ESP 下载插件使用简介

拟制	版本	时间	备注
FAE(Puff)	V1.2	20170724	修改第三章的配置项

ESP 下载插件用于仪器厂家的 download 工具，嵌入到客户厂家其他工具内，做到稳定，通用，易用。

1：工具介绍

工具功能简介：

- RAM/Flash download
- Write custom MAC address

工具支持平台：

- ESP8089
- ESP8266
- ESP8285
- ESP32
- ESP8689

2：工具说明

ESP 下载插件包中包括三个文件夹：

图 2-1 ESP 下载插件包包含内容

（1）config 文件夹下包括 settings.txt 和 ESP_MAC_create_tool 文件夹。

(a) settings.txt 是参数配置文件。配置文件中参数的设置将在第三章中详述。

(b) ESP_MAC_create_tool 文件夹下有可执行文件 creat_MAC_file.exe。该文件的功能是生成 MAC 地址文件，以便客户需要自定义 MAC 地址的时候，可以自行生成一个符合要求的 MAC 地址表。如果用户不需要自定义 MAC 地址，则不需要运行这个文件。

（2）doc 文件夹下是通用下载工具用户指南。

（3）image 文件夹下包括工具使用中需要烧录的固件。

（4）工具根目录下是 esp_cmm_download_tool_v1.0.0.0.exe。功能是下载固件到 RAM 或者 FLASH 中。它会读取配置文件中的参数，根据对应的参数执行相应的下载工作。

3：参数配置说明

在执行工具之前，一定要先修改配置文件中的参数，否则工具不能成功运行！

配置参数全部写在 config/settings.txt 文件中。该文件中的参数包括：

（1）CHIP_TYPE：

支持平台：如 ESP32 或 ESP8266

（2）COMPORT：

Com 口：选择对应的 COM 口，如“COM2”

（3）BAUDRATE：

下载波特率：如 460800

（4）RAM_PATH：

烧写到 RAM 中固件的存放路径，程序会将该路径下的固件烧写到芯片的 RAM 中运行

（5）PO_NUMBER：

批次号：用于自定义 MAC 地址烧写时使用的校验机制，校验目标烧写 MAC 地址文件和当前烧写台的配置对应，与生成 MAC 地址文件中的 PO number 对应

（6）ESP_DOWNLOAD_MODE：

下载模式：选择 RAM download 或者 FLASH download。当前只支持 RAM download 功能

（7）ESP_WRITE_CUS_MAC_EN：

自定义 MAC 地址使能：配置是否需要烧写用户自定义的 MAC 地址

（8）ESP_DOWNLOAD_TOOL_VER：

ESP 下载工具当前版本号。该版本号会与工具中的版本号作比较，若不同，则提示工具使用错误

（9）CONNECT_WAITING_MAX_TIME：

连接超时时间设置。当超过所设定的值时，则跳过连接过程，直接返回连接失败状态

（10）READ_FLASH_INFO：

是否读取 FLASH 信息的标志位。置为 1，则读取 flash 信息，此功能的返回结果会在 stage 2 中打印。如果只是烧写 mac 地址的话，则不会打印该信息

