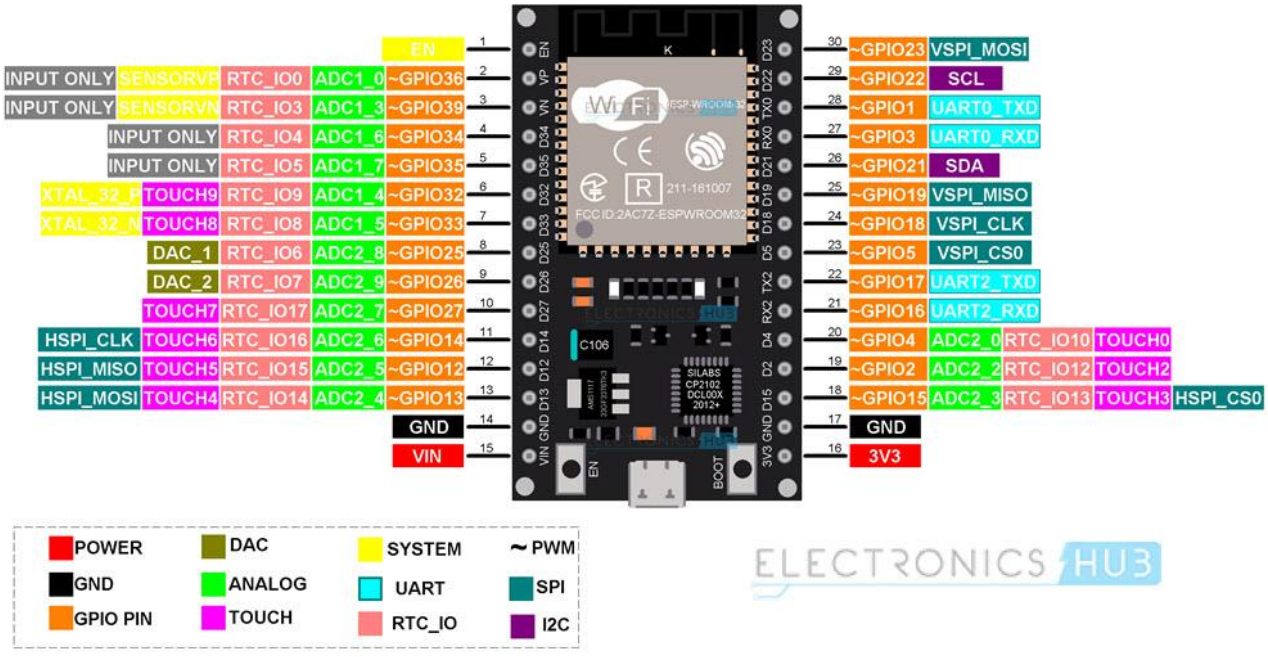

Figure 7:ESP32 pinout

2.3.1 UART communication between Arduino Uno and ESP32:

In this project I have read some data from Arduino Uno using ESP32 serially using UART communication protocol. To do this first we need to connect both the boards serially. Challenge over here is that our ESP32 board works on 3.3V whereas Arduino Uno works on 5V. To establish a proper communication channel between the two it is required to bring the voltage of Arduino board to 3.3V. To achieve this, I have made a voltage divider circuit using two 10k resistors.

