Rendu TP Microcontrôleur STM32

Année scolaire 2021-2022

Enseignant

M. Fréderic Lecomte

Etudiants

OMAR Omar
Théo BOUVIER-LECLERC

Introduction

Ce rendu constitue une base de nos Tps en microcontrôleur. En partant du programme global des capteurs, notre mission est de manipuler chacun seul, en modifiant, voire supprimant tous les codes et les librairies qui ne sont pas indispensables pour le fonctionnement du capteur concerné. Cela nécessite une compréhension des pins relié à chacun et aux différentes configurations y compris les fonctions et les variables propre au stm32 et aux capteurs.

1 Capteur de température

1.1 Modifications et suppression des fichiers

Tout d'abord, on s'introduit dans le fichier **main.c** car il constitue la base du programme et c'est ce dernier qui nous dirige vers les autres fichiers et librairies. On se rende compte que dans while(1) à la ligne 99, il y a des fonctions qui font appel à autres capteurs se trouvant dans le fichier sensors.c dans figure 1 donc on les supprime et on laisse que le fichier Temperaure_Test(). Les fonctions QSPI font partie de la configuration de termite du coup on n'y touche pas.

```
  /* Infinite loop */
  while (1)
  {
    QSPI_demo();
    QSPI_MemoryMapped_demo();
    Temperature_Test();
   // Humidity_Test();
   /* Pressure_Test();
    Magneto_Test();
    Gyro_Test();*/
   //Accelero_Test();
  }
}

/**
  * @brief  System Clock Configuration
  *         The system Clock is configured as follow :
  *            System Clock source            = PLL (MSI)
  *            SYSCLK(Hz)                     = 80000000
  *            HCLK(Hz)                       = 80000000
```

Figure.1

En faisant un clic droit sur la fonction temperature_Test (), on se retrouve dans le fichier sensors.c. On constate qu'on ne trouve que la syntaxe qui concerne ce capteur donc même pas besoin de la modifier. Pourtant, en continuant la lecture comme illustré en figure 2, on remarque qu'il contient que les fonctions concernant les autres capteurs et pour cela il sera évident de supprimer le fichier et de copier juste la fonction **temperature_Test ()** puis on la colle dans main.c. Maintenant il ne reste plus rien à modifier dans le module **Example/User.**

```c
void Pressure_Test(void)
{
  uint32_t ret = 0;
  float press_value = 0;

  printf("\n*****************************************************************\n");
  printf("\n********************** Pressure Test **********************\n");
  printf("\n*****************************************************************\n\n");
  BSP_PSENSOR_Init();
  printf("\n*** Type n or N to get a first Pressure data ***\n\n");
  printf("\n*** Type q or Q to quit Pressure Test ***\n\n");
  while(1)
  {
    ret = Serial_Scanf(255);
    if((ret == 'n') || (ret == 'N'))
    {
      printf("\n*** This is a new data ***\n\n");
      press_value = BSP_PSENSOR_ReadPressure();
      printf("PRESSURE is = %.2f mBar \n", press_value);
      printf("\n*** This is a new data ***\n\n");
      printf("\n*** Type n or N to get a new data ***\n\n");
      printf("\n*** Type q or Q to quit Pressure Test ***\n\n");
    }
    else if((ret == 'q') || (ret == 'Q'))
    {
      printf("\n*** End of Pressure Test ***\n\n");
      return;
    }
    else
    {
      printf("\n*** Type n or N to get a new data ***\n\n");
      printf("\n*** Type q or Q to quit Pressure Test ***\n\n");
    }
  }
}

/**
  * @brief  Test of HTS221 humidity sensor.
  */
void Humidity_Test(void)
{
  uint32_t ret = 0;
  float humidity_value = 0;

  printf("\n*****************************************************************\n");
  printf("\n********************** Humidity Test **********************\n");
  printf("\n*****************************************************************\n\n");
  BSP_HSENSOR_Init();
  printf("\n*** Type n or N to get a first Humidity data ***\n\n");
  printf("\n*** Type q or Q to quit Humidity Test ***\n\n");
  while(1)
  {
    ret = Serial_Scanf(255);
    if((ret == 'n') || (ret == 'N'))
    {
      printf("\n*** This is a new data ***\n\n");
      humidity_value = BSP_HSENSOR_ReadHumidity();
      printf("HUMIDITY is = %.2f %%\n", humidity_value);
      printf("\n*** This is a new data ***\n\n");
      printf("\n*** Type n or N to get a new data ***\n\n");
      printf("\n*** Type q or Q to quit Humidity Test ***\n\n");
    }
    else if((ret == 'q') || (ret == 'Q'))
    {
      printf("\n*** End of Humidity Test ***\n\n");
      return;
```

