ESP32 as wireless temperature sensor and actuator controlled through internet.

Author: Moises Reyes Pardo.

January 11, 2018

Contents

1 Main goal

The main goal of this project is measure the temperature in a place and also generate an action to control it remotely through internet.

The final purpose is accomplished crossing frontiers, linking different networking tools, hardware and programming languages.

2 Equipment used

2.1 Modem / Router

The whole system is deployed over a LAN, mixing in this case wired and wireless connections. To reach the system through internet the system itself has to be connected to it. To do that, every house with an internet connection has a modem/router, which is used in this project to interconnect the devices and give internet access to the system.

There is not a technical description of this device in this document because the equipment used was given by the internet service provider (ISP) and all companies provide different devices, but with quite similar characteristics.

2.2 BeagleBone Black

Table 1: BeagleBone Black - Relevant specifications.

BeagleBone model:	BeagleBone Black Rev C
O.S. installed:	bone-debian-9.2-iot-armhf-2017-10-10-4gb.img
Kernel release [uname -r]:	4.4.91-ti-r133
O.S. running [uname -o]:	GNU/Linux
More about this device:	http://beagleboard.org/black

2.3 ESP32 module

Table 2: ESP32 - Relevant specifications.

ESP32 model:	ESP32 DevKit v1
Key features:	2.4 GHz Wi-Fi and Bluetooth combo chip. MCU + Advanced Peripheral Interfaces
More about this device:	`http://esp32.net`

3 Final system and subsystems that make it up.

3.1 General system diagram.

The "Figure 1" shows a general system diagram. How the devices interact each other, which role play each one and which tasks are executed inside of them.

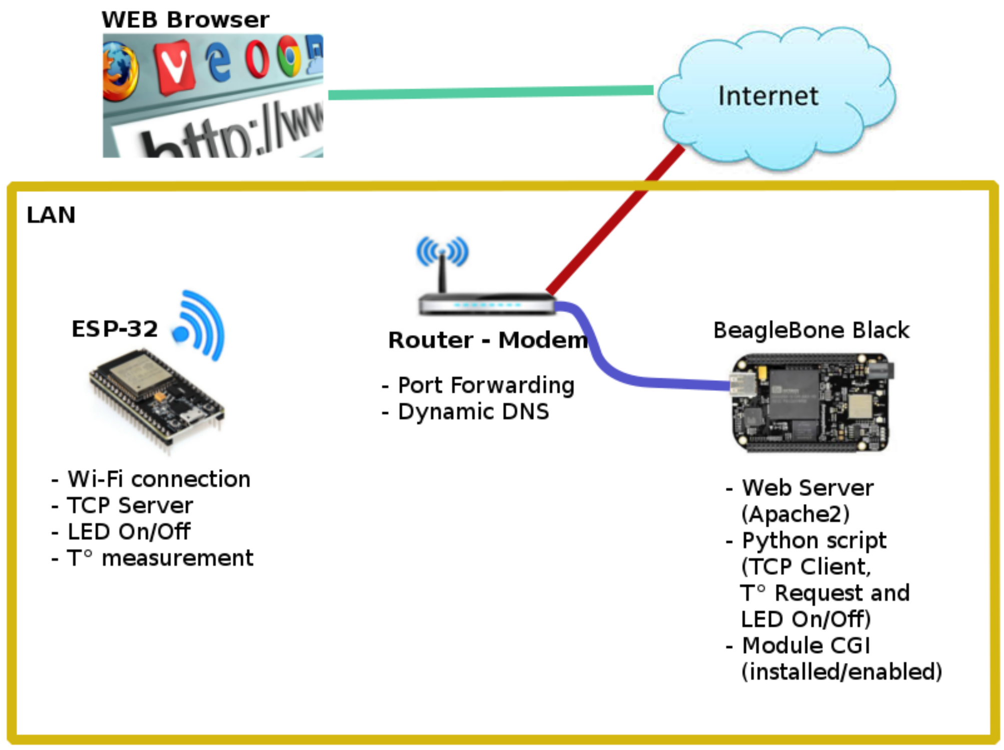


Figure 1: General system diagram.