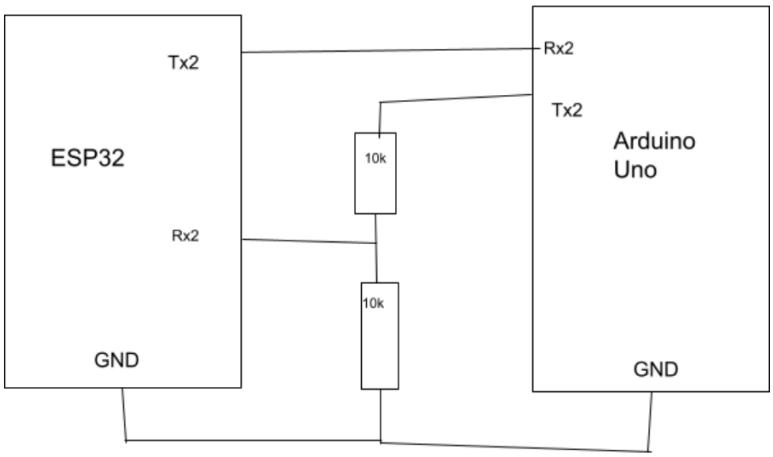

Figure 8:Arduino-ESP32 UART-Circuit schematic

2.3.2 Interfacing ESP32 using Arduino IDE

To print a string in a website from ESP32, we should connect the ESP32 to a Wi-Fi network. Then using the functions available in Arduino IDE and following the HTTPS protocol we can print the any string in an IP address, which can easily be accessed by any devices connected to the same network.

1. The string st is printed using the function available in ESP32.

```
client.println(st);
```

2. The string is followed by a line break.

```
client.println("<br>");
```

2.3.3 Interfacing ESP32 using Python

In this section, we discuss how to send a string to a webpage using ESP32 from Python. The ESP32 should be connected to the Arduino Uno board (as in Fig.12) before doing these experiments and the Arduino Uno needs to be connected to the computer with a USB cable. Give the SSID and password of the Wi-Fi network in the ESP32 firmware code. The Firmware code given in appendix should be uploaded to the Arduino board and the ESP32.

Note down the IP address from the Serial port of ESP32, which is usually same every time, when connected to the same Wi-Fi network. The blue light of ESP32 glows when its successfully connected to the Wi-Fi network. In case the LED doesn't glow press the EN(restart) button on the ESP32 and wait for 5 seconds. Arduino.py code has to be in the same folder in which the LM335.py code is present. After doing all the above, run the LM335.py code.

1. Declare the pins used for the Rx and Tx using a variable.

```
self.rx=2
self.tx=3
```

2. The python-based command to print the string st in webpage is:

```
self.obj_arduino.wifi_out(1,st,self.rx,self.tx )
```

3. Finally open the IP address obtained from the ESP32 Serial port on any device connected to the same Wi-Fi network to view the temperature.

