HACETTEPE UNIVERSITY COMPUTER ENGINEERING DEPARTMENT

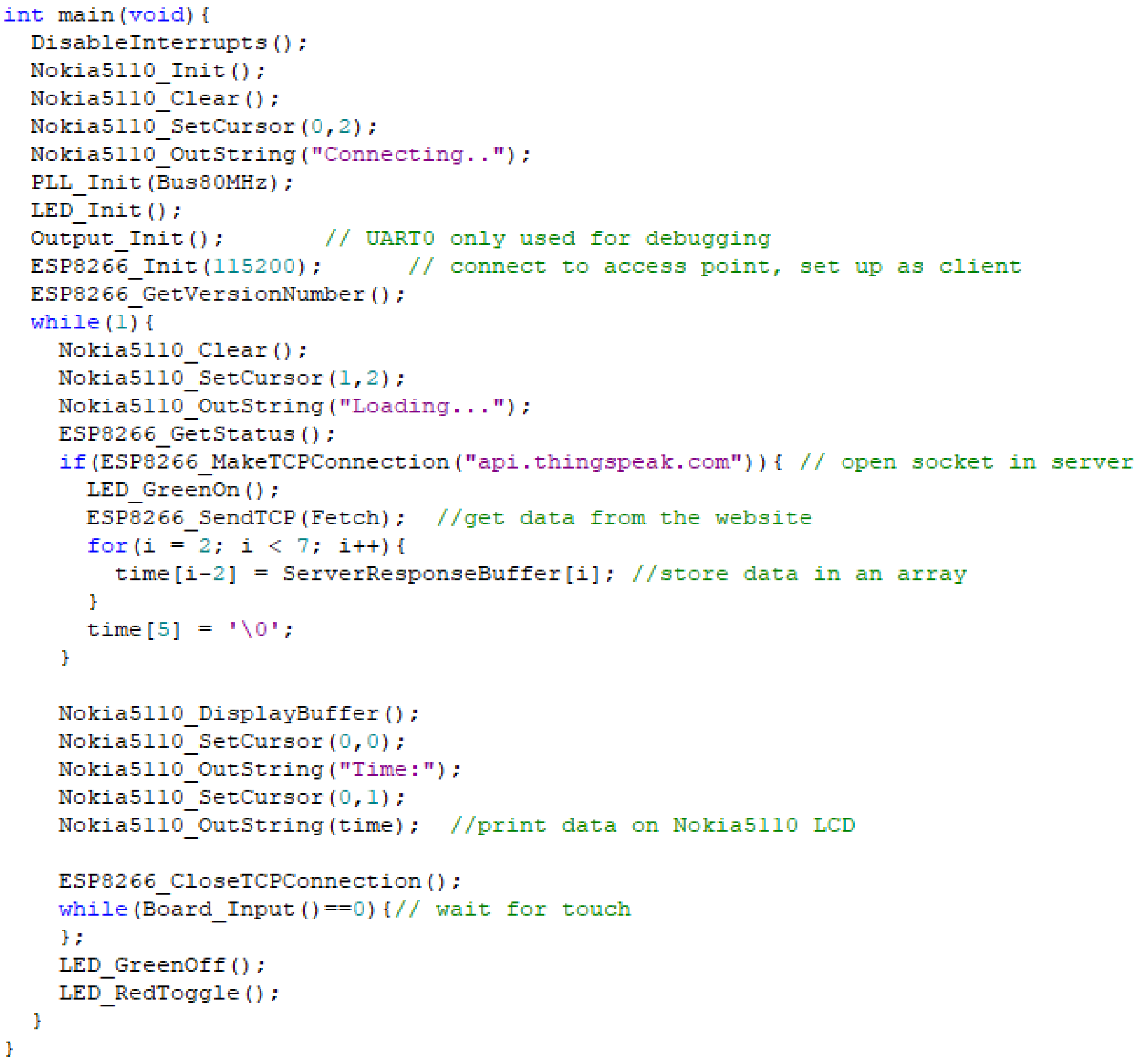
Student Information:	Merve Müge Deliktaş - 21526896
	Onur Cankur – 21526791
Course Information:	BBM 434 - Embedded Systems Laboratory
	Spring – 2019
Report Information:	Lab-08 Experiment Report

Short Brief of Lab-08 and Function Explanations

In this lab, we created a watch that can be connected to Wi-Fi using ESP8266 Wi-Fi module and Nokia5110 LCD screen.

If one of the switches on the launchpad is pressed, it gets the data again and updates result on the Nokia screen.

- **Main**

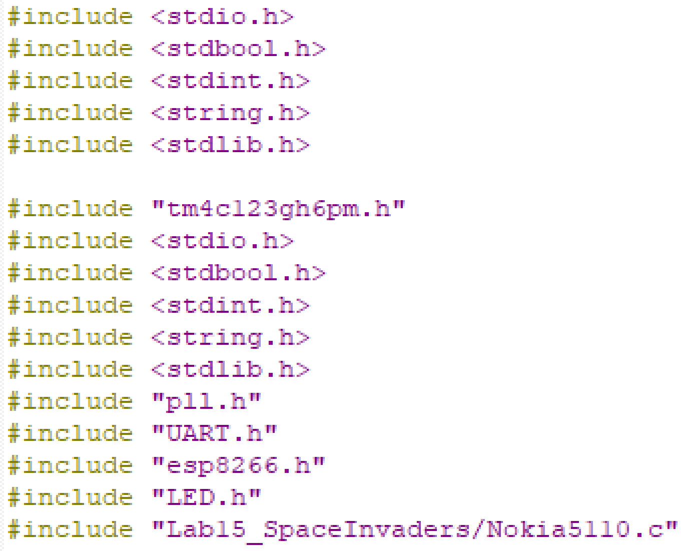


```
int main(void){
  DisableInterrupts();
  Nokia5110_Init();
  Nokia5110_Clear();
  Nokia5110_SetCursor(0,2);
  Nokia5110_OutString("Connecting..");
  PLL_Init(Bus80MHz);
  LED_Init();
  Output_Init();       // UART0 only used for debugging
  ESP8266_Init(115200);      // connect to access point, set up as client
  ESP8266_GetVersionNumber();
  while(1){
    Nokia5110_Clear();
    Nokia5110_SetCursor(1,2);
    Nokia5110_OutString("Loading...");
    ESP8266_GetStatus();
    if(ESP8266_MakeTCPConnection("api.thingspeak.com")){ // open socket in server
      LED_GreenOn();
      ESP8266_SendTCP(Fetch);  //get data from the website
      for(i = 2; i < 7; i++){
        time[i-2] = ServerResponseBuffer[i]; //store data in an array
      }
      time[5] = '\0';
    }

    Nokia5110_DisplayBuffer();
    Nokia5110_SetCursor(0,0);
    Nokia5110_OutString("Time:");
    Nokia5110_SetCursor(0,1);
    Nokia5110_OutString(time);  //print data on Nokia5110 LCD

    ESP8266_CloseTCPConnection();
    while(Board_Input()==0){// wait for touch
    };
    LED_GreenOff();
    LED_RedToggle();
  }
}
```

Figure 1-main function

You can see the main function from Figure-1. It initializes Nokia, UART, PLL, and ESP8266. Then in an infinite while loop, it makes TCP connection and sends the IP address to get the data. After getting and storing the data, writes it to the Nokia5110 LCD screen. Then, waits for the switch button to be pressed and when it pressed, it gets the new data.