Figure.2

1.1.1 DRIVERS/BSP/B-L475E-IOT01

Ce module est parlant car il contient des fichiers nommés selon le capteur concerné comme illustré dans la figure ci-après.

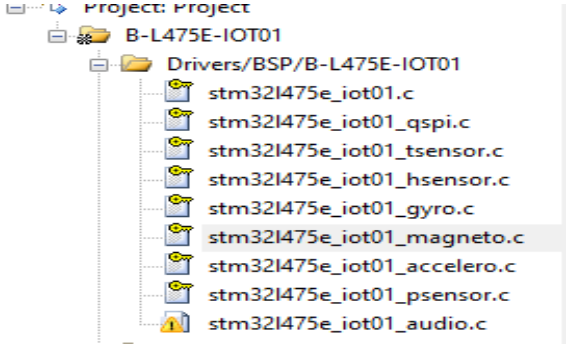

Figure.3

Si on veut vérifier pour chacun que le contenu n'est pas lié aux autres capteurs, on s'introduit en faisant double clic sur chacun. Pour ce module on va s'intéresser au contenu de tous les fichiers mais directement au fichier _tsensor.c, au iot01.c et qspi.c car ces derniers sont importants pour configurer le **GPIO** et **I2C**. On laisse le fichier _tsensor.c car il est nécessaire pour la programmation du capteur de température donc on supprime les autres dans ce module.

1.1.2 Drivers/BSP/Components

On commence par supprimer le fichier **LIS3MDL** car ce dernier ne concerne que le capteur magnetometer, et le **lsm6dsl** car celui-ci est utilisé pour la configuration du capteur LSM6DSL (accelero et gyro). En dernier on supprime le fichier **lps22hb car** il concerne le capteur de pression. Finalement pour le fichier **hts221** on supprime les lignes 39 -> 47 puis 63 -> 146 car il ne contient que les fonctions du capteur d'humidité. Evidemment on a eu des erreurs dès la première compilation, car on a laissé des variables ou fonctions liées aux autres capteurs. Pour résoudre ce problème il faut tout simplement entrer dans la fenêtre output, appuyer sur l'erreur, puis voir dans quelle fonction ou variable on est ramené pour les supprimer. Pourtant il ne faut pas oublier que certains fichiers sont en format read (lecture) et qu'il est donc impossible de les modifier, comme par exemple les fichiers du module **Drivers/BSP/Components.** Pour ce faire, comme le montre la figure 4, on a créé un nouveau fichier que l'on a choisit de type file.c, on lui a donné le même nom que celui de **hts221,** puis on a copié-collé le code en modifiant quoi que ce soit dans le nouveau fichier.

