Wireless Temperature Monitoring System

By
Nithin K Shine
—
Under the guidance
of
Prof. Chanthini Baskar
——
Vellore Institute of Technology, Chennai

FOSSEE Project
Indian Institute of Technology Bombay

Contents

List of Figures

List of Tables

List of Arduino Code

List of Python Code

List of Acronyms

ACM	Abstract Control Model
ADC	Analog to Digital Converter
ADK	Accessory Development Kit
ALU	Arithmetic and Logic Unit
ARM	Advanced RISC Machines
BIOS	Basic Input/ Output System
CD	Compact Disc
CNES	National Centre for Space Studies
COM Port	Communication Port
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DC	Direct Current
DIY	Do It Yourself
DVD	Digital Versatile Disc
EEPROM	Electronically Erasable Programmable Read-Only Memory
FPGA	Field-programmable Gate Array
GNU	GNU's Not Unix
GPS	Global Positioning System
GPL	General Public License
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
LAPACK	Linear Algebra Package
LCD	Liquid Crystal Display
LDR	Light Dependent Resistor
LED	Light Emitting Diode

MRI	Magnetic Resonance Imaging
MISO	Master Input, Slave output
MOSI	Master out, Slave input
NTC	Negative Temperature Coefficient
OGP	Open Graphics Project
OS	Operating System
OSHW	Open Source Hardware
PCB	Printed Circuit Board
PTC	Positive Temperature Coefficient
PWM	Pulse width modulation
RAM	Random-access Memory
ROM	Read Only Memory
RS	Recommended Standard
RTC	Real Time Clock
Rx	Receiver
SD Card	Secure Digital Card
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TCL	Tool Command Language
Tx	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Chapter 1

Interfacing LM335, 16x2 LCD display and ESP32

This 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 Uno . Arduino Uno 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.

1.1 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: $Vout = VoutT_0 * \frac{T}{T_0}$, where T_0 is the known reference temperature and $VoutT_0$ is the corresponding output voltage. The nominal $VoutT_0$ is equal to

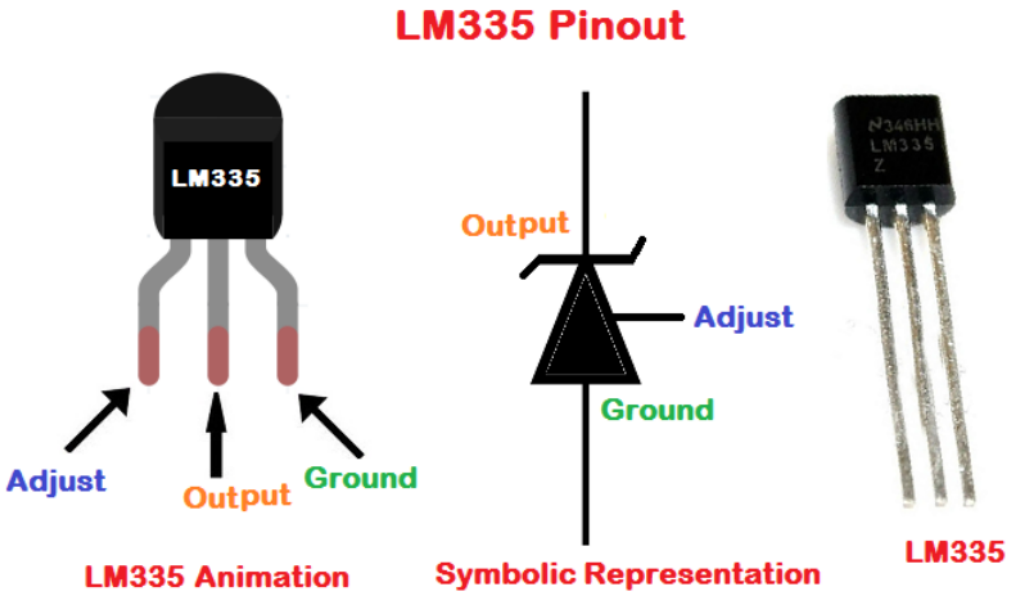

Figure 1.1: LM335 pinout

T_0 * 10 mV/K. So, at 25 C, $VoutT_0$ 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.

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 A5 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 A5 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.

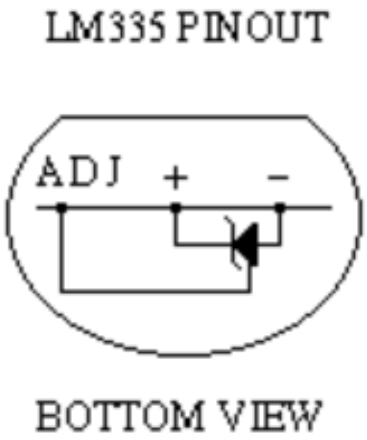

Figure 1.2: LM335 Circuit Schematic

1.1.1 Interfacing 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 `A5` 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 Fig. 1.3. Then run the code given in Arduino Code 1.1 using 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 by

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

   where `A5` represents the analog pin 5 to be read and the read LDR values are stored in the variable `val`.

