Final Project Report IoT Application

Embedded Systems

Ahmed Afify

900151894

Spring 2020

Outline

1. Introduction

2. Hardware Required

3. System Architecture

4. System Overview and description

5. Implementation

6. Results

Introduction

In this project, a simple IoT application that allows users to communicate with a microcontroller to retrieve date/time and control a led status through a web interface will be implemented. The web server will be hosted on a Nodemcu that communicates with both the user and the microcontroller.

Hardware Required:

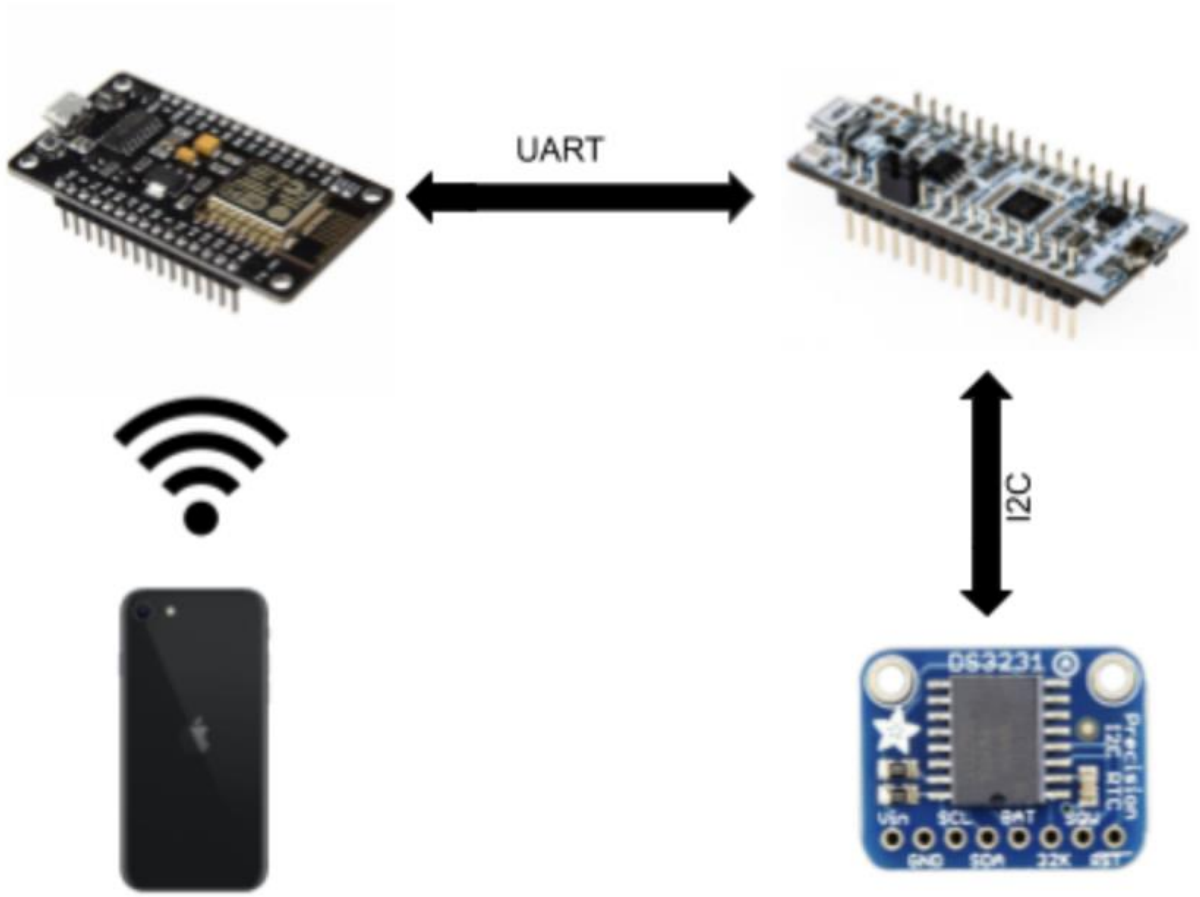
- Nodemcu

- Nucleo-32 STM32L432

- I2C RTC.

- Wifi compatible device (Laptop , Smart Phone..etc).

