

**MT7681 IoT FAQ**

Version: 0.04

Release date: 2014-6-

© 2014 MediaTek Inc.

This document contains information that is proprietary to MediaTek Inc.

Unauthorized reproduction or disclosure of this information in whole or in part is strictly prohibited.

Specifications are subject to change without notice.

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Author | Description |
| 04.12.2014 | First v0.01 | Jinchuan | Initial draft for MT7681 IoT FAQ |
| 05.17.2014 | V0.02 | Jinchuan |  |
| 05.28.2014 | V0.03 | Jinchuan | Add APK compile Error in Android4.2 |
| 06.16.2014 | V0.04 | Jinchuan | Update “4.3 如何改变MAC地址” with Hex input  Add “4.6 XIP, OVL 机制” |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[1 Source Code Compile 5](#_Toc390717269)

[1.1 How to Setup AndeSight SDK 5](#_Toc390717270)

[1.2 How to create new project in AndeSight / How to import MTK7681 SDK to AndeSight 5](#_Toc390717271)

[1.3 How to compile source code for Recovery FW , Sta FW, AP FW 6](#_Toc390717272)

[1.4 SDK编译后生成ap，sta，recovery，all等文件，哪个文件才是我最终要烧录的， 6](#_Toc390717273)

[1.5 Compile Source Code fail -- .BSS is not within region SRAM 6](#_Toc390717274)

[2 FW Upgrade 6](#_Toc390717275)

[2.1 FW upgrade method 6](#_Toc390717276)

[2.2 How to change AP mode and Station Mode 6](#_Toc390717277)

[2.3 Why there is a Recovery Mode 7](#_Toc390717278)

[2.4 How to set recovery mode duration 7](#_Toc390717279)

[3 APK Install and usage 8](#_Toc390717280)

[3.1 手机安装IoTManager\_v0.94\_1\_android4.0.apk，显示是“解析包时出现问题” 8](#_Toc390717281)

[3.2 将APK SourceCode 放到Android Codebase 4.2中编译会失败 8](#_Toc390717282)

[4 System Coding 8](#_Toc390717283)

[4.1 Printf\_High() /DBGPRINTF\_HIGH() 输出LOG的开关 8](#_Toc390717284)

[4.2 flash分区里reserved的区域有没有其他特殊用途？可不可以拿来保存一些数据 9](#_Toc390717285)

[4.3 如何改变MAC地址 9](#_Toc390717286)

[4.4 打开 CFG\_SUPPORT\_TCPIP 宏使用AT命令的TCP功能。编译时出现错误 10](#_Toc390717287)

[4.5 为什么使用IoT\_gpio\_output() 设定某个GPIO为1后，一会后莫名其妙又被拉低 10](#_Toc390717288)

[4.6 XIP, Overlay机制 10](#_Toc390717289)

[5 Connection Development 11](#_Toc390717290)

[5.1 MT7681 Support mode 和Bandwidth 11](#_Toc390717291)

[5.2 一般 C程式有entry point,即main(), 請問一個IoT application 的entry point在哪裡? 11](#_Toc390717292)

[5.3 How to get mt7681 MAC address 12](#_Toc390717293)

[5.4 在sta模式下连接ap，但是我要怎么来判断连接成功与否？？？ 12](#_Toc390717294)

[5.5 如何自动获取IP，是通过uip中的dhcpc相关函数吗？？？ 12](#_Toc390717295)

[5.6 如何進行SW reset. 將IP/SSID 等參數 reset成出廠設定值 13](#_Toc390717296)

[5.7 现在的1.10版本好像没有mem\_alloc及mem\_free函数 13](#_Toc390717297)

[5.8 结构体\_WIFI\_STATE的说明性文档，每个状态代表什么意义 14](#_Toc390717298)

[5.9 Smart Connection的station config数据结构 14](#_Toc390717299)

[5.10 Smart Connection的(station config)数据存取及清除 14](#_Toc390717300)

[5.11 使用宏 CFG\_SUPPORT\_MTK\_SMNT=1  来控制是否使用 MTK的smart connection 15](#_Toc390717301)

[5.12 How to get PMK 15](#_Toc390717302)

[6 Application Development 16](#_Toc390717303)

[6.1 How to send TCP/UDP packet 16](#_Toc390717304)

[6.2 IoT application內要scan AP, 連到AP, 建TCP + SSL ...等,要參考哪些 APIs document? 16](#_Toc390717305)

[6.3 数据从服务器转发到7681，数据包格式是怎样的，经过解包之后是怎样的数据类型？ 17](#_Toc390717306)

[6.4 SDK中有TCP和UDP两种方式， UDP/TCP是什么情况下使用的呢？ 17](#_Toc390717307)

[6.5 MT7681透过AP Router与Internet Server，请问这个Internet Server做"转发"作用外，还有哪些作用呢？ 17](#_Toc390717308)

[6.6 是否有获取网络时间的协议Support SNTP, NTP 17](#_Toc390717309)

[6.7 Set up the TCP connection on MT7681 17](#_Toc390717310)