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 in `Serial Monitor` using the code given below,

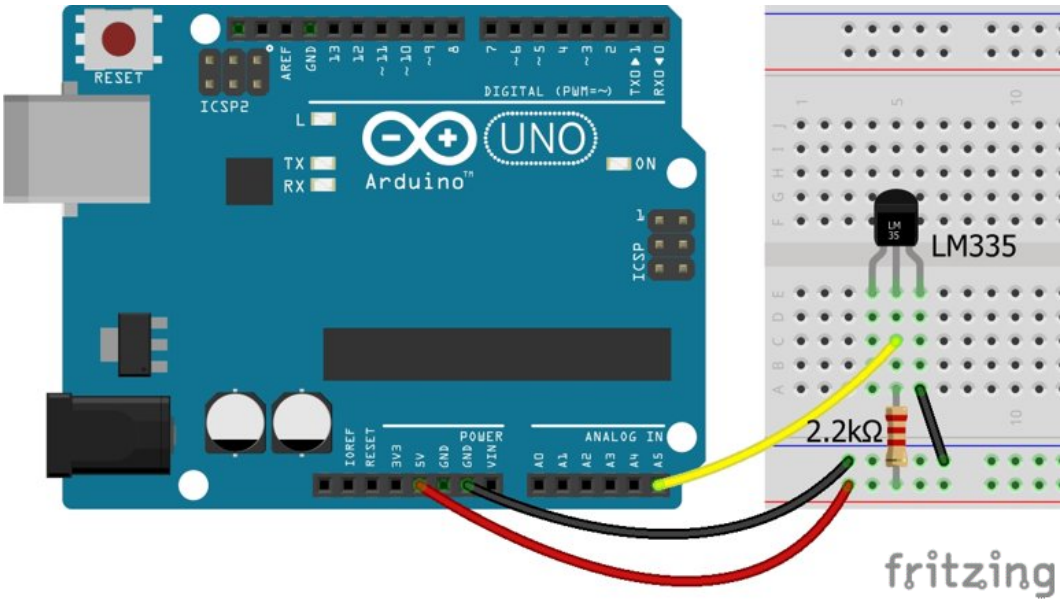

Figure 1.3: LM335 and Arduino Uno connected using a breadboard

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

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

```
1 delay(500);
```

Arduino Code 1.1 To Read and display the output from LM335. Available at `Origin/user-code/Tmp-Mon/arduino/LM335/LM335.ino`.

```
1  void setup()
2  {
3  Serial.begin(9600);
4  }
5  //main loop
6  void loop()
7  {
8  val = analogRead(A5);//value of LM335
9  val = val*(500/1023)-273;
10 Serial.println(val);//for display
11 delay(500);
12 }
```

1.1.2 Interfacing LM335 through Python

In this section, we shall describe how to read the voltage values from a LM335 connected to the analog pin A5 of the Arduino Uno board using Python. The firmware code given should be uploaded to the Arduino 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 Fig. 1.3. After doing all the above, run the Python Code 1.1.

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

```
1          self.lm335 = 5
```

.

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

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

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

```
1              val=val*0.489-273
2              val=int(val)
```

and printed ,

```
1              print(val)
2              sleep(2)
```

Python Code 1.1 To Read and display the output from LM335. Available at Origin/user-code/Tmp-Mon/python/TMP-mon.py.

```
1  import os
2  import sys
3  cwd = os.getcwd()
4  (setpath,Examples) = os.path.split(cwd)
5  sys.path.append(setpath)
6
7  from Arduino.Arduino import Arduino
8  from time import sleep
9
10 class LM335:
11     def __init__(self, baudrate):
12         self.baudrate = baudrate
13         self.setup()
14         self.run()
15         self.exit()
```

Figure 1.4: Temperature display in Python IDLE

```
16
17      def setup(self):
18          self.obj_arduino = Arduino()
19          self.port = self.obj_arduino.locateport()
20          self.obj_arduino.open_serial(1, self.port, self.baudrate)
21
22      def run(self):
23          #initializing pins
24          self.lm335 = 5
25          self.rs=12
26          self.en=11
27          self.d4=7
28          self.d5=6
29          self.d6=5
30          self.d7=4
31          self.rx=2
32          self.tx=3
33          #infinite loop
34          while True:
35              val = self.obj_arduino.cmd_analog_in(1, self.lm335)
36              val=val*0.489-273
37              val=int(val)
38              #String to be send
39              st="Temperature:"+str(val)+"C "
40              #sending string to LCD
41              self.obj_arduino.lcd_out(1,st,self.rs, self.en, self.d4,
```

```
    self.d5, self.d6, self.d7 )
42              sleep(2)
43              #Sending string to Webpage
44              self.obj_arduino.wifi_out(1,st,self.rx,self.tx )
45              print(val)
46              sleep(2)
47
48      def exit(self):
49          self.obj_arduino.close_serial()
50
51  def main():
52      obj_lm335 = LM335(115200)#Creating a object
53
54  if __name__=='__main__':
55      main()
```

1.2 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's 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.

