



MT7681 IoT FAQ

Version: 0.05
Release date: 2014-7-

© 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”
07.09.2014	V0.05	Jinchuan	Enable TCP Tx ReTransmit, and Http Client Update: 5.11 使用宏 CFG_SUPPORT_MTK_SMNT=1 来控制 是否使用 MTK 的 smart connection Add: 5.12 Wifi State Machine Flow. Add: 6.9 Enable TCP Tx ReTransmit, and Http Client Add: 4.6 HW Timer1 Add: Capter7: Interface Customization 7.1 PWM Level 7.2 Set UartTx as Interrupt Mode or Poll mode

Contents

1	Source Code Compile	5
1.1	How to Setup AndeSight SDK.....	5
1.2	How to create new project in AndeSight / How to import MTK7681 SDK to AndeSight.....	5
1.3	How to compile source code for Recovery FW , Sta FW, AP FW	6
1.4	Ap/Sta/recovery/all.bin are created after compile done , which binary we can use.....	6
1.5	Compile Source Code fail -- .BSS is not within region SRAM	6
2	FW Upgrade.....	6
2.1	FW upgrade method.....	6
2.2	How to change AP mode and Station Mode.....	6
2.3	Why there is a Recovery Mode.....	7
2.4	How to set recovery mode duration.....	7
3	APK Install and usage	8
3.1	Install “IoTManager_v0.94_1_android4.0.apk” , show “解析包时出现问题”	8
3.2	Compile APK Source Code failure in Android Codebase 4.2	8
4	System Coding	8
4.1	Printf_High() /DBGPRINTF_HIGH() 输出 LOG 的开关	8
4.2	The usage of “reserved” region in flash ? is possible to store other data in it.....	9
4.3	How to change MAC Address	9
4.4	Open Macro:CFG_SUPPORT_TCPIP, Compile Error	10
4.5	XIP, Overlay Mechanism	10
4.6	HW timer1 EINT Freq Adjustment	11
5	Connection Development.....	11
5.1	MT7681 Support mode 和 Bandwidth	11
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
5.3	How to get mt7681 MAC address.....	12

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
5.5	如何自动获取 IP，是通过 uip 中的 dhcpc 相关函数吗？？？	12
5.6	How to reset IP/SSID and other parameter to default	13
5.7	There is no “mem_alloc, mem_free” API in SDK v1.10.	13
5.8	Structure _WIFI_STATE Introduction，The means for each state	14
5.9	Smart Connection 的 station config 数据结构	14
5.10	Smart Connection 的(station config)数据存取及清除	14
5.11	使用宏 CFG_SUPPORT_MTK_SMNT=1 来控制是否使用 MTK 的 smart connection	15
5.12	Wifi Connection State Machine Flow	16
5.13	How to get PMK	17
6	Application Development	17
6.1	How to send TCP/UDP packet.	17
6.2	IoT application 內要 scan AP, 連到 AP, 建 TCP + SSL ...等,要參考哪些 APIs document?	18
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？	18
6.4	SDK 中有 TCP 和 UDP 两种方式，UDP/TCP 是什么情况下使用的呢？	18
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？	18
6.6	是否有获取网络时间的协议 Support SNTP, NTP	18
6.7	Set up the TCP connection on MT7681	19
6.8	Cannot use AT Cmd “Netmode ,Channel, SoftAPConf and TCP_Connect”...	20
6.9	Enable TCP Tx ReTransmit, and Http Client	20
7	Interface Customization	20
7.1	Set UartTx as Interrupt Mode or Poll mode	20
7.2	Set PWM Level	20
7.3	Why GPIO-x set Back to Low，after use IoT_gpio_output() to set GPIO-x to High	21

1 SOURCE CODE COMPILE

1.1 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

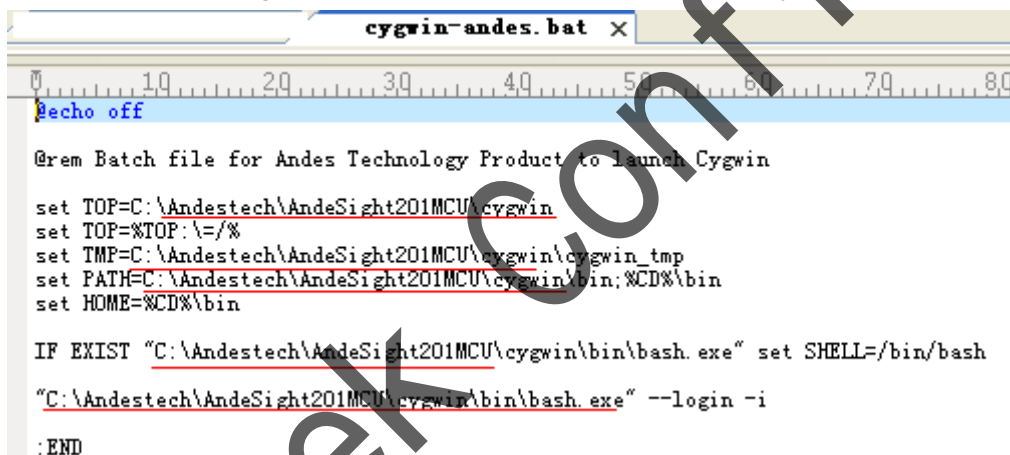
Q2: 按照文档《MTK_AndesToolChains_Usage_20140212.pdf》的方法用 MTK 的“nds32le-elf-newlib-V2j”替换了原始的 toolchain, 启动后马上退出, 没有打开窗口。

