YEDİTEPE UNIVERSITY

Term Project Final Report

EE242 Microprocessor Systems

Fall 2020

IoT Colorful LED Strip Flames

January 24, 2021

Ayşe Vildan ÖLÇEK

Instructor
Gökhan ŞAHİN

TABLE OF CONTENTS

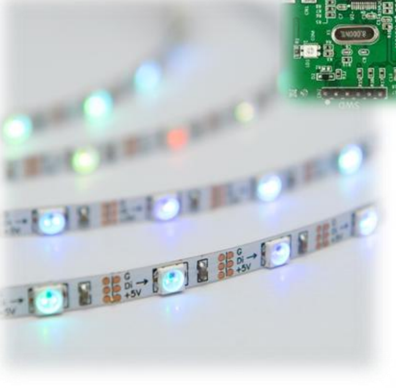
1.GOAL OF THE PROJECT

This project is an implementation of the Arduino project “Play with Fire Over Wi-Fi ” to the STM32F4 Discovery board. The original project aims to make a lamp that has a flame-effect and can be controlled by a phone application over wifi. My approach will be to create the project with the STM32f4 discovery board instead of Arduino using the same idea and same materials.

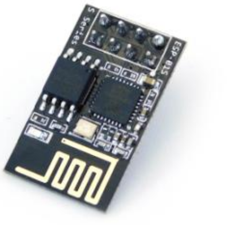
2.WHAT İS USED?

•STM32F4 Discovery board

•WS2812B RGB addressable led strip



•ESP8266 Wifi module



3.WHAT IS THE PROCEDURE?

I divide my project to 3 stage.
•First, how WS2812B works; how to connect and communicate. How to create flame effect.

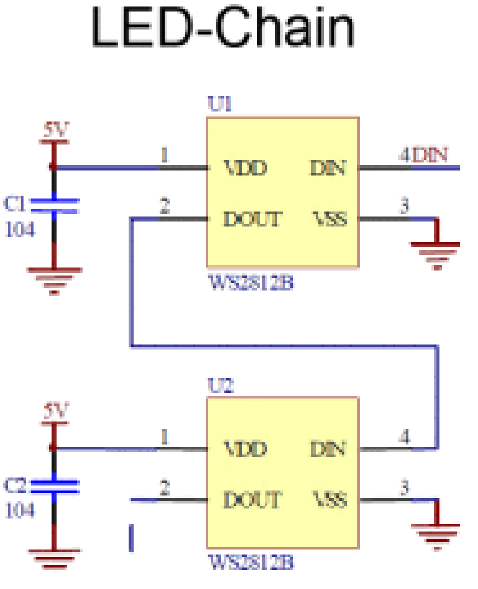
•Secondly, how ESP8266 works; how to connect and communicate.

•And lastly, how the application works; what will be the design and functions.

3.1.LEDs

3.1.1 How the WS2812B Led Strip Works

WS2812B is a led strip that can be arranged any color by changing the three main colors red, green and blue. It is also an addressable strip so any LED on the line can be changed independently.



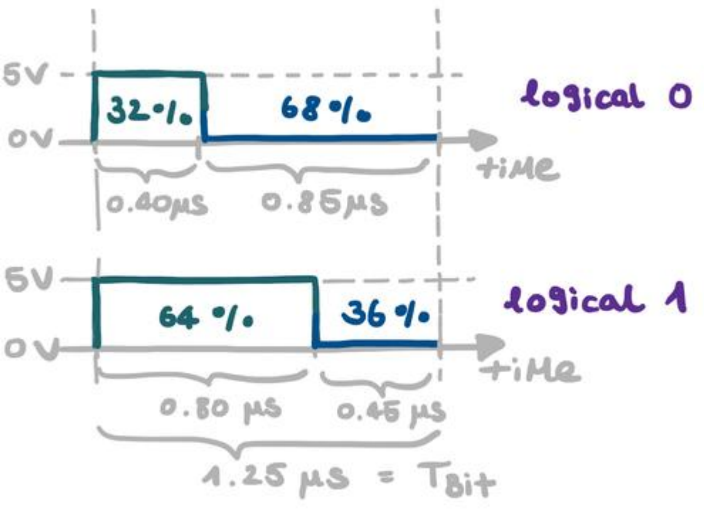
The strip has only 3 pins: power, ground and data input. A 5V power supply is required from the power pin. And of course it must be grounded. All LEDs actually have 4 pins, the additional one is data out pin. Every Dout in the line is connected to the Din serially.