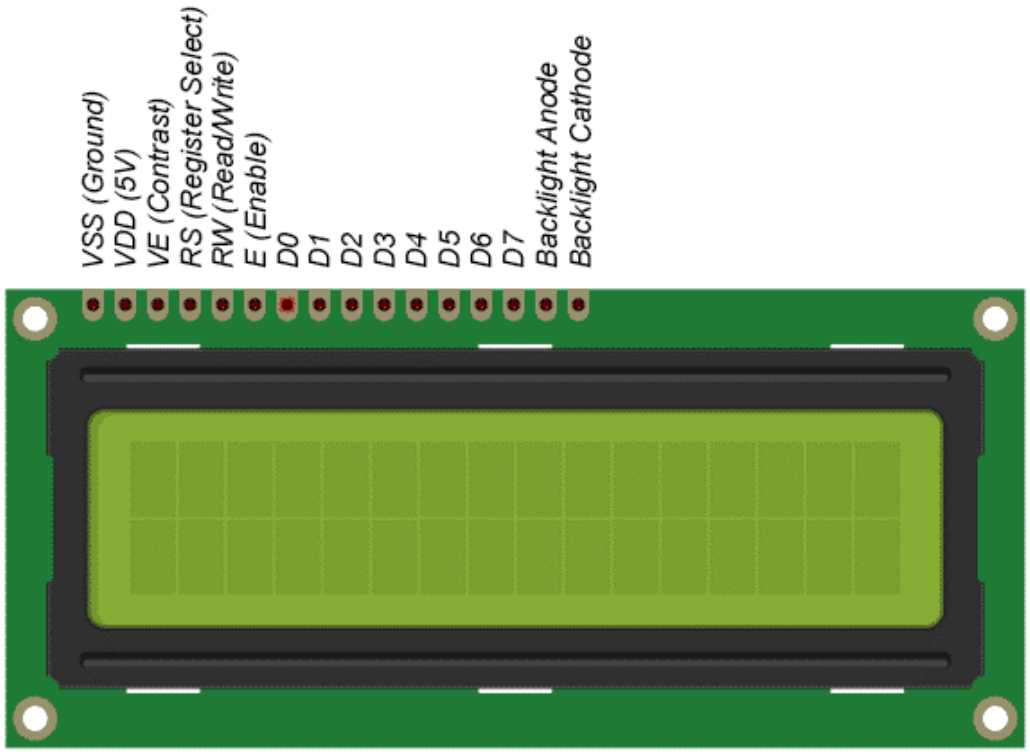

Figure 1.5: LCD pinout

- *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.

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 are connected as follows:

- 6 pins (RS, EN, D4, D5, D6, and D7) are connected to Arduino's pin.

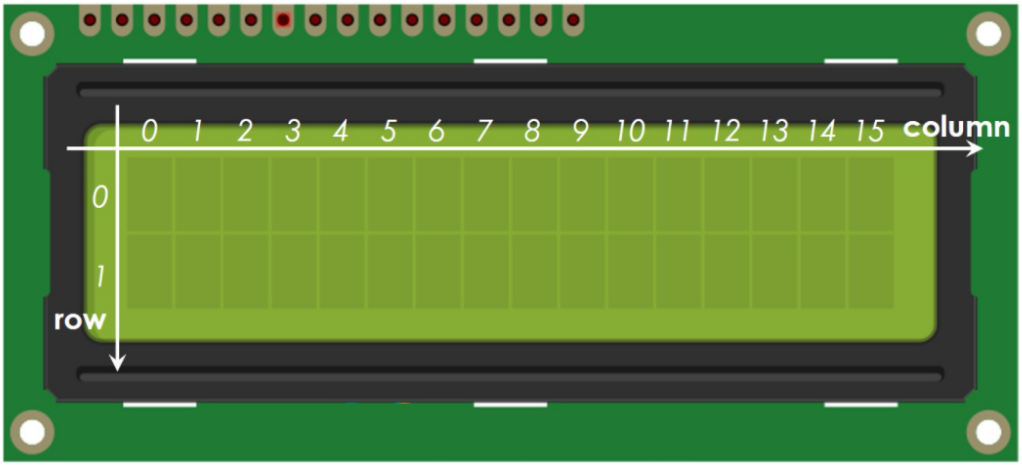

Figure 1.6: LCD Coordinates

- 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.

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.

1.2.1 Interfacing LCD display through the 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 as shown in Fig. 1.7 and use the functions of the library.

1. Include the library:

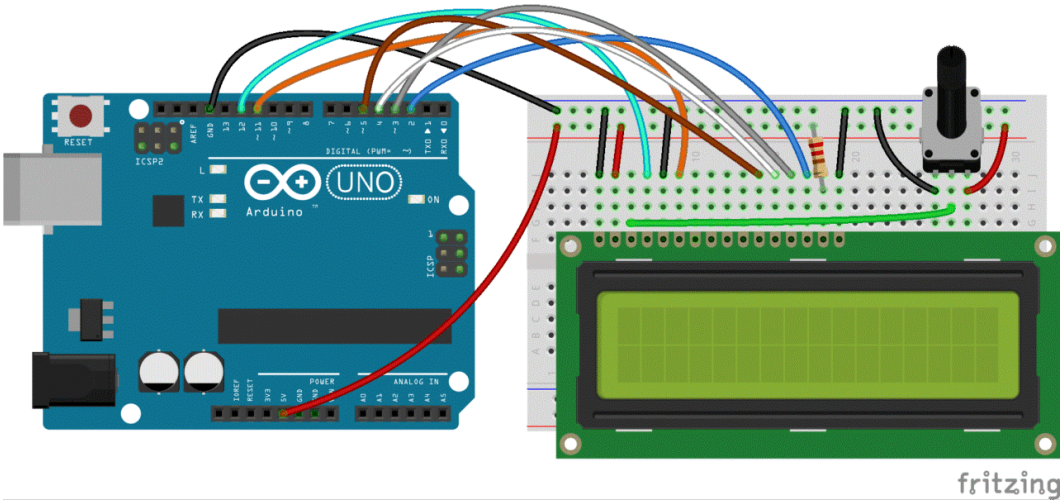

Figure 1.7: LCD and Arduino Uno connected using a breadboard

```
1 #include <LiquidCrystal.h>//For LCD display
```

