STM32 FOTA 例程之 ESP8266 使用

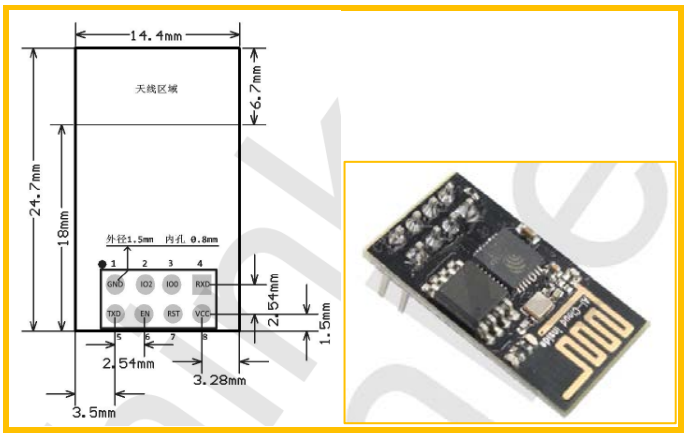
前言

这一节，我们来聊聊 STM32 的 FOTA 例程中用到的 wifi 模块：ESP-01。ESP-01 是安信可公司基于 ESP8266 wifi 芯片的 WIFI 模块。在 STM32 FOTAdemo 里，用来实现无线通信。下面我们将来认识一下这个模块，并介绍 demo 里相关底层驱动的实现。

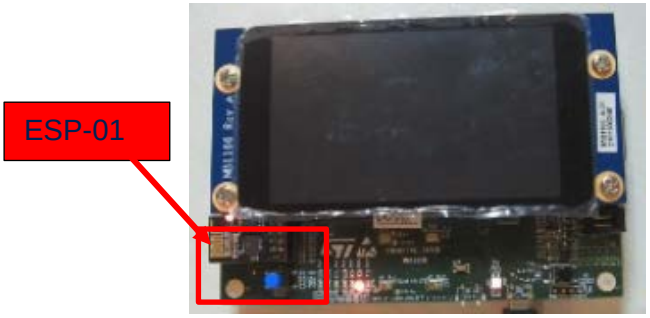
ESP-01 模块

ESP-01 模块集成 esp8266EX WIFI 芯片，支持 802.11b/g/n 协议，支持 UART/GPIO 等接口，内嵌 LwIP 协议栈，支持 STA/AP/STA+AP 工作模式，是一款低成本的无线模块。

ESP-01 模块采用 DIP-8 封装。尺寸和引脚定义见下图。提供一个 UART 接口和两个 GPIO 口。



STM32F769 探索板的 CN2 接口支持对 ESP-01 模块的扩展，可以直接将 ESP-01 模块插在 CN2 接口上。STM32F769 和 ESP-01 之间通过串口通信，串口配置为：115200 波特率，8 位数据位，无奇偶校验，1 位停止位。连接如下：



AT 指令格式

ESP-01 的 AT 指令集就是 ESP8266 的 AT 指令集。可以细分为四种类型：

类型	指令格式	描述
测试指令	AT+<x>=?	该命令用于查询设置命令或内部程序设置的参数以及其取值范围。
查询指令	AT+<x>?	该命令用于返回参数的当前值。
设置指令	AT+<x>=<...>	该命令用于设置用户自定义的参数值。
执行指令	AT+<x>	该命令用于执行受模块内部程序控制的变参数不可变的功能。

要注意的是，

不是每条 AT 指令都具备上面 4 种类型，具体要去看该条 AT 指令的说明。

使用双引号表示字符串数据。比如“123”，就是一个字符串，而 123 就是数字。

开头的 AT 指令两个字符必须大写，每条命令以回车换行符结尾“\r\n”

ESP8266 的 AT 指令集又分为：基础 AT 命令（对模块的配置，串口设置等），WIFI 功能 AT 命令（设置 wifi 模式，连接 AP 等），TCP/IP 功能 AT 命令（建立 TCP 连接，收发数据等）。