（11）EFUSE_CHECK_MODE：

选择校验 EFUSE 的模式。置为 0，则不校验 EFUSE；置为 1，则以乐鑫通用的结构来校验 EFUSE；置为 2，则以小米自定义的结构来校验 EFUSE

（12）FLASH 下载：

(a) FLASH_FREQ：晶振频率

(b) FLASH_MODE：FLASH 访问方式

(c) FLASH_SIZE：FLASH 大小

(d) FLASH_PATH：烧入 FLASH 的固件路径

(e) FLASH_ADDR：烧入 FLASH 的下载地址

注：（1）由于 MAC 地址是烧写到芯片中，一旦烧入就不能更改，请谨慎对待。

（2）将固件放在 image 文件夹下，并修改 RAM_PATH 或者 FLASH_PATH 的路径，就可以下载客户自定义的固件。

4：工具使用说明

4.1：返回说明

程序运行中的返回值：

（1）STAGE：

返回 0：程序在运行中出错，错误信息会打印出来

返回 1：等待同步

返回 2：同步完成，下载中

返回 3：下载完成

（2）ESP_MAC：在 ESP 芯片出厂时，内部烧写的 MAC 地址

（3）BT_MAC：在 ESP 芯片出厂时，内部烧写的蓝牙 MAC 地址

（4）CUS_MAC：客户需要自定义 MAC 地址时，会在烧写完成后读出 CUS_MAC。如果客户没有自定义 MAC 地址，则 CUS_MAC 始终为 00:00:00:00:00:00

（5）CUS_MAC_STAGE：如果客户需要自定义 MAC 地址，则在完成烧写之后会返回一个状态。

返回 0：未烧写 MAC

返回 1：烧写 MAC 地址成功

返回 2：MAC 地址 CRC 校验失败

返回 3：写入 EFUSE 失败

客户可以根据返回的 STAGE 值，来判断该程序当前的运行状态，从而进行进一步的处理。

4.2：调用参数说明

该工具在调用的时候需要额外的输入参数，例如如下的操作：

```
p=subprocess.Popen("esp_cmm_download_tool_v1.0.4.0.exe 1,0", stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

工具后面跟的输入参数分为以下几种：

该版本工具无论传递参数是什么，都会打印 stage 3 字段。可以将 stage 3 当做工具运行结束标志。

（1）如果输入参数为“0,0”，既不下载 ram，也不烧写 mac 地址，则工具不会进行同步，因此无法读取 mac 地址，打印的 mac 地址皆为空；

（2）如果输入参数为“1,0”，下载 ram，但不烧写 mac 地址，工具会依次打印 stage 1-3，并输出相应的 mac 地址；

（3）如果输入参数为“0,1”，则烧录 mac 地址。该输入只有在下载 ram 之后操作才有效；

（4）如果输入参数为“1,1”，下载完 ram 之后，直接烧写 mac 地址；

（5）如果输入参数为“0,0,param3”（第三个参数为 mac 地址，mac 地址格式为 0x010203040506），则将用户输入的 mac 地址烧录进去。该输入只有在下载 ram 之后操作才有效，且只有前两位参数均为 0 时才有效。

4.3：结果演示

```
D:\FAE_puff\ESP_DOWNLOAD_TOOL\ESP_DOWNLOAD_TOOL_v1.1.0.1>esp_cmm_download_tool_v1.1.0.1.exe 1,0
CHIP_TYPE is:  ESP8266
BAUDRATE is:  460800
COMPORT is:  COM21
RAM_PATH is:  ./image/ESP8266_RF_TEST_BIN_V113_26m_20170518.bin
PO_NUMBER is:  c0392
ESP_WRITE_CUS_MAC_EN is:  0
ESP_DOWNLOAD_MODE is:  1
ESP_DOWNLOAD_TOOL_VER is:  v1.1.0.1

SYS_STAGE: 1
ESP_MAC: 00:00:00:00:00:00
CUS_MAC: 00:00:00:00:00:00
CUS_MAC_STAGE: 0
Connecting........_
esp connect success

SYS_STAGE: 2
Manufacturer: a1
Device: 4016
ESP_MAC: 5C:CF:7F:F5:E4:BE
CUS_MAC: 2C:3A:E8:08:11:98
CUS_MAC_STAGE: 1
Downloading... Plese wait...