2. Define which Arduino's pin connected to six LCD's pins: `RS`, `EN`, `D4`, `D5`, `D6`, `D7`

```
1 int rs=12,en=11,d4=7,d5=6,d6=5,d7=4;    /* pins used for LCD
      display       */
```

3. Declare a LiquidCrystal object,

```
1 LiquidCrystal lcd(rs, en, d4, d5, d6, d7); //Declare a
      LiquidCrystal object
```

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

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

5. Move cursor to the desired position,

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

6. Print a message to the LCD.

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

Arduino Code 1.2 To display a string in LCD. Available at `Origin/user-code/Tmp-Mon/arduino/LCD/LCD.ino`.

```
#include <LiquidCrystal.h>//For LCD display
int rs=12,en=11,d4=7,d5=6,d6=5,d7=4;   /* pins used for LCD display
    */
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); //Declare a LiquidCrystal
    object
int column_index=0, row_index=0;
void setup() {
  // put your setup code here, to run once:
  lcd.begin(16, 2);
  lcd.setCursor(column_index, row_index);
  lcd.print("Hello World!");
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

1.2.2 Interfacing LCD display through 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. 1.7. The firmware code should be uploaded to the Arduino board. After doing all the above, run the code given in Python Code 1.2.

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 )
```

Python Code 1.2 To display a string in LCD. Available at `Origin/user-code/Tmp-Mon/python/TMP-mon.py`.

```
import os
import sys
cwd = os.getcwd()
```

```
4 (setpath,Examples) = os.path.split(cwd)
5 sys.path.append(setpath)
6
7 from Arduino.Arduino import Arduino
8 from time import sleep
9
10 class LM335:
11     def __init__(self, baudrate):
12         self.baudrate = baudrate
13         self.setup()
14         self.run()
15         self.exit()
16
17     def setup(self):
18         self.obj_arduino = Arduino()
19         self.port = self.obj_arduino.locateport()
20         self.obj_arduino.open_serial(1, self.port, self.baudrate)
21
22     def run(self):
23         #initializing pins
24         self.lm335 = 5
25         self.rs=12
26         self.en=11
27         self.d4=7
28         self.d5=6
29         self.d6=5
30         self.d7=4
31         self.rx=2
32         self.tx=3
33         #infinite loop
34         while True:
35             val = self.obj_arduino.cmd_analog_in(1, self.lm335)
36             val=val*0.489-273
37             val=int(val)
38             #String to be send
39             st="Temperature:"+str(val)+"C "
40             #sending string to LCD
41             self.obj_arduino.lcd_out(1,st,self.rs, self.en, self.d4,
    self.d5, self.d6, self.d7 )
42             sleep(2)
43             #Sending string to Webpage
44             self.obj_arduino.wifi_out(1,st,self.rx,self.tx )
45             print(val)
46             sleep(2)
47
48     def exit(self):
49         self.obj_arduino.close_serial()
50
51 def main():
```

```
52      obj_lm335 = LM335(115200)#Creating a object
53
54  if __name__=='__main__':
55      main()
```

1.3 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.

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 I2C, 2 x I2S, 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.