System Architecture



System Overview and Description

As the above diagram shows, the wifi device, in this case the Smart Phone, communicates with the Nodemcu (ESP8266) using a web interface that the Nodemcu hosts. The Nodemcu connects to the same network as that of the smartphone and acts as a server with a specific IP address. When the client (smartphone) issues a command via a URL the Nodemcu will receive the command and forward it to the Nucleo. Depending on the command, whether it's a Led control or date/time retrieval, the Nucleo will either turn a led on or off or retrieve the date/time from the RTC and send it back to the Nodemcu. The Nodemcu will then reply to the client with the retrieved date and time.

Implementation

Following a modular approach, the following listed milestones describe how the above set of requirements were implemented incrementally.

- Develop a server on the Nodemcu
    - Develop the API commands to
        - Control LED status on Nodemcu
        - Retrieve dummy date/time data
- Utilize UART on Nodemcu
    - Integrate API commands to transmit and receive data to/from TeraTerm
- Nucleo-32 & RTC integration
    - Retrieve date/time from RTC and output to TeraTerm
    - Receive commands from TeraTerm to retrieve date time and control Led
- Integrate Subsystems (Laptop <--> NodeMcu <--> Nucleo <--> RTC)

Develop a server on the Nodemcu

In order to allow the Nodemcu to act as a server, it must first be connected to the same network as that of the client.

```
// Create AsyncWebServer object on port 80
ESP8266WebServer server(80);
const char* ssid = "ID"
const char* password = "Pass";
//connect to your local wi-fi network
  WiFi.mode(WIFI_STA); // add if network is a mobile hotspot
  WiFi.begin(ssid, password);
//check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");  Serial.println(WiFi.localIP());
```

The above excerpt initializes the ESP8266Server object, connects to the local network and prints the IP given to Nodemcu.

The following step was defining and implementing the APIs. All APIs used where HTTP get requests that return text of type HTML. The following APIs were implemented.

- IP/
  - This loads the GUI or the starter page
- IP/on
  - This API is called upon the user clicking the “Led On” button to turn on the led
- IP/OFF
  - This API is called upon the user clicking the “Led Off” button to turn on the led
- IP/time
  - This API is called upon the user clicking the “Get time” button

Before integrating the Nodemcu with the Nucleo. All APIs were tested first to ensure no errors on the Nodemcu’s end. The on & off APIs would turn on the Nodemcu’s led and the time API would retrieve a constant dummy timestamp.

```
server.on("/on",turnon);
server.on("/off",turnoff);
server.on("/time",getTime);
server.on("/",turnoff);
```

The above excerpt creates the APIs and sets the handler function for each API. At this stage, the handler functions merely returned a string in case of time request or turned a led on or off, for example.

```
void turnon(){
  pinMode(2,HIGH);
  state=onn;
  server.send(200, "text/html", SendHTML(true));
  Serial.print("Led On\n");
}
```

The pin “2” corresponds to the Nodemcu’s led and the SendHTML function is the one that returns the HTML text. It will be shown in the GUI section to follow. The variable ‘state’ is also used in determining the HTML text.

After testing all the mentioned APIs and making sure everything was working properly. The next step was to Integrate API commands to transmit and receive data to/from TeraTerm

Utilize UART on Nodemcu

In order to use the UART, the library 'Software Serial' was used.

```
#include <SoftwareSerial.h>
SoftwareSerial s(D6,D5);//(Rx,Tx)
```

The above excerpt initializes our UART object with pin D6 as the Rx & D5 as the Tx.

Before actually utilizing the UART, the protocol first had to be determined. In other words, what should the Nodemcu transmit upon receiving different commands? Since there were only 3 APIs, the protocol I chose was as simple as it could be in order to minimize any errors. Upon receiving any of the three requests, the Nodemcu will transmit one of the three characters 'o','f','t'. The first would be sent in the case of the 'led on' command, the next for the 'led off' and finally the third for the 'get time' command. The only difference between the first two commands and the last is that in the case of 'get time' command, the Nodemcu must wait for a response. Hence the handler functions were now adjusted to the following.

```
void turnon(){
  s.write("o");
  state=onn;
  pinMode(2,HIGH);
 server.send(200, "text/html", SendHTML(true));
 Serial.print("Led On\n");
}
```