Communication occurs as follows:
LEDs in the chain use the protocol that single-line serial communication. Every time when signal get into the LED, it takes the first 24 bits of information for itself and let the rest of the signal continue through the Dout to the next LED's Din. That means if I use 6 pieces of 8 LEDs long strip,
I need 24*8*6 bits of information. Every LED needs 24 bits of information because colors are made up of a mixture of red, green and blue defined by 8 bits of each color.

In each LED, one period signal is used to detect 1 bit. The cycle time is 1.25 us. To detect 1 and 0 values, it uses different duty cycles. My LEDs need 0.35 us high and 90 us low for logical 0 ; 0.90 us high and 0.35 us low for logical 1. It is also required 50 us low for reset.

That means I need 1.25 period, %28-30 duty cycle for logical 0 and %72-70 for

logical 1. That is also work if I divide my period into 3 then one third will be logical 0 and two thirds will be logical 1.



3.1.2 How the Code Shapes

Until now it was "how WS2812B works". So I learned how it works then I need to consider the follows:
I need 1.25 us (800 kHz) periodic pulses:
I should generate PWM signals. I will use one of the timers in my board. I choose Timer3.
I will use pin connected to Tim3 Channel3 This is not important that which timer or channel is used unless some of them uses different timer clocks (APB1 for timers 2,3,4,5,6,7,12,13,14 and APB2 for timers 1,8,9,10,11).
I should check the frequency of the APB1 timer clock and arrange my signals to 800 kHz:
800 kHz = 84 MHz / (Prescale*ARR (counter period))
(Prescale*ARR (counter period)) should equal to 105
I will give the numbers Prescale= 5 and ARR= 21 (should 5-1 and 21-1 because it starts from 0)

After arranging those from MX perspective I will need some codes.

•I need 0.90 us high and 0.35 us low for logical 1:
0.90 / 1.25 = 72% duty cycle, that is approximately 15 of 21. (15-1)!
and also for logical 0:
0.35/ 1.25 = 28% duty cycle, that is approximately 6 of 21. (6-1)!

•I need 24 bits (mix of 3 color, 8 bits per each) data per LEDs. That means I should send 24 pulses with correspond duty cycles to the zero or one values for each LEDs. To do that I need specific numbers of pulses also determine each with what duty cycle.

First a function to count a cycle (The timer for this function is seperately and arranged as 1.25 us period):

```
void n_cycle_wait (uint16_t n)
                                        //1.25 us counts = eactly 1 period of 1 bit
{
   __HAL_TIM_SET_COUNTER(&htim1,0);              // set the counter value 0
       while (__HAL_TIM_GET_COUNTER(&htim1) < n);       // wait for the counter to
                                              reach the n input in the parameter
}
```

Functions for one bits as logic zero or one :

```
void logic1 ()
{
     HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
     __HAL_TIM_SetCompare(&htim3,TIM_CHANNEL_2,PWM_Hi);
     n_cycle_wait(1);
     HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_2);

}
```

```
void logic0 ()
{
		HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
		__HAL_TIM_SetCompare(&htim3,TIM_CHANNEL_2,PWM_Lo);
		n_cycle_wait(1);
		HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_2);

}
```

The idea is starting the pulse generator with a spesific duty cycle (#define PWM_Hi (67) , #define PWM_Lo (33)) also counter for the wait function, after one wait(which is one cyle time)stop the pulse generator.