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 bat 中 的红色底线位置改成和我一样。

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



```
@rem Batch file for Andes Technology Product to launch Cygwin

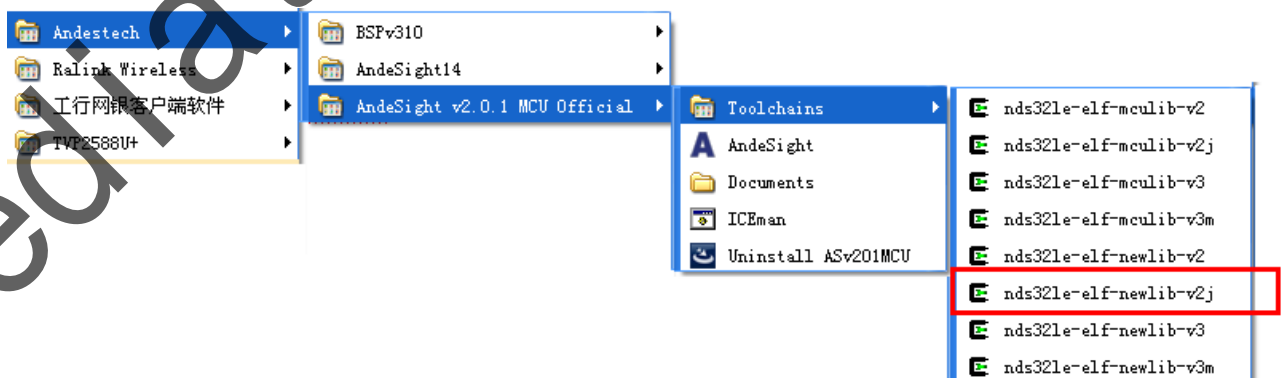
set TOP=C:\Andestech\AndeSight201MCU\cygwin
set TOP=%TOP:\/=%
set TMP=C:\Andestech\AndeSight201MCU\cygwin\cygwin_tmp
set PATH=C:\Andestech\AndeSight201MCU\cygwin\bin;%CD%\bin
set HOME=%CD%\bin

IF EXIST "C:\Andestech\AndeSight201MCU\cygwin\bin\bash.exe" set SHELL=/bin/bash

"C:\Andestech\AndeSight201MCU\cygwin\bin\bash.exe" --login -i

:END
```

然后点击 红色方框的连接 启动即可。



1.2 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

1.3 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
```

1.4 Ap/Sta/recovery/all.bin are created after compile done, which binary we can use

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

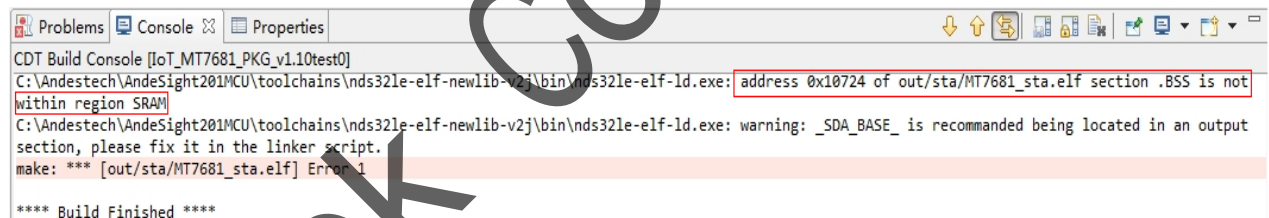
这些文件是 用于 Uart FW upgrade 的,

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_Firmware_Upgrade_v0.0*_*****.pdf 有描述

1.5 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

2 FW UPGRADE

2.1 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_Firmware_Upgrade_v0.0*_*****.pdf

- Not support FOTA yet, 客户可通过现有 API 开发

2.2 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

2.3 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

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

```

[IoT_custom.c (src\api)]
ew Window Help
00111: /*Default setting of STA Config Block*/
00112:
00113: /*Default setting of AP Config Block*/
00114:
00115:
00116: /*bit0- read Calibration settings (TxPower/Tx Freq Offset) from [0:Flash, 1:Efuse]*/
00117: UINT8 gCaliFrEfuse = 0x00;
00118:
00119:
00120: /*unit:ms indicated recovery mode duration*/
00121: #if (ATCMD_RECOVERY_SUPPORT==1)
00122:   UINT16 gRecoveryModeTime = 4000;
00123: #endif
00124:

```

3 APK INSTALL AND USAGE

3.1 Install “IoTManager_v0.94_1_android4.0.apk”, show “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

3.2 Compile APK Source Code failure in Android Codebase 4.2

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

```

-4.4
make: *** No rule to make target `out/target/product/panda/obj/SHARED_LIBRARIES/
libSmartConnection_intermediates/export_includes', needed by `out/target/product
/panda/obj/SHARED_LIBRARIES/libIoT_manager_jni_intermediates/import_includes'.
Stop.

```

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:

```

[mtk54425@mcswgl10 android-4.4]$mkdir ./out/target/product/panda/obj/SHARED_LI
BRARIES/libSmartConnection_intermediates
[mtk54425@mcswgl10 android-4.4]$touch ./out/target/product/panda/obj/SHARED_LI
BRARIES/libSmartConnection_intermediates/import_includes
[mtk54425@mcswgl10 android-4.4]$touch ./out/target/product/panda/obj/SHARED_LI
BRARIES/libSmartConnection_intermediates/export_includes

```

2: 提示没有 libSmartConnection.so 文件

```

make: *** No rule to make target `out/target/product/panda/obj/lib/libSmartConne
ction.so', needed by `out/target/product/panda/obj/SHARED_LIBRARIES/libIoT_manag
er_jni_intermediates/LINKED/libIoT_manager_jni.so'. Stop.
make: Leaving directory `/proj/mtk54425/WCN/TRUNK/APEX/customer/android/android-
4.4'

```

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

```

[mtk54425@mcswgl10 lib]$cp libSmartConnection.so ../../../../../../out/target/pro
duct/panda/obj/lib/

```

4 SYSTEM CODING

4.1 Printf_High() /DBGPRINTF_HIGH() 输出LOG的开关

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

