<u>Embedded Systems Final Project Report</u>
<u>Simple IoT Application</u>

<u>Submitted to</u>

Dr. Mohamed Shalan

<u>Submitted by</u>

Kirollos N. Sorour 900153483

Table of Contents

Project Disclaimer: Simple IoT Application

In this project we are going to utilize the ESP8266 module to create a small IoT application that can enable the user to perform I/O operations with the STM32 module through a web interface. The I/O operations include:

- Retrieving the date/time from RTC module connected to the STM32 module
- Control the STM32 module LEDs status To build this application:
- The ESP8266 module will be programmed to act as a WiFi Access Point and run a simple HTTP server.
- The ESP8266 module communicates with the STM32 module using Asynchronous Serial link (using UART)
- The STM32 module runs code that receives commands from the ESP8266 module to control the LEDs and to retrieve the current time/date from the i2c RTC module attached to it
- Notes:
- To program the ESP8266 module, you will either use Arduino IDE or Lua/microPython scripting language. There are tons of online resources to show you how to do that.
- There are 2 types of ESP8266 modules with differences. Please check yours and get information from:
    - The Adafruit HUZZAH (https://www.adafruit.com/product/2471)
    - The Wemos mini D (https://www.wemos.cc/en/latest/)

Project Proposal

System Description

This project is the first step towards achieving smart home in a world driven by the powerful IoT technologies. In this project, I will develop a time-aware, embedded smart solution to keep some loads (LEDs) connected to the Internet. This is basically similar to making a dumb device intellegient, connected all the time, and remotely controllable. In order to do so, I would need a method to connect to the Internet; we will be using a Wi-Fi module in this project in order to do so. This Wi-Fi module will be responsible to take commands from an authorized online user and direct these commands to the microcontroller.

This embedded project contains in its core two partitions: the hardware package and the software one. For the hardware of this project, I have used limited resources for the sake of compatibility of future considerations. I have used a STM32L432KC microcontroller, a DS3231 RTC module, an Adafruit Feather HUZZAH ESP8266 Wi-Fi module, and a Usb to Serial/TTL Adapter. The first 3 components will be connected together to form the hardware of our project. The adapter is only there for uploading code to the Wi-Fi module itself. The connections will be described later on in this proposal

The software package of this project is pretty simple. It only contains source code running on the microcontroller collecting data from the RTC and the Wi-Fi module using I^2C and UART respectively, and another source code running on the Wi-Fi module specifying the HTML Server details and how to interact with clients. Although bluntly stated, the coding process is not as easy as it might seem. Therefore, I’ll be dividing the coding process into three sections which will be mentioned later.

Block Diagrams and Schematic

Actual System Schematic

This figure below shows the connections between the 3 main hardware components in the proposed system. The one to the left is the D3231 RTC module, the one in the middle is the Adafruit Feather ESP8266 Wi-Fi module, and finally the one to the right is the STM32L432KC microcontroller also known as the Nucleo board. A hidden connection is assumed here which is the power connection which is assumed to be provided by an external battery to the Nucleo board or through connecting it temporarily to the computer using a USB cable.