So this helps for generating 1 bit of "0" or "1".
Waiting 1152 cycle with high duty cyle will turn on all LEDs with white color. This could be used in a test function:

```
void test_all ()
{
		HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
		__HAL_TIM_SetCompare(&htim3,TIM_CHANNEL_2,PWM_Hi);
		n_cycle_wait(1152);
		HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_2);

}
```

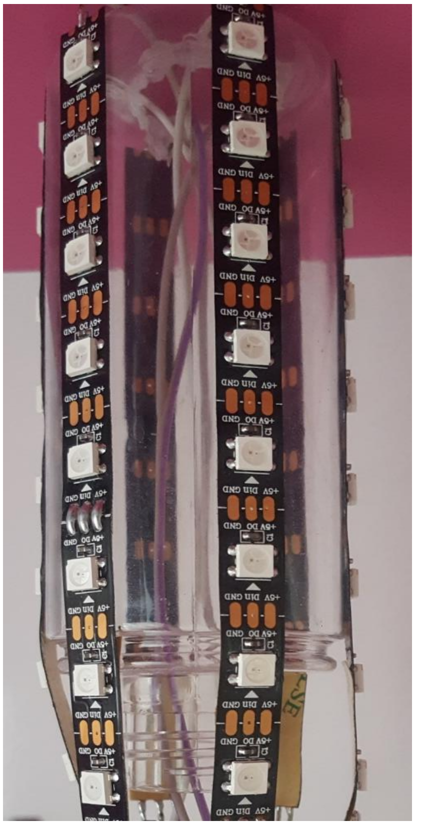
With the same idea 1152 cycle long 0s will turn of the all LEDs. So the reset function is:

```
void reset ()
{
		HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
		__HAL_TIM_SetCompare(&htim3,TIM_CHANNEL_2,PWM_Lo);
		n_cycle_wait(1152);
		HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_2);

}
```

So to light one LED this funtion handles the job after giving the necessary rgb information:

```
uint8_t r_array[8] ={0};        // I need 8 array for 8 bit of r color code
uint8_t g_array[8] ={0};                // I need 8 array for 8 bit of g color code
uint8_t b_array[8] ={0};                // I need 8 array for 8 bit of b color code


void rgb (uint8_t r_array[8],uint8_t g_array[8],uint8_t b_array[8]){
                                        //produces the necessary pulses for 1 led

        for(uint8_t i = 0; i < 8; ++i){
                if(g_array[i]==1) {
                        logic1();
                }else {
                        logic0();
                }
        }
        for(uint8_t i = 0; i < 8; ++i){

                        if(r_array[i]==1) {
                                logic1();
                        }else {
                                logic0();
                        }
                }
        for(uint8_t i = 0; i < 8; ++i){

                        if(b_array[i]==1) {
                                logic1();
                        }else {
                                logic0();
                        }
                }
}
```

Now all I need to do is, in the main function calling this function and say I want this color.
I can arrange every LEDs with different colors but nobody wants 48 different colors to choose and even want, it doesn't look fancy.
What I need is the harmony of the colors and the fire effect.