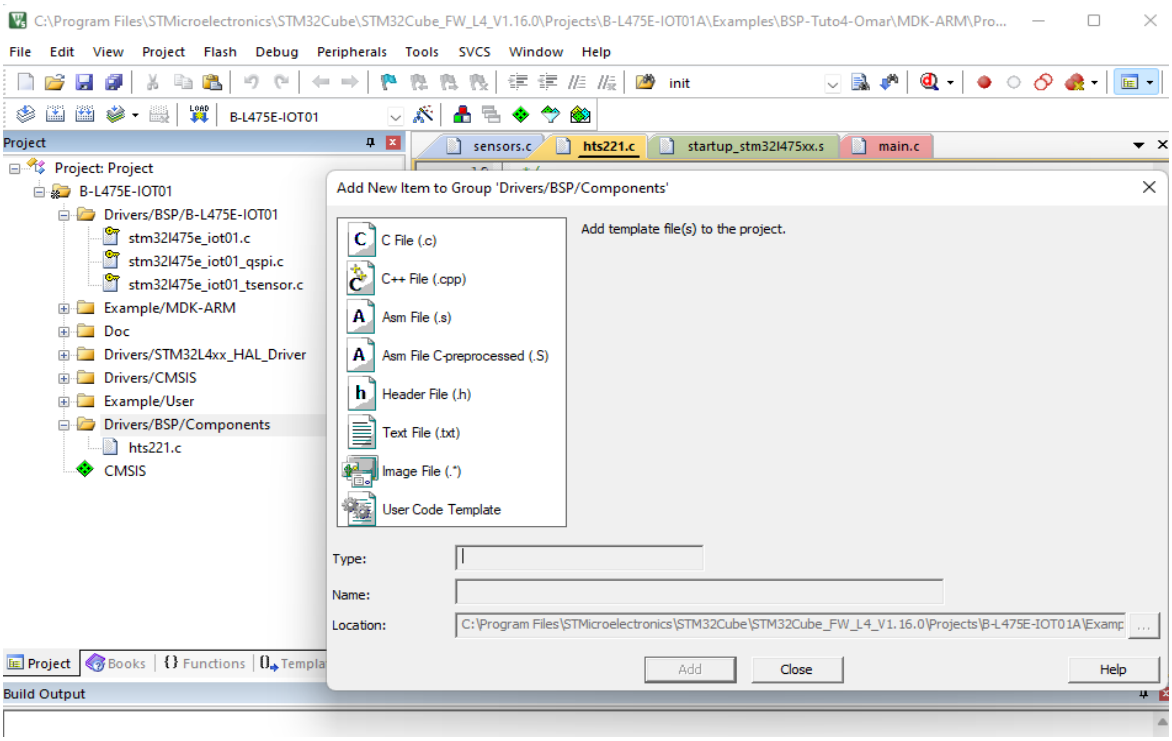

Figure.4

1.1.3 Décortication du code

```
                                        Temperature Program ! :)
                    ------------------------------------------------------------------------------------------


In this file we show the different layers of the temperature program.The indent means a function nested in the one just before.

1.Temperature_test(void)
   1.1 BSP_TSENSOR_init() Initialize the sensor
        1.1.1 SENSOR_IO_Init( initialize I2C communication bus) Low level and tells whether the sensor
                                            is was well initialised or not(it returns TSENSOR_OK)

   1.2 Serial_Scan(uint32_t value) It's used to get numeric values from hyperterminal(termit) with a maximum value allowed.
        1.2.1 getchar(); It returns tmp which is the value given by the user.
   1.3 BSP_TSENSOR_ReadTemp return the temperature read thanks to I2C protocol. The value type returned "tsensor_drv" is static.
        1.3.1 ReadTemp(TSENSOR_I2C_ADDRESS) is a float variable that belongs to the TSENSOR_DRV structure, and equal to HTS221_I2C_ADDRESS,
                                        defined in the stm32l475e_iot01.h as (uint8_t)0xBE.
                                        This function reads directly from the bus the temperature values.



                    ------------------------------------------------------------------------------------------
```

Figure 5

Dans cette partie, on s'intéresse à reformuler notre code en plusieurs couches afin de bien comprendre son fonctionnement. Pour rappel, les fonctions de QSPI sont utilisées pour afficher les information fournis par les capteurs, et recevoir des commandes de l'utilisateur. La figure ci avant montre

les différentes liaisons entre les fonctions qui sont imbriqués dans la fonction Temperature_test. Les fonctions placées sur le même niveau montrent qu'elles ne sont pas liées, mais qu'elles sont placées l'une après l'autre car elles font partie du fonctionnement du capteur. Les autres fonctions qui sont imbriqués dans une première, sans bien espacés (même concept que python).

1.1.4 Validation du TP

Pour vérifier que notre code est correct avec toutes les modifications qu'on a faites, on commence par connecter la carte et on s'assure qu'elle est bien connectée, comme illustré dans la figure ci-dessous.