```
/*TRUE: Printf_High()/DBGPRINT_HIGH() is enabled, FALSE: Printf_High/DBGPRINT_HIGH() is disabled*/  
BOOLEAN PRINT_FLAG = TRUE;
```

4.2 The usage of “reserved” region in flash? is possible to store other data in it

12 FLASH PARTITIONS

Flash Layout					
Offset	Section	Size (KB)	HEX (Byte)	DEC Offset	
0x0000	Loader	20	0x5000	0	Store Loader program
0x5000	reserved 1	4	0x1000	20480	Store Recovery Mode program
0x6000	Recovery Mode FW	64	0x10000	24576	
0x16000	reserved 2	4	0x1000	90112	Store Calibration Settings
0x17000	EEPROM	4	0x1000	94208	
0x18000	Common config	4	0x1000	98304	
0x19000	Station Mode Config	4	0x1000	102400	
0x1A000	AP Mode Config	4	0x1000	106496	
0x1B000	User Config	4	0x1000	110592	
0x1C000	reserved 3	12	0x3000	114688	
0x1F000	STA Mode FW	64	0x10000	126976	Store Station Mode Program
0x2F000	reserved 4	4	0x1000	192512	
0x30000	STA Mode-XIP FW	60	0xF000	196608	
0x3F000	STA Mode-OVL FW	60	0xF000	268048	
0x4E000	reserved 5	4	0x1000	319488	
0x4F000	AP Mode FW	64	0x10000	323584	Store AP Mode Program
0x5F000	reserved 6	4	0x1000	389120	
0x60000	AP Mode-XIP FW	60	0xF000	393216	
0x6F000	AP Mode-OVL FW	60	0xF000	454656	
0x7E000	reserved 7	4	0x1000	516096	
0x7F000	Flash Write Buffer	4	0x1000	520192	Flash Write时的Data中转，以减少RAM BufSize
0x80000	reserved 8	0	0x0	524288	

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。

4.3 How to change MAC Address

默认情况下，Mac 地址存在 Flash 的 EEPROM 区域，

Flash Layout					
Offset	Section	Size (KB)	HEX (Byte)	DEC Offset	
0x0000	Loader	20	0x5000	0	Store Loader program
0x5000	reserved 1	4	0x1000	20480	Store Recovery Mode program
0x6000	Recovery Mode FW	64	0x10000	24576	
0x16000	reserved 2	4	0x1000	90112	Store Calibration Settings
0x17000	EEPROM	4	0x1000	94208	
0x18000	Common config	4	0x1000	98304	
0x19000	Station Mode Config	4	0x1000	102400	
0x1A000	AP Mode Config	4	0x1000	106496	
0x1B000	User Config	4	0x1000	110592	
0x1C000	reserved 3	12	0x3000	114688	

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

2 MT7681U EEPROM Layout

Offset ²⁾	Default ²⁾ (hex) ²⁾	b15 ~ b8 ²⁾	b7 ~ b0 ²⁾
04h ²⁾	²⁾	Mac Address [15:0] ²⁾	
06h ²⁾	²⁾	Mac Address [31:16] ²⁾	
08h ²⁾	²⁾	Mac Address [47:32] ²⁾	

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。

4.4 Open Macro:CFG_SUPPORT_TCPIP, Compile Error

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

4.5 XIP, Overlay Mechanism

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

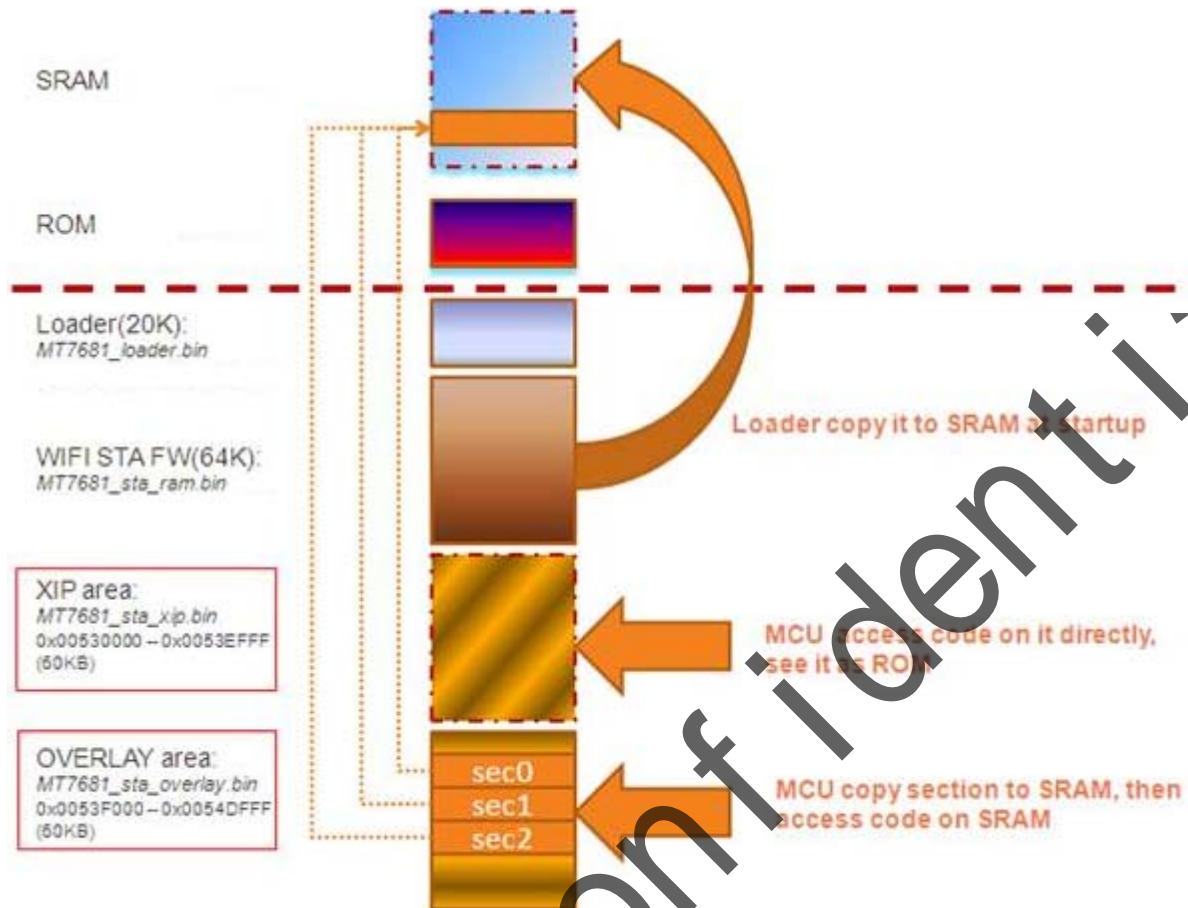
XIP function 定义只需在 function 声明的地方, 加上 **XIP_ATTRIBUTE(“.xipsec0”)** 即可

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。



4.6 HW timer1 EINT Freq Adjustment

The Frequency for hardware timer 1 interrupt, Range [1~10]