3.2 Subsystems - what the devices do and how they do it.

This project is internally divided in several subsystems and all of them work together to achieve an unique bound system. Each device plays more than one role and execute more than one task.

3.2.1 ESP32 module

The "Figure 2" shows a flow diagram to explain the logic behind the code used to program the execution of all needed tasks on the ESP32 module.

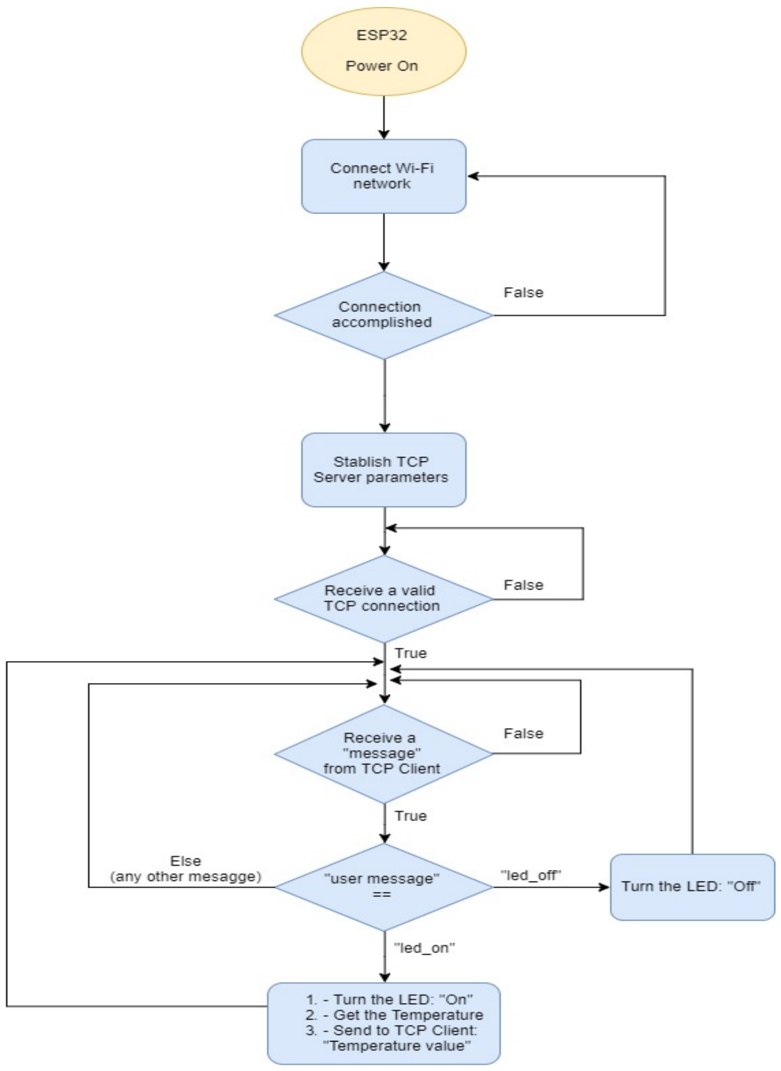


Figure 2: Flow diagram, "ESP32_LED_TEMP" software.

A chip DS18B20 was used as temperature sensor. To program the ESP32 device the Arduino IDE and his libraries were used. Now the complete source code of "ESP32_LED_TEMP" software is given.

```
#include <WiFi.h>
#include <OneWire.h>

/* Global statements required for WiFi connection */
const char* ssid = "your_Wi-Fi_network";
const char* password =  "your_Wi-Fi_password";

/* Global statements required for a TCP server */
int TCP_port = 1234;
WiFiServer server(TCP_port);
String data;

/* Global statements required for T   sensor and LED */
int ledPin = 5;
int DS18S20_Pin = 15;
OneWire ds(DS18S20_Pin);
int rec=0;
float temperature;

void setup() {
  pinMode(ledPin, OUTPUT);   //Set the pin as output

/* Set the convertion resolution on DS18B20 */
  ds.reset();
  ds.write(0xcc);
  ds.write(0x4E);          // write on scratchPad
  ds.write(0x00);          // User byte 0 - Unused
  ds.write(0x00);          // User byte 1 - Unused
  ds.write(0x1F);          // setup to 9 bits resolution
```

```

 /* Start the wi-fi connection */
 WiFi.begin(ssid, password);

 while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
 }

 /* Start the TCP server */
 server.begin();
 delay(100);
}

void loop() {
 WiFiClient client = server.available(); //TCP server waiting for a client
 if (client.connected()) {   //TCP client connected
   while (client.connected()) {
    data = client.readStringUntil('\r');
    if (data == "led_on"){
     digitalWrite(ledPin, HIGH);     //LED turn on
     temperature = getTemp();  //Get the temperature
     delay(20);
     client.print(temperature);  //Send the Temperature to TCP client
    }
    else if (data == "led_off"){
     digitalWrite(ledPin, LOW);      //LED turn off
    }
   }
   delay(100);
 }
}