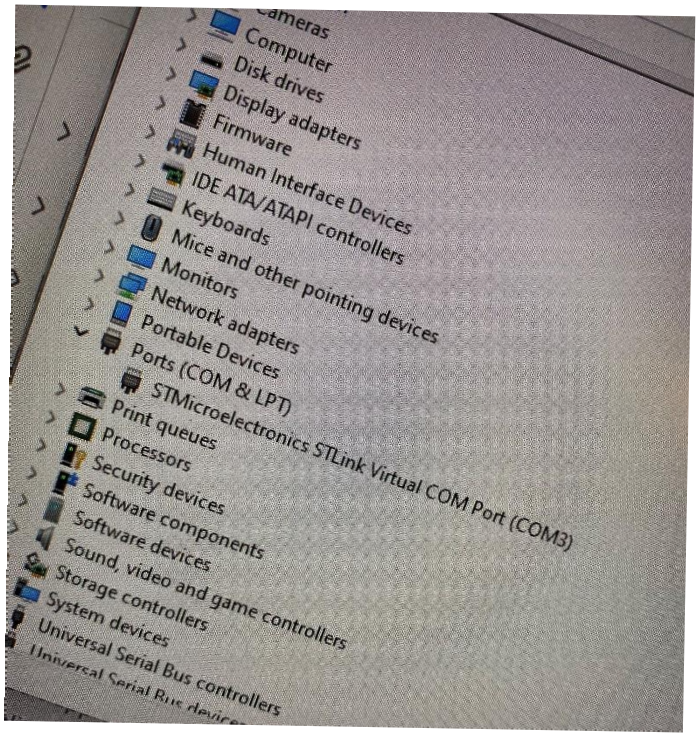

Figure.5

On ouvre le projet dans *keil* et on fait *Project -> Build target* puis *start/stop debugging*, après on ouvre **termite**, et on appuie sur le bouton utilisateur pour allumer led2, et on appuie sur la touche q une première fois pour terminer le test de QSPI Test, puis une seconde fois pour terminer le test de QSPI Memory Mapped Test. Maintenant on arrive au test de température et on appuie su la touche n du clavier pour afficher la température. Notre code a fonctionné et la figure 6 (il faut zoomer un peu) illustre bien le nom de notre projet en haut, les fichiers supprimés à gauche, la température affichée (20.87 degrés Celsius) dans la fenêtre de termite à droite, et le code en mode running en bas.

