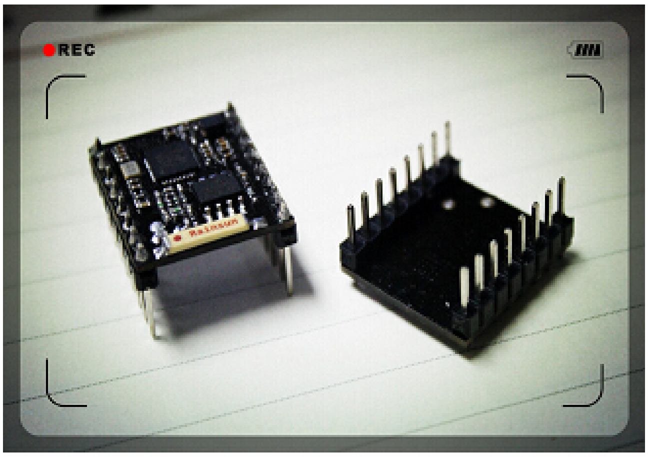
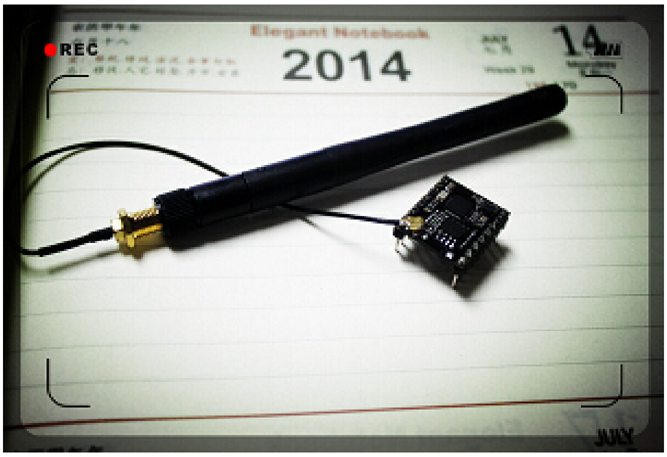




深圳市安信可科技 无线模块专家 无敌性价比！

MT7681 无线模块成品价格：

批量价格大于500：	30元
大于5K：	28元
大于10K：	25元

以上价格只降不涨，每隔6个月降低10%

网址：http://www.ai-thinker.com

MT7681
802.11 b/g/n single chip
Preliminary datasheet

Version:	0.00
Release date:	2014-01-08

Specifications are subject to change without notice.

MT7681 IoT Wi-Fi Firmware Programming Guide

Version: 0.07

Release date: 2014-5-16

Specifications are subject to change without notice.

Revision History

Date	Revision	Author	Description
01.03.2014	First v0.01	Jinchuan	Initial draft for MT7681 IoT Firmware Programming Guide.
01.16.2014	v0.02	Jiayi	Update Flash Layout and Flash API Update Folder Structure
01.24.2014	v0.03	Jinchuan	Update Flash Layout and Flash API： spi_flash_write() Add Section: AT Command Usage
03.22.2014	v0.04	Jinchuan Jerry	Add FIRMWARE boot UP flow Add Customer Hook Function Add Flash settings Load/Storage Add Timer APIs Modify Interfae APIs Modify Flash Partitions
04.15.2014	v0.05	Jinchuan Jerry	Modify File Structure Modify GPIO Interface Description and GPIO/Pin Mode Set Modify Flash Partitions Add New PWM API Add Customer UART-TO-WIFI Function Add Customer Hook Function for Smart Connection
05.16.2014	v0.07	Jinchuan	Add IoT_gpio_read() Add IoT_Cust_Set_GPIINT_MODE(); IoT_Cust_Get_GPIINT_MODE(); IoT_Cust_GPIINT_Hdlr() Add GetMsTimer() Add Security API Update Firmware Boot Up Flow Update Customer Hootk Functioin Update File Structure Update IoT_led_pwm(): led_num range in soft pwm mode Delete ATCmd about TCP/UDP

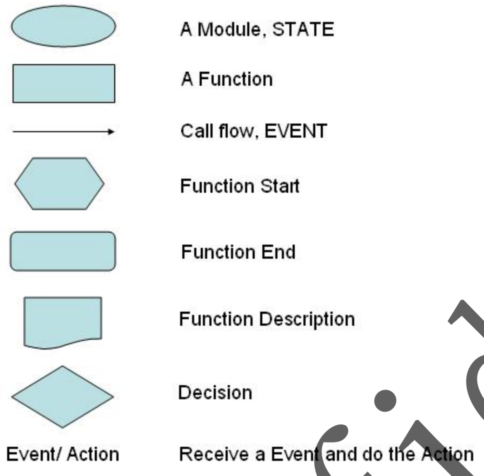
Contents

1 INTRODUCTION

The 7681 IoT Wi-Fi structure could be divided into two layers (HW layer, Firmware Layer);