[6.8 Cannot use AT Cmd “Netmode ,Channel, SoftAPConf and TCP\_Connect ”… 19](#_Toc390717311)

# Source Code Compile

## How to Setup AndeSight SDK

Refer to the document：MTK\_AndesToolChains\_Usage\_v0.0\*\_\*\*\*\*\*\*\*.pdf

Q1：AndeSight IDE我download的版本是：Andestech\AndeSight201MCU   ， 你们文档中提到要用版本C:\Andestech\AndeSight14\，是不是有问题？

A1：没有问题， 只是我们在准备文档的时候 还是用的1.4 的AndeSight，

          您download的 是最新的 2.01版本， 我们有实验过，配置方法同1.4一样， 所以文档没有再做更新

Q2：按照文档《MTK\_AndesToolChains\_Usage\_20140212.pdf〉的方法用MTK的“nds32le‐elf‐newlib‐V2j”

替换了原始的toolchain，启动后马上退出，没有打开窗口。

A2:   这里先了解下，您实验的操作系统是win7 还是XP ？

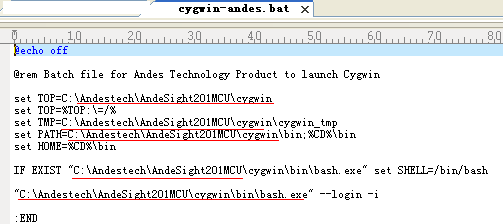
         在XP 32bit OS上 安装了AndeSight201MCU 是OK 的， (有人员在Win7 64bitOS 中也有安装成功)

使用默认的安装路径C:\Andestech\AndeSight201MCU\toolchains

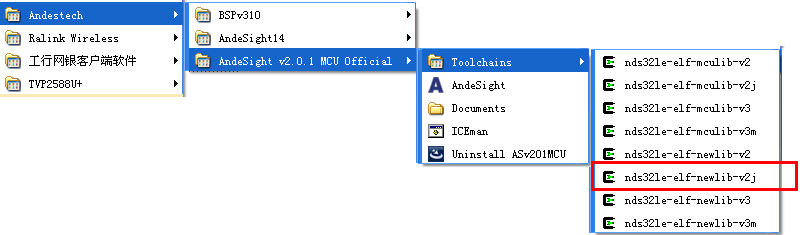
         下图是我修改后的cygwin-andes.bat 档，如果您的安装路径  以及Toolchain 替换路径 同我一样

         可以将如下路径的dat 中 的红色底线位置改成和我一样。

         C:\Andestech\AndeSight201MCU\toolchains\nds32le-elf-newlib-v2j



         然后点击 红色方框的链接 启动即可。



## How to create new project in AndeSight / How to import MTK7681 SDK to AndeSight

AndeSight SDK 我现在也只是 使用它的Cygwin 环境, 并未使用它来 Create new project 。

Create new project  我还是通过source insight 这些tool来创建

## How to compile source code for Recovery FW , Sta FW, AP FW

make b=0 clean;make b=0      -> create recovery bin

make b=1 clean;make b=1      -> create sta bin

make b=2 clean;make b=2      -> create ap bin

## SDK编译后生成ap，sta，recovery，all等文件，哪个文件才是我最终要烧录的，

MT7681\_IoT\_Package\_v1.10\Src 中Readme-ForEachBinDescript.xlsx 文件会介绍产生出的binary关系

编译后 会 生成:  MT7681\_sta.bin， MT7681\_recovery\_old.bin， MT7681\_ap.bin ，

这些文件会再加上(loader\_0322\_94973.bin，MT7681E2\_EEPROM layout\_20140330.bin )

按照 Flash layout 中指定的位置  生成MT7681\_all.bin

MT7681\_all.bin  就是我们通过flash 烧录器 进行整个flash 烧录的程序

编译后还会生成:   MT7681\_recovery\_header.bin,   MT7681\_sta\_header.bin, MT7681\_ap\_header.bin

另外Package中 有我手动对loader和Recovery的binary加上header，

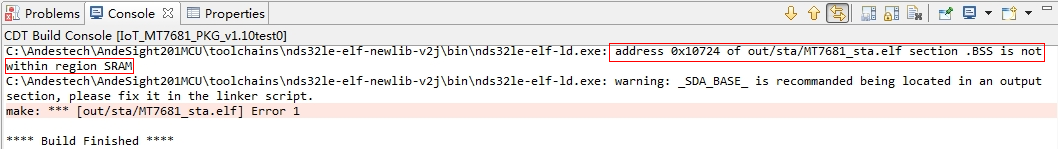
生成的loader\_header\_0322\_94973.bin,   MT7681\_recovery\_header.bin

这些文件是 用于Uart FW upgrade 的，

具体Uart升级方法，以及 \*\*\*\_Header\_\*\*.bin   与\*\*\*.bin  的 差异

在MT7681\_IoT\_Package\_v1.10\Doc\MT7681\_Uart\_Firwmare\_Upgrade\_v0.0\*\_\*\*\*\*\*\*\*\*.pdf  有描述