```
#define TICK_HZ_HWTIMER1 10 /*T = 1/TICK_HZ_HWTIMER1*/
```

Above example: IoT_Cust_HW_Timer1_Hdlr will be triggered every 100ms (That is $T=1/10$)

5 CONNECTION DEVELOPMENT

5.1 MT7681 Support mode 和 Bandwidth

在 Package v1.10 中，Channel 选择的 API: IoT_Cmd_Set_Channel

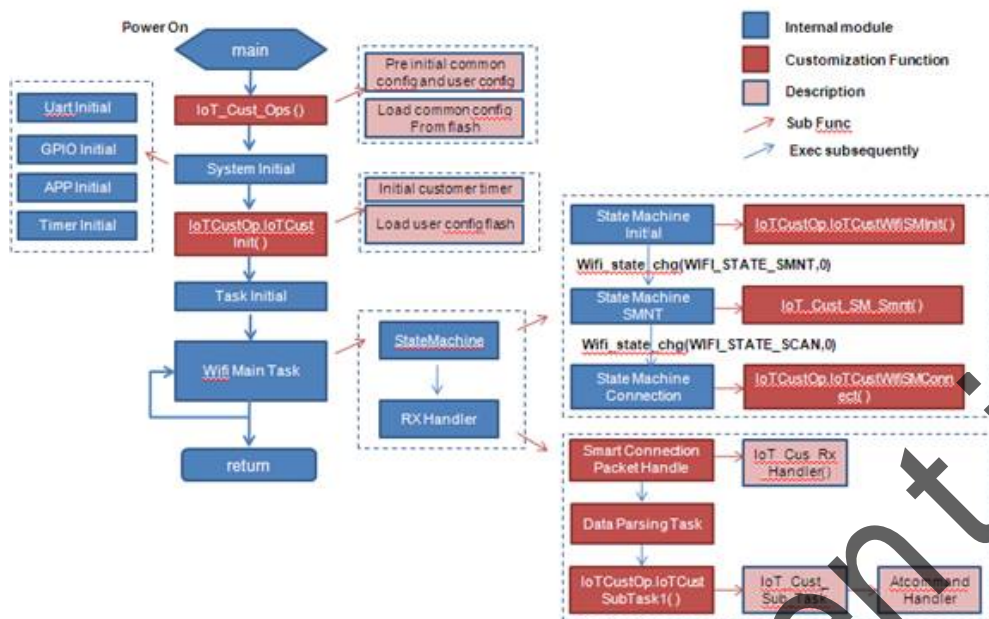
对于 Mode 和 Bandwidth: 目前 Support 80.211 b/g , BW_20M

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

是的，我們也有 main(), 只是我們未完全開放 main function

但我們有 創建 hook function 以便客戶可以開發自己的程序，大部分集中在 iot_custom.c 中

可參考 MT7681_IoT_WiFi_Firmware_Programming_Guide_v0.0*.pdf 的 Section 5 “Customer Hook Function”



5.3 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]`

Flash Layout				
Offset	Section	Size (KB)	HEX (Byte)	DEC Offset
0x0000	Loader	20	0x5000	0
0x5000	reserved 1	4	0x1000	20480
0x6000	Recovery Mode FW	64	0x10000	24576
0x16000	reserved 2	4	0x1000	60112
0x17000	EEPROM	4	0x1000	64208
0x18000	Common config	4	0x1000	68304

```
000056: #define VALID_EEPROM_VERSION 1
000057: #define EEPROM_VERSION_OFFSET 0x02
000058:
000059: #define EEPROM_MAC_12_OFFSET 0x04
000060: #define EEPROM_MAC_34_OFFSET 0x06
000061: #define EEPROM_MAC_56_OFFSET 0x08
000062:
000063: #define EEPROM_NIC1_OFFSET 0x34
000064: #define EEPROM_NIC2_OFFSET 0x36
000065:
```

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???

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 `pl0TMIme->CurrentWifiState` 这个变量中。以此来控制状态机的 flow 如果 `pl0TMIme->CurrentWifiState = WIFI_STATE_CONNED (6)`, 即表示与 AP 的连线成功 下一步会开始获取 IP

5.5 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 DHCP 以获取 IP，具体文件和 function 如下图