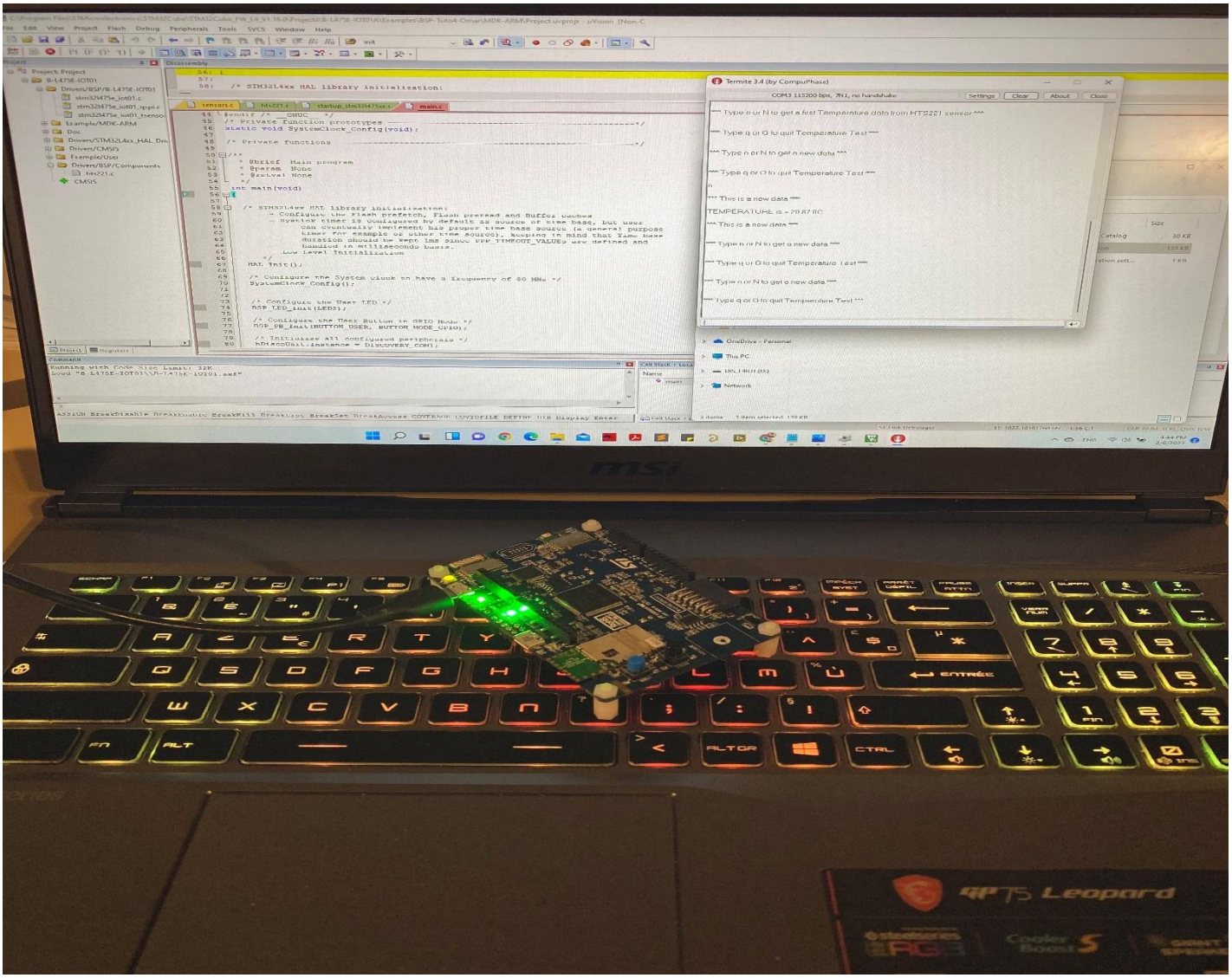

Figure.6

2.Client-serveur (WIFI)

2.1 Fonctionnement

Dans cette dernière partie, on doit trouver un programme impliquant l'utilisation du module wifi esp. On en a trouvé un sur internet, donc on va expliquer son fonctionnement. Pour cet exemple le module **Es_Wifi** est utilisé en mode TCP et commandé grâce à AT pour l'initialisation de la connexion client, et en utilisant l'interface SPI pour communiquer avec elle. Ce mode consiste à connecter le module **Es_Wifi** à un point d'accès (**Soft Access point**) comme un module routeur, comme illustré avec la figure ci-après.

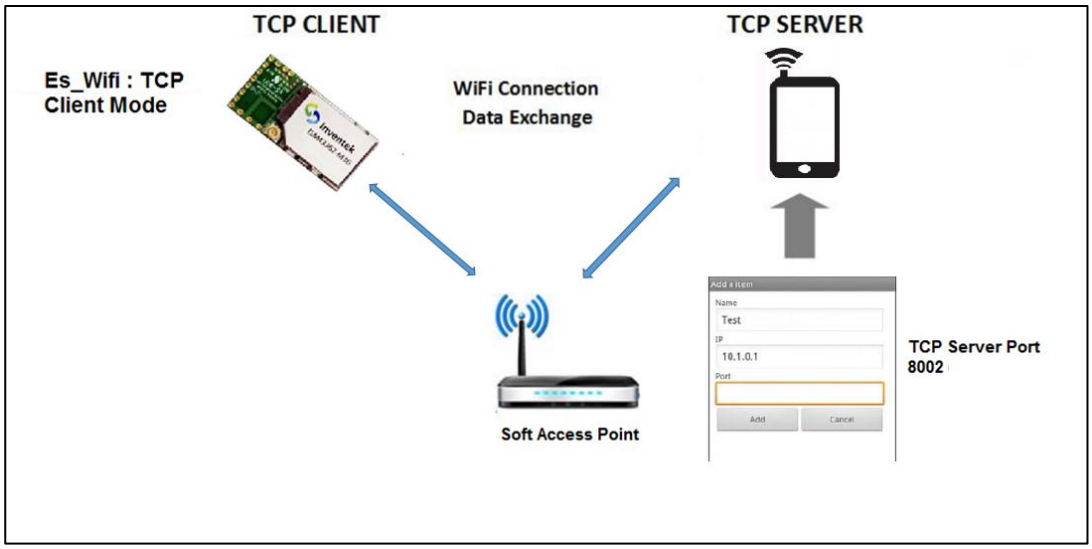

Figure.7

Puis on se connecte à une adresse IP déjà connectée sur cette dernière, notamment dans le cas du portable illustré dans la figure ci-avant, en l'initialisant grâce à RemoteIP[] dans main.c et en définissant l'identifiant (SSID) et le mode de passe de L'AP associé au router. De plus, il faut configurer le téléphone en mode **Access point** en choisissant la porte 8002. Désormais le module **Es_Wifi** est considérée comme client, et va vérifier la connexion. Si cette dernière est bien faite, un message "STM32 : Hello !" est envoyé au serveur.