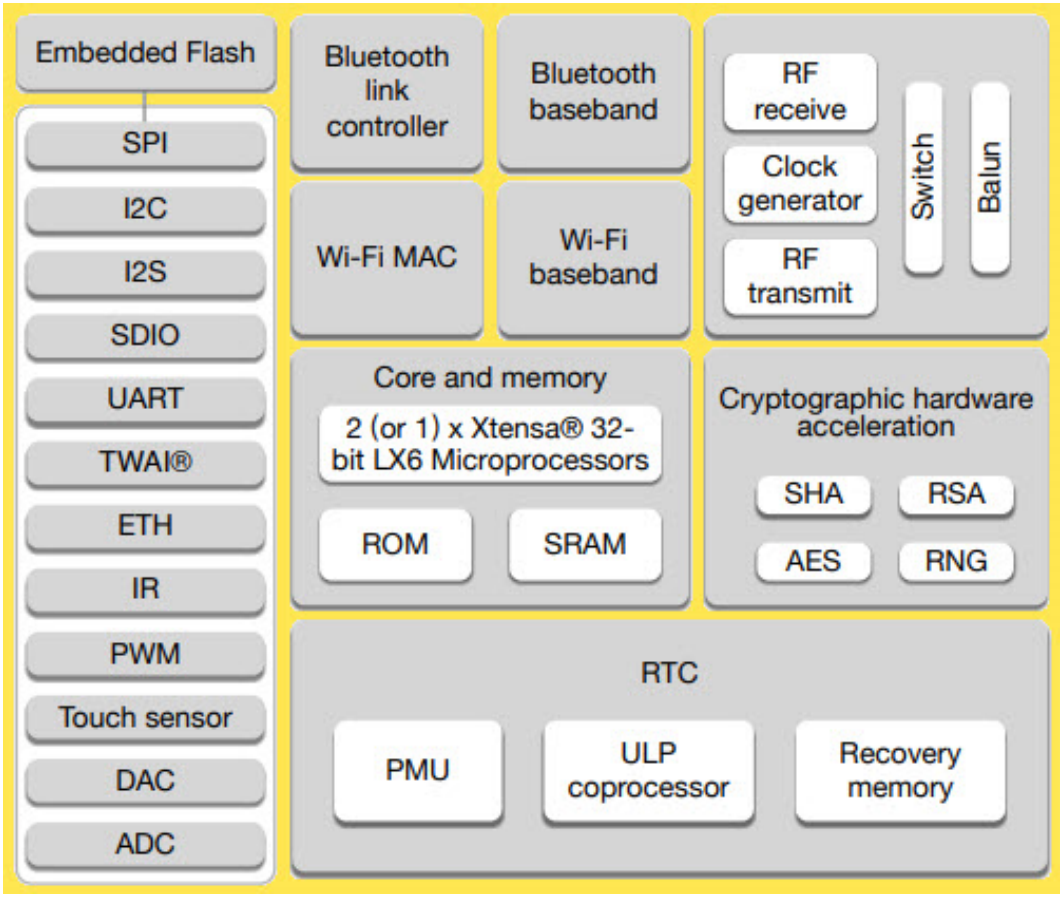

Figure 1.8: ESP32-Block-Diagram

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.

1.3.1 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

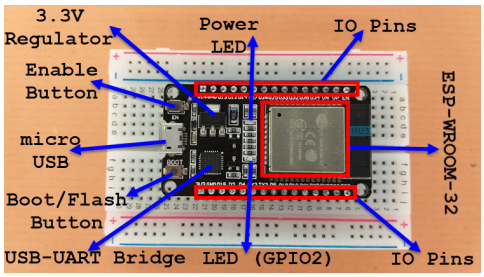

Figure 1.9: ESP32 DevKit Board Layout

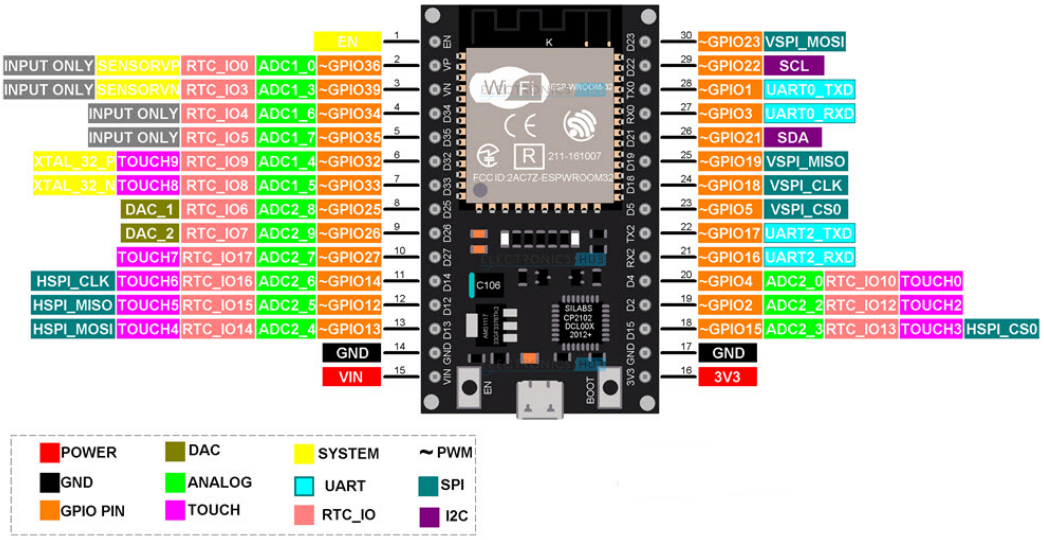

Figure 1.10: ESP32 pinout

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. As shown in Fig. 1.11

1.3.2 Interfacing ESP32 through the 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 any string in an IP address, which can easily be accessed by any devices connected to the same network. The code for this is given below, Arduino Code 1.3 is to be uploaded to the ESP32 and Arduino Code 1.4 is to be uploaded to Arduino Uno. The circuit is given in Fig. 1.12