## Compile Source Code fail -- .BSS is not within region SRAM

这个表明当前程序  已经超过ram的大小

以Package v1.10  的SDK来看

在默认的flag\_sta.mk (for Station mode) 功能配置下，可供使用的memory 大小为 11KB

在默认的flag\_ap.mk  (for AP  mode)功能配置下，可供使用的memory 大小为 13KB

# FW Upgrade

## FW upgrade method

1: FW upgrade by Flash Writer

2: FW upgrade by Uart

Just like descript in “**1.4**” , 也请先阅读MT7681\_Uart\_Firwmare\_Upgrade\_v0.0\*\_\*\*\*\*\*\*\*\*.pdf

* Not support FOTA yet，客户可通过现有API开发

## How to change AP mode and Station Mode

通过AT#FLASH -s98305 –v\* 命令，修改Flash Offset: 0x18001的值, 再关开机即可实现AP/STA切换

[0x18001]=[0x00]  🡪 表示Boot as STA mode,

[0x18001]=[0x01]  🡪 表示Boot as AP mode

Example: AP Mode的切换比较简单， 就是修改Flash Offset: 0x18001 的值为1

Step1： 通过Flash Writer 烧录MT7681\_all\_v1.10.bin  到Flash

Step2： 开机后显示 如下信息，表示进入到STA Mode

        ==> Recovery Mode

<== RecoveryMode

(-)

SM=0, Sub=0

SM=1, Sub=0

[WTask]9811

Step3:     可通过 AT#Flash 命令读取0x18001 位置的BootIndex值 double Check

               AT#FLASH -r98305

                 [0x18001]=[0x00]  🡪 表示Boot as STA mode, 如为[0x18001]=[0x01]表示Boot as AP mode

Step4：  通过 AT#Flash 命令修改0x18001 位置的BootIndex值 为1

                     AT#FLASH -s98305 -v1

Step5：  7681重新上电后，会以AP mode方式启动，

SSID name为“MT7681\_Softap”, 只Support Open Mode

==> Recovery Mode

<== RecoveryMode

(-)

APStartUp ok

Start AP ...

[WTask]9318

[WTask]14322

如果此时有手机连接上来, 会显示如下连线信息

Assoc request sanity success

i = 5, j = 0

client ip addr: 192.168.81.2

[WTask]39372

## Why there is a Recovery Mode

主要目的是用于Uart FW Upgrade 和Production Calibration

上电/重启后会自动进入到Recovery Mode, 并停留4s (default), 在这4s内 通过Uart 输入

AT#UpdateFW 🡪 会启动Uart FW upgrade Process, 通过xmodem 进行FW 升级

AT#ATECAL -S 🡪 会进入 Calibration Mode, 进行Tx/Rx 的Calibration

我们需考量到STA FW 如果更新失败，系统如何恢复，此时Recovery Mode的存在就比较关键

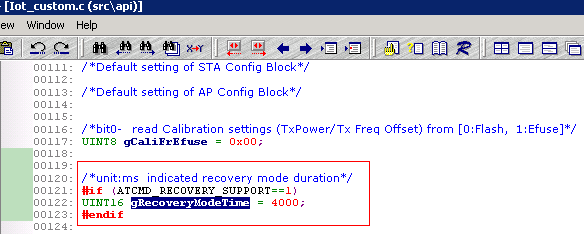
Recovery Mode 的4s等待时间，是考虑到目前HDK  v1.0  不能接上Uart Rx开机，所以上电后，暂时保留足够的时间让  开发者可以输入AT#UpdateFW 等命令

这4s的时间是可以调整的， 如2.4

## How to set recovery mode duration

现在recovery 模式下，如果不做任何操作，会在4s后 自动跳出recovery 模式

这个4s 的时间  在v1.2 SDK 中可以由 通过设定变量  来控制



# APK Install and usage

## 手机安装IoTManager\_v0.94\_1\_android4.0.apk，显示是“解析包时出现问题”

IoTManager\_v0.94\_1\_android4.0.apk 是for  Android 4.0 及以上的手机 都可以

 很可能出现问题的手机不是android4.0及以上的版本

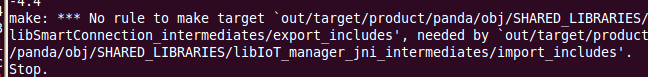
APK 的SourceCode 也有放在Package v1.10中， 您也可自己编译出对应的APK for demo