```

- Source Insight - [Iot_udp_app.c (src\...\iot_udp_app)]
Options View Window Help
00043: void
00044: iot_udp_appcall(void)
00045: {
00046:     struct uip_udp_conn *udp_conn = uip_udp_conn;
00047:     u16_t lport, rport;
00048:     lport=HTONS(udp_conn->lport);
00049:     rport=HTONS(udp_conn->rport);
00050:
00051:     if(lport == DHCP_CLIENT_PORT) {
00052:         handle_dhcp();
00053:     }
00054:     #if CFG_SUPPORT_DNS
00055:     } else if (rport == DNS_SERVER_PORT) {
00056:         handle_resolv();
00057:     #endif
00058:     /* Customer APP start. */
00059:
00060:     } else if (lport == 7682) {
00061:         udp_server_sample();
00062:     /* } else if (lport == 6666) {
00063:         udp_client_sample();
00064:     } else if (lport == 8888) {
00065:         resolv_usage_sample(); */
00066:
00067:     /* Customer APP end. */
00068: }

```

5.6 How to reset IP/SSID and other parameter to default

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

Flash Layout					
Offset	Section	Size (KB)	HEX (Byte)	DEC Offset	
0x0000	Loader	20	0x5000	0	Store Loader program
0x5000	reserved 1	4	0x1000	20480	
0x6000	Recovery Mode FW	64	0x10000	24576	Store Recovery Mode program
0x16000	reserved 2	2	0x4000	90112	
0x17000	EEPROM	4	0x1000	94208	Store Calibration Settings
0x18000	Common config	4	0x1000	98304	
0x19000	Station Mode Config	2	0x1000	102400	
0x1A000	AP Mode Config	4	0x1000	106496	
0x1B000	User Config	4	0x1000	110592	
0x1C000	reserved 3	12	0x3000	114688	
0x1F000	STA Mode RAM FW	64	0x10000	126976	
0x2F000	reserved 4	4	0x1000	192512	
0x30000	STA Mode-XIP FW	60	0xF000	196608	Store Station Mode Program
0x3F000	STA Mode-OVL FW	60	0xF000	268048	
0x4E000	reserved 5	4	0x1000	319488	
0x4F000	AP Mode FW	64	0x10000	323584	
0x5F000	reserved 6	4	0x1000	389120	Store AP Mode Program
0x60000	AP Mode-XIP FW	60	0xF000	393216	
0x6F000	AP Mode-OVL FW	60	0xF000	454656	
0x7E000	reserved 7	4	0x1000	516096	
0x7F000	Flash Write Buffer	4	0x1000	520192	Flash Write时的Data中转，以减少RAM Buf Size
0x80000	reserved 8	0	0x0	524288	

5.7 There is no “mem_alloc, mem_free” API in SDK v1.10

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。

5.8 Structure _WIFI_STATE Introduction, The means for each state

```
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;
```

5.9 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

```
00642:      }  
00643:      /* Smnt connection done */  
00644:  
00645:  
00646:      /* After smnt connection done */  
00647: 1  /* need set Smnt connection information and start to scan*/  
00648:      IoTSmntInfo.AuthMode = 0;  
00649:      IoTSmntInfo.SsidLen = strlen(Ssid);  
00650:      IoTSmntInfo.PassphraseLen = strlen(Passphrase); //sizeof(Passphrase);  
00651:      memcpy(IoTSmntInfo.Ssid, Ssid, IoTSmntInfo.SsidLen);  
00652:      memcpy(IoTSmntInfo.Passphrase, Passphrase, IoTSmntInfo.PassphraseLen);  
00653:      memcpy(IoTSmntInfo.PMK, PMK, strlen(PMK));  
00654:  
00655:      IoT_Cust_smnt_info(); /*Sync the IoTSmntInfo to other StateMachine, must call this func if IoT_Si  
00656:  
00657: 2  /* change wifi state to SCAN*/  
00658:      wifi_state_chg(WIFI_STATE_SCAN, 0);  
00659: #endif
```

5.10 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)