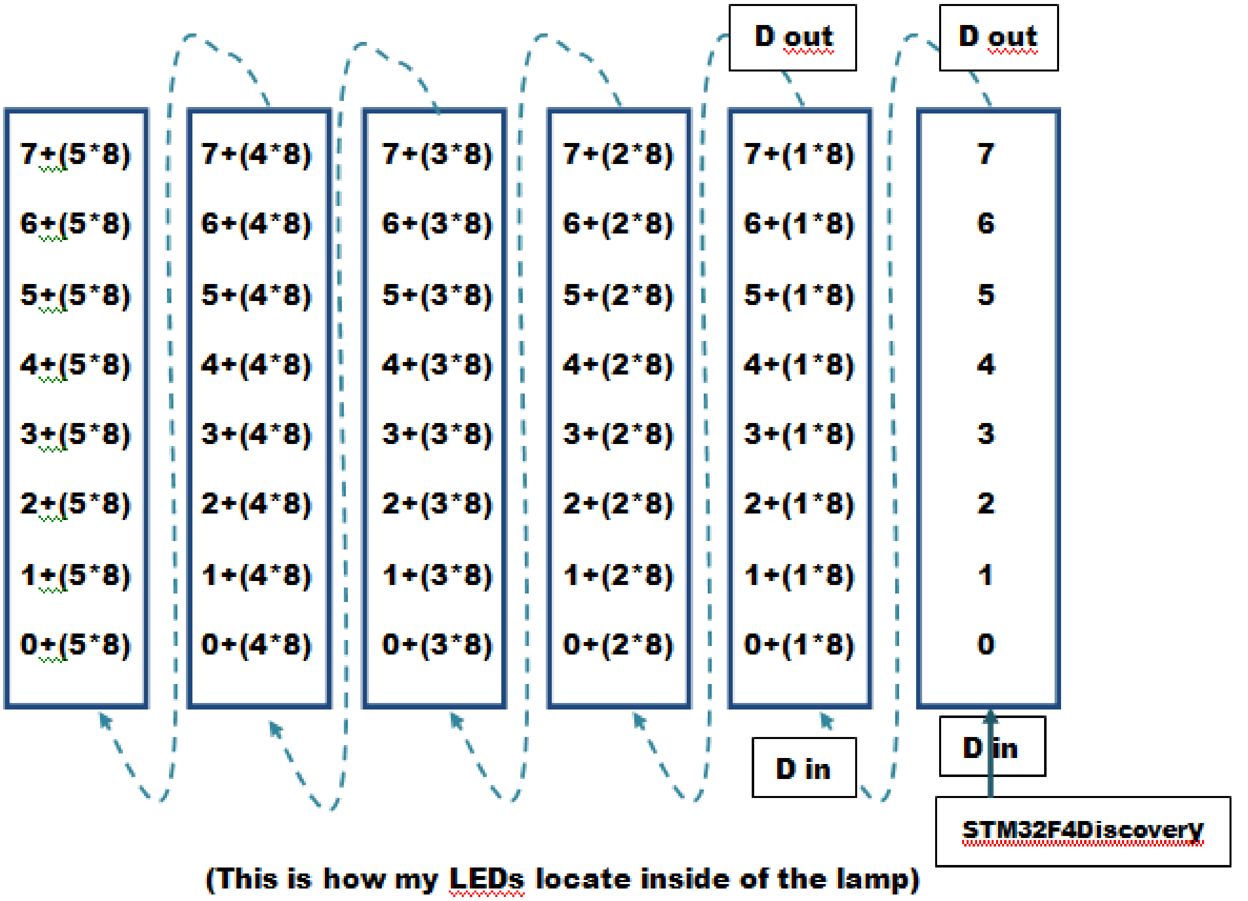


I have 8 LEDs long 6 strips. It is enough to arrange 1 column then the next ones can be arranged as same with only to add 1*8 for the second column, 2*8 for the third and so on. With this idea the necessary information would be only the color desired for each row. For better color transitions instead of 8 different color preference only 3 choices would be given. Other colors will be arranged with STM by calculating the. middle colors.

So the rows would be like this:

R	G	B	
100	50	100	→ #3 User Preference
147	58	70	((#2 - #3) / 3) + #3 seperately for each r,g,b
193	67	40	(2 * ((#2 - #3) / 3)) + #3
240	75	10	→ #2 User Preference
168	127	60	((#1 - #2) / 3) + #2
97	178	110	(2 * ((#1 - #2) / 3)) + #2
25	230	160	→ #1 User Preference
25	230	160	↗

I only need a function to arrange all LEDs in a row with same color, request 3 color information from user and finally let STM to start the party!



(This is how my LEDs locate inside of the lamp)

As it can be seen, with some function like below all rows can be arranged as same:

```
uint8_t num_of_columns = 6;
uint8_t num_of_rows = 8;
void row_color (uint8_t nth_row, uint8_t r, uint8_t g, uint8_t b)
{
for(uint8_t c=0;  c< num_of_columns ; c++ ) rgb (nth_row +(c*num_of_rows)),r,g,b);

}
```

All 48 LEDs lighted up with just 3*24 bits. What left is adding some flames.