## 将APK SourceCode 放到Android Codebase 4.2中编译会失败

我们APK中没有release SmartConnection.lib的source code，

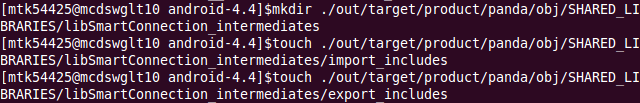
所以在android4.2以上的codebase中编译可能存在以下问题：

**1：提示没有export\_includes**

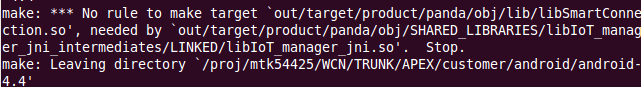


         这是因为我们没有build SmartConnection的lib所以不会存在该临时文件夹。

解决办法为手动创建两个文件，如下：



**2：提示没有libSmartConnection.so文件**



出现该问题后请将IoTManager/lib/libSmartConnection.so copy到对应目录中，如下：

cid:image005.png@01CF7AB9.4B0F5C30

# System Coding

## Printf\_High() /DBGPRINTF\_HIGH() 输出LOG的开关

V1.2 SDK中，会在Iot\_custom.c  加上如下Boolean全局变量，来控制是否打印LOG

cid:image001.png@01CF6973.A2DF4350

## flash分区里reserved的区域有没有其他特殊用途？可不可以拿来保存一些数据



Reserved 主要用于 各区域之间的有效隔离

          如果您在flash中 需要额外的使用空间，  建议可以用0x1C000 这个位置的reserved,  total 12KB

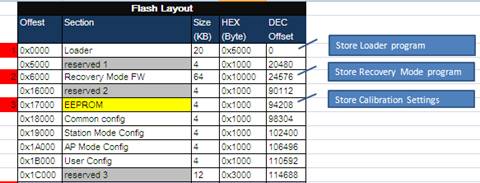
          只是需注意  不要越界到0x1F000    即STA FW 的位置。

 还有，我们Chip 能Access到的Flash Size是1MB,   目前Flash  Layout 中定义的是512KB 的排列

          如果您的产品  采用1MB Flash ，那么 [0x8000 ~ 0xFFFFF]   这个区域(512KB) 也是可以使用的.

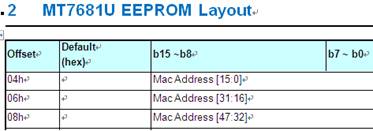
## 如何改变MAC地址

默认情况下，Mac地址存在Flash 的EEPROM 区域，



EEPROM区域的Layout  在MT7681U-EEPROM Content \_\*\*\*.docx文档中有描述

其中 EEPROM 中的 0x04~0x09 的位置 是 存放MAC Address的



读取Flash MAC 地址的AT Command 有：

AT#FLASH –r94212          会LOG输出 [0x17004]=[0x00]

AT#FLASH –r94213          会LOG输出 [0x17005]=[0x0c]

AT#FLASH –r94214          会LOG输出 [0x17006]=[0x43]

AT#FLASH –r94215          会LOG输出 [0x17007]=[0x26]

AT#FLASH –r94216          会LOG输出 [0x17008]=[0x60]

AT#FLASH –r94217          会LOG输出 [0x17009]=[0x40]

在v1.30 SDK开始，可以支持: AT#FLASH –r0x17004 十六进制的输入

设置Flash MAC 地址的AT Command 有：

AT#FLASH –s94212   -vX      x为设定值

AT#FLASH –s94213   -vX      x为设定值

AT#FLASH –s94214   -vX      x为设定值

AT#FLASH –s94215   -vX      x为设定值

AT#FLASH –s94216   -vX      x为设定值

AT#FLASH –s94217   -vX      x为设定值

上面Command 的-r, -s, -v 的参数，都是**十进制**

在v1.30 SDK开始，可以支持: AT#FLASH -s0x17004 -v0x0c 十六进制的输入

设定完后，重新上电/Reboot，系统会使用新的MAC地址, 并赋值给gCurrentAddress。

## 打开 CFG\_SUPPORT\_TCPIP 宏使用AT命令的TCP功能。编译时出现错误

因为我们有开放出uIP的SourceCode

ATCMD\_TCPIP\_SUPPORT 就没有再维护了，所以目前是把它给关掉的

## 为什么使用IoT\_gpio\_output() 设定某个GPIO为1后，一会后莫名其妙又被拉低

原因很可能是：有地方调用到IoT\_gpio\_input()的缘故

IoT\_gpio\_input() à

如果当前GPIOx是Output mode，则会将其设定为Input mode，此时得到的input值会改成0

          如果当前GPIOx是Input  mode，此时得到的input值 就是外面电路给到该GPIO的状态

在v1.2 SDK ,会再提供API: IoT\_gpio\_read (INT32 GPIONum, UINT8 \*Val, UINT8 \*Polarity)

可以只对gpio 做read 的操作，而不改变gpio状态/mode

## XIP, Overlay机制

目前有些客户的客制化代码 比较多，可能超过了 剩余Ram的可使用范围，

所以这里介绍MTK对该项目 有在使用的两种机制：XIP, OVL 机制，

大概的流程如下，供参考：

* 定义为XIP的Function，会直接在Flash上执行，

各个function不会有资源重叠，所以比较不会有资源冲突的情况

XIP function定义只需在 function声明的地方，加上 **XIP\_ATTRIBUTE(“.xipsec0”)** 即可

但定义XIP 还是会有一些考量, 比如如下情形，最好不要定义成XIP:

1:   对flash 进行r/w 的function

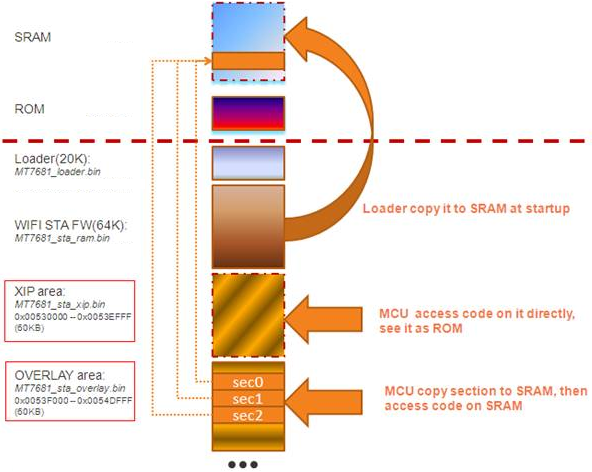
2:   对实时性/效能 要求 非常高的function

* 定义为Overlay的Function,   因为可能是多个function  会用到同一块RAM，  就很容易出现冲突

所以Overlay的function，我们是先逐一做function review，

确保这个function不会和其他的overlay function 同时用到，或相互调用，

越到后期这种机制的维护会越困难，所以这里做个大概介绍，不建议客户使用该方法 。



# Connection Development

## MT7681 Support mode 和Bandwidth

在Package v1.10中，Channel选择的API:   IoT\_Cmd\_Set\_Channel

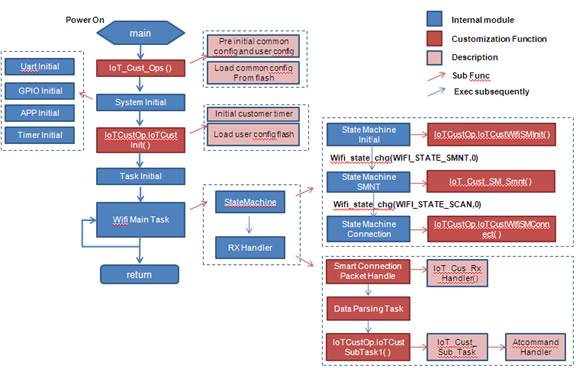
对于Mode 和Bandwidth：目前Support 80.211 b/g  ，BW\_20M

## 一般 C程式有entry point,即main(), 請問一個IoT application 的entry point在哪裡?

是的，我们也有main(),    只是我们未完全开放main  function

但我们有 创建 hook function  以便客户可以开发自己的程序，大部分集中在iot\_custom.c中

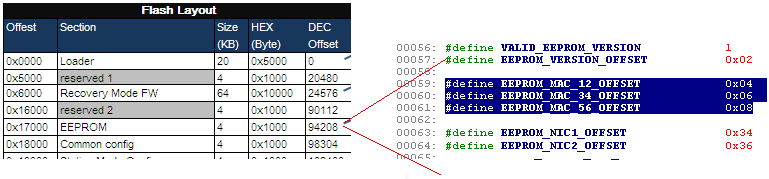
可参考MT7681\_IoT\_WIFI\_Firmware\_Programming\_Guide\_v0.0\*.pdf的Section 5 “Customer Hook Function”



## How to get mt7681 MAC address

The MAC address of 7681 is stored on Flash offset [0x17004~0x17009],

and this MAC shall be set to global parameter gCurrentAddress[MAC\_ADDR\_LEN]



## 在sta模式下连接ap，但是我要怎么来判断连接成功与否？？？

A1：   目前系统会有Initial-> Smart Connection -> Scan 🡪 Auth-> Assoc 🡪 4 way -> Connected的过程，

             这个State 会 设定在pIoTMlme->CurrentWifiState 这个变量中。 以此来控制状态机的flow

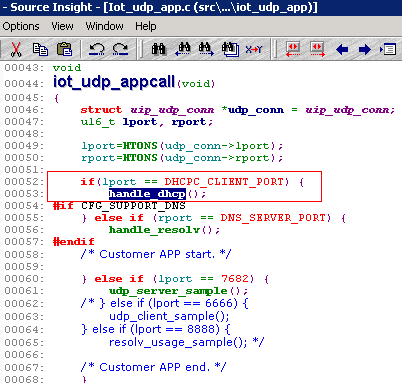
             如果pIoTMlme->CurrentWifiState =  WIFI\_STATE\_CONNED  (6)，即表示与AP的连线成功