```

==> Recovery Mode
<== RecoveryMode
(-)
SM=0, Sub=0
SM=1, Sub=0
SM=2, Sub=0
SM=3, Sub=0
SM=3, Sub=1
SM=4, Sub=0
SM=4, Sub=1
SM=5, Sub=0
SM=6, Sub=0
Got IP:192.168.43.61
[wTask]11782
[wTask]19640
[wTask]27246
==> Recovery Mode
<== RecoveryMode
(-)
SM=0, Sub=0
SM=2, Sub=0
SM=3, Sub=0
SM=3, Sub=1
SM=4, Sub=0
SM=4, Sub=1
SM=5, Sub=0
SM=6, Sub=0
Got IP:192.168.43.61
[wTask]9745
[wTask]15083
[wTask]20497

```

第一次上电开始做
Smart Connection

连线成功，此时会存储
Smart Connection的数据
这样下次再上电就不用再做
smart connect了

再次上电后，直
接进入Scan状态

5.11 使用宏 CFG_SUPPORT_MTK_SMNT=1 来控制是否使用 MTK的smart connection

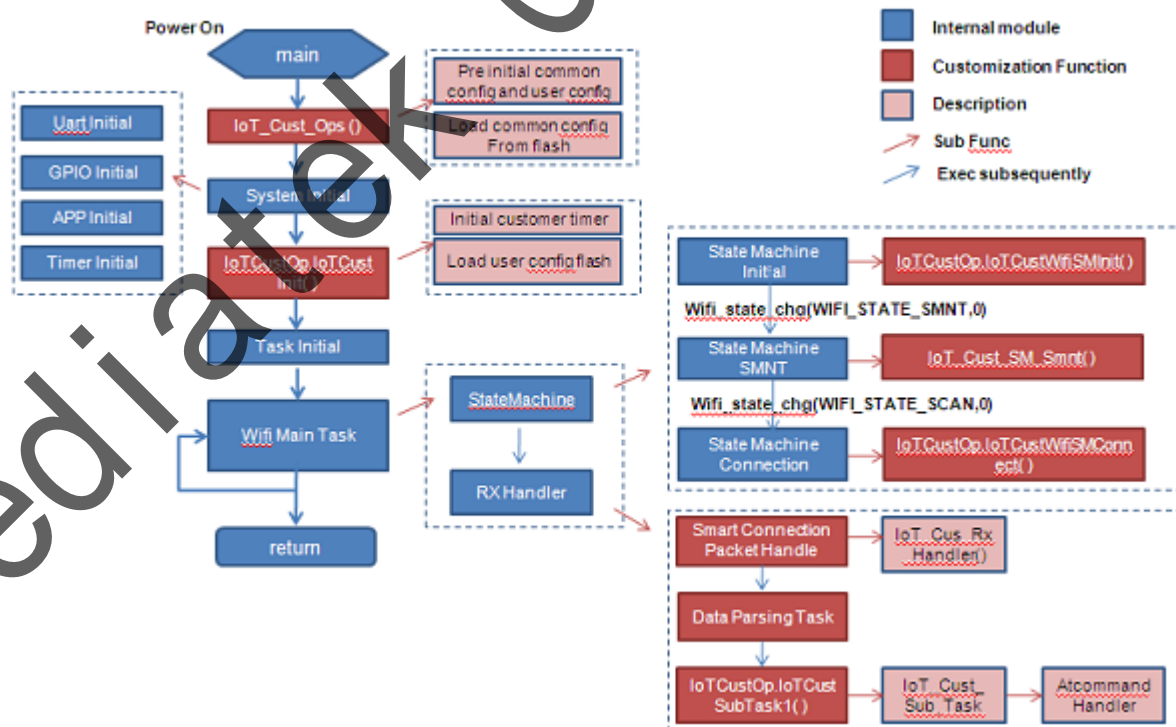
如果客户有自己的 Smart Connection，就可将该宏 值设为 0

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

SDKv1.20 之前：在 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

SDKv1.30 及之后：IoT_Cus_Rx_Handler() 被 STARxDoneInterruptHandle 替代

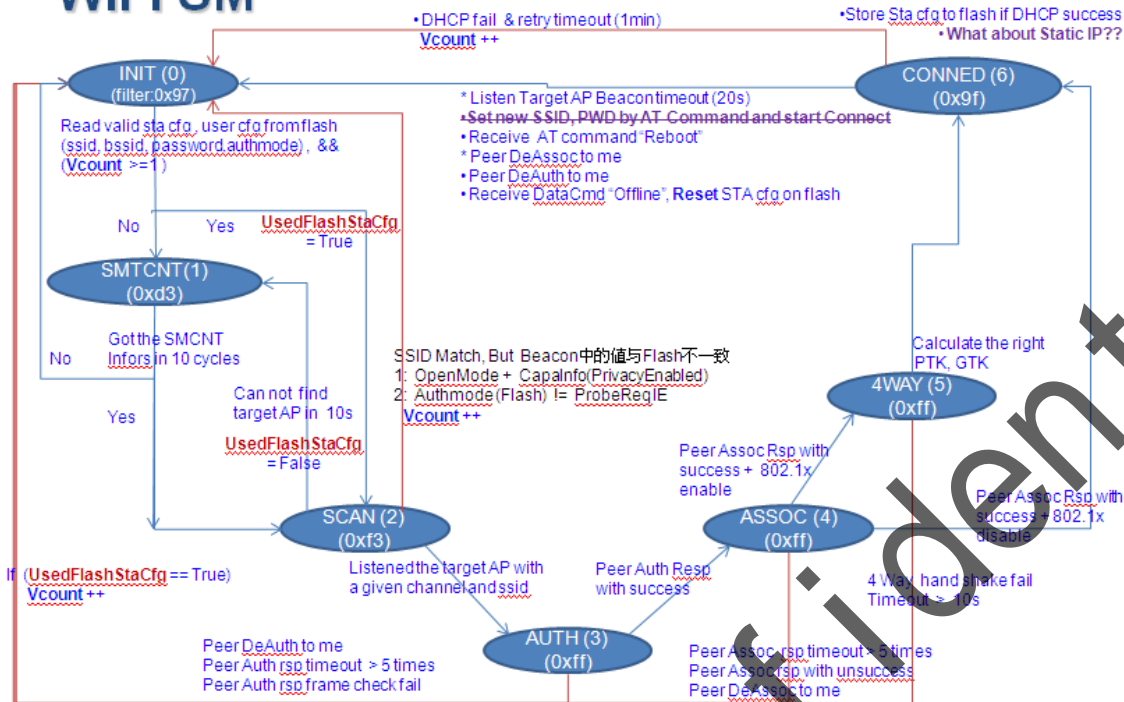
IoT_Cut_SM_Smnt() 是 smart connection 的状态机。结合代码可做进一步了解



5.12 Wifi Connection State Machine Flow

Wifi Connection State Machine Flow (STA Mode v1.30):

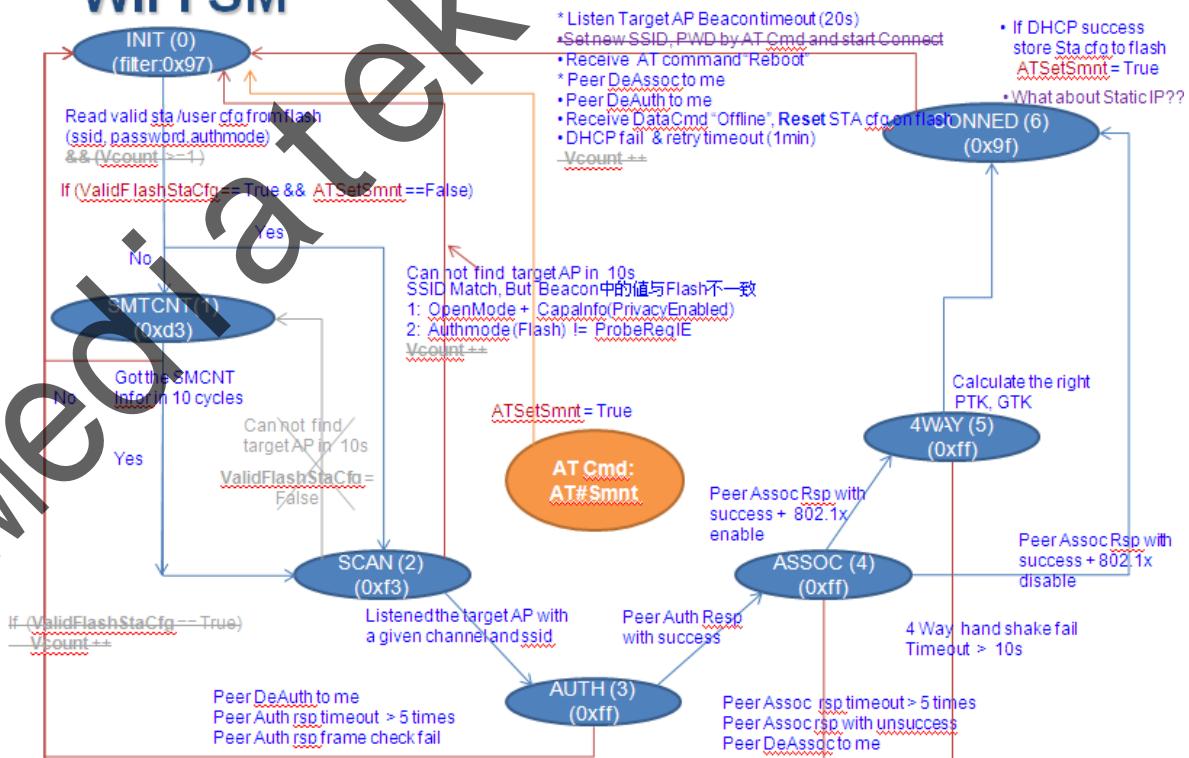
WIFI SM



Wifi Connection State Machine Flow (STA Mode v1.40):

- * when Scan, Auth, Assoc, 4Way, DHCP fail, not back to smart connection state
- * Smart Connection state only be triggered by following method:
 - 1: There is no valid content in Flash Sta Config region
 - 2: 7681 receive AT Cmd: AT#Smnt

WIFI SM



5.13 How to get PMK

PMK 会由 SSID, Passphrase 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, Passphrase, 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()