- **Includes**

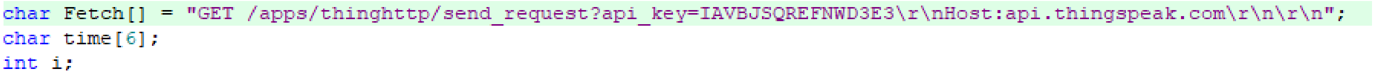


```
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>

#include "tm4c123gh6pm.h"
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include "pll.h"
#include "UART.h"
#include "esp8266.h"
#include "LED.h"
#include "Lab15_SpaceInvaders/Nokia5110.c"
```

Figure 2-includes

UART, ESP8266, LED and Nokia5510 libraries are used in this lab.

- **Variables**

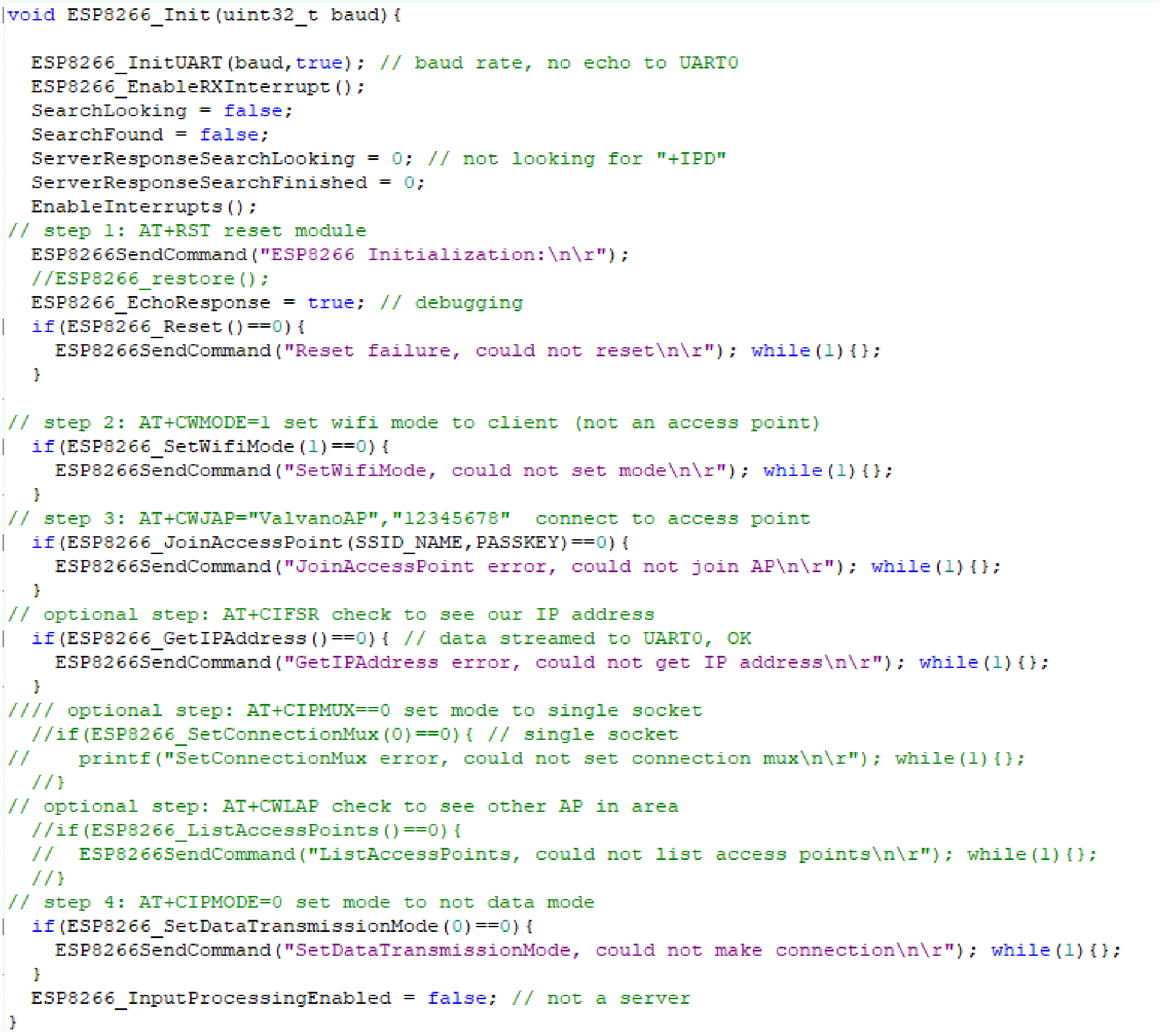


```
char Fetch[] = "GET /apps/thinghttp/send_request?api_key=IAVBJSQREFNWD3E3\r\nHost:api.thingspeak.com\r\n\r\n";
char time[6];
int i;
```

Figure 3-variables

The data that will be got from the server can be seen in above figure.