                   下一步会开始获取IP

## 如何自动获取IP，是通过uip中的dhcpc相关函数吗？？？

A2：进入到Connected的State后，会开始调用 tcpip\_periodic\_timer 进行TCP/UDP 的相关数据交互

这里会开始做DHCPC  以获取IP ， 具体文件和function 如下图



## 如何進行SW reset. 將IP/SSID 等參數 reset成出廠設定值

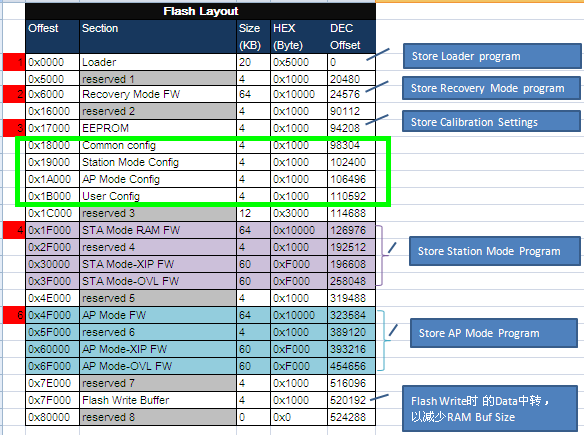
目前的Code 已经有提供AT#Default的功能

对于参数设定 我们会在Flash的如下绿色 方框区域 来存储，

在MT7681\_IoT\_WIFI\_Firmware\_Programming\_Guide\_v0.0\*.pdf 的Section “11 FLASH PARTITIONS “

有对 里面的每个Byte做定义

另外，在iot\_customer.c 中，有common cfg,  AP cfg,  user cfg 的 参数load, reset 的函数实现，这部分有开放出来 对于station cfg ， 也有reset\_sta\_cfg（） 来做初始化



## 现在的1.10版本好像没有mem\_alloc及mem\_free函数

因为没有OS的support,   我们并没有完整的mem\_alloc及mem\_free  的API

目前的malloc, free函数 有限制 不能做嵌套的调用，即

malloc(A) -> free(A) 🡪 malloc(B) 🡪 free(B) === OK

malloc(A) 🡪 malloc(B) -> free(A) 🡪 free(B) === NG

所以该API 现在只在很少的地方用到，但考虑到MTK和客户 都会开发代码，建议不要使用该API

目前大部分情况 大的buffer 都是用全局/局部 数组 来申请！

所以在程序设计时，我们对数组的size 会多做一些考量。

## 结构体\_WIFI\_STATE的说明性文档，每个状态代表什么意义

    typedef enum **\_WIFI\_STATE**{

    WIFI\_STATE\_INIT = 0, -> //进行wifi state machine的初始化，会去读flash 中存取的sta cfg

设定，如果已读到有效地SSID, Password, PMK, AuthMode，则直接

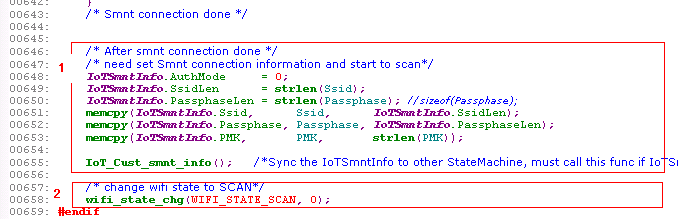
跳到SCAN阶段，否则到SMNT阶段  
    WIFI\_STATE\_SMTCNT,    -> //执行smart connection,  收集sta cfg 中的设定  
    WIFI\_STATE\_SCAN,      -> //通过 Init 或Smnt 阶段得到的sta cfg, 去scan特定ssid ，

然后固定channel  
    WIFI\_STATE\_AUTH,      -> //向该ssid AP, 发送Auth Request frame并获取Auth Response  
    WIFI\_STATE\_ASSOC,     -> //向该ssid AP, 发送Assoc Request frame并获取Assoc Response  
    WIFI\_STATE\_4WAY,     -> //与该ssid AP 进行4-way handshark, 产生GTK,PTK  
    WIFI\_STATE\_CONNED    -> //会开始出发 dhcpc，获取AP分配的IP  
    }WIFI\_STATE;

## Smart Connection的station config数据结构

如果不使用MTK的smart connection

在IoT\_Cust\_SM\_Smnt中有 如下处理， 可填写IoTSmntInfo 结构体，并call wifi\_state\_chg到SCAN状态 , 此后系统会自动完成 Scan->Auth -> Assoc 。。。。。 的过程



## Smart Connection的(station config)数据存取及清除

有关cfg 存放每家客户 也可能有自己的做法

我们默认的做法是:  存储到Flash中的SSID/Password/PMK…    在绝大多数情况下 是不允许被改掉的

* 只有在 处理AT cmd: Default ，  及处理Data Cmd: Offline 时才清除

清除调用function:    reset\_sta\_cfg ()

* 或在使用原有flash中的值，无法连接AP,   重新做Smart Connection，连接到新的AP 获得IP时 才会覆盖旧的flash sta cfg值

存储调用function有:    在smnt阶段先call    IoT\_Cust\_smnt\_info(VOID),        到**获得IP**后会再由ws\_got\_ip()  call到store\_sta\_cfg(VOID)