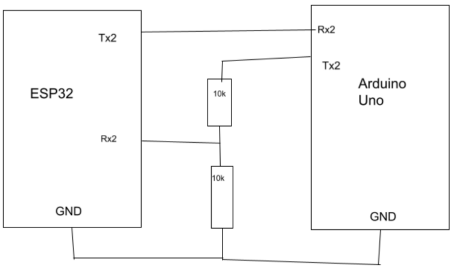

Figure 1.11: Arduino-ESP32 UART-Circuit schematic

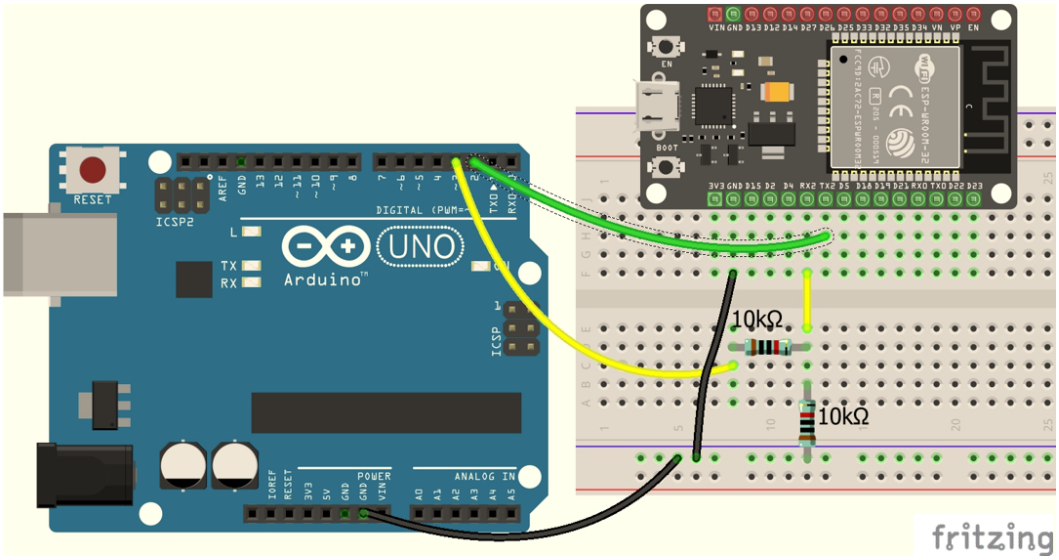

Figure 1.12: Arduino-ESP32 UART connection using breadboard

1. The string `st` is received from Arduino through UART communication protocol and is send to a website using the functions available in WiFi.h library

```
client.println(Serial2.readString());
```

2. The string is followed by a line break,

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

Arduino Code 1.3 Uploaded to ESP32 to read and display the value sent by Arduino Uno(through UART protocol) in a webpage. Available at `Origin/user-code/Tmp-Mon/arduino/esp-uart/esp-uart.ino`.

```
/* This file is meant to be used with the FLOSS-Arduino toolbox*/

#include <WiFi.h>
const char* ssid     = "SSID";      //SSID of the WiFi network to which ESP32 has to be connected
const char* password = "PASSWORD";
WiFiServer server(80);
#define RXp2 16
#define TXp2 17
char c;

void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, RXp2, TXp2);
  pinMode(LED_BUILTIN,OUTPUT);
}

void loop() {
    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    digitalWrite(LED_BUILTIN,HIGH);//to turn on LED
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    server.begin();
    while(1){
         WiFiClient client = server.available();   // listen for incoming clients

        if (client) {                             // if you get a client,
          Serial.println("New Client.");           // print a message out the serial port
          String currentLine = "";                // make a String to hold incoming data from the client
          while (client.connected()) {            // loop while the client's connected
            if (client.available()) {             // if there's bytes to read from the client,
```

```
43                char c = client.read();             // read a byte, then
44                Serial.write(c);                    // print it out the
      serial monitor
45                if (c == '\n') {                    // if the byte is a
      newline character
46
47                  // if the current line is blank, you got two newline
      characters in a row.
48                  // that's the end of the client HTTP request, so send a
       response:
49                  if (currentLine.length() == 0) {
50                    // HTTP headers always start with a response code (e.
      g. HTTP/1.1 200 OK)
51                    // and a content-type so the client knows what's
      coming, then a blank line:
52                    client.println("HTTP/1.1 200 OK");
53                    client.println("Content-type:text/html");
54                    client.println();
55
56                    // the content of the HTTP response follows the
      header:
57                    while(Serial2.available()){
58                      client.println(Serial2.readString());
59                      client.print("   <br>");
60                    }
61                    // The HTTP response ends with another blank line:
62                    client.println();
63                    // break out of the while loop:
64                    break;
65                  } else {    // if you got a newline, then clear
      currentLine:
66                    currentLine = "";
67                  }
68                } else if (c != '\r') {  // if you got anything else but
      a carriage return character,
69                  currentLine += c;      // add it to the end of the
      currentLine
70                }
71              }
72            }
73            // close the connection:
74            client.stop();
75            Serial.println("Client Disconnected.");
76          }
77      }
78  }
```

Arduino Code 1.4 Uploaded to Arduino to Sent data to ESP32 through UART protocol. Available at `Origin/user-code/Tmp-Mon/arduino/ard-uart/ar`

d-uart.ino.

```
#include "SoftwareSerial.h"

SoftwareSerial wifi(2,3) ; // rx , tx

void setup()
{
Serial.begin(9600);
wifi.begin(9600);

}

void loop()

{

  Serial.println(" I am sending message to Webserver using ESP32");
  wifi.println(" Message from Arduino   ");
  delay(1000);
}
```

1.3.3 Interfacing ESP32 through 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. 1.12) 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 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. After doing all the above, run the Python Code 1.3.

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.

Python Code 1.3 To Send a string to a website. Available at `Origin/user-code/Tmp-Mon/python/TMP-mon.py`.

```
import os
import sys
cwd = os.getcwd()
(setpath,Examples) = os.path.split(cwd)
sys.path.append(setpath)

from Arduino.Arduino import Arduino
from time import sleep

class LM335:
    def __init__(self, baudrate):
        self.baudrate = baudrate
        self.setup()
        self.run()
        self.exit()

    def setup(self):
        self.obj_arduino = Arduino()
        self.port = self.obj_arduino.locateport()
        self.obj_arduino.open_serial(1, self.port, self.baudrate)

    def run(self):
        #initializing pins
        self.lm335 = 5
        self.rs=12
        self.en=11
        self.d4=7
        self.d5=6
        self.d6=5
        self.d7=4
        self.rx=2
        self.tx=3
        #infinite loop
        while True:
            val = self.obj_arduino.cmd_analog_in(1, self.lm335)
            val=val*0.489-273
            val=int(val)
            #String to be send
            st="Temperature:"+str(val)+"C "
            #sending string to LCD
            self.obj_arduino.lcd_out(1,st,self.rs, self.en, self.d4,
    self.d5, self.d6, self.d7 )
            sleep(2)
            #Sending string to Webpage
            self.obj_arduino.wifi_out(1,st,self.rx,self.tx )
            print(val)
```