- **ESP8266 Initialization**

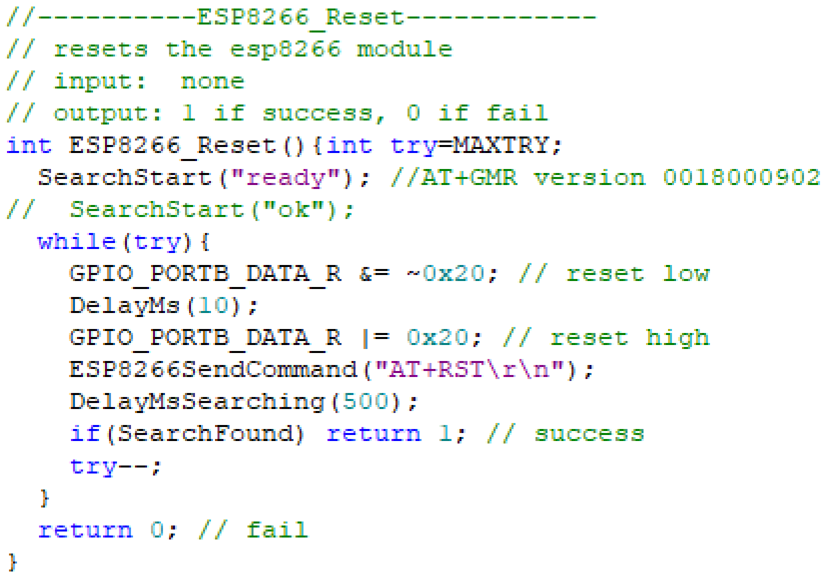


```
void ESP8266_Init(uint32_t baud){

  ESP8266_InitUART(baud,true); // baud rate, no echo to UART0
  ESP8266_EnableRXInterrupt();
  SearchLooking = false;
  SearchFound = false;
  ServerResponseSearchLooking = 0; // not looking for "+IPD"
  ServerResponseSearchFinished = 0;
  EnableInterrupts();
// step 1: AT+RST reset module
  ESP8266SendCommand("ESP8266 Initialization:\n\r");
  //ESP8266_restore();
  ESP8266_EchoResponse = true; // debugging
  if(ESP8266_Reset()==0){
    ESP8266SendCommand("Reset failure, could not reset\n\r"); while(1){};
  }

// step 2: AT+CWMODE=1 set wifi mode to client (not an access point)
  if(ESP8266_SetWifiMode(1)==0){
    ESP8266SendCommand("SetWifiMode, could not set mode\n\r"); while(1){};
  }
// step 3: AT+CWJAP="ValvanoAP","12345678"  connect to access point
  if(ESP8266_JoinAccessPoint(SSID_NAME,PASSKEY)==0){
    ESP8266SendCommand("JoinAccessPoint error, could not join AP\n\r"); while(1){};
  }
// optional step: AT+CIFSR check to see our IP address
  if(ESP8266_GetIPAddress()==0){ // data streamed to UART0, OK
    ESP8266SendCommand("GetIPAddress error, could not get IP address\n\r"); while(1){};
  }
//// optional step: AT+CIPMUX==0 set mode to single socket
  //if(ESP8266_SetConnectionMux(0)==0){ // single socket
//    printf("SetConnectionMux error, could not set connection mux\n\r"); while(1){};
  //}
// optional step: AT+CWLAP check to see other AP in area
  //if(ESP8266_ListAccessPoints()==0){
  //  ESP8266SendCommand("ListAccessPoints, could not list access points\n\r"); while(1){};
  //}
// step 4: AT+CIPMODE=0 set mode to not data mode
  if(ESP8266_SetDataTransmissionMode(0)==0){
    ESP8266SendCommand("SetDataTransmissionMode, could not make connection\n\r"); while(1){};
  }
  ESP8266_InputProcessingEnabled = false; // not a server
}
```

Figure 4-ESP8266_Init()

First, it resets the device. Then sets the Wi-Fi mode as 1 to set it as client. Then it joins the access point which is our Wi-Fi. Finally, sets the data transmission mode as 0.

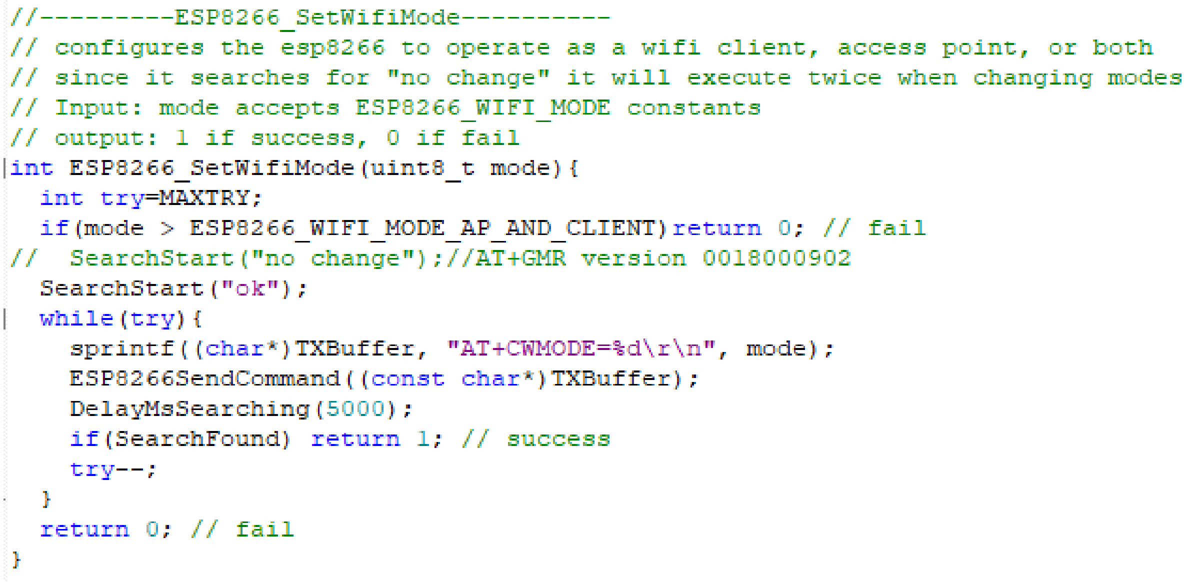
- **AT Commands**
  - **AT+RST**



```
//----------ESP8266_Reset------------
// resets the esp8266 module
// input:  none
// output: 1 if success, 0 if fail
int ESP8266_Reset(){int try=MAXTRY;
  SearchStart("ready"); //AT+GMR version 0018000902
//  SearchStart("ok");
  while(try){
    GPIO_PORTB_DATA_R &= ~0x20; // reset low
    DelayMs(10);
    GPIO_PORTB_DATA_R |= 0x20; // reset high
    ESP8266SendCommand("AT+RST\r\n");
    DelayMsSearching(500);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 5-AT+RST

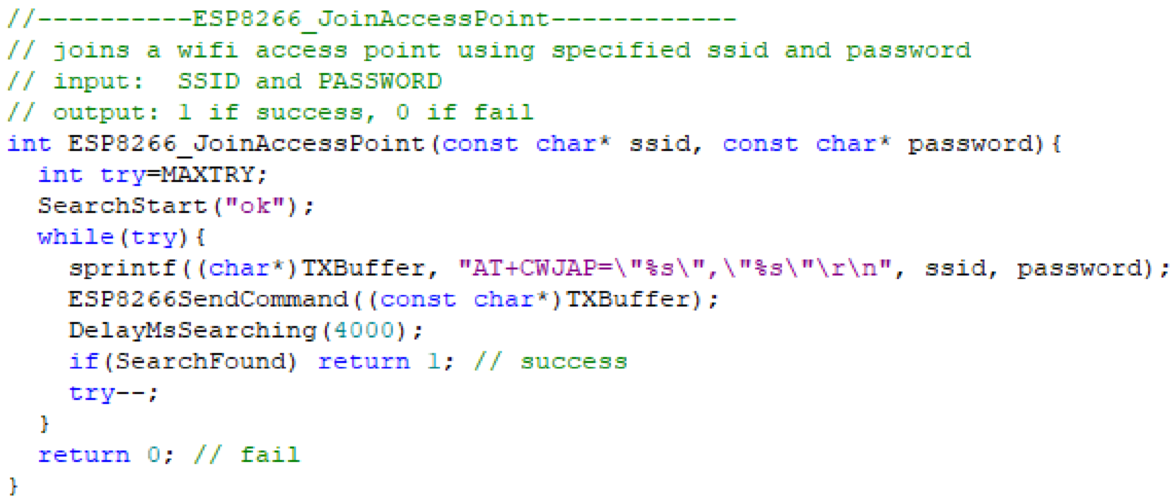
  - **AT+CWMODE**