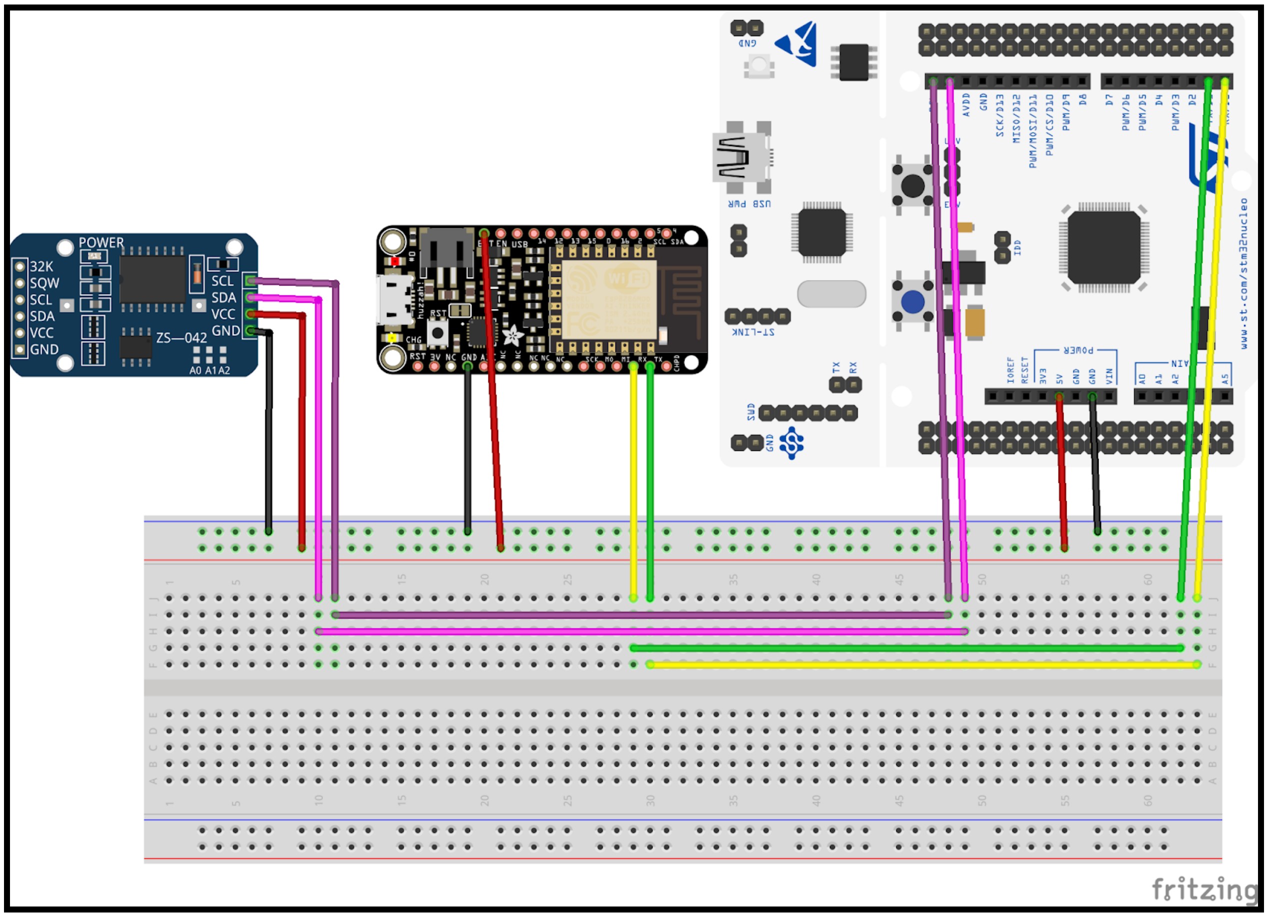


Figure 1: System Schematic; developed on Fritzing software

Conceptual System Block Diagram

Moreover, the figure below is more of a conceptual block diagram which shows the idea of the project simply. This basically says that the microcontroller will be able to do two things. The first of which is to tell the time due to the information coming from the RTC module. Second of which is to be able to control an LED through the commands incoming from an HTML server on the Wi-Fi module to the microcontroller.