2.2 Décortication du code

Maintenant on passe à la partie codage pour voir un peu comment ça se passe. La figure 8 illustre les différentes couches composants notre programme. Tout d'abord, et avant de rentrer dans int main(void), on se rend compte qu'il y a des variables existantes et qu'on doit les définir au début du programme, notamment SSID, PASSWORD, Remote_ip et remote_port mais cette partie est déjà développé dans 2.1. Ensuite en regardant les fonctions et derrière ce qui se passe, on conclut la figure ci-après. De 1 à 4 cela n'a rien à voir directement avec le module **Es_Wifi**. Wifi_init permet d'initialiser l'ensemble de configurations concernant les fonctions du wifi permettant un démarrage correct, et notamment tester la communication à travers SPI. Dans Wifi_getMac on récupère l'adresse physique du module. Si on ne réussit pas à avoir ce dernier, la led2 s'allume. Puis la fonction wifi_connect permet de vérifier la connexion. Si cette dernière est bonne, elle retourne ret(ret=wifi_status_ok), grâce à AT_ExecuteCommand. Cette commande est assez présente car c'est grâce à elle qu'on puisse communiquer avec le module **Es_Wifi** à travers SPI. Après on récupère les configurations de la connexion auquel on s'est connecté grâce à AT_ExecuteCommand() et AT_parseIsconnected(). Maintenant on procède à la récupération de l'IP auquel on s'est connecté puis on établit une connexion (grâce à "wifi_OpenClientConnection" et on vérifie si on a reçu des données grâce à wifi_receiveDATA et si c'est le cas, on envoie "STM32 : Hello!\n" grâce à WIFI_SendData.

```
             1_HAL_Init();
/Reset of all peripherals, Initializes the Flash interface and the Systick.

             2_SystemClock_Config();
/Configure the system clock

             3_BSP_LED_Init(Led2)
            /Configuration de Led2

            4_BSP_COM_Init()
/enable GBIO,USART CLOCK, TX AND RX alternate functions

            5 Wifi_Init
            6 WIFI_GetMAC_Address(MAC_Addr)
            7 wifi_Connect
                    7.1 ES_WIFI_Connect()
                            7.1.1 AT_ExecuteCommand()
                    7.2 ES_WIFI_GetnetworkSettings()
                            7.2.1 AT_ExecuteCommand
                            7.2.2 AT_ParseConnSettings
            8   WIFI_GetIP_Address()
                    8.1 ES_WIFI_IsConnected()
                            8.1.1 AT_ExecuteCommand()
                            8.1.2 AT_ParseIsConnected()
            9 WIFI_OpenClientConnection
                    9.1 ES_WIFI_StartClientConnection
                    9.2 AT_ExecuteCommand
            10  WIFI_ReceiveData
                    10.1 ES_WIFI_ReceiveData
                            10.1.1 AT_ExecuteCommand
                            10.1.2 AT_RequestReceiveData
            11 WIFI_SendData
                    11.1 ES_WIFI_SendData
                            11.1.1 AT_ExecuteCommand
                            11.1.2 AT_RequestSendData
```

Figure.8

Conclusion

Au niveau de ce TP en microcontrôleur, on a appris à comprendre chaque module et son contenu, à modifier un code de façon à l'adapter pour un seul capteur, et à corriger les erreurs. Puis, nous avons appris à créer de nouveaux fichiers pour pouvoir modifier un fichier sous format read, et identifier les lignes de code liées au fonctionnement du stm32 seulement. Enfin on a maîtrisé la décortication d'un programme afin de mieux comprendre son fonctionnement. Ce travail constitue une base pour faire fonctionner les autres capteurs chacun seul, et un point fort dans la programmation pour laquelle même si on n'est pas expert, on arrive à bien comprendre son contenu et à le modifier selon notre application.