```
//---------ESP8266_SetWifiMode----------
// configures the esp8266 to operate as a wifi client, access point, or both
// since it searches for "no change" it will execute twice when changing modes
// Input: mode accepts ESP8266_WIFI_MODE constants
// output: 1 if success, 0 if fail
int ESP8266_SetWifiMode(uint8_t mode){
  int try=MAXTRY;
  if(mode > ESP8266_WIFI_MODE_AP_AND_CLIENT)return 0; // fail
//  SearchStart("no change");//AT+GMR version 0018000902
  SearchStart("ok");
  while(try){
    sprintf((char*)TXBuffer, "AT+CWMODE=%d\r\n", mode);
    ESP8266SendCommand((const char*)TXBuffer);
    DelayMsSearching(5000);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 6-AT+CWMODE

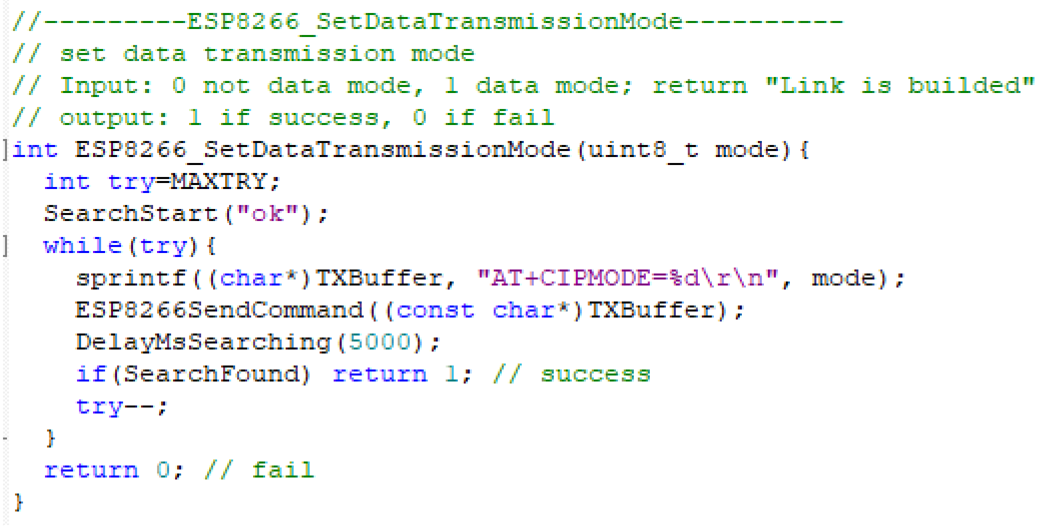
- **AT+CWJAP**



```
//----------ESP8266_JoinAccessPoint------------
// joins a wifi access point using specified ssid and password
// input:  SSID and PASSWORD
// output: 1 if success, 0 if fail
int ESP8266_JoinAccessPoint(const char* ssid, const char* password){
  int try=MAXTRY;
  SearchStart("ok");
  while(try){
    sprintf((char*)TXBuffer, "AT+CWJAP=\"%s\",\"%s\"\r\n", ssid, password);
    ESP8266SendCommand((const char*)TXBuffer);
    DelayMsSearching(4000);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 7-AT+CWJAP

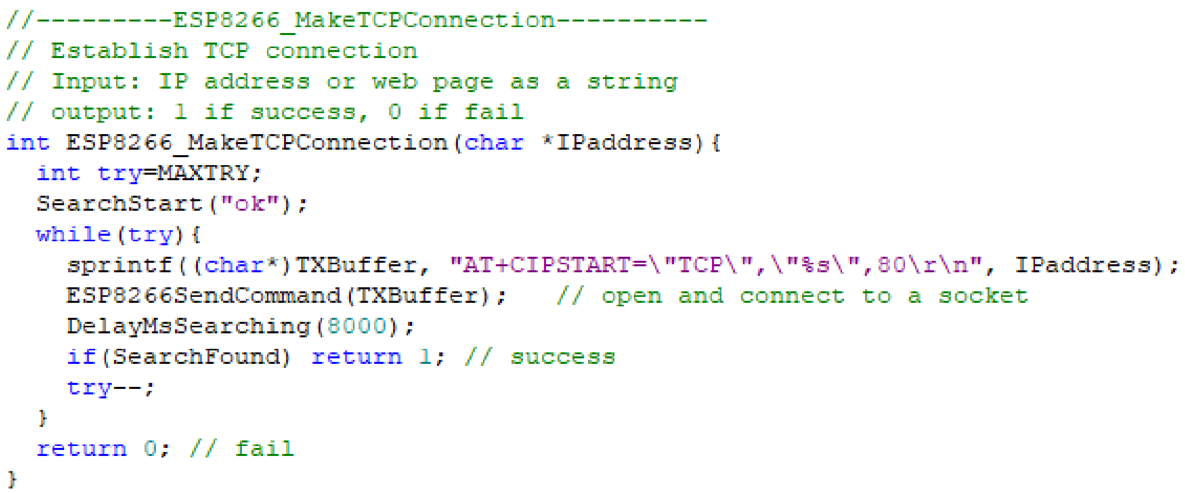
- **AT+CIPMODE**