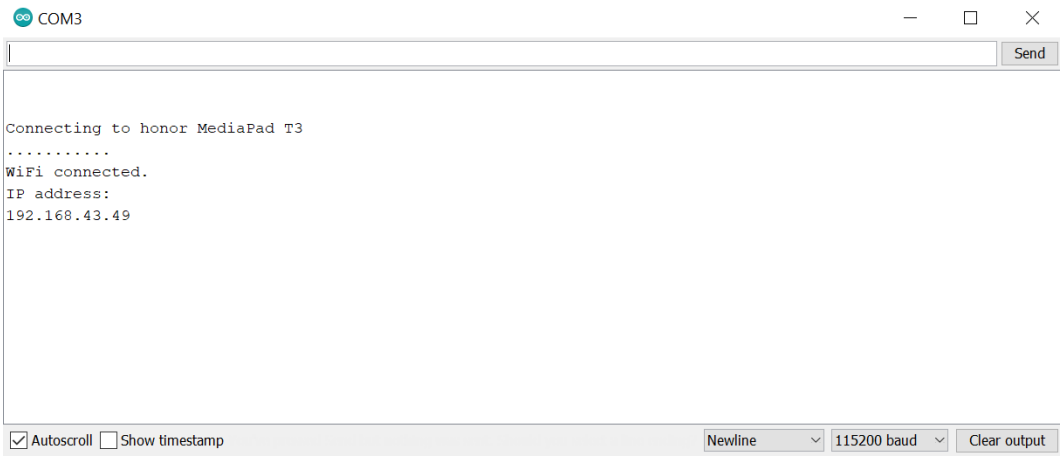

Figure 1.13: ESP-32 Serial port

```
46              sleep(2)
47
48      def exit(self):
49          self.obj_arduino.close_serial()
50
51  def main():
52      obj_lm335 = LM335(115200)#Creating a object
53
54  if __name__=='__main__':
55      main()
```