2.4 Implemented Circuits:

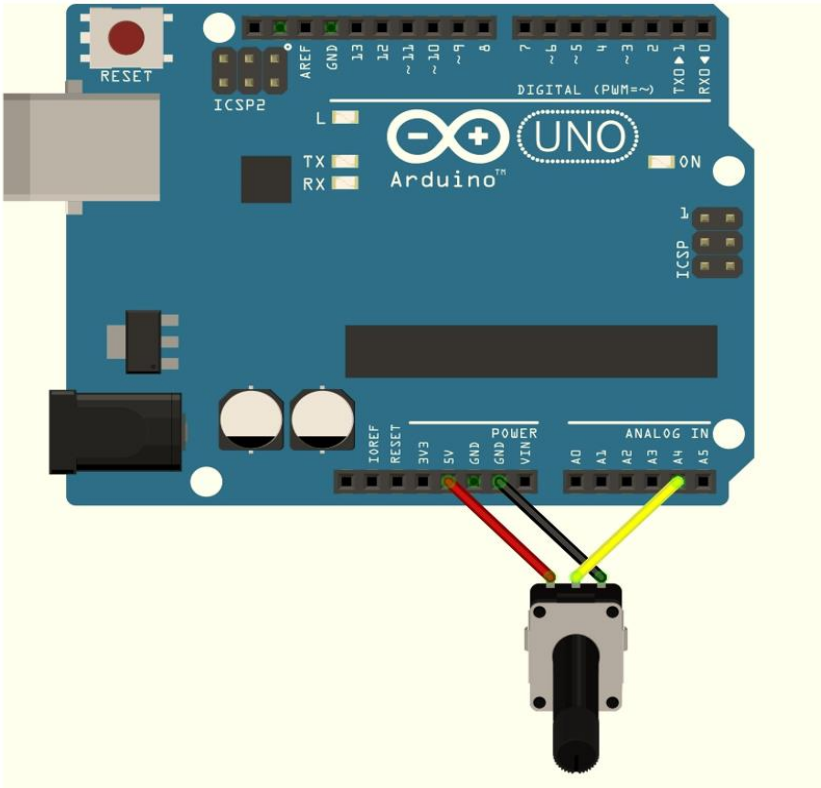

Figure 9:ADC

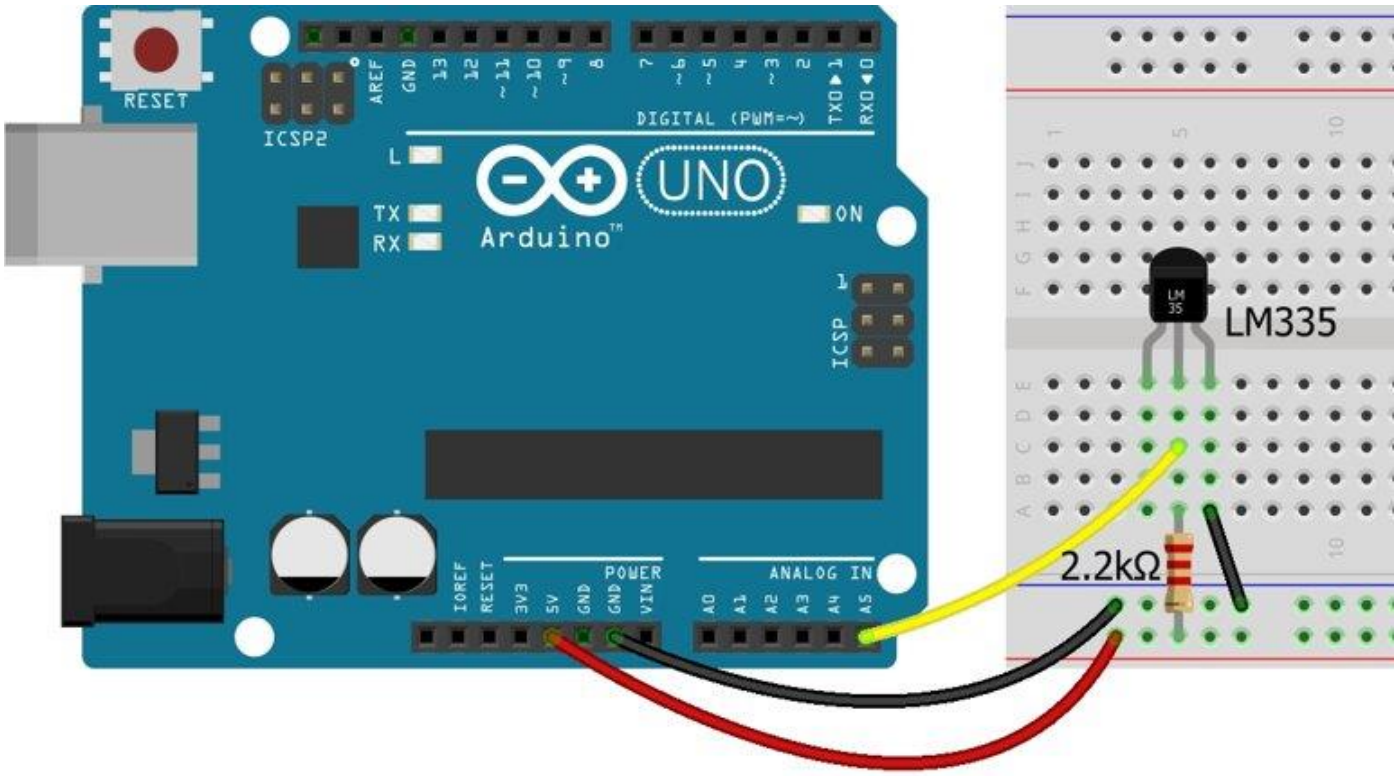

Figure 10:LM335