3.1.3 Another Code with DMA

There is also another ways to do that. Source code using DMA is follow:

```
#include "stm32f4xx_hal.h"
#include "WS2812B.h"
extern TIM_HandleTypeDef htim3;
extern DMA_HandleTypeDef hdma_tim3_ch3;
//constants
#define PWM_Hi (15)
#define PWM_Lo (4)
// LED parameters
#define Num_BPP (3)                         // I have 3 color parameter for each led = grb
#define Num_Pixels (48)                     // I have 48 leds in the strip
#define Num_Bytes (Num_BPP * Num_Pixels)    // I need 3*48=144 times 8bit arrays for every led's every parameter.

// LED color buffer                         // For WS2812B the RGB order is actually "GRB" !
uint8_t rgb_array[Num_Bytes] ={0};          /** rgb_array[0] will be 0th LED's 8 bit g, rgb_array[1]
                                            /* will be 0th LED's 8 bit r, rgb_array[2] will be 0th LED's
                                            /* 8 bit b, rgb_array[3] will be 1st LED's 8 bit g and so on.
                                                                                    **/

//LED write buffer
#define Write_Buf_Len (Num_BPP * 8 * 2)
uint8_t wr_buf[Write_Buf_Len] ={0};
uint_fast8_t wr_buf_p = 0;                  //write buffer position tracker



void led_set_RGB(uint8_t index, uint8_t r, uint8_t g, uint8_t b) {    // index gives the which number of LED we set
 rgb_array[3 * index    ] = g;
 rgb_array[3 * index + 1] = r;              // for led 0 index is zero so the order is 0,1,2
 rgb_array[3 * index + 2] = b;              // then led 1 index is also 1 so the order is 3,4,5 now

}
```

```
void led_render() {
                                                  // If there is an ongoing transfer, it cancel to continue
if(wr_buf_p != 0 || hdma_tim3_ch3.State != HAL_DMA_STATE_READY) {
   for(uint8_t i = 0; i < Write_Buf_Len; ++i) wr_buf[i] = 0;
   wr_buf_p = 0;
   HAL_TIM_PWM_Stop_DMA(&htim3, TIM_CHANNEL_3);
   return;
 }

 for(uint_fast8_t i = 0; i < 8; ++i) {
   wr_buf [i      ] = PWM_Lo << (((rgb_array[0] << i) & 0x80) > 0);
   wr_buf [i + 8 ] = PWM_Lo << (((rgb_array[1] << i) & 0x80) > 0);
   wr_buf [i + 16] = PWM_Lo << (((rgb_array[2] << i) & 0x80) > 0);
   wr_buf [i + 24] = PWM_Lo << (((rgb_array[3] << i) & 0x80) > 0);
   wr_buf [i + 32] = PWM_Lo << (((rgb_array[4] << i) & 0x80) > 0);
   wr_buf [i + 40] = PWM_Lo << (((rgb_array[5] << i) & 0x80) > 0);



 HAL_TIM_PWM_Start_DMA(&htim3, TIM_CHANNEL_3, (uint32_t *)wr_buf, Write_Buf_Len);
 wr_buf_p = 2;                                              // Since it's ready for the next buffer
}


void HAL_TIM_PWM_PulseFinishedHalfCpltCallback(TIM_HandleTypeDef *htim) {

    if(wr_buf_p < Num_Pixels) {

           for(uint_fast8_t i = 0; i < 8; ++i) {
              wr_buf[i    ] = PWM_Lo << (((rgb_array[3 * wr_buf_p   ] << i) & 0x80) > 0);
              wr_buf[i + 8] = PWM_Lo << (((rgb_array[3 * wr_buf_p + 1] << i) & 0x80) > 0);
              wr_buf[i + 16] = PWM_Lo << (((rgb_array[3 * wr_buf_p + 2] << i) & 0x80) > 0);
             }
```

```
            wr_buf_p++;
            } else if (wr_buf_p < Num_Pixels + 2) {
                  for(uint8_t i = 0; i < Write_Buf_Len / 2; ++i) wr_buf[i] = 0;
                    wr_buf_p++;
                  }


            }

void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim) {
      if(wr_buf_p < Num_Pixels) {
            for(uint_fast8_t i = 0; i < 8; ++i) {
              wr_buf[i + 24] = PWM_Lo << (((rgb_array[3 * wr_buf_p  ] << i) & 0x80) > 0);
              wr_buf[i + 32] = PWM_Lo << (((rgb_array[3 * wr_buf_p + 1] << i) & 0x80) > 0);
              wr_buf[i + 40] = PWM_Lo << (((rgb_array[3 * wr_buf_p + 2] << i) & 0x80) > 0);
             }
            wr_buf_p++;
            } else if (wr_buf_p < Num_Pixels + 2) {
                  for(uint8_t i = Write_Buf_Len / 2; i < Write_Buf_Len; ++i) wr_buf[i] = 0;
                    ++wr_buf_p;
                  } else {
                        wr_buf_p = 0;
                         HAL_TIM_PWM_Stop_DMA(&htim3, TIM_CHANNEL_3);
                        }
                      }
```