```
00646: /* After smnt connection done */
00647: /* need set Smnt connection information and start to scan*/
00648: IoTSmntInfo.AuthMode = 0;
00649: IoTSmntInfo.SsidLen = strlen(Ssid);
00650: IoTSmntInfo.PassphraseLen = strlen(Passphrase); //sizeof(Passphrase);
00651: memcpy(IoTSmntInfo.Ssid, Ssid, IoTSmntInfo.SsidLen);
00652: memcpy(IoTSmntInfo.Passphrase, Passphrase, IoTSmntInfo.PassphraseLen);
00653:
00654:
00655: {
00656:     /*Deriver PMK by AP 's SSID and Password*/
00657:     UCHAR keyMaterial[40] = {0};
00658:     if (IoTSmntInfo.AuthMode != Ndis802_11AuthModeOpen)
00659:     {
00660:         RtmpPasswordHash(IoTSmntInfo.Passphrase, IoTSmntInfo.Ssid,
00661:                         IoTSmntInfo.SsidLen, keyMaterial);
00662:         NdisMoveMemory(IoTSmntInfo.PMK, keyMaterial, 32);
00663:     }
00664: }
00665: //memcpy(IoTSmntInfo.PMK, PMK, strlen(PMK));
00666:
00667: IoT_Cust_smnt_info(); /*Sync the IoTSmntInfo to other StateMachine, must call this func if IoTSmntInfo changed*/
00668:
00669: /* change wifi state to SCAN*/
00670: wifi_state_chg(WIFI_STATE_SCAN, 0);
00671: #endif

/*
 * password - ascii string up to 63 characters in length
 * ssid - octet string up to 32 octets
 * ssidlength - length of ssid in octets
 * output must be 40 octets in length and outputs 256 bits of key
 */
int RtmpPasswordHash(PSTRING password, PCHAR ssid, INT ssidlength, PCHAR output)
```

6 APPLICATION DEVELOPMENT

6.1 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.

```

: void IoT_Cust_uart2wifi_data_handler(UCHAR *uart_content, UINT16 uart_content_count)
:
: {
:     IoT_uart_output(uart_content, uart_content_count);
:
:     /*should not call uip_send here, all uip_send need to be implemented
:     in iot_udp_appcall() / iot_tcp_appcall(), as the reason of the uIP app
:     management (Connection/Port...) is controlled in the iot_***_appcall()*/
:     //uip_send(uart_content, uart_content_count); //mask
:
:     /*here should allocate a buffer or flag,
:     Let iot_***_appcall() detected it and call uip_send()*/
:
:     return;
: }

```

6.2 IoT application 内要 scan AP, 連到 AP, 建 TCP + SSL ...等, 要参考哪些 APIs document?

MTK 代码中, 有 Initial-> Smart Connection-> Scan -> Auth-> Assoc-> 4 way -> DHCP 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 Source Code 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

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?

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

6.4 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 Source Code 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

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?

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,

6.6 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

6.7 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