The turn on function only had the 's.write' function call added, which as shown, sends the character 'o' which will later be decoded by the Nucleo as the led on command. The turnoff function is a mirror of the above one except with 'f' instead of the 'o' and 'LOW' instead of 'HIGH' to turn off the local led,

```
void getTime(){
 Serial.print("Getting time\n");
 timee="";
 bool received=false;
 prnt();
  while (!received)
```

```
{
  if(s.available()>0){
  data=s.read();
  timee=timee+data;
  if(data=='\n')
  received=true;
  Serial.println(data);
  }
}


    pinMode(2,LOW);
 server.send(200, "text/html",SendHTML(state==onn));


}
```

As shown above, the ‘get time’ handler is a little different. That is due to the fact that it must wait for a response which is the time. A variable of type bool ‘received’, is initialized as false. A while loop loops on the ‘not received’ condition waiting for the full timestamp to be received. The ‘s.available()’ function checks whether there is anything stored received in the UART buffer. If there is, ‘s.read()’ would read one character from the buffer which is then concatenated with the variable ‘time’ that hold the actual timestamp to be sent back to the client. The loop goes on until a ‘\n’ is received indicating the completion of the transaction.

After testing with TeraTerm and making sure everything was working, the next step was to work on the other subsystem, Nulceo & RTC.

Nucleo-32 & RTC integration

The first milestone in building this subsystem was receiving time from the RTC and outputting it to TeraTerm, since eventually it will be sent back to the Nodemcu in the same manner. The very first step was utilizing CubeMX to set up the project. Knowing that both the UART and I2C peripherals would be utilized, I enabled both of them along with the Led. To be specific, UART2 and I2C1 were used. Consulting the datasheet of the RTC, the SDA & SCL pins of the RTC were connected to those of Nucleo’s I2C1. Before doing anything else, the RTC needed to be configured first. This was of course done with close consultation with the datasheet.

```
        // seconds
   secbuffer[0] = 0x00; //register address
   secbuffer[1] = 0x00; //data to put in register --> 0 sec
   HAL_I2C_Master_Transmit(&hi2c1, 0xD0, secbuffer, 2, 10);
```

```
// minutes
minbuffer[0] = 0x01; //register address
minbuffer[1] = 0x30; //data to put in register --> 30 min
HAL_I2C_Master_Transmit(&hi2c1, 0xD0, minbuffer, 2, 10);
// hours
hourbuffer[0] = 0x02; //register address
hourbuffer[1] = 0x41; //data to put in register 01011001 --> 11 am
HAL_I2C_Master_Transmit(&hi2c1, 0xD0, hourbuffer, 2, 10);
```

By writing to specific addresses, the seconds, minutes, and hours of the RTC were set up to be 11:30 am.

```
//send seconds register address 00h to read from
HAL_I2C_Master_Transmit(&hi2c1, 0xD0, secbuffer, 1, 10);
//read data of register 00h to secbuffer[1]
HAL_I2C_Master_Receive(&hi2c1, 0xD1, secbuffer+1, 1, 10);
//prepare UART output
out[6] = hexToAscii(secbuffer[1] >> 4 );
out[7] = hexToAscii(secbuffer[1] & 0x0F );
HAL_I2C_Master_Transmit(&hi2c1, 0xD0, minbuffer, 1, 10);
HAL_I2C_Master_Receive(&hi2c1, 0xD1, minbuffer+1, 1, 10);
out[3] = hexToAscii(minbuffer[1] >> 4 );
out[4] = hexToAscii(minbuffer[1] & 0x0F );
HAL_I2C_Master_Transmit(&hi2c1, 0xD0, hourbuffer, 1, 10);
HAL_I2C_Master_Receive(&hi2c1, 0xD1, hourbuffer+1, 1, 10);
out[0] = hexToAscii(hourbuffer[1] >> 4 &0001 );
out[1] = hexToAscii(hourbuffer[1] & 0x0F);
   // transmit time to UART
HAL_UART_Transmit(&huart2,out, sizeof(out), 10);
```

The above piece of code reads the secs, mins, and hours in stores them in an array of characters that will be sent over the UART to TeraTerm. The address to read from (sec, mins or hours) is sent and then the data is sequentially sent back. The minutes or secs for example, are sent back (one byte) in hexadecimal form where each 4 bits correspond to a decimal number. For example. 01010011 would translate to 53. That is why the received data is (4 bits followed by another 4 bits) is sent to the function hexToAscii which returns the asci of the corresponding hex. For example, sending 0100 would give back char 4. The hours however do not follow the

exact same form. Only the last bit of the first of the most significant 4 bits is considered in the 12-hour mode of the RTC. For example, xxx00001 is 1, while xxx10001 is 11.