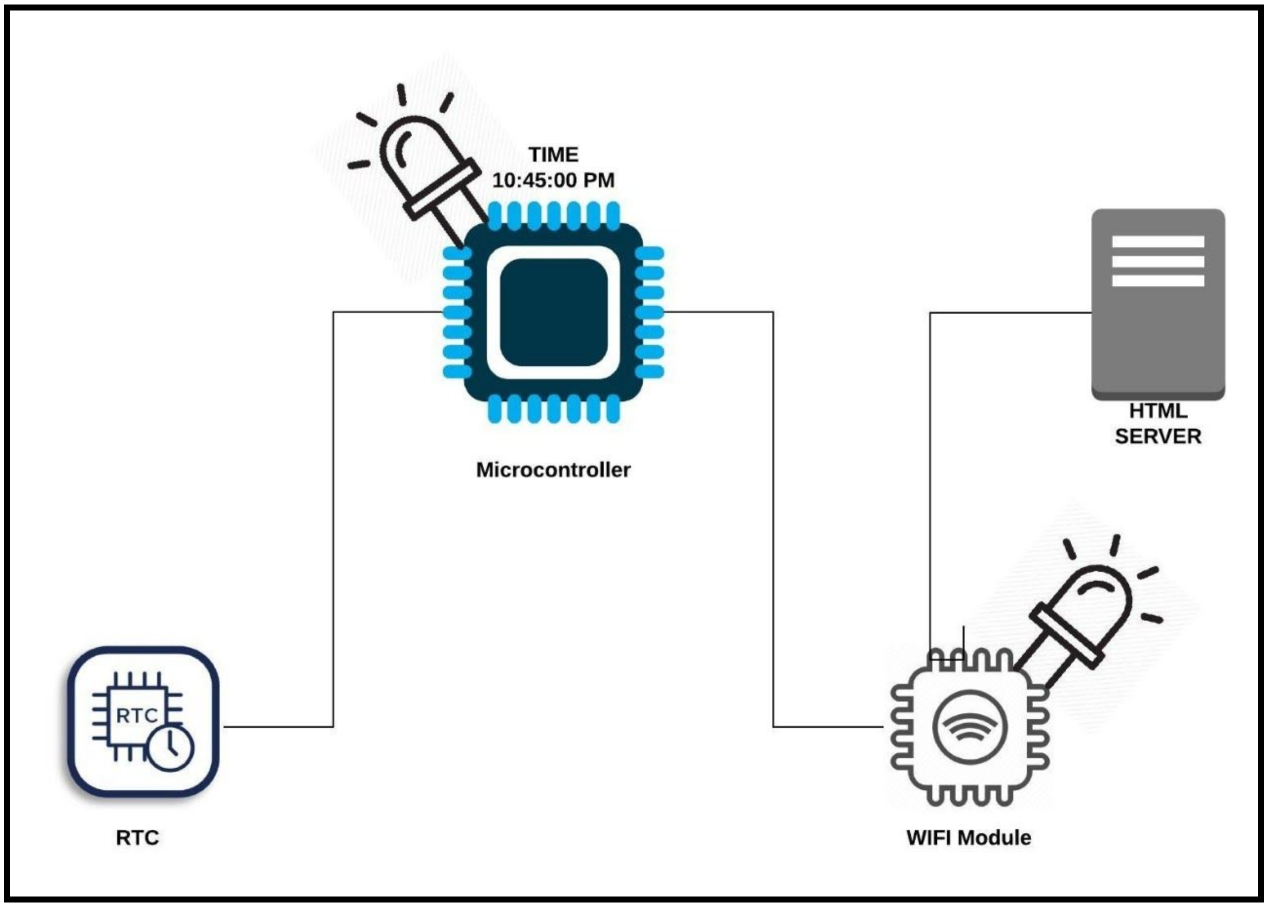


Figure 2: Conceptual System Block Diagram; developed on LucidChart Software

Programming Milestones

Milestone 1: Reading from RTC using I^2C

The first milestone in the programming phases is the simplest. We just need to do two things. First of all is to set the timer inside the RTC only once to match real time. Then next, we would read the values for the hours, minutes, and seconds from the RTC using the I^2C protocol. Also additionally we can set the date, month, year, and day of week and retrieve them later. The DS3231 RTC module is self-explanatory and makes it very simple to set any value from it by just reading its datasheet. Knowing the date and time can be very beneficial in any sort of application. One good example would be the one where a house owner needs to set an alarm remotely to wake up his/her family. Having a time-aware system is more than beneficial.