Total download time used is: 2.24s
SYS_STAGE: 3
ESP_MAC: 5C:CF:7F:F5:E4:BE
CUS_MAC: 2C:3A:E8:08:11:98
CUS_MAC_STAGE: 1

D:\FAE_puff\ESP_DOWNLOAD_TOOL\ESP_DOWNLOAD_TOOL_v1.1.0.1>
```

图 4-1 工具运行成功示例

5：调用 ESP 下载插件流程示例

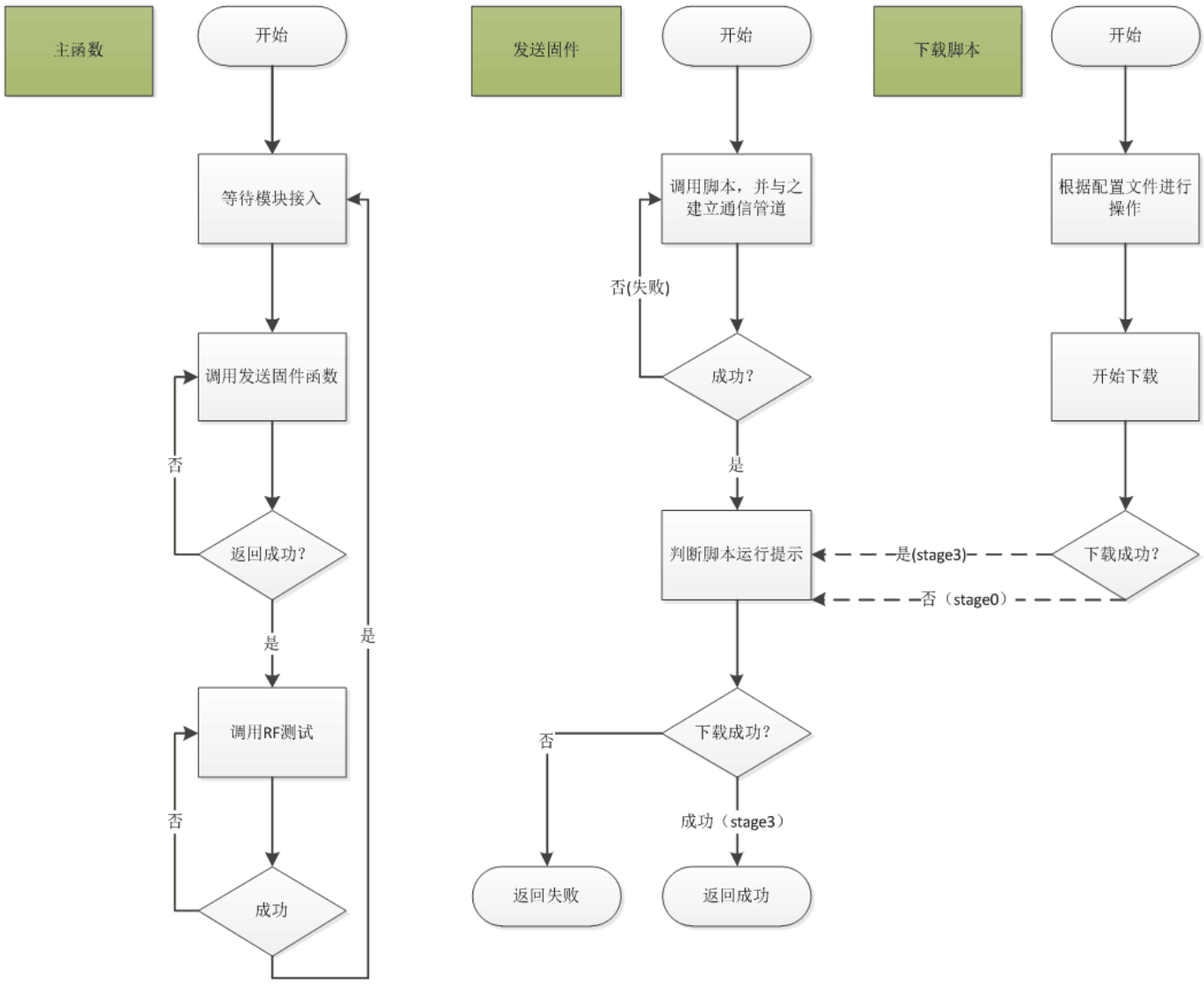


图 5-1 ESP 下载插件调用示例图

ESP 下载插件工具的调用主要分为三个主要部分：

（1）读取配置参数；

（2）根据配置参数，将对应的固件下载到对应的芯片中；

（3）完成下载后，在外部主程序中即可打开串口，并发送串口指令，测试设备的 RF 性能等。

附录：调用示例

在客户的应用工具中，可以直接调用该 exe 完成下载，下载成功后，即可发送命令进行对应的测试。

调用以下代码时，请根据实际存放地址，修改 exe 所在路径。可以使用相对路径调用 exe。

A. C++ 调用示例