作为 TCP 客户端工作

模块初始化

程序上电运行后，在开始使用 WIFI 模块前，必须先对其进行初始化。除了 GPIO 口以及串口的初始化外，还需要对 WIFI 模块进行配置，使其工作在我们希望的模式下。在 STM32 FOTA demo 里 ESP8266 应该工作在 STATION 模式和多连接模式下。所以在初始化的时候，需要通过相应的 AT 指令进行配置。下面是初始化的代码以及 AT 指令执行的过程。

```
ESP8266_StatusTypeDef ESP8266_Init(void)
{
  ESP8266_StatusTypeDef Ret;

  /* Configuration the IO low layer */
  if (ESP8266_IO_Init() < 0)
  {
    return ESP8266_ERROR;
  }

  /* Disable the Echo mode */
  /* Construct the command */
  memset (AtCmd, '\0', MAX_AT_CMD_SIZE);
  sprintf((char *)AtCmd, "ATE0%c%c", '\r', '\n');

  /* Send the command */
  Ret = runAtCmd(AtCmd, strlen((char *)AtCmd), (uint8_t*)AT_OK_STRING,NULL);

  /* Exit in case of error */
  if (Ret !=  ESP8266_OK)
  {
    return ESP8266_ERROR;
  }
```

关回显功能

```
  /* Setup the module in Station Mode*/
  /* Construct the command */
  memset (AtCmd, '\0', MAX_AT_CMD_SIZE);
  sprintf((char *)AtCmd, "AT+CWMODE=1%c%c", '\r', '\n');

  /* Send the command */
  Ret = runAtCmd(AtCmd, strlen((char *)AtCmd), (uint8_t*)AT_OK_STRING,NULL);

  //colse all socket connection
  if(ESP8266_CheckMUXStatus()==1)
  {

    uint8_t i=0;
    for(i=0;i<4;i++)
    {
      ESP8266_CloseConnection(i); //don't check and return the returned status
    }
  }
  else //this application is mult-connection application,if MUXMODE is 0,it means this is
the first time run
    ESP8266_SetMuxMode(1);

  return Ret;
}
```

设置 wifi 模块工作在 STATION 模式

设置多连接模式

AT 指令的运行记录：

```
ATE0..
..OK..
AT+CWMODE=1..
..OK..
AT+CIPMUX=1..
..OK..
```

关回显功能

设置 wifi 模块工作在 STATION 模式

设置多连接模式

连接到 AP

调用 ESP8266_JoinAccessPoint 函数，输入 AP 的 SSID 和密码，连接到对应的 wifi 热点。

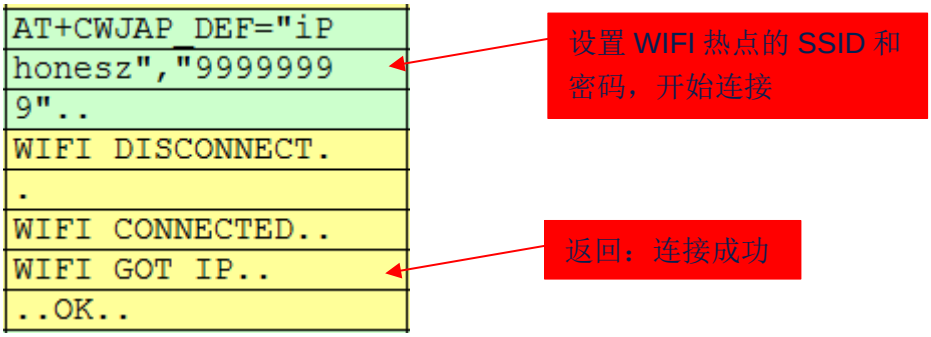
```
ESP8266_StatusTypeDef ESP8266_JoinAccessPoint(uint8_t* Ssid, uint8_t* Password)
{
  ESP8266_StatusTypeDef Ret;

  /* List all the available Access points first
   then check whether the specified 'ssid' exists among them or not.*/
  memset(AtCmd, '\0', MAX_AT_CMD_SIZE);
  sprintf((char *)AtCmd, "AT+CWJAP_DEF=\"%s\",\"%s\"%c%c", Ssid, Password, '\r', '\n');

  /* Send the command */
  Ret = runAtCmd(AtCmd, strlen((char *)AtCmd), (uint8_t*)AT_OK_STRING,NULL);
```

```
  return Ret;
}
```