```

```
/*Function to get the temperature DS18S20 in DEG Celsius*/
float getTemp(){
  byte data[12];
  byte addr[8];
  float tempRead;
  float TemperatureSum;
  byte MSB;
  byte LSB;
  byte SignBit;

  ds.reset();
  ds.write(0xcc); //Skyp the ROM address
  delay(5);
  ds.write(0x44); //start conversion, without parasite
     power
  delay(5);

  byte present = ds.reset();
  ds.write(0xcc); //Skyp the ROM address
  delay(5);
  ds.write(0xBE); // Read Scratchpad

  for (int i = 0; i < 9; i++) { // we need 9 bytes
    data[i] = ds.read();
  }

  ds.reset_search();

  MSB = data[1];
  LSB = data[0];

  /* Handling the data in function of the sign -/+ */
  SignBit = (MSB & 0xf8);  //Masking to expose the sign

```

```
    if(SignBit == 0xf8){ // Check if the sign is -
      MSB = (~(MSB));
      LSB = (~(LSB));
      tempRead = ((MSB << 8) | LSB); //using two's compliment
      TemperatureSum = (tempRead / (16.0))*(-1);
    }
    else{  // if the sign is +
      tempRead = ((MSB << 8) | LSB); //using two's compliment
      TemperatureSum = tempRead / (16.0);
    }
  rec=0;
  return TemperatureSum;
}
```

3.2.2 BeagleBone Black

The BeagleBone Black has to implement a web server and execute a software to reach the ESP32 device. The web server was built using Apache 2, in this document will be not reference to this topic and only the main routine (index.html) will be shown. In internet there is a lot of literature and tutorials related to this subject.

A Python script was used to reach the ESP32 device. The python script must then be made executable, because it runs over a Linux O.S. Finally the permissions must be changed to allow the access from an other module. In this document there is no explicit code to fulfil this requirements but there is plenty of information about this topic on the web.

Once the web server is implemented and the Python script is ready a third part to join them is needed. To execute this task a *Common Gateway Interface (CGI)* module was installed and used. The Python script was located inside the folder `/usr/lib/cgibin/`.

The "Figure 3" shows a flow diagram to explain the logic behind the code used and how interact the `"index.html"` file and `"TCP_Cli_Temp.py"` software.