```
//---------ESP8266_SetDataTransmissionMode----------
// set data transmission mode
// Input: 0 not data mode, 1 data mode; return "Link is builded"
// output: 1 if success, 0 if fail
int ESP8266_SetDataTransmissionMode(uint8_t mode){
  int try=MAXTRY;
  SearchStart("ok");
  while(try){
    sprintf((char*)TXBuffer, "AT+CIPMODE=%d\r\n", mode);
    ESP8266SendCommand((const char*)TXBuffer);
    DelayMsSearching(5000);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 8-AT+CIPMODE

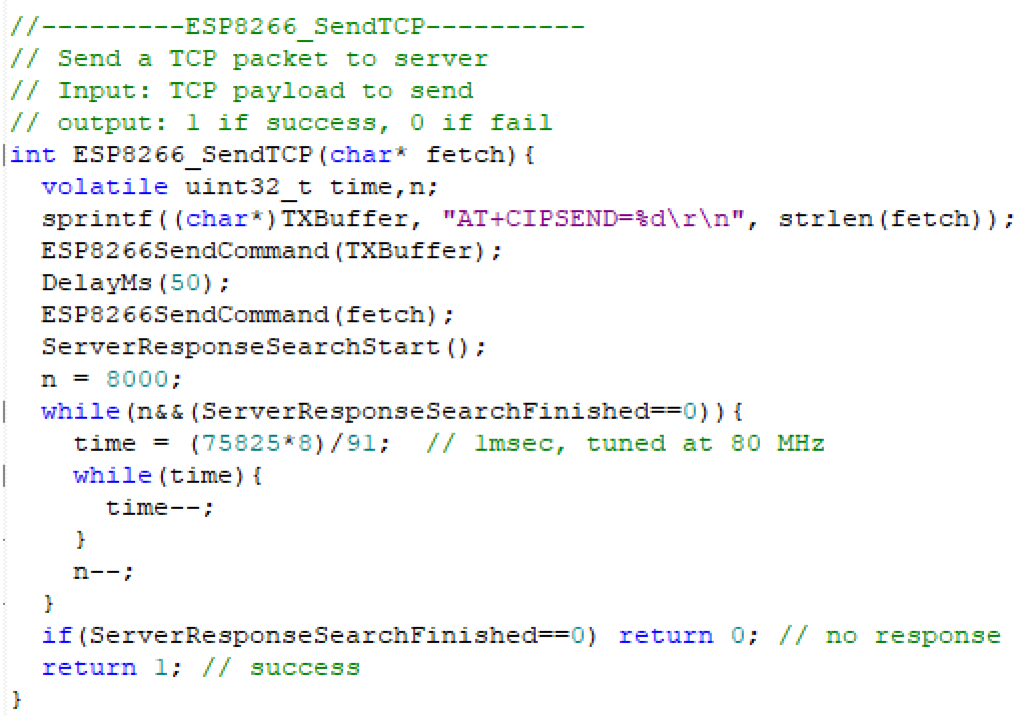
- **AT+CIPSTART**



```
//---------ESP8266_MakeTCPConnection----------
// Establish TCP connection
// Input: IP address or web page as a string
// output: 1 if success, 0 if fail
int ESP8266_MakeTCPConnection(char *IPaddress){
  int try=MAXTRY;
  SearchStart("ok");
  while(try){
    sprintf((char*)TXBuffer, "AT+CIPSTART=\"TCP\",\"%s\",80\r\n", IPaddress);
    ESP8266SendCommand(TXBuffer);    // open and connect to a socket
    DelayMsSearching(8000);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 9-AT+CIPSTART

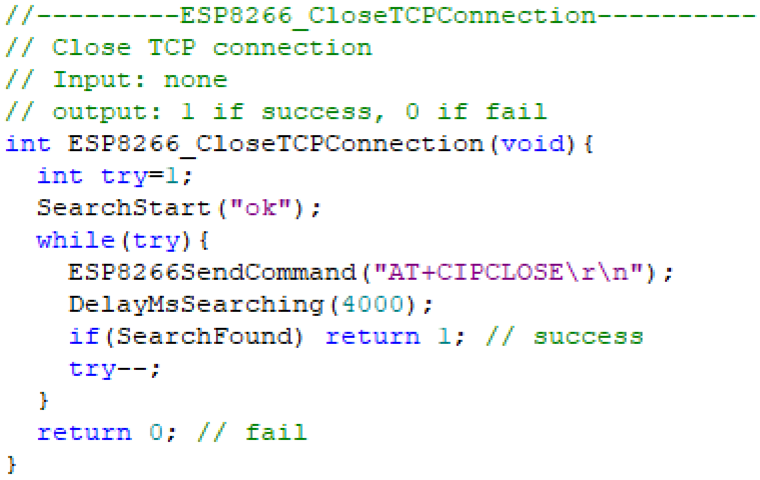
- **AT+CIPSEND**