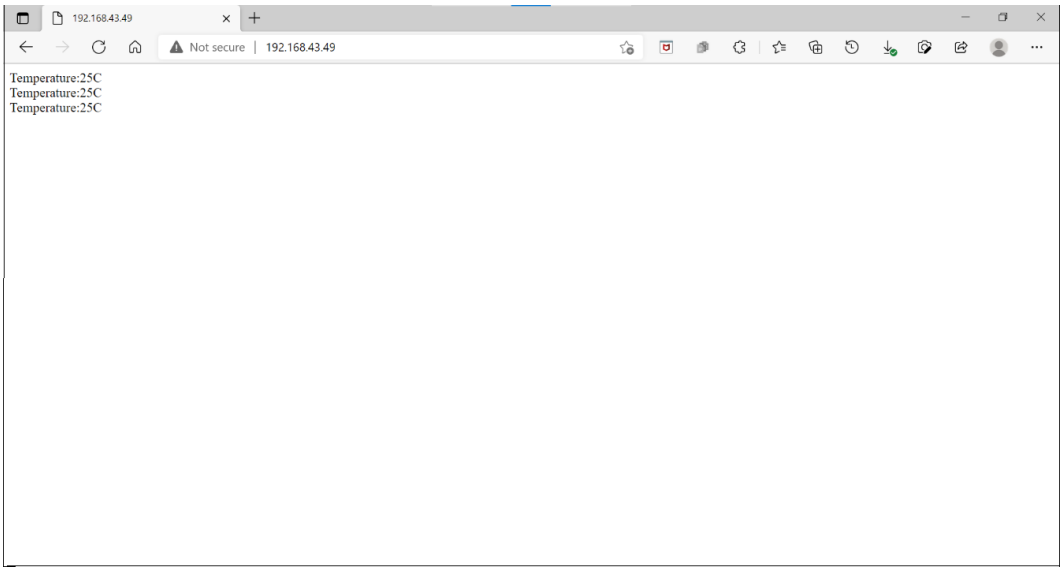

Figure 1.14: Temperature display in Webpage

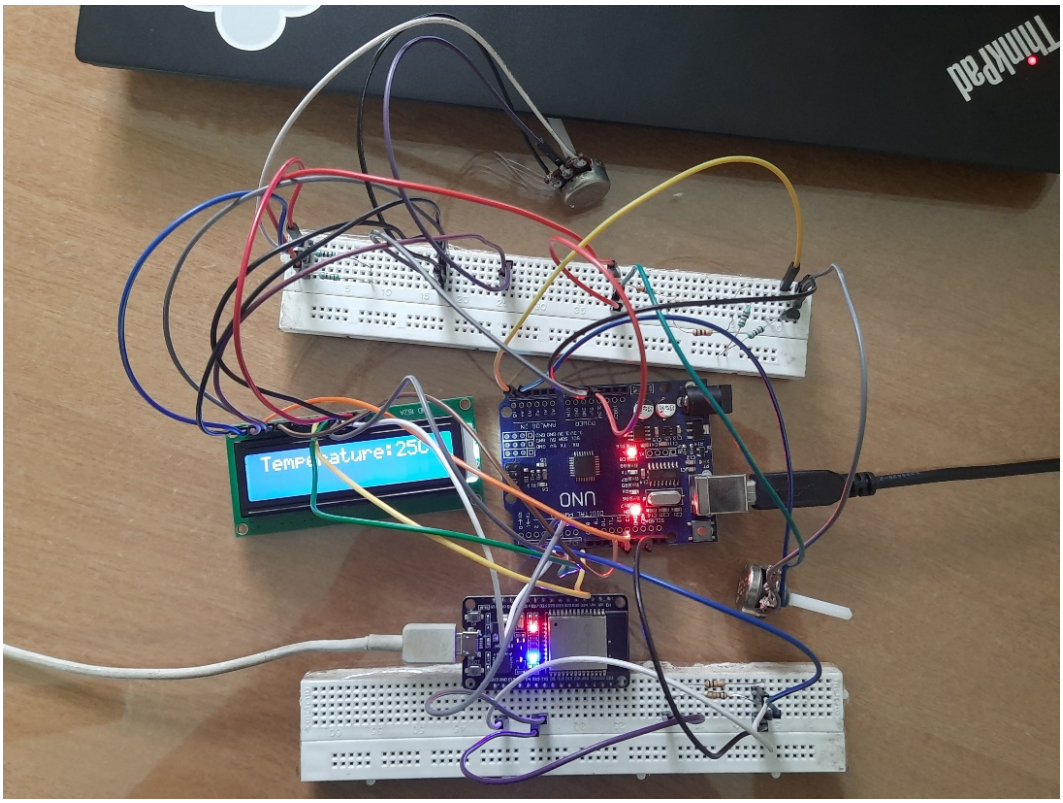

Figure 1.15: Project Setup

References

[1] T. Martin. Use of scilab for space mission analysis. https://www.scilab.org/community/scilabtec/2009/Use-of-Scilab-for-space-mission-analysis. Seen on 28 June 2015.

[2] B. Jofret. Scilab arduino toolbox. http://atoms.scilab.org/. Seen on 28 June 2015.

[3] oshwa.org. http://www.oshwa.org/definition. Seen on 28 June 2015.

[4] Mateo Zlatar. Open source hardware logo. http://www.oshwa.org/open-source-hardware-logo. Seen on 28 June 2015.

[5] Arduino uno. https://www.arduino.cc/en/uploads/Main/ArduinoUnoFront240.jpg. Seen on 28 June 2015.

[6] Arduino mega. https://www.arduino.cc/en/uploads/Main/ArduinoMega2560_R3_Fronte.jpg. Seen on 28 June 2015.

[7] Lilypod arduino. https://www.arduino.cc/en/uploads/Main/LilyPad_5.jpg. Seen on 28 June 2015.

[8] Arduino phone. http://www.instructables.com/id/ArduinoPhone/. Seen on 28 June 2015.

[9] Candy sorting machine. http://beta.ivc.no/wiki/index.php/Skittles_M%26M%27s_Sorting_Machine. Seen on 28 June 2015.

[10] 3d printer. http://www.instructables.com/id/Arduino-Controlled-CNC-3D-Printer/. Seen on 28 June 2015.

[11] Shield. http://codeshield.diyode.com/about/schematics/. Seen on 28 June 2015.

[12] scilab.org. http://www.scilab.org/scilab/about. Seen on 28 June 2015.

[13] scilab.org. http://www.scilab.org/scilab/interoperability. Seen on 28 June 2015.

[14] scilab.org. http://www.scilab.org/scilab/features/xcos. Seen on 28 June 2015.

[15] python.org. https://www.python.org/doc/essays/blurb/. Seen on 24 February 2021.

[16] pyserial - pypi. https://pypi.org/project/pyserial/. Seen on 21 April 2021.

[17] julialang.org/. https://julialang.org/. Seen on 12 April 2021.

[18] Juliaio/serialports.jl: Serialport io streams in julia backed by pyserial. https://github.com/JuliaIO/SerialPorts.jl. Seen on 15 April 2021.

[19] Openmodelica. https://www.openmodelica.org/. Seen on 2 April 2021.

[20] Thermistor - wikipedia. https://en.wikipedia.org/wiki/Thermistor. Seen on 2 May 2021.

[21] Secrets of arduino pwm. https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM. Seen on 5 May 2021.

[22] Servo. https://www.arduino.cc/reference/en/libraries/servo/. Seen on 6 May 2021.

[23] Modbus. https://modbus.org/. Seen on 6 May 2021.

[24] Paavni Shukla, Sonal Singh, Tanmayee Joshi, Sudhakar Kumar, Samrudha Kelkar, Manas R. Das, and Kannan M. Moudgalya. Design and development of a modbus automation system for industrial applications. In *2017 6th International Conference on Computer Applications In Electrical Engineering-Recent Advances (CERA)*, pages 515–520, 2017.

[25] Simply modbus. https://simplymodbus.ca/. Seen on 6 May 2021.

[26] Online crc. https://www.lammertbies.nl/comm/info/crc-calculation. Seen on 2 May 2021.

[27] Floating point converter. https://www.h-schmidt.net/FloatConverter/IEEE754.html. Seen on 6 May 2021.