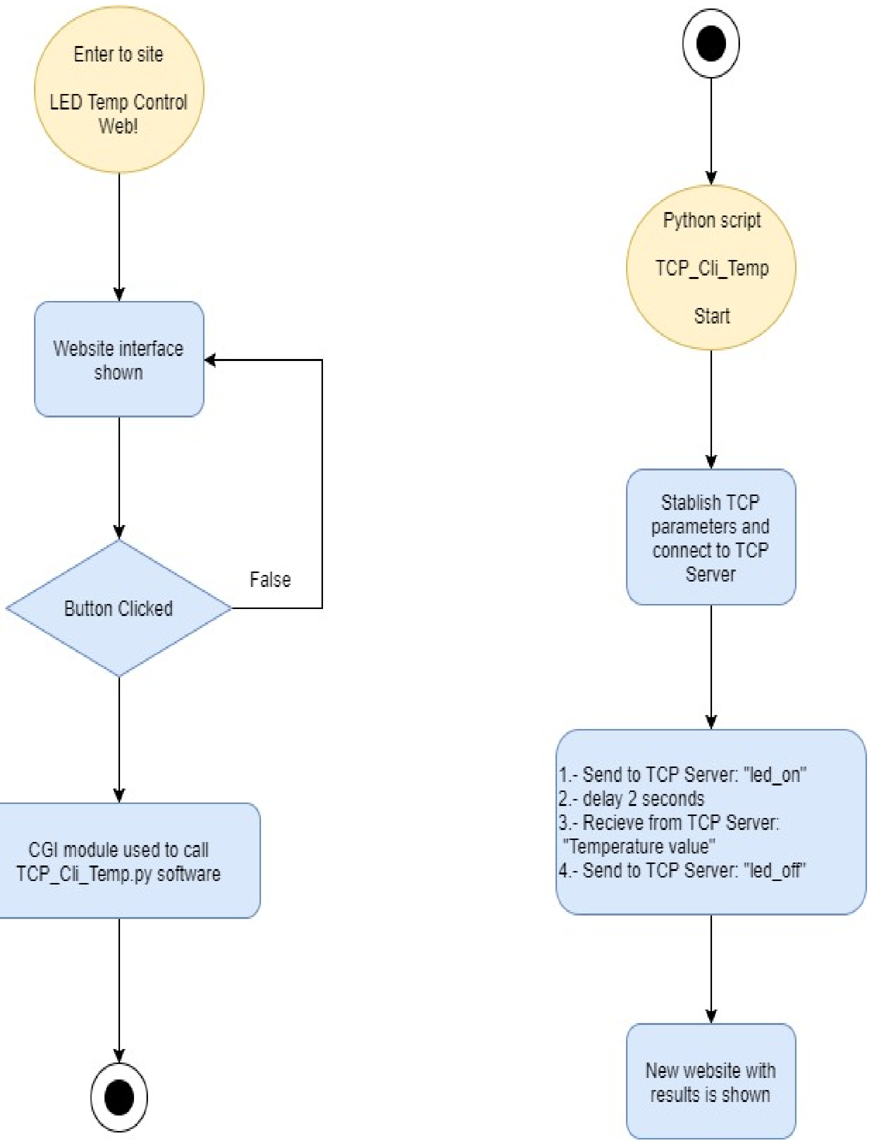

Enter to site
LED Temp Control Web!
Website interface shown
Button Clicked
False
CGI module used to call TCP_Cli_Temp.py software
Python script
TCP_Cli_Temp
Start
Stablish TCP parameters and connect to TCP Server
1.- Send to TCP Server: "led_on"
2.- delay 2 seconds
3.- Recieve from TCP Server: "Temperature value"
4.- Send to TCP Server: "led_off"
New website with results is shown

Figure 3: Flow diagram, `"BBB_Web_LED_Temp"` system.

Now the "TCP_Cli_Temp.py" source code is given.

```python
#!/usr/bin/python3

import socket
import time
import cgitb
cgitb.enable()

client=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
ip="192.168.2.112"
port=1234
address=(ip,port)
client.connect(address)
data = 'led_on'
client.sendall(data.encode('utf-8'))
time.sleep(2)
temp = client.recv(1024)
data = 'led_off'
client.sendall(data.encode('utf-8'))
time.sleep(0.02)
client.close()

print("Content-type: text/html\n\n")
print("<html>\n<body>")
print("<div style=\"with:100%; font-size: 40px; font-
    weight bold; text-align: center:\">")
print("The LED was turned On and then turned Off again.")
print('<br />')
print("Temperature: ",temp.decode('utf-8'))
print("Celsius grads")
print("</div>\n</body>\n</html>")
```

Now the "`index.html`" source code (php language) is given.

```
<!DOCTYPE html>
<html>
<head>
<title>Web Server MReyesP!</title>
<style>
.button {
  padding: 15px 25px;
  font-size: 24px;
  text-align: center;
  cursor: pointer;
  outline: none;
  color: #fff;
  background-color: #4CAF50;
  border: none;
  border-radius: 15px;
  box-shadow: 0 9px #999;
}

.button:hover {background-color: #3e8e41}

.button:active {
  background-color: #3e8e41;
  box-shadow: 0 5px #666;
  transform: translateY(4px);
}
</style>
</head>
<body>

<h1>Success! The Web Server is working :)</h1>
<h2>Click the Button to run - "LED toggle and Temperature
```

```
    measure software"</h2>

<input class=button onClick="location.href='http://ftpubu.
    ddns.net/cgi-bin/TCP_Cli_Temp.py'" value='click here'>

</body>
</html>
```

3.2.3 Modem / Router

The web server require internet access, so a port in the modem/router settings must be opened. Literally that is all, but some modem/router allows a Dynamic DNS configuration. If that is the case, the use of this feature is recommendable. The best way to reach the web server or any webpage is using a DNS, instead their physical addresses.