```
//---------ESP8266_SendTCP----------
// Send a TCP packet to server
// Input: TCP payload to send
// output: 1 if success, 0 if fail
int ESP8266_SendTCP(char* fetch){
  volatile uint32_t time,n;
  sprintf((char*)TXBuffer, "AT+CIPSEND=%d\r\n", strlen(fetch));
  ESP8266SendCommand(TXBuffer);
  DelayMs(50);
  ESP8266SendCommand(fetch);
  ServerResponseSearchStart();
  n = 8000;
  while(n&&(ServerResponseSearchFinished==0)){
    time = (75825*8)/91;  // 1msec, tuned at 80 MHz
    while(time){
      time--;
    }
    n--;
  }
  if(ServerResponseSearchFinished==0) return 0; // no response
  return 1; // success
}
```

Figure 10-AT+CIPSEND

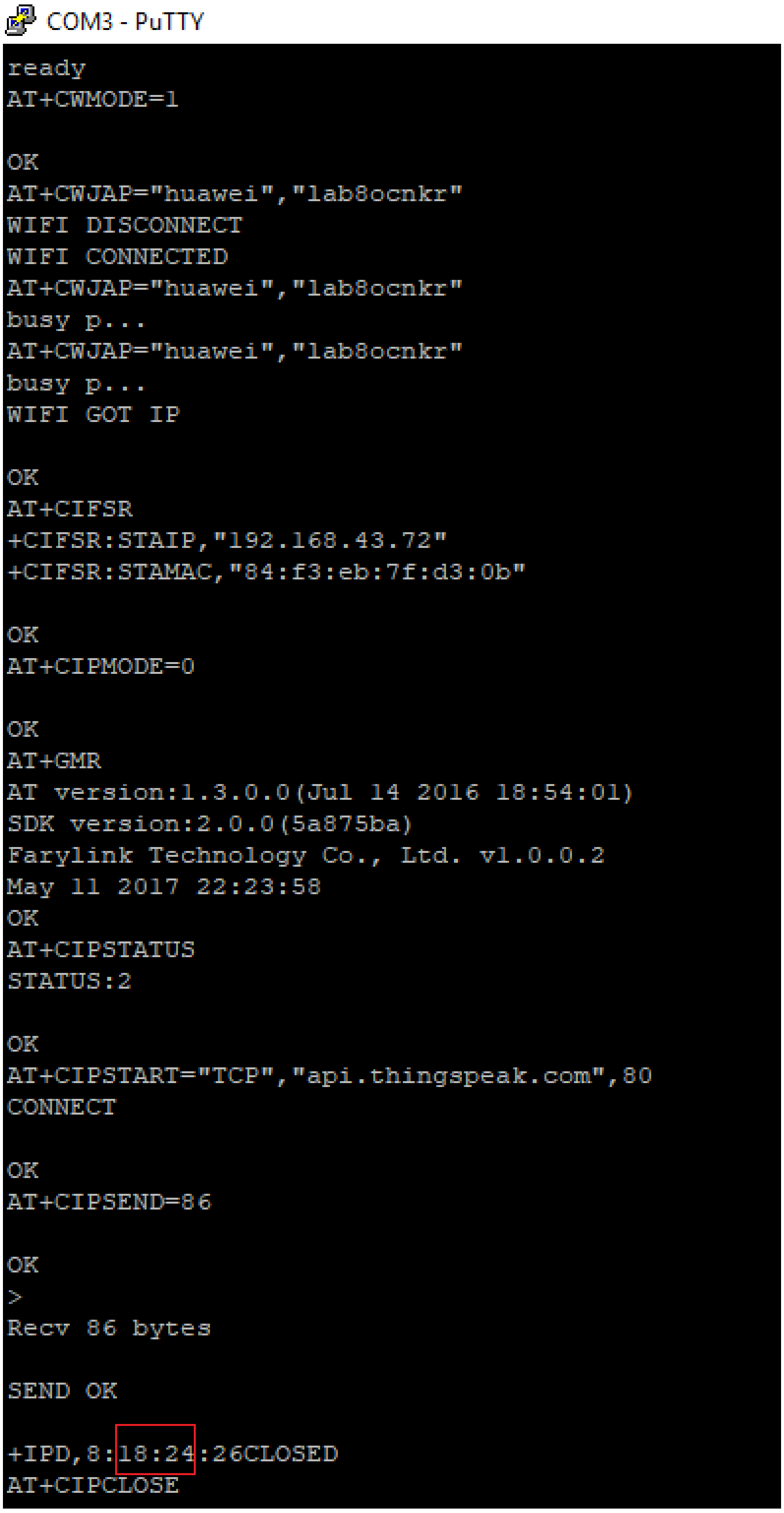
- **AT+CIPCLOSE**



```
//---------ESP8266_CloseTCPConnection----------
// Close TCP connection
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_CloseTCPConnection(void){
  int try=1;
  SearchStart("ok");
  while(try){
    ESP8266SendCommand("AT+CIPCLOSE\r\n");
    DelayMsSearching(4000);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 11-AT+CIPCLOSE

- **Putty**

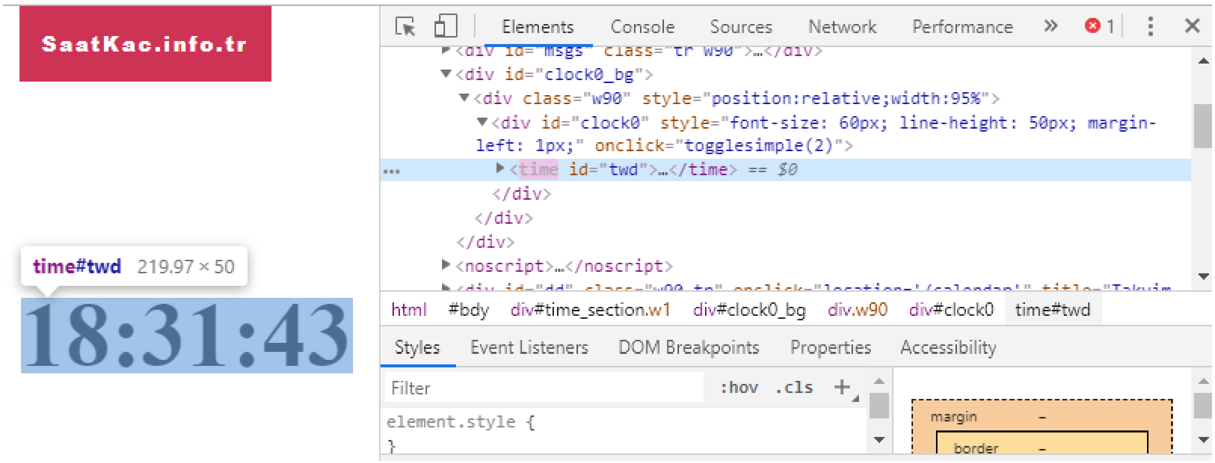


```
ready
AT+CWMODE=1

OK
AT+CWJAP="huawei","lab8ocnkr"
WIFI DISCONNECT
WIFI CONNECTED
AT+CWJAP="huawei","lab8ocnkr"
busy p...
AT+CWJAP="huawei","lab8ocnkr"
busy p...
WIFI GOT IP

OK
AT+CIFSR
+CIFSR:STAIP,"192.168.43.72"
+CIFSR:STAMAC,"84:f3:eb:7f:d3:0b"

OK
AT+CIPMODE=0

OK
AT+GMR
AT version:1.3.0.0(Jul 14 2016 18:54:01)
SDK version:2.0.0(5a875ba)
Farylink Technology Co., Ltd. v1.0.0.2
May 11 2017 22:23:58
OK
AT+CIPSTATUS
STATUS:2

OK
AT+CIPSTART="TCP","api.thingspeak.com",80
CONNECT

OK
AT+CIPSEND=86

OK
>
Recv 86 bytes

SEND OK

+IPD,8:18:24:26CLOSED
AT+CIPCLOSE
```

Figure 12-Putty results

You can see the results on putty. Red section shows the time data that we got from the web.