After testing, the next step was to receive a character from TeraTerm, ‘o’,’f’, or ‘t’ and either turn on the led, turn off the led, or retrieve the time from the RTC and output it to TeraTerm.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    HAL_UART_Receive_IT(&huart2, &x, 1);

    if(x=='t' || x=='T') // get time
      getTime();
    else if( x=='o' || x=='o') // turn on led
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3,GPIO_PIN_SET);
    else if(x=='f' || x=='F') // turn off led
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3,GPIO_PIN_RESET);

}
```

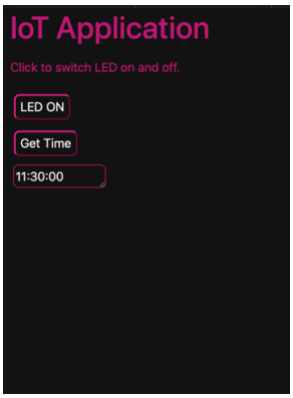
The function above shows the UART interrupt receive call back function. Upon receiving a character in the global variable ‘x’, one of the following courses of action would take place, either the led would be turned on/off or the function getTime() would be called, which is an exact replica of the code excerpt shown preceding the above one.

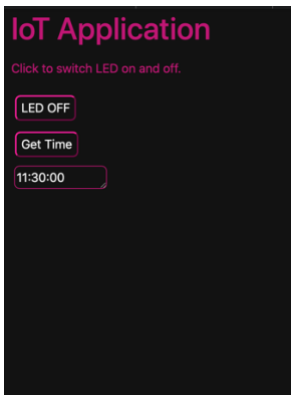
After testing with TeraTerm and making sure everything was properly it was time for the last step, Integration.

Integrate Subsystems

Due to the modular approach followed, the integration was profoundly easy and smooth. The Nodemcu’s UART pins were connected with that of the Nucleo and the system was ready.

GUI





The GUI is simply composed of two buttons and a text field. The first button controls the led and toggles between “led on” & “led off” depending on the current state of the led. The second button loads the time into the text field.

```
String SendHTML(uint8_t led){
  String ptr = "<!DOCTYPE html>\n";
  ptr +="<html>\n";
  ptr +="<head>\n";
  ptr +="<title>LED Control</title>\n";
  ptr+="<style>body{margin:0;font-family:-apple-system,BlinkMacSystemFont,\"Segoe UI\",Roboto,\"Helvetica Neue\",Arial
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr+= "<div class=\"container\">";
  ptr +="<h1>IoT Application</h1>\n";
  ptr +="<p>Click to switch LED on and off.</p>\n";
  ptr +="<form method=\"get\">\n";
  if(led)
  ptr +="<input type=\"button\" class= \"btn-primary\" value=\"LED OFF\" onclick=\"window.location.href='/off'\">\n";
  else
  ptr +="<input type=\"button\" class= \"btn-primary\" value=\"LED ON\" onclick=\"window.location.href='/on'\">\n";
  ptr +="</form>\n";
  ptr +="<form method=\"get\">\n";

  ptr +="<input type=\"button\" class= \"btn-primary\" value=\"Get Time\" onclick=\"window.location.href='/time'\">\n"

  ptr +="</form>\n";
  ptr +="<textarea rows=\"1\" cols=\"10\" readonly class= \"bg-dark shadow-none\" name=\"comment\" form=\"usrform\">";
  ptr += timee;
  ptr +="</textarea>";
  ptr+= "</div>";
  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Lastly the above function is responsible for generating the HTML string to be sent back by the Nodemcu. Since the function of the first button and its text depend on the led state, an if condition if used to determine its attributes.