AT 指令运行记录：



上图的黄色部分是 WIFI 模块的返回状态。必须要接收到"OK\r\n"，才能去读取 IP 地址。连接 WIFI 热点的过程，需要的时长不一定，有时 2、3 秒，有时 6、7 秒。所以这里最好把等待的时间 留长一点，否则经常会出现连接 WIFI 热点失败的情况。

与服务器建立连接

连接到 WIFI 热点后，就可以开始与服务器建立连接了。ESP8266 支持 5 个并发连接。

一般我们知道的不是目标服务器的 IP 地址，而是域名。所以在开始创建连接之前需要先通过 DNS 服务获该取域名对应的 IP 地址。ESP8266 也提供了相应的 AT 指令。

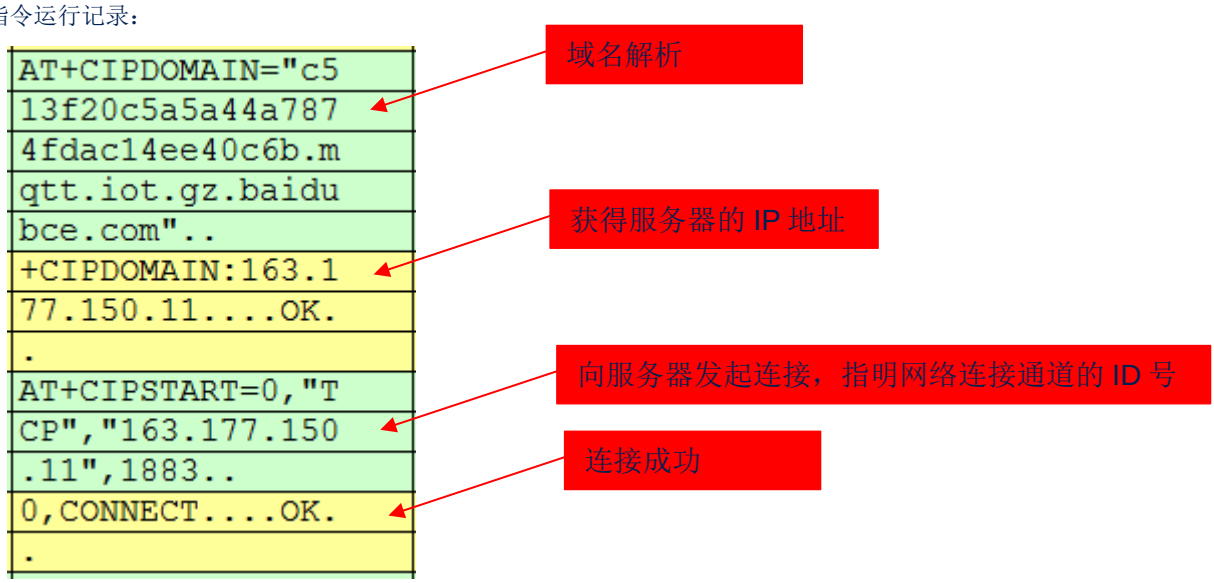
下面是代码中的一段和建立连接相关的代码：

域名解析

根据获得的 IP 地址连接服务器

```
    if (WIFI_GetHostAddress((char *)hostname, ip_addr) != WIFI_STATUS_OK)
    {
      // TODO: Defect report on WIFI_GetHostAddress() which return code is not
informative.
      // NB: This blocking call may take several seconds before returning. An
asynchronous interface should be added.
      msg_info("The address of %s could not be resolved.\n", hostname);
      rc = NET_ERR;
    }
    else
    {
      if( WIFI_STATUS_OK == WIFI_OpenClientConnection(
            (uint32_t) sock->underlying_sock_ctxt, WIFI_TCP_PROTOCOL, "", ip_addr,
dstport, 0) )
      {
        rc = NET_OK;
      }
      else
      {
        underlying_socket_busy[(int) sock->underlying_sock_ctxt] = false;
        msg_error("Failed opening the underlying Wifi socket %d.\n", (int) sock-
>underlying_sock_ctxt);
        sock->underlying_sock_ctxt = (net_sockhnd_t) -1;
      }
    }
```