Getting Data from the Web

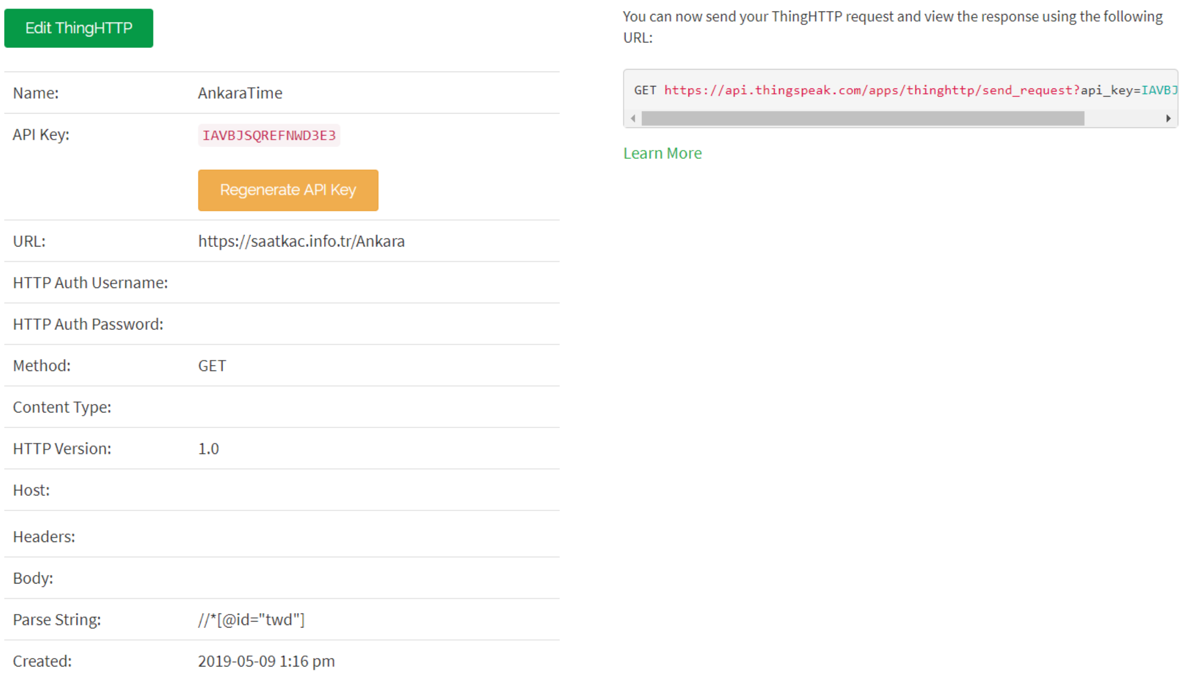


```
<div id="msgs" class="tr w90">…</div>
▼<div id="clock0_bg">
  ▼<div class="w90" style="position:relative;width:95%">
    ▼<div id="clock0" style="font-size: 60px; line-height: 50px; margin-
    left: 1px;" onclick="togglesimple(2)">
      ▶<time id="twd">…</time> == $0
    </div>
  </div>
</div>
▶<noscript>…</noscript>
▶<div id="dd" class="w90 tn" onclick="location='/calendar'" title="Takvim
```

html #bdy div#time_section.w1 div#clock0_bg div.w90 div#clock0 time#twd

```
element.style {
}
```

Figure 13-https://saatkac.info.tr/Ankara



Name:	AnkaraTime
API Key:	IAVBJSQREFNWD3E3 Regenerate API Key
URL:	https://saatkac.info.tr/Ankara
HTTP Auth Username:	
HTTP Auth Password:	
Method:	GET
Content Type:	
HTTP Version:	1.0
Host:	
Headers:	
Body:	
Parse String:	//*[@id="twd"]
Created:	2019-05-09 1:16 pm

You can now send your ThingHTTP request and view the response using the following URL:

```
GET https://api.thingspeak.com/apps/thinghttp/send_request?api_key=IAVBJ
```

Figure 14-ThingSpeak

We got the time information from this website and using ThingSpeak. We used this information in our Fetch array.