reference: **https://www.thevfdcollective.com/blog/stm32-and-sk6812-rgbw-led**

3.1.4 Construction and the Flame Effect

All the LEDs has 4 pin to be connected. All grounds and the 5 V power pins should connect to the stm32 gnd and 5V parallely. Very first LED of the strip should connected to pin that PWM is produced. Every LEDs Dout should connected to Din of next. What I have is serially connected data line of 48 LEDs.

For the flame affect there should be random swipe ups of a very light or white color. To make it more realistic it should be completely random, more frequently light pops or even one row is completely lighter in the bottom and less bright pixels, more faster in the top.

3.2. Connecting to WiFi

3.2.1. How the ESP8266 Works

At this stage I need to connect the wifi module Esp8266 to my board to communicate over wifi.
By connecting wifi I can get the information that which color user wants to arrange the lamp. Then with this information my MCU will drive the LEDs.
Wifi and Esp use At commends to communicate. This is the language that I can control-connect the ESP and send receive datas so to control my lamp over wifi.
I should use Stm with UART communication. Two of the pins should be arrange as Tx and Rx with using UART. communication. Then what left for the MCU is sending AT commends and interprets the answers transmitted via Tx.
The challenge here is sending strings and reading received strings corretly. Anything else is just using the information about AT commends.

3.2.2. What is AT commends

At first to be connected we should use that commends respectively:

•Send "AT"
•Then should get an "OK" response
This says my module is working.

Send "AT+CWMODE=1"

That says which mode we want to use, =1 STA mode, =2 AP mode, =3 both. In AP the module acts as a Wi-Fi network, or access point (AP), allowing other devices to connect to it.

In STA the Esp can connect to an AP such as the Wi-Fi network from house. This allows any device connected to that network to communicate with the module.

The third permits the module to act as both an AP and a STA.

Also sending "AT+CWMODE? " checks the mode

I will choose STA, now it needs to connect wifi.

•Send "AT+CIFSR" , response should be the IP address if we already connected to wifi. If the response is nothing we should connect with this way:

•AT+CWJAP= "SSID","Password"
I should get an "OK" response.
I need to enable multiple connections before configuring the module as a server.

•"AT+CIPMUX=1"
0 for single, 1 for multiple connections.

After that the following transactions are about how to start a server:

•"AT+CIPSERVER=1,80"

The first number is used to indicate whether we want to close server mode (0), or open server mode (1). The second number indicates the port that the client uses to connect to a server. Port 80 is the default port for HTTP protocol.

When a web browser is opened and typed the IP address of Esp, the following response should be shown.

```
+IPD,1,379:GET / HTTP/1.1
Host: 192.168.1.40
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,es;q=0.6


OK
```

This is the HTTP request that computer sends to the server to fetch a file. It contains some information such as what file to retrieve, name of the browser and version, what operating system to being used, what language to be prefered to receive the file in, and more…

- **"AT+CIPSEND=0,5"**

It is the command to send some data and display it in our web browser's window

The "0" indicates the channel through which the data is going to be transferred; while "5" represents the number of characters that are going to be sent.