## 使用宏 CFG\_SUPPORT\_MTK\_SMNT=1  来控制是否使用 MTK的smart connection

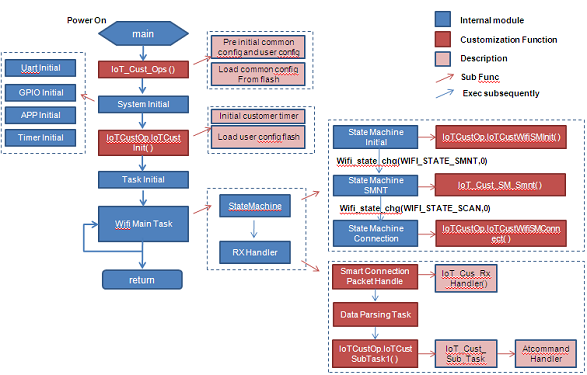
    如果客户有自己的Smart Connection，就可将该宏 值设为0

    在MT7681\_IoT\_WIFI\_Firmware\_Programming\_Guide\_v0.05.pdf 中， 有列出当前Smart connection开放出的对应callback，

    其中IoT\_Cus\_Rx\_Handler() 可看到每一包Rx Packet的内容, 并可对其做进一步处理

    IoT\_Cut\_SM\_Smnt() 是 smart connection的 状态机。

结合代码可做进一步了解



## How to get PMK

PMK 会由SSID, Passphase 通过SHA1 算法  计算得到

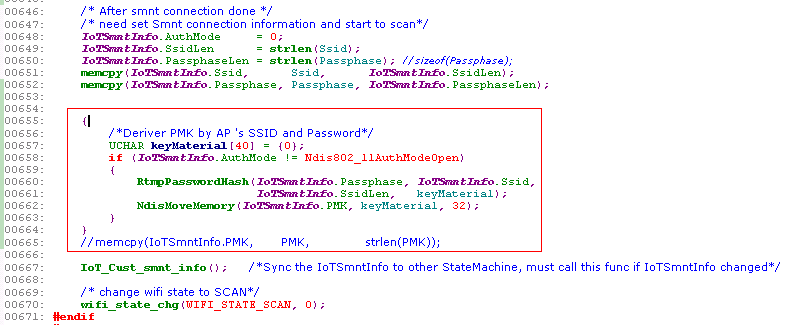
我们之前试验7681  软件来计算该值，但会花费近6s的时间

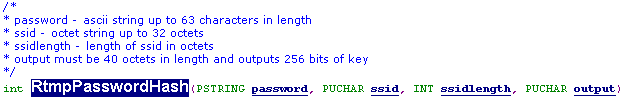
所以我们有修改smart connection的做法，

让手机端先计算好PMK， 然后通过smart connection将SSID, Passphase, PMK, AuthMode传递出来。

这样7681 就不用再计算PMK ，节省时间， 所以后面7681 是将软件计算PMK的function给拿掉

如果您需要使用，可以在IoT\_Cust\_SM\_Smnt（）中 调用API:     RtmpPasswordHash()





# Application Development

## How to send TCP/UDP packet

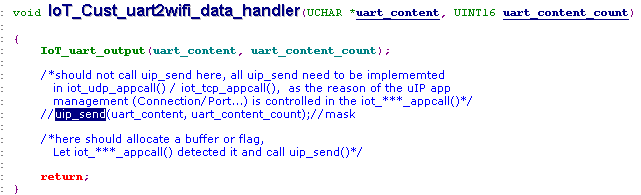
UDP，TCP app的开发 包括connection建立和 packet发送

都应该在Tcpip\_main.c à uip\_process()🡪 UIP\_APPCALL / UIP\_UDP\_APPCALL中进行,

目前uIP 完成Code是有开出来的，可以进行自定义的开发

在其他地方 (AT cmd handler中) 直接调用 udp\_send()发送Packet 会失败，

因为直接调用该function，不会指定具体发给哪个connection.



## IoT application內要scan AP, 連到AP, 建TCP + SSL ...等,要參考哪些 APIs document?

MTK代码中，有Initial-> Smart Connection-> Scan -> Auth-> Assoc-> 4 way -> DHCPC的过程，

手机上安装的APK 会 发送出SSID, Password, PMK，MT7681在Smart Connect阶段会获取这些信息

然后依此来 Scan 对应的AP,并进行连线 和从AP获得IP .