```
#include <iostream>
#include <string>
#include   "windows.h "
#include   "shellapi.h "

using namespace std;

int main(){
      cout << "call esp download start" << endl;

      SECURITY_ATTRIBUTES sa          = {0};
   STARTUPINFO         si          = {0};
   PROCESS_INFORMATION pi          = {0};
   HANDLE              hPipeOutputRead  = NULL;
   HANDLE              hPipeOutputWrite = NULL;
   BOOL               bTest = 0;
   DWORD               dwNumberOfBytesRead = 0;
   CHAR               szBuffer[500];

   sa.nLength = sizeof(sa);
   sa.bInheritHandle = TRUE;
   sa.lpSecurityDescriptor = NULL;


   // Create pipe for standard output redirection.
   CreatePipe(&hPipeOutputRead,  // read handle
         &hPipeOutputWrite, // write handle
         &sa,      // security attributes
         0      // number of bytes reserved for pipe - 0 default
         );
```

```
// Make child process use hPipeOutputWrite as standard out,
si.cb = sizeof(si);
si.dwFlags     = STARTF_USESHOWWINDOW | STARTF_USESTDHANDLES;
si.wShowWindow = SW_HIDE;
si.hStdInput   = NULL;//hPipeInputRead;
si.hStdOutput  = hPipeOutputWrite;
si.hStdError   = hPipeOutputWrite;

CreateProcess (
   NULL, "D:\\hmj\\Desktop\\ESP_DOWNLOAD_TOOL_v1.0.0\\esp_cmm_download_tool_v1.0.0.exe",
   NULL, NULL,
   TRUE, 0,
   NULL, "D:\\hmj\\Desktop\\ESP_DOWNLOAD_TOOL_v1.0.0\\",
   &si, &pi);

// Now that handles have been inherited, close it to be safe.
CloseHandle(hPipeOutputWrite);

while(TRUE)
{
   bTest=ReadFile(hPipeOutputRead,      // handle of the read end of our pipe
            &szBuffer,           // address of buffer that receives data
            256,                 // number of bytes to read
            &dwNumberOfBytesRead, // address of number of bytes read
            NULL                 // non-overlapped.
            );

        if (!bTest){
            MessageBox(NULL, "esp download running over", "Test", MB_OK);
            break;
        }

        // do something with data.
        szBuffer[dwNumberOfBytesRead] = 0;  // null terminate
        cout << szBuffer <<endl;
}

// Wait for CONSPAWN to finish.
WaitForSingleObject (pi.hProcess, INFINITE);

// Close all remaining handles
CloseHandle (pi.hProcess);
CloseHandle (hPipeOutputRead);
```