When the symbol ">" appears, this indicates that now the characters to send to the browser can be typed.

Response should be "SEND OK." This means that the data has been transmitted

successfully to the client. It is required to close the channel first in order to display the characters now.

"AT+CIPCLOSE=0"

"0" indicates the channel that is being closed.

Now I can send characters to the server as well as receive responses from it and receive information about user requests such as color and brightness over wifi. My goal is to transfer the data showing the colors and brightness to be selected in the application. The STM board will use this information that receives over wifi, to control the flame color and brightness of the lamp.

Reference:https://www.instructables.com/Getting-Started-With-the-ESP8266-ESP-01/

3.3.3. How the Code Shape

It is possible to shape an awesome code with using a nice library like in this project:

https://controllerstech.com/data-logger-using-stm32-and-esp8266/

UartRingbuffer.c and ESPDataLogger.c is very helpful to use but needed to be arranged which board is being used.

Beyond this the code is actually that simple:

```
uint8_t reset[] = "AT+RST\r\n";
uint8_t check[] = "AT\r\n";
uint8_t STAmode[] = "AT+CWMODE=1\r\n";
char data[80];
char SSID[] = “yourwifissıd";
char PASSWD[] = “yourpassword";
sprintf (data, "AT+CWJAP=\"%s\",\"%s\"\r\n", SSID, PASSWD);
uint8_t singleconnectn[] = "AT+CIPMUX=0\r\n";

HAL_UART_Transmit (&huart3, reset, sizeof (reset),HAL_MAX_DELAY);
HAL_UART_Transmit (&huart3, check, sizeof (check), HAL_MAX_DELAY);
HAL_UART_Transmit (&huart3, STAmode, sizeof
(STAmode),HAL_MAX_DELAY);
HAL_UART_Transmit (&huart3, data, sizeof (data),HAL_MAX_DELAY);
HAL_UART_Transmit (&huart3, singleconnectn, sizeof (singleconnectn),
HAL_MAX_DELAY);
```

What is done here only transmitting AT commends without checking the responces. Not a usable way but the main idea.

3.3 Phone Application

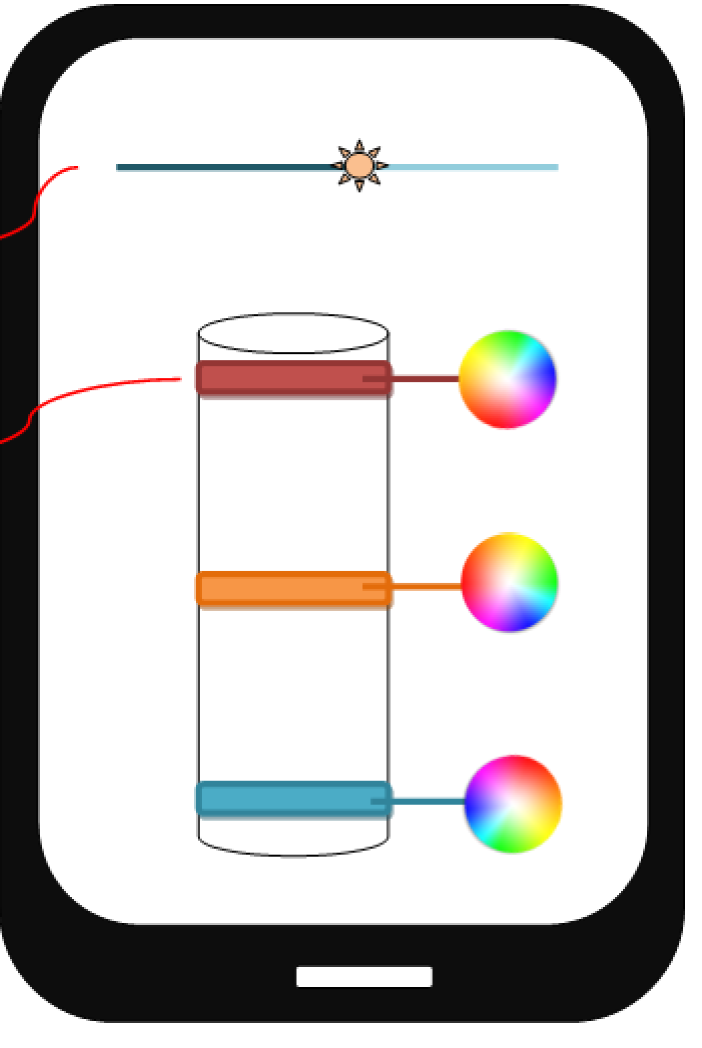
Application will be the interface between the person and the microcontroller. The functions would be allow to choose 3 different color throughout the lamp and arranging the brightness.