```
00060: #define DEFAULT_LOCAL_UDP_SRV_PORT 7681
00061:
00062: #define DEFAULT_TCP_UDP_CS 1 /*0: UDP, 1:TCP (Default 3*Client, 1*Server is Open)*/
00063: // #define DEFAULT_IOT_TCP_SRV_PORT 7681 /*The IoT Server TCP Port in the internet */
00064: #define DEFAULT_IOT_TCP_SRV_PORT 12345 /*The IoT Server TCP Port in the internet */
00065: #define DEFAULT_LOCAL_TCP_SRV_PORT 7681 /*The TCP Port if 7681 as a TCP server */
00066: #define DEFAULT_IOT_UDP_SRV_PORT 7681 /*The IoT Server UDP Port in the internet */
00067: #define DEFAULT_LOCAL_UDP_SRV_PORT 7681 /*The UDP Port if 7681 as a UDP server */
00068:
00069: #define DEFAULT_USE_DHCP 1 /*0: Static IP, 1:Dynamic IP*/
00070: #define DEFAULT_STATIC_IP {192,168,0,99}
00071: #define DEFAULT_SUBNET_MASK_IP {255,255,255,0}
00072: #define DEFAULT_DNS_IP {192,168,0,1}
00073: #define DEFAULT_GATEWAY_IP {192,168,0,1}
00074: // #define DEFAULT_IOT_SERVER_IP {182,148,123,91}
00075: // #define DEFAULT_IOT_SERVER_IP {172,26,74,63}
00076: #define DEFAULT_IOT_SERVER_IP {192,168,1,89}
00077:
00078: #define DEFAULT_IOT_CMD_PWD {0xFF,0xFF,0xFF,0xFF}
```

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_ServerIP`

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

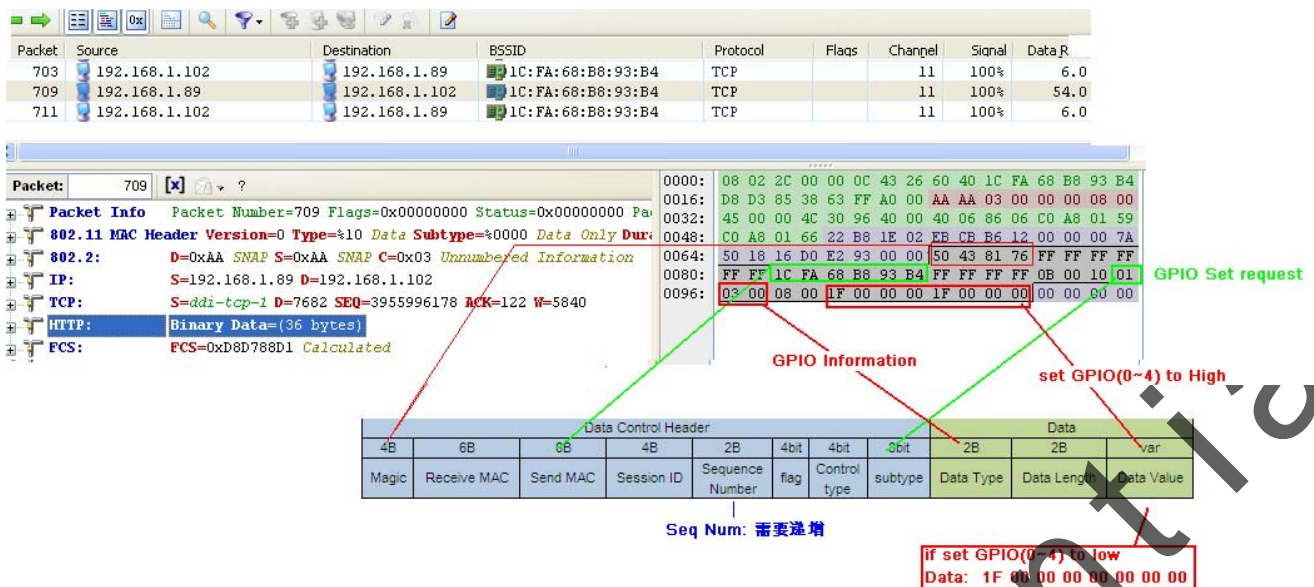
正常的话, LOG 中会有如下提示:

```
SM=2, Sub=0
SM=3, Sub=0
SM=3, Sub=1
SM=4, Sub=0
SM=4, Sub=1
SM=6, Sub=0
Got IP:192.168.1.102
[WTASK]185880
Connected fd:0,lp:7682,ra:192.168.1.89,rp:8888
```

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)



6.8 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，

6.9 Enable TCP Tx ReTransmit, and Http Client

In uip_conf.h

- 1: Enable TCP Re-Transmit
#define CFG_SUPPORT_TCP_REXMIT 1
- 2: Enable Http Client
#define UIP_HTTP_CLIENT_SUPPORT 1

7 INTERFACE CUSTOMIZATION

7.1 Set UartTx as Interrrupt Mode or Poll mode

```
Insight - [Iot_custom.c (src\api)]
Options View Window Help
00145: #if (UART_INTERRUPT == 1)
00146: /*
00147:  * We can use UART TX POLL scheme(such as debug/test),default is FALSE
00148:  */
00149: BOOLEAN UART_TX_POLL_ENABLE = FALSE;
00150:
```

7.2 Set PWM Level

```

e Insight - [lot_api.h (src\api)]
Options View Window Help
00752: #elif (IOT_PWM_SUPPORT == 1 && IOT_PWM_TYPE == 2)
00753:
00754: #define PWM_R_GPIO 0
00755: #define PWM_G_GPIO 1
00756: #define PWM_B_GPIO 3 // gpio2 for button input
00757: #define PWM_HIGHEST_LEVEL 20 /* PWM Freq = 1000/PWM_HIGHEST_LEVEL */
00758: #define MAX_PWM_COUNT 5

```

7.3 Why GPIO-x set Back to Low, after use IoT_gpio_output() to set GPIO-x to High

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