Simple IoT Application

Embedded Systems

Ahmed Sharaf - 900162790
AMERICAN UNIVERSITY IN CAIRO

1. Project Background and Description

In this project, I will implement an IoT Application that controls an embedded system through a web interface. That will be achieved using two micro-controllers one will act as a webserver that users will interact with through the web interface, and the other will do the processing part of data including controlling leds, buzzers, etc. using the data from the first micro controller.

2. Project Scope

In this project, the ESP8266 module will be used as an access point and run an http web server. Users initiate some requests to that webserver through well-defined API endpoints. The webserver will propagate payload of the request to the STM32 module through a UART connection for processing. Based on the data received in the STM32 module, it will do some I/O operations on the GPOIs. Sample Operation could be controlling status of LEDs, buzzers, etc. The new system must include the following:

- Ability to connect to the webserver from any device that is connected to the same Network router as the ESP8266 Module
- Ability to get feedback responses from the ESP8266 Module.
- Ability to retrieve data from the STM32, Date and time of a connected RTC module for example, and retrieve in the web interface
- Ability to make multiple I/O operations in STM32 in a single http request to the webserver.

3. Deliverables

1. A standalone webserver hosted in ESP8266 Module that is capable of receiving requests and sends responses to the user through http responses.
2. Extension for the webserver functionality to including logging requests payload through UART connection.
3. A standalone embedded system on STM32 module that receives data from UART and make I/O operations according (utilizing RTOS for scheduling those processes).
4. Connecting the systems together.

4. Testing

- For the standalone webserver requests and responses functionality, A platform for initiating http requests will be used on a machine that is connected to the same WI-FI as the ESP8266 module and initiates requests to the local IP of the module and display the response. For that, I will use Postman tool.
- For testing the logging functionality of the webserver through a UART, I will connect it to PC and create a serial communication with a terminal on the PC, I will use TeraTerm for that.
- For testing the STM32 functionality I will connect it to PC as well with UART and TeraTerm and send data through Teraterm and observe board reaction. I will also receive data from STM32 contains some feedback messages and display it on TeraTerm,
- After insuring all components works correctly independently, I will connect the two modules, ESP8266 and STM32, together through UART connection instead of connecting them to PC and verify that everything works.

5. Implementation Plan

- I will finish the webserver part. I will use Arduino IDE to program the ESP8266 to work as a webserver and handle requests/responses. The Module will be connected to usb-to-ttl then to PC.
- After that I will test the API with Postman and verify the responses are correct.
- I will implement the logging part of the webserver through UART using TeraTerm. And veryfiy the requests payload shown on TeraTerm terminal match the one I initiated from Postman.
- When all goes well, I will implement a website to initiate the requests instead of postman and configure it to send requests to the endpoints on theserver on ESP8266. And configure it to display responses in a pretty way etc.
- I will implement the STM32 logic to do the required I/O operations based on data coming from UART and test that with TeraTerm
- Connecting the two modules together through Tx/Rx of uart instead of PC.

Implementation

Implementation has two sides, the web server and the controller. For the webserver, and ESP8266 board is used and coded using Arduino IDE to operate as a server with a given IP and handles requests coming from user. For the Controller STM32L432KCU6 is used with an RTC module.

WebServer:

1- <u>Philosophy</u>:

The webserver is an http webserver listens to specific port on specific IP to any incoming requests. The requests supported by the webserver are:

a. URL: /led/{Number}/{Status: 0,1}

   Method: POST

   Example: /led/3/1 => Turns on the LED number thrsee

b. /led/{Number}/toggle

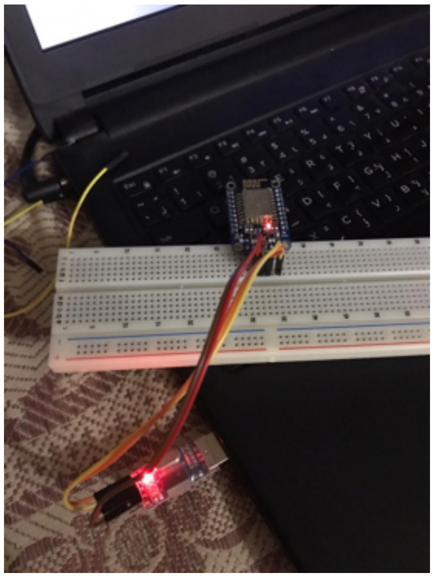
c. /time/get

   Methos: GET

d. /alarm/set

   Method: POST

2- <u>Connection</u>



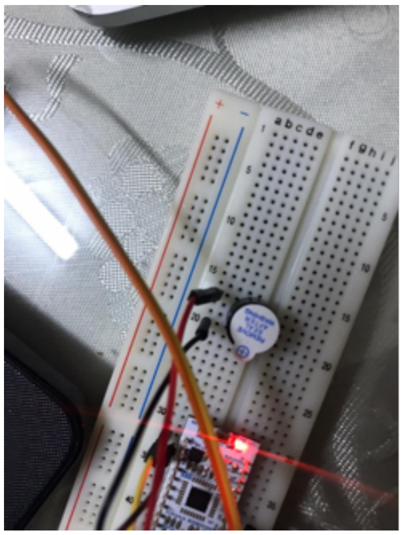
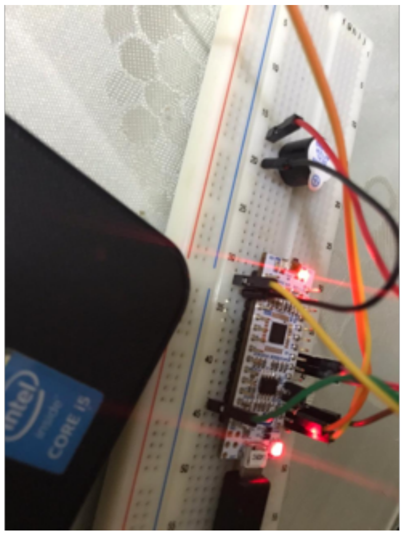
Controller (STM32L432K) Board:

1- **Philosophy:**

The STM controller contains a parser as a part of its logic to parse the incoming data from the webserver and serve it accordingly, more like a router.

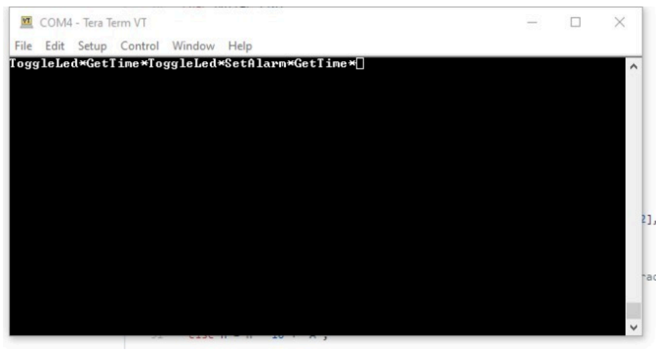
So, for example when a user make a requests to /led/3/toggle, the server sends "ToggleLed3" and the parser rule is to know that this is a toggling command for the third LED. After the parser is done parsing, a proper callback function is being called to handle the correct I/O operation and after that the callback sends the feedback through UART transmission for the server to consume it. Based on the feedback, the server will prepare a proper response with proper response code.

2- **Connection:**

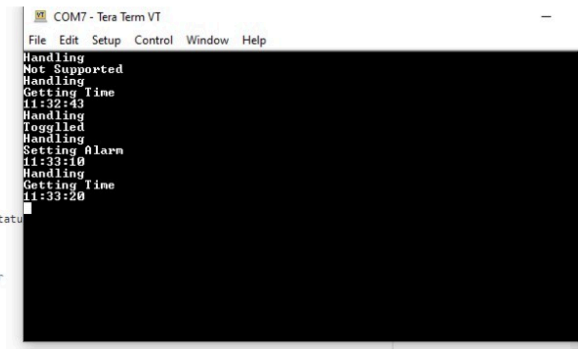
3- 


As a part of the system capabilities is to retrieve data and time and also set timers. Such functionality is supported using Real Time Clock (RTC) Module connected to the STM board. The controller communicates with RTC using I2C bus. The controller when it first get powered it sets the time in RTC and after that any incoming request asking for the time will get a time relative to the initial time the system was powered in. the time can be set also and that can be supported easily through adding a URL for it for the server and a callback function for it in the STM controller that set the time in the RTC module.

Results:



```
ToggleLed*GetTime*ToggleLed*SetAlarm*GetTime*
```



```
Handling
Not Supported
Handling
Getting Time
11:32:43
Handling
Toggled
Handling
Setting Alarm
11:33:10
Handling
Getting Time
11:33:20
```

COM4 is the data transmitted from the server to the STM Controller and COM7 is where the controller posts its feedback. All supported commands are working correctly and tested.

Adding new Command for the system can be achieved through:

1- Create A URL for that command

2- Create a message that the STM will recognize and add it to the constants section

3- Create a callback to handle it

Example: A command to provide a white light through 3 led

1- Define a function to handle that in the server: /leds/white

2- Send a message to the STM for example “whiteColor”

3- Add constant in the STM controller torecognize the new command

```
const char TOGGLE_LED[16] = "ToggleLed";
const char GET_TIME[16] = "GetTime";
const char SET_ALARM[16] = "SetAlarm";
```

So new command can be: const char WHITE_COLOR[16] = “whiteColor”

The name must match the name sent by thr server

4- Create A handler callback method to handle that

Void WhiteColor(){

//Turn on red led

//Turn on Blue led

//Turn on Green led

//Send a feedback message through uart.

```
HAL_UART_Transmit(&huart2, "White Color Done\n\r", 21,500);
```

}