There is a lot of DDNS providers, some of them for free. Besides they offer very flexible options in case that the used modem/router do not has the DDNS configuration feature.

4 Testing the whole system.

In order to test the system and show how it works in a real environment, a video was recorded. The video ”Demo_ESP32_LED_Temp_video” shows the web server home page and how the ESP32 module (ESP32, LED and DS18B20 sensor) response.

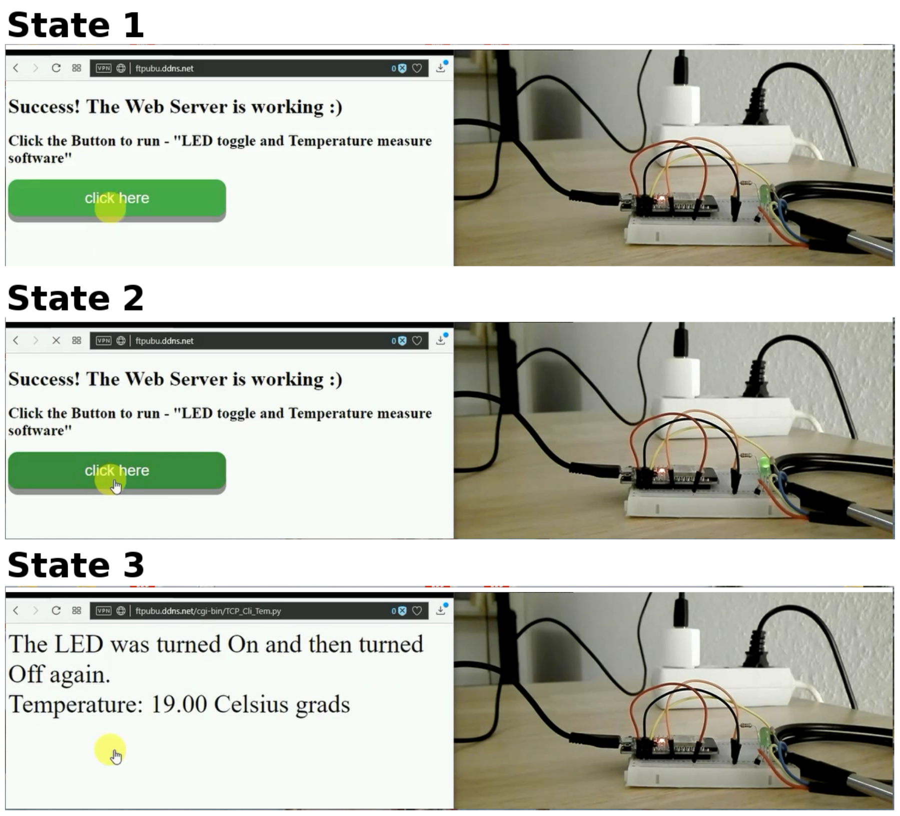
The video shows also that the ESP32 module requires nothing but energy to start working, so after all the configurations are done, his behaviour is just like a plug and play device.

The BeagleBone Black is not shown because there is not a visual output besides the web server home page.

The ”Figure 4” shows visually the three system states that ”the user” can perceive, while the ”Table 3” shows a small explanation of them.

Table 3: System states (user perspective).

State 1:	Web server home page is shown.
State 2:	After the home page button was clicked, the LED is On.
State 3:	The temperature and other messages are shown in a new webpage.



Success! The Web Server is working :)

Click the Button to run - "LED toggle and Temperature measure software"

Success! The Web Server is working :)

Click the Button to run - "LED toggle and Temperature measure software"

The LED was turned On and then turned Off again.
Temperature: 19.00 Celsius grads

Figure 4: System states (user perspective).