Board Pictures

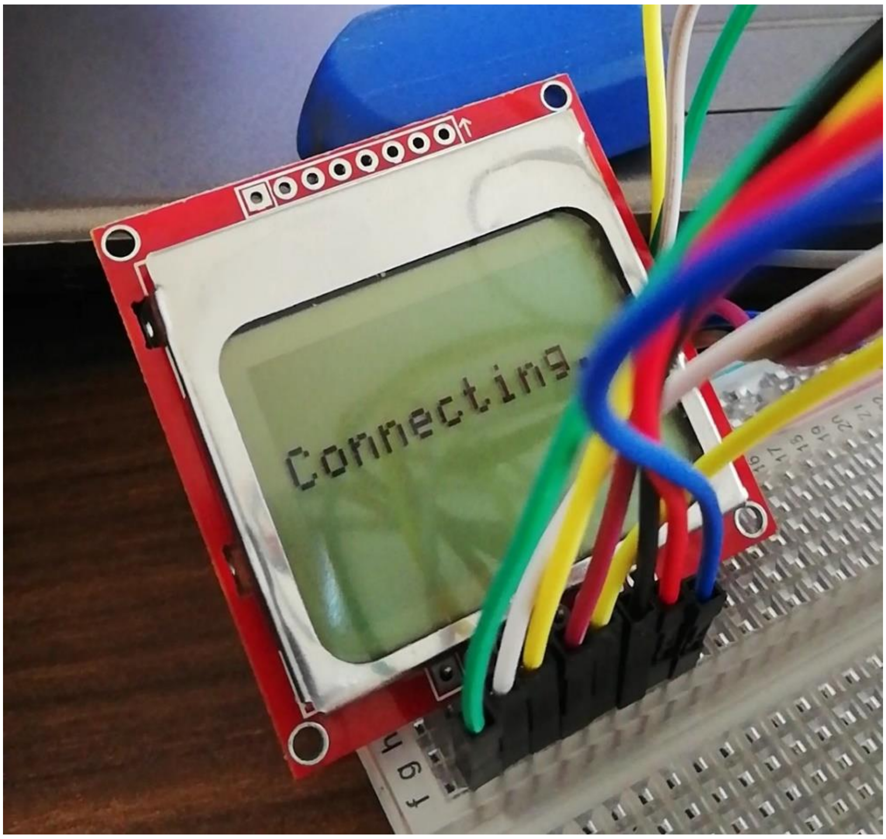


Figure 15-Connecting on board

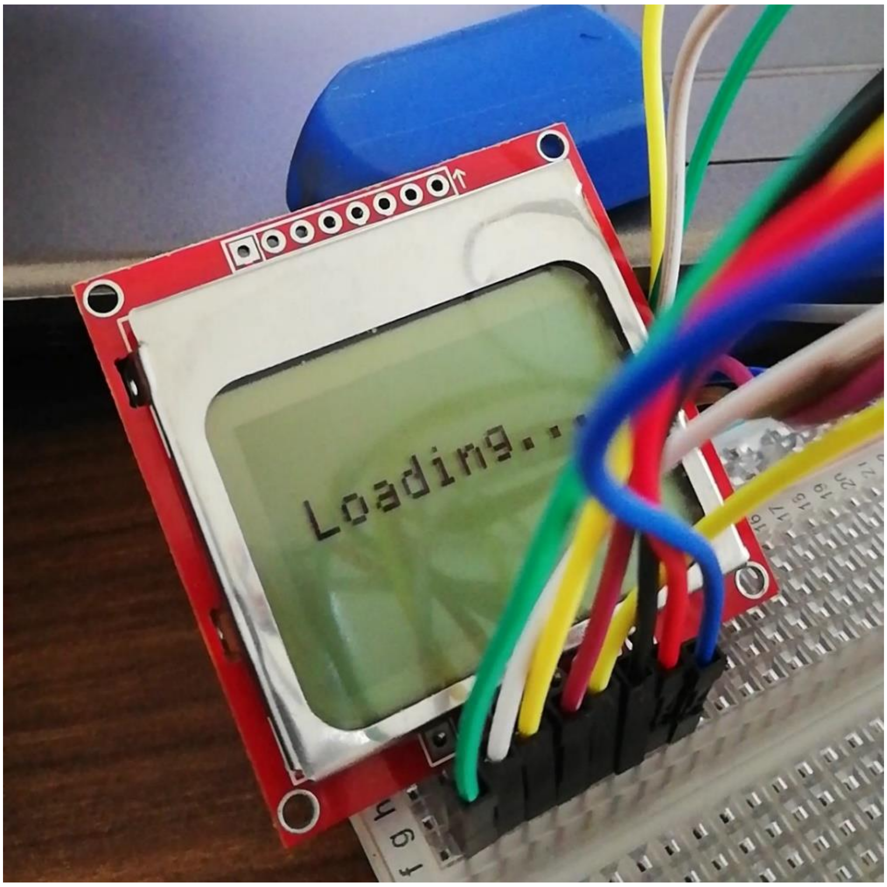


Figure 16-Loading on board

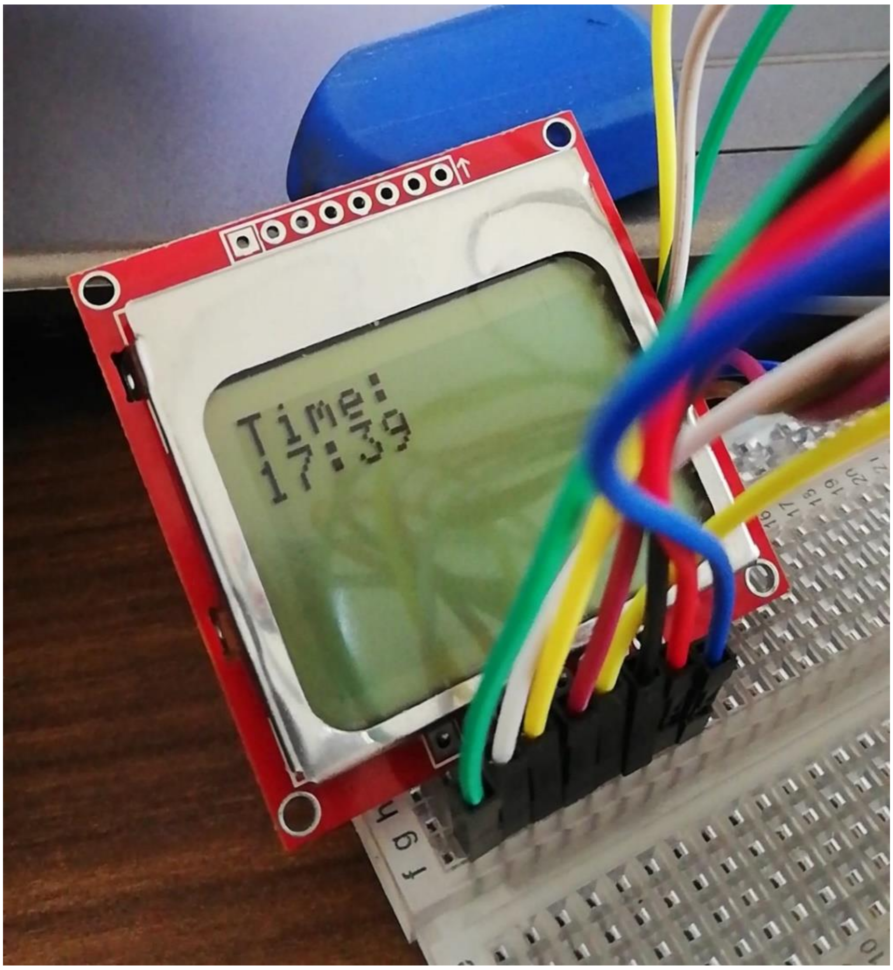
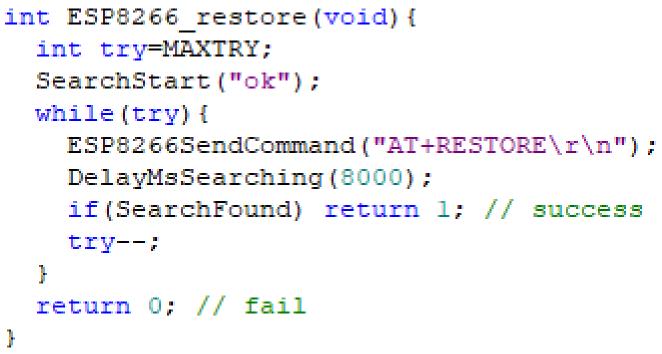


Figure 17-Time on board

Interesting Problem During the Experiment

The next day after working all day and getting some good results, suddenly our ESP8266 module started to give DNS Fail error when connecting to TCP. Fortunately, we found an AT command to solve it and wrote the function below.



```
int ESP8266_restore(void){
  int try=MAXTRY;
  SearchStart("ok");
  while(try){
    ESP8266SendCommand("AT+RESTORE\r\n");
    DelayMsSearching(8000);
    if(SearchFound) return 1; // success
    try--;
  }
  return 0; // fail
}
```

Figure 18-AT+RESTORE

Using this command, we restored the factory defaults and problem solved.