AT 指令运行记录：



发送数据

发送数据过程分两步：

1.发送 AT+CIPSEND=<LINK ID>,<LENGTH>命令。说明要往哪个 socket 通道，发送多少字节的数据。

2.收到 WIFI 模块返回的"OK\r\n>"后，再发送数据。

下面是例程中，ESP8266 发送数据的代码实现。

```
ESP8266_StatusTypeDef ESP8266_SendData(uint8_t socket,uint8_t* Buffer, uint32_t Length)
{
  ESP8266_StatusTypeDef Ret = ESP8266_OK;

  if (Buffer != NULL)
  {
      /* Construct the CIPSEND command */
    memset(AtCmd, '\0', MAX_AT_CMD_SIZE);
    sprintf((char *)AtCmd, "AT+CIPSEND=%lu,%lu%c%c", socket,Length, '\r', '\n');

    /* The CIPSEND command doesn't have a return command
       until the data is actually sent. Thus we check here whether
       we got the '>' prompt or not. */
    Ret = runAtCmd(AtCmd, strlen((char *)AtCmd), (uint8_t*)AT_SEND_PROMPT_STRING,NULL);

    /* Return Error */
    if (Ret != ESP8266_OK)
      {
      return ESP8266_ERROR;
      }

     /* Send the data */
     Ret = runAtCmd(Buffer, Length, (uint8_t*)AT_SEND_OK_STRING,NULL);
```

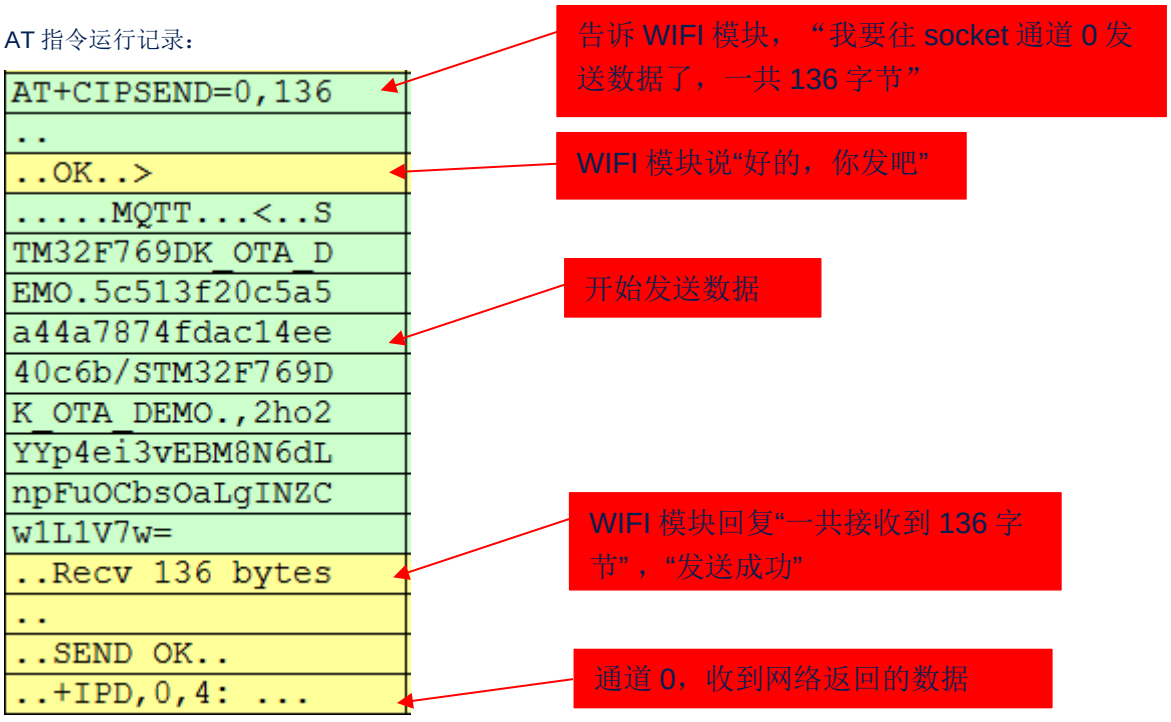
准备发送数据，等待 WIFI 模块发的“发送数据提示符”

```
    if (Ret != ESP8266_OK)
    {
      return ESP8266_ERROR;
    }
  }

  return Ret;
}
```