Next, I also set up a code to retrieve the temperature of the room using the temperature sensor that is embedded on the DS3231 RTC module. It has a small tolerance of + or - 3 C, which is acceptable. This is also useful since one can program the microcontroller later on to turn on the A/C if the room temperature exceeds a certain threshold. This is the true power of IoT where your house is full of smart devices communicating data to each other and analyzing the data to take actions based on it.

The source code files for this milestone, and others, are to be found on the GitHub repository I made for this project. It is well documented, and the code contains as much comments as I could to increase readability. A link for the GitHub repository is described here:

https://github.com/KirollosNagi/IoTSimpleApp

Milestone 2: Developing a basic HTML Server using the Wi-Fi module

This milestone is a bit different than what we have been doing during this course and its labs. This milestone deals with programming an Arduino-powered chip that should be programmed very differently than what we are used to. Nevertheless, it is not hard at all. As a matter of fact, it is much easier than what we normally used to do.

The real challenge was to read, learn, understand, and write HTML code on arduino to start an HTML server from the Wi-Fi module itself, and deal with clients accessing the page that was developed. It was a bit hectic but I managed to do it. However, it is still subject to improvements. It has come a long way since the very first time I investigated it almost a month ago. Moreover, I'll be showing the final HTML server side output. It will be shown in the figures below.