```
    cout << "call esp download end" << endl;
    system("PAUSE");
}
```

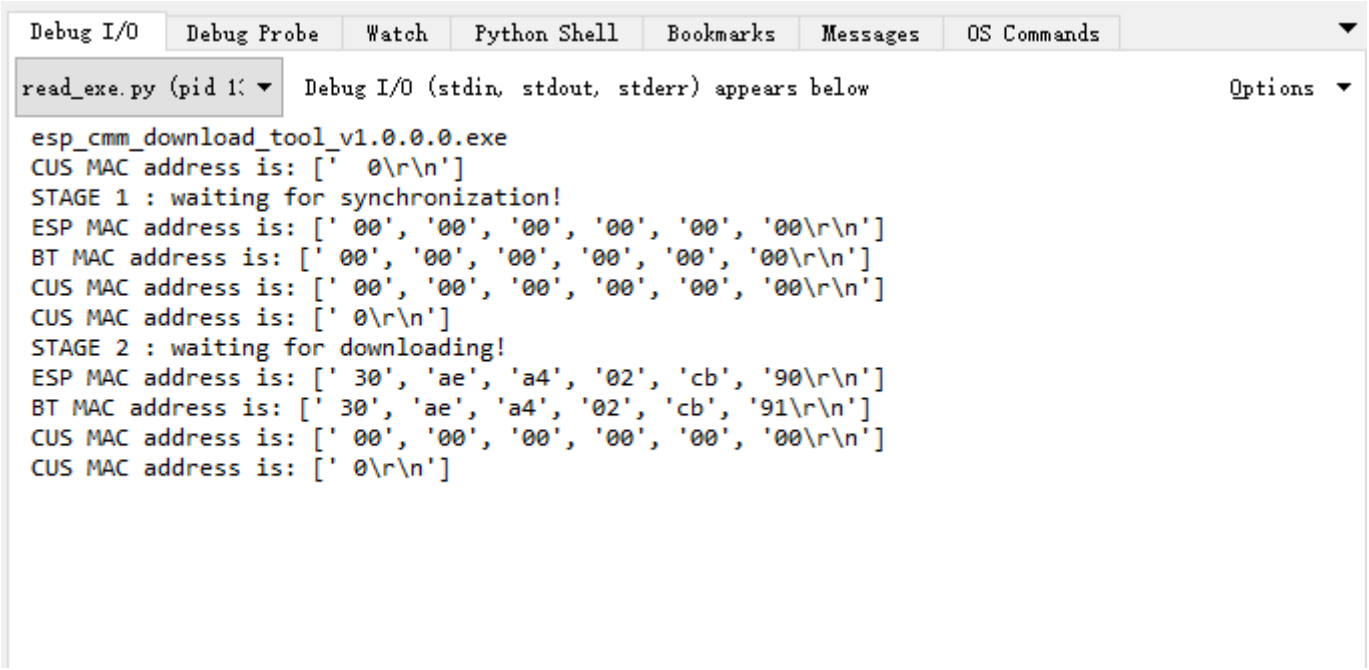
C++示例代码

B. PYTHON 调用示例

```
import os
import subprocess
import time

if __name__ == "__main__":

    exe_1 = "esp_cmm_download_tool_v0.8.3.exe" +  " 1,0"
    print exe_1
    try:
        p=subprocess.Popen(exe_1, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    except:
        print "ERROR!! exe name is not right!"
        exit()

    while(True):
        stdout = p.stdout.readline()
        if stdout != '':
            if(stdout.find("SYS_STAGE: 0") >= 0):
                print "ERROR happen"
                break
            elif(stdout.find("STAGE: 1") >= 0):
                print "STAGE 1 : waiting for synchronization!"
            elif(stdout.find("STAGE: 2") >= 0):
                print "STAGE 2 : waiting for downloading!"
            elif(stdout.find("STAGE: 3") >= 0):
                print "STAGE 3 : downloading ok!"
                break
            elif(stdout.find("ESP_MAC") >= 0):
                esp_mac = stdout.split(":")[1:]
                print "ESP MAC address is: %s" %esp_mac
            elif(stdout.find("BT_MAC") >= 0):
                bt_mac = stdout.split(":")[1:]
                print "BT MAC address is: %s" %bt_mac
```

```
        elif(stdout.find("CUS_MAC") >= 0):
            cus_mac = stdout.split(":")[1:]
            print "CUS MAC address is: %s" %cus_mac
    else:
        break
time.sleep(10)
```

PYTHON 示例代码



附图-1 Python 脚本执行结果