It is needed to arrange ESP to interrupt mode then when application sends an information it will be interrupted, receive data and transfer to the microcontroller.

There will be only 1 main page. It will look like as fallow:

Brightness Control

3 part of the colour of the lamp can be arranged according to users desires. Middle parts will be arranged as the colour between the choosen two.



4. CONCLUSION

This study intends to implement an Arduino project to an STM32.

The benefit will be the ability to implement any project with any board on any platform.

Of course, as long as the software language is known and a data sheet is available.

Specifically in this report, how each element used works, the physical and algorithmic connections between one another and basically what kind of coding should be done were discussed.

Apart from references, suggested codes are very simple and written at a low level.

All the information has been tried to be given in detail and well explained with graphics.

Many different sources have been used and care has been taken to address all of them.

5.REFERENCES

•The Arduino Project:
https://electropeak.com/learn/controllable-led-flame-fire-by-arduino-esp8266-android-app-wifi/
•WS2812B RGB addressable led strip
http://hypnocube.com/2013/12/design-and-implementation-of-serial-led-gadgets/
https://cpldcpu.wordpress.com/2014/01/14/light_ws2812-library-v2-0-part-i-understanding-the-ws2812/
https://www.thevfdcollective.com/blog/stm32-and-sk6812-rgbw-led
http://www.martinhubacek.cz/arm/improved-stm32-ws2812b-library
http://fabioangeletti.altervista.org/blog/stm32-interface-ws2812b/?doing_wp_cron=1609008050.1861259937286376953125
https://stm32f4-discovery.net/2018/06/tutorial-control-ws2812b-leds-stm32/
https://github.com/OutOfTheBots/STM32_NEOpixels_timer/blob/master/main.c
https://github.com/cpldcpu/light_ws2812/tree/master/light_ws2812_ARM
https://github.com/hubmartin/WS2812B_STM32F4
https://www.bitcraze.io/2014/04/neopixel-driver-for-the-crazyflies-stm32/
https://github.com/evilwombat/stm32f103_fastNP
•About PWM
https://www.eng.auburn.edu/~nelsovp/courses/elec3040_3050/LabLectures/ELEC30x0%20Lab7%20PWM%20Slides.pdf
https://controllerstech.com/create-1-microsecond-delay-stm32/
•About RGB colors
https://www.tutorialspoint.com/c-program-to-change-rgb-color-model-to-hsv-color-model
https://www.cs.utah.edu/~germain/PPS/Topics/color.html
•ESP8266 Wifi module
https://www.instructables.com/Getting-Started-With-the-ESP8266-ESP-01/
https://circuitdigest.com/microcontroller-projects/getting-started-with-esp8266-module
https://stm32f4-discovery.net/2016/01/esp8266-at-parser-library-beta-version-released/
https://controllerstech.com/esp8266-webserver-using-stm32-hal/
•About At Commends
http://room-15.github.io/blog/2015/03/26/esp8266-at-command-reference/
https://docs.espressif.com/projects/esp-at/en/latest/AT_Command_Set/index.html
https://docs.espressif.com/projects/esp-at/en/latest/AT_Command_Examples/TCP-IP_AT_Examples.html#exam-UWFPTT