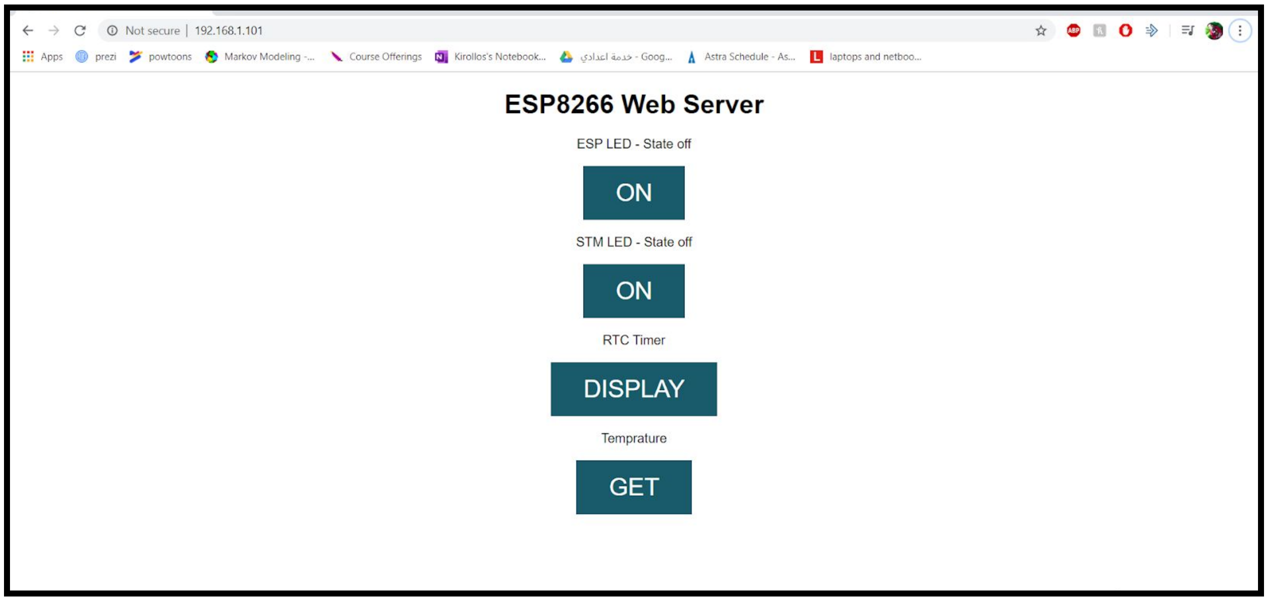


Figure 3: HTML Server for ESP8266

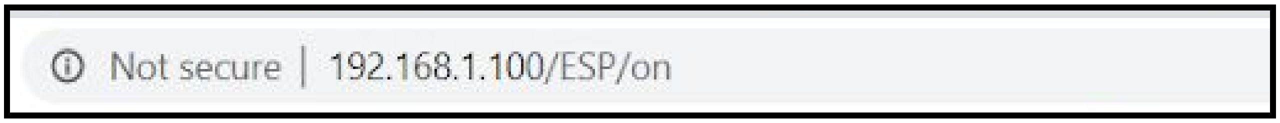


Figure 4: Effect on header upon button click

Milestone 3: Interfacing microcontroller with Wi-Fi Module using UART

A) Commands/Communications Design

In this part, I will be discussing the commands I decided to use in the communication between the STM32 microcontroller and the ESP Wi-Fi module. It is a two-way half-duplex communication where initially the the STM32 board initiates contact with the ESP Wi-Fi module by sending a command to retrieve the IP address from the ESP and display it on the UART to the screen.

After this step, Communication could be initiated from the ESP web server where the user would find four buttons to click; three of which would send a command to the STM32 Microcontroller whether to turn on/off the STM32 LED, to display the time or to retrieve the room temperature. All these commands are summarized in the following Table:

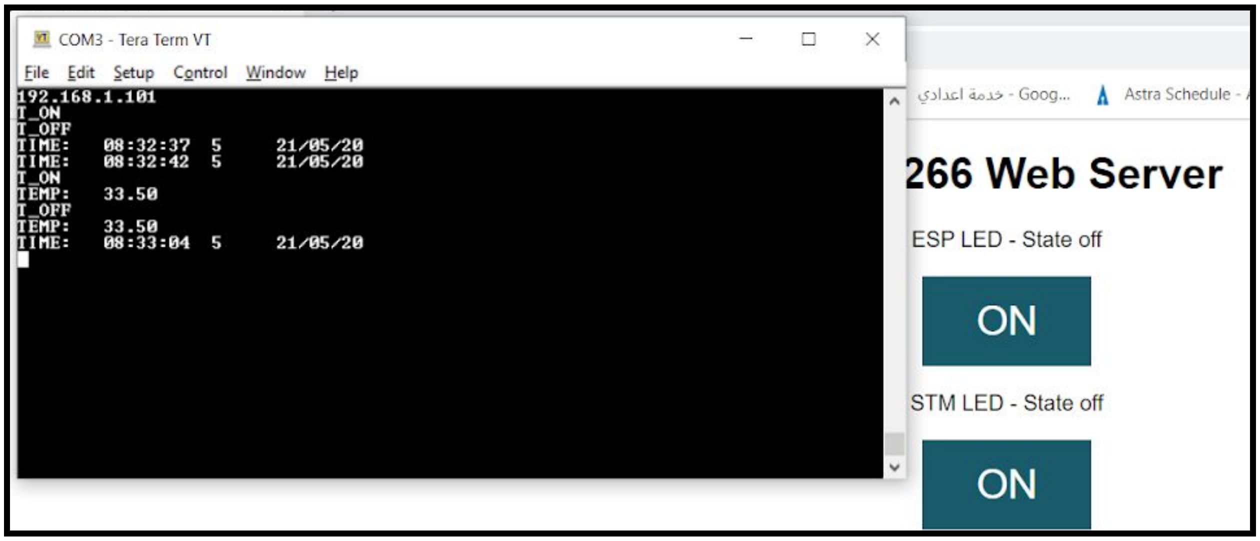
Command	Char count	Sender	Receiver	Functionality
“!”	1	STM32	ESP8266	Retrieve IP
“T_ON”	4	ESP8266	STM32	Turn LED on
“T_OF”	4	ESP8266	STM32	Turn LED off
“TIME”	4	ESP8266	STM32	Display Time
“TEMP”	4	ESP8266	STM32	Display Temperature

The main idea driving this approach of designing commands was its simplicity since all the commands received at the STM32 side were of char size of 4 which made them easier to process using static containers. Moreover, an easier approach that I took was to design each command to have a different last character. Therefore, The STM32 board can easily analyze the commands based on the last character of each one of these commands.