前面这些基本很少客制化，所以未开发出对应的API

 获得IP后，可以使用现有的uip 进行 TCP/UDP Connection的建立，

这部分有SourceCode 和Sample Code开放出来，他们的.c / .h 档有放在如下目录

 IoT\_MT7681\_PKG\cust\tcpip\,     IoT\_MT7681\_PKG\src\tcpip\

您可参考Package中的 文档 **MT7681\_TCP\_IP\_\*\*\*\*\*\*\*\_ToCust.pdf**

并结合这些Source File 来了解

## 数据从服务器转发到7681，数据包格式是怎样的，经过解包之后是怎样的数据类型？

可参考Release Package v1.10 的  IoT\_Control\_Protocol\_v0.2.pdf  了解数据包的具体格式

         解包后的数据  会在iot\_parse.c 的IoT\_process\_app\_packet() 中做进一步处理

         客户也可定义直接的格式

## SDK中有TCP和UDP两种方式， UDP/TCP是什么情况下使用的呢？

是的会有两种方式，不过这些方式都可以是客户来确定的

        有关TCP，UDP 的API和Source Code 我们并未采用标准 socket，而使用uip

主要原因是当前MT7681 并未采用multi task或复杂的OS

这部分有SourceCode 和Sample Code开放出来，它们的.c / .h 档有放在如下目录

 IoT\_MT7681\_PKG\cust\tcpip\,     IoT\_MT7681\_PKG\src\tcpip\

您可参考Package v1.10 中的 文档 MT7681\_TCP\_IP\_\*\*\*\*\*\*\*\_ToCust.pdf

结合这些Source File 来了解具体的使用

## MT7681透过AP Router与Internet Server，请问这个Internet Server做"转发"作用外，还有哪些作用呢？

同4.3一样，Internet Server实际也是客户来确定的

它可以做简单的转发，也可以做其他更多的事情，

## 是否有获取网络时间的协议Support SNTP, NTP

应该需要SNTP, NTP 协议，在MTK的当前代码中并未包含这些 比较上层的协议

## Set up the TCP connection on MT7681

Behavior example:

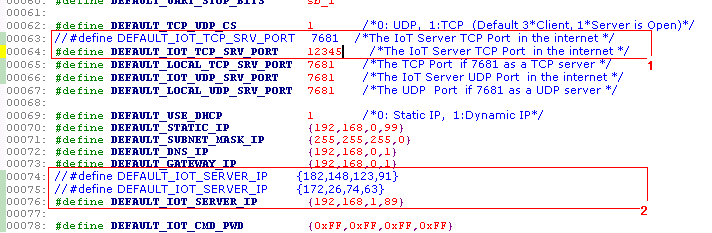
          (1) Connection Type: TCP on port 12345 with data payload "0" or "1" only.

          (2) Send data to MT7681 to Pull GPIO high/Low

相关修改如下：

**Step1:**

在iot\_custom.c 中有如下default 定义

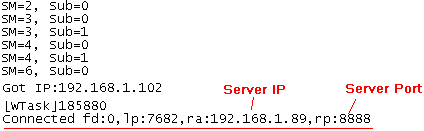


首先 修改 方框1 的Server Port 为12345,    然后修改方框2的Server IP

修改方框2的Server IP  最后会被   赋值给**IoTpAd.ComCfg.IoT\_ServeIP**

这样在7681上电 获取IP address后，会与该TCP server建立连接

正常的话，LOG中会有如下提示：

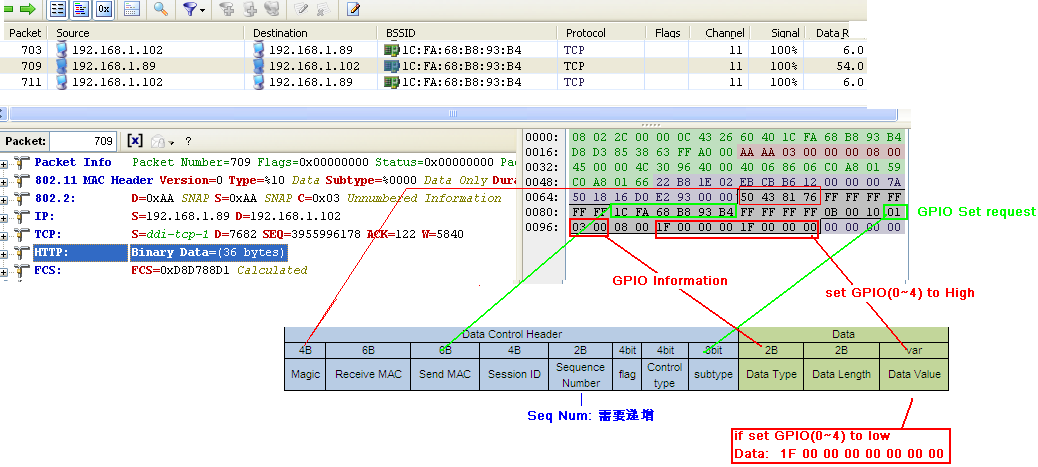


**Step2:**

TCP Connection建立连接后，TCP Server就可以 发送Data Command来控制 7681的GPIO了

Data Command 的Protocol  可以参考 Package v1.10 的文档: **IoT\_Control\_Protocol\_v0.\*.pdf**

这里有个TCP packet 包含Data Cmd 的实例 (Server  向7681 发送GPIO set 的command)



## Cannot use AT Cmd “Netmode ,Channel, SoftAPConf and TCP\_Connect ”…

Tcp\_Connect命令等TCP\_IP配置的条件是？

STA模式smartconnection配置后应该默认开启的相应的TCP\_IP进程。

Tcp\_Connect命令等TCP\_IP配置命令是否还能再配置？

     A3-1:   SoftAPConf命令 有在代码中被CONFIG\_SOFTAP 宏给包起来，  该宏只在flag\_ap.mk 中指定上，

所以SoftAPConf命令  只在SoftAP mode才会enable起来

     A3-2:   因为有开放出所有的UIP source code , 所以AT cmd中Tcp\_Connect命令  现在是默认disable的，