开始发送数据

AT 指令运行记录：



接收数据的模式

在上一节“发送数据”的 AT 指令执行记录中，我们看到最后 WIFI 模块回复了一条以“+IPD”开头的消息。这是从网络上的服务器返回的数据。使用 Esp8266 模块时，接收网络上发来的数据，不需要 MCU 先发“接收”的“AT 指令。只要收到数据，ESP8266 就会通过“+IPD”开头的数据包，将数据传递给 MCU。

demo 里接收数据，我们采用了下面的这种模式：

串口中断程序（UART ISR）负责把从 UART 端口接收到的数据放到 ring buffer 中。

上层应用程序需要数据的时候，不直接从 ring buffer 中拿，而是从各自的 socket buffer 中拿，每个通道（ESP8266 支持 5 个通道）都有一个对应的 socket buffer。本示例代码里有两个应用，MQTT 和 HTTP，因此使用了 wifi 模块的两个 socket 通道。只有在 socket buffer 中没有有效数据的时候，再到 ring_buffer 中去查看是否有收到数据，然后将数据拷贝到 socket buffer 中。除了读取 wifi 传过来的云端数据，上层应用程序还需要就发送的 AT 指令检查响应。这个是在 ring buffer 中去读取。如果发现 ring buffer 里面有以“+IPD”开头的来自云端的数据，会“帮忙”拷贝到对应的 socket buffer 里。

这么做虽然多了一次数据的拷贝（集中的 ring buffer 到各自的 socket buffer），但是方便上层应用程序的操作。因为有时候，我们会一下子从网络端收到上百个字节的数据，这些数据都在一个“+PID”开头的数据包中传给了 MCU。但也许上层应用程序，希望一个或几个几个字节的来解析这个数据包。比如，MQTT 中，要先读一个字节的数据判断一下收到的是什么类型的

数据包，是 CONNACK 还是 PUBLISH？在这种情况下，增加了 socket buffer 这一层的话，就可以做到，每次从 ring buffer 中读数据，都是将整个+IPD 的数据包读出，而上层应用程序，可以想从 socket buffer 中每次读几个字节都可以。

将数据从 ring buffer 拷贝到 socket buffer 是由 ExtractDatafromIPDpacket（）函数来完成的。

在 esp8266_io.h 文件中，分别定义了 socket buffer 和 ring buffer 的结构：esp_sock_ctx_t， RingBuffer_t。

注意：该示例中 MQTT 和 HTTP 应用都在一个 task 里面调用，如果在不同的 task 实现，接收驱动需要做相应修改。

小结

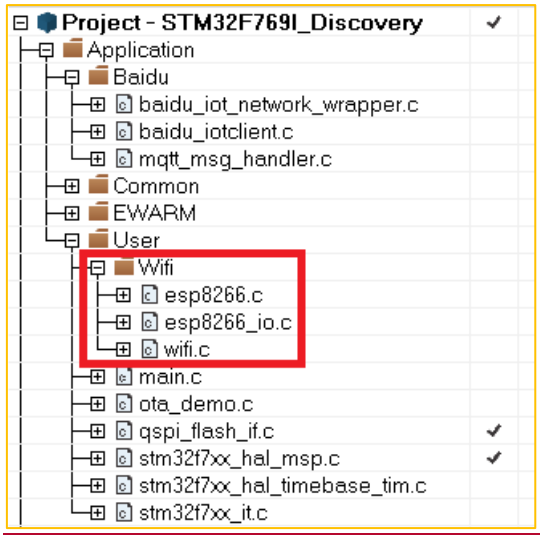
在例程中，和 WIFI 模块相关的驱动由三层组成：

在 ESP8266_io.c 中，是最底层的跟 STM32 外设打交道的部分。包括初始化引脚，从串口读取和发送数据；

Esp8266.c 中是对 AT 指令的实现。

Wifi.c 是 wifi 底层驱动和上层的一个接口。我们可以看到它的函数和 ESP8266.c 中的某些函数名字都很类似。

我们前面讲的内容基本都在这三个文件中，对于如果想更换 WIFI 模块来讲，它主要涉及到的代码也就是这三部分。



重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。