B) Connection Issues

This part is mainly discussing the UART connections in general between two communicating devices that don't run at the same frequency. This is mainly because the ESP runs on an 80 MHz or 160 MHz, where the STM board can operate on a maximum of 80 MHz. Therefore, making them both run at the same frequency at 80 MHz is a valid way of doing things. However, I decided to opt for dynamic power consumption and delay the transmission from the ESP to the UART to an effective delay of 10 ms between each char and the next one. This way we are saving energy on both ends.

The figure below shows the final output of the system outputted to the virtual UART. Mainly there are three functionalities which are displayed on the previous page in the HTML page.



```
192.168.1.101
T_ON
T_OFF
TIME:   08:32:37  5     21/05/20
TIME:   08:32:42  5     21/05/20
T_ON
TEMP:   33.50
T_OFF
TEMP:   33.50
TIME:   08:33:04  5     21/05/20
```

Figure 5: Final Output of the System

Lastly, the third and fourth buttons on the HTML page send a command to the STM board to display the current time and the room temperature. This is also more efficient than outputting the time each a certain amount of time. This way power consumption is being reduced to a minimum so that we can put the microcontroller to standby or sleep mode in the future. This would be amazing for Wireless Sensor Networks (WSN) applications where they all must be power efficient to prolong the lifetime of the battery used to power the device.

Future Considerations

In this section I'll present a few considerations that might be added to the system in the future to increase the learning curve and exposure to more hardware and software integration details. Following are four future considerations for this project.

1- Adding an indicator screen

Given that this project will be utilized in smart homes as a product in the future, a good idea would be to add an LCD screen indicating the status of the system and individual components of it.

2- Power calculation and choosing a sufficient battery

Since this project is going to be deployed heavily, we need to take into consideration the power consumption of the circuitry used in our system. Also, this system could be deployed with a fixed power source or using a battery. In the latter case, a good idea would be to examine the batteries in the market to provide the most reasonable lifetime for our system to be attractive for customers.

3- Using advanced AT commands with the Wi-Fi module

I have spent some time searching many datasheets till I found the one for the Wi-Fi module we use, and it seems that this Wi-Fi module is very advanced and has many capabilities. One of these is that it can read and process AT commands which would be a great learning experience and also a great advancement for the controllability of the proposed system.

4- Board PCB design and enclosure

Finally after all of the above is covered and the system is running and user friendly, I would propose to increase the system's usability by introducing a factor of compactness into the game and going over the PCB design process and provide also an enclosure casing to it to make it look like an attractive on-the-shelf product.

Conclusions

In conclusion, I would like to mention that this project has been an eye opener to me in so many different ways. This is mainly due to the fact that I got to deal with new devices such as the ESP8266 Wi-Fi Module, use new software tools like Arduino IDE to program the Wi-Fi module, and to discover new languages such as HTML which I managed to develop a good idea about in this project.

That being said, I would also like to mention that this project was an introductory project, for me as well, in the embedded systems scope when used in WSN/IoT applications. This project has so much potential to add more and more features to it as one develops a better understanding of HTML capabilities. The more one tries to investigate the useful scenarios where this project is to be deployed, one finds them to be too many. All in all, I am really thankful this project existed in the first place such that I could apply the knowledge I received in this course and its lab and to also investigate the world of IoT technologies.

I also got to learn a bit about GitHub as I added to it one of the first repositories there is in it. This is the link to my project on GitHub:

https://github.com/KirollosNagi/IoTSimpleApp