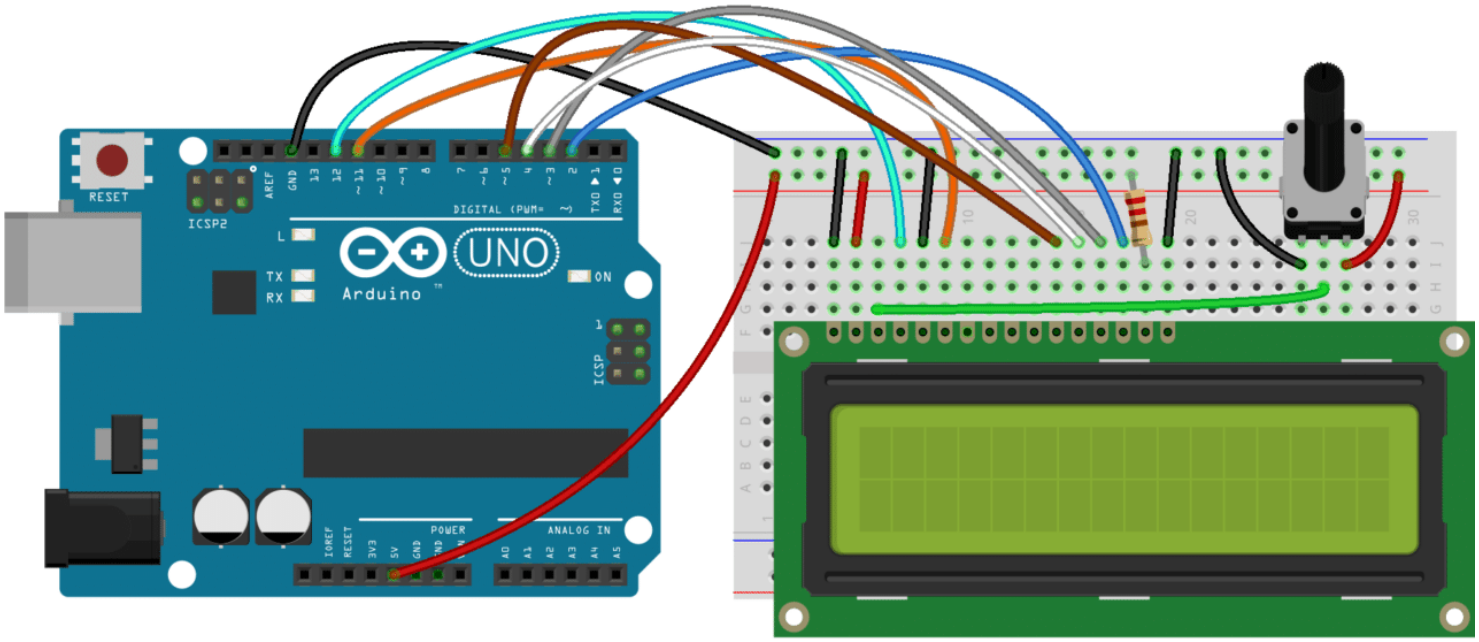

Figure 11:LCD display

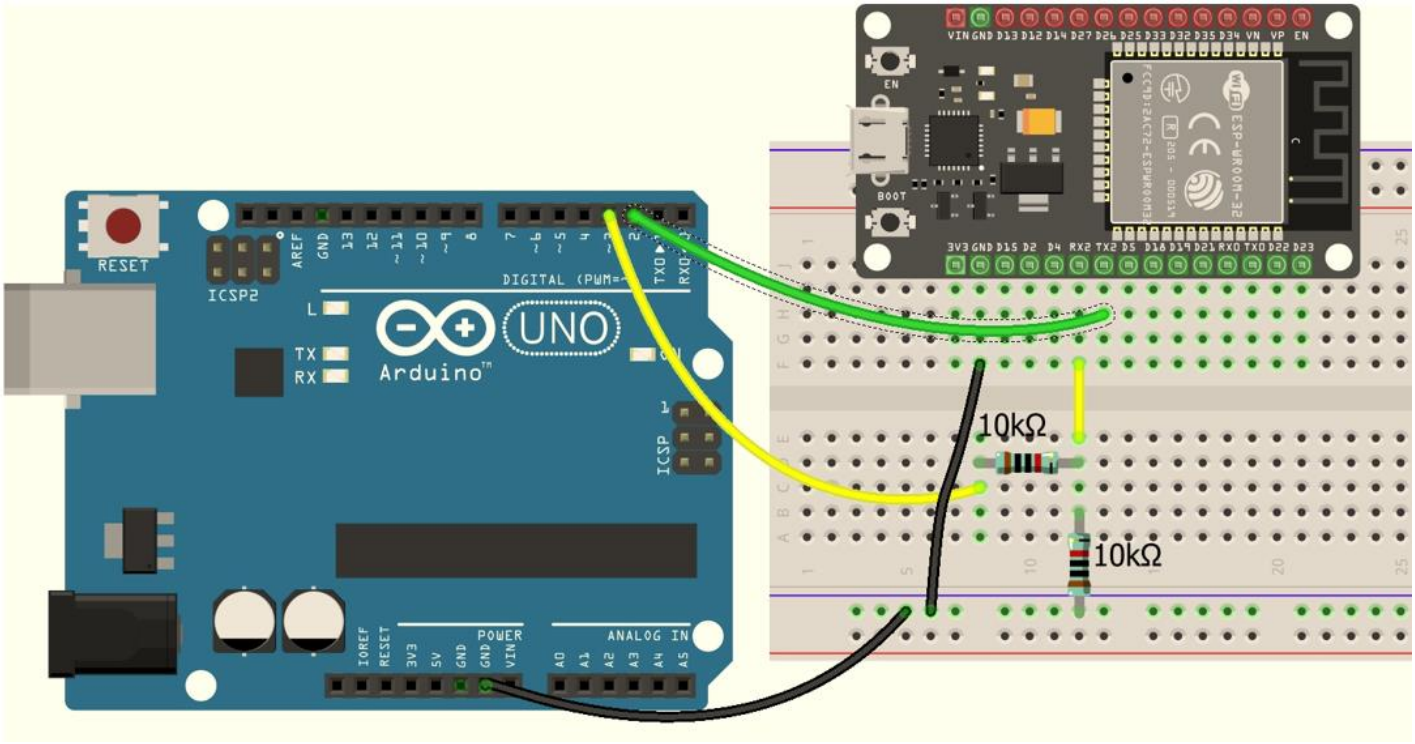

Figure 12:Arduino-ESP32 UART

3. Output screenshots

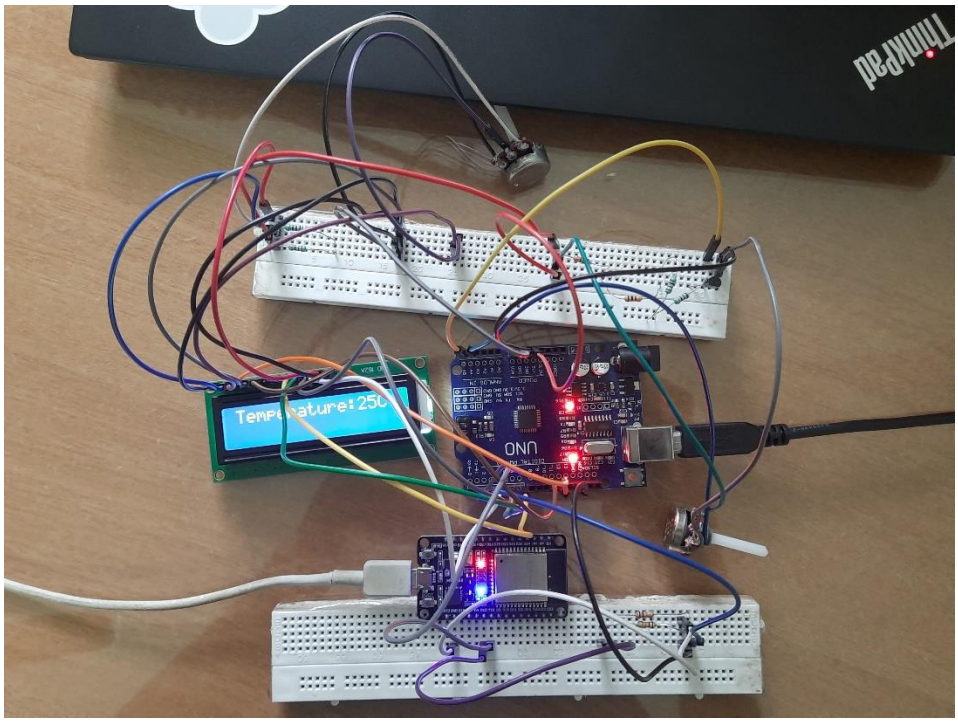

Figure 13: Project setup

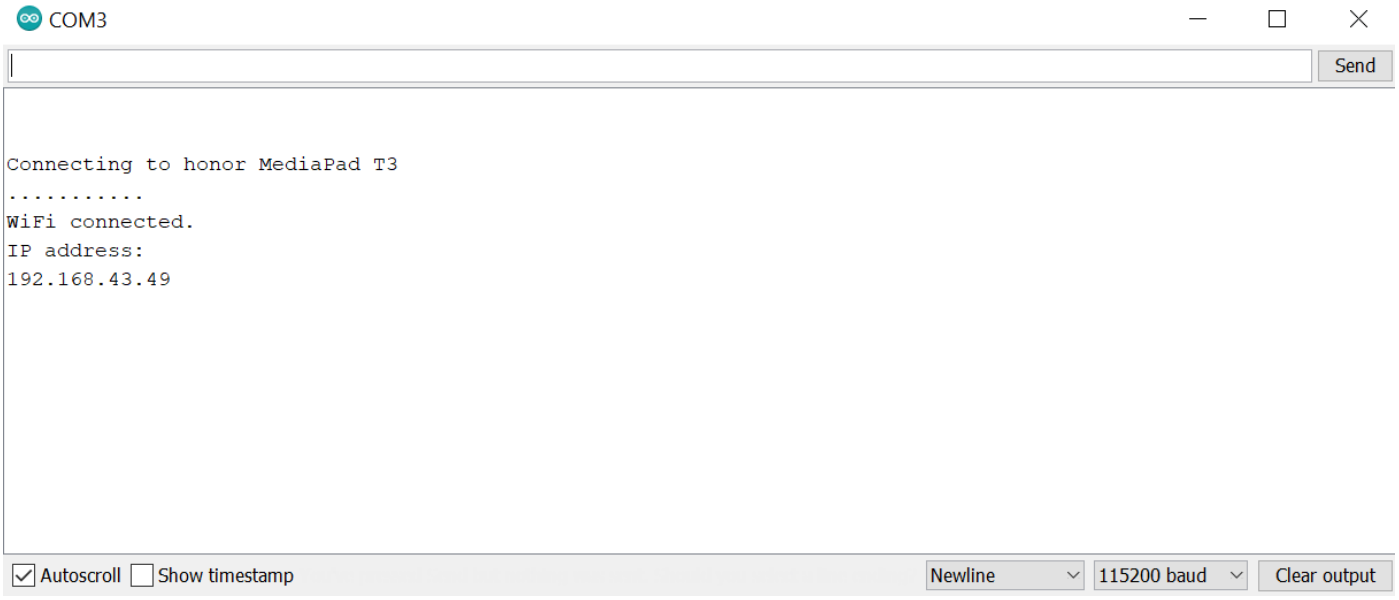

Figure 14: ESP-32 Serial port

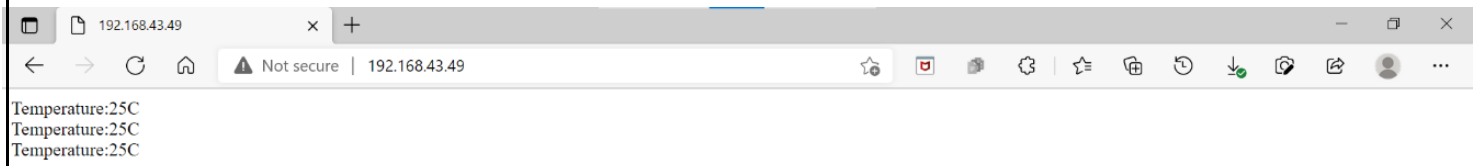

Figure 15:Temperature display in Webpage

Figure 16:Temperature display in Python IDLE