This document aims to help the programmers understand the 7681 Wi-Fi Firmware architecture and how to do the customization, such as AT command or Data command

1.1 Flow Chart Symbols

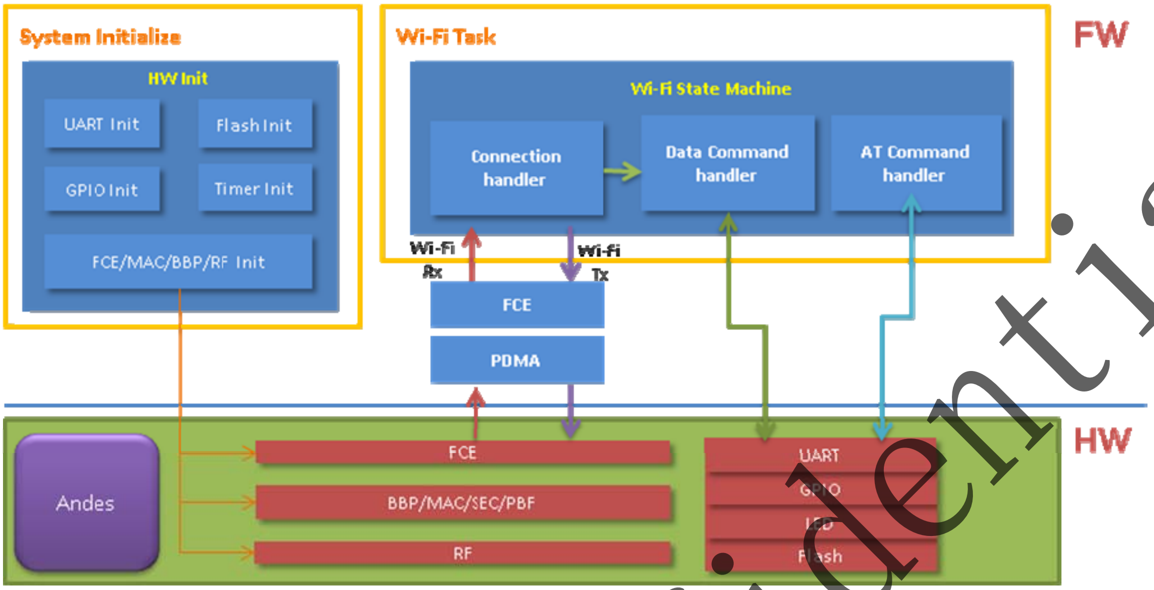


1.2 Keywords

BBP:	Base Band Processor
SEC:	Security Engine
PBF:	Packet Buffer
PDMA:	Programmable Direct Memory Access
FCE:	Frame Control Engine

2 SW STRUCTURE

2.1 Flow chart



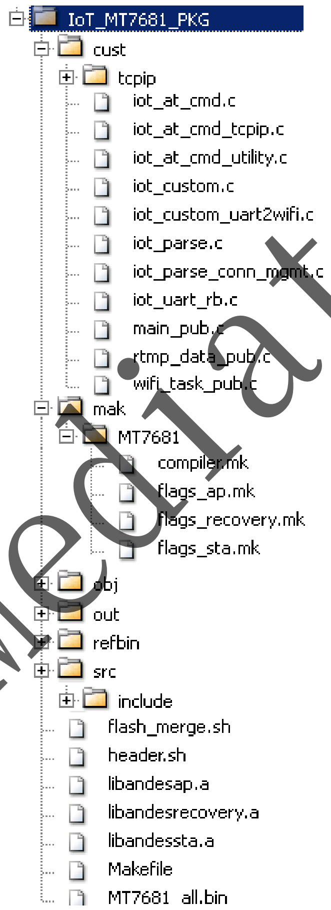
HW init: initialize the HW registers to enable HW interfaces and Wi-Fi function

Wi-Fi State machine: a single loop to control Wi-Fi Tx, Rx, and process the Data command, AT command

Data Command handler: handle the Data command which received from Wi-Fi

AT Command handler: handle the AT command which received from UART

3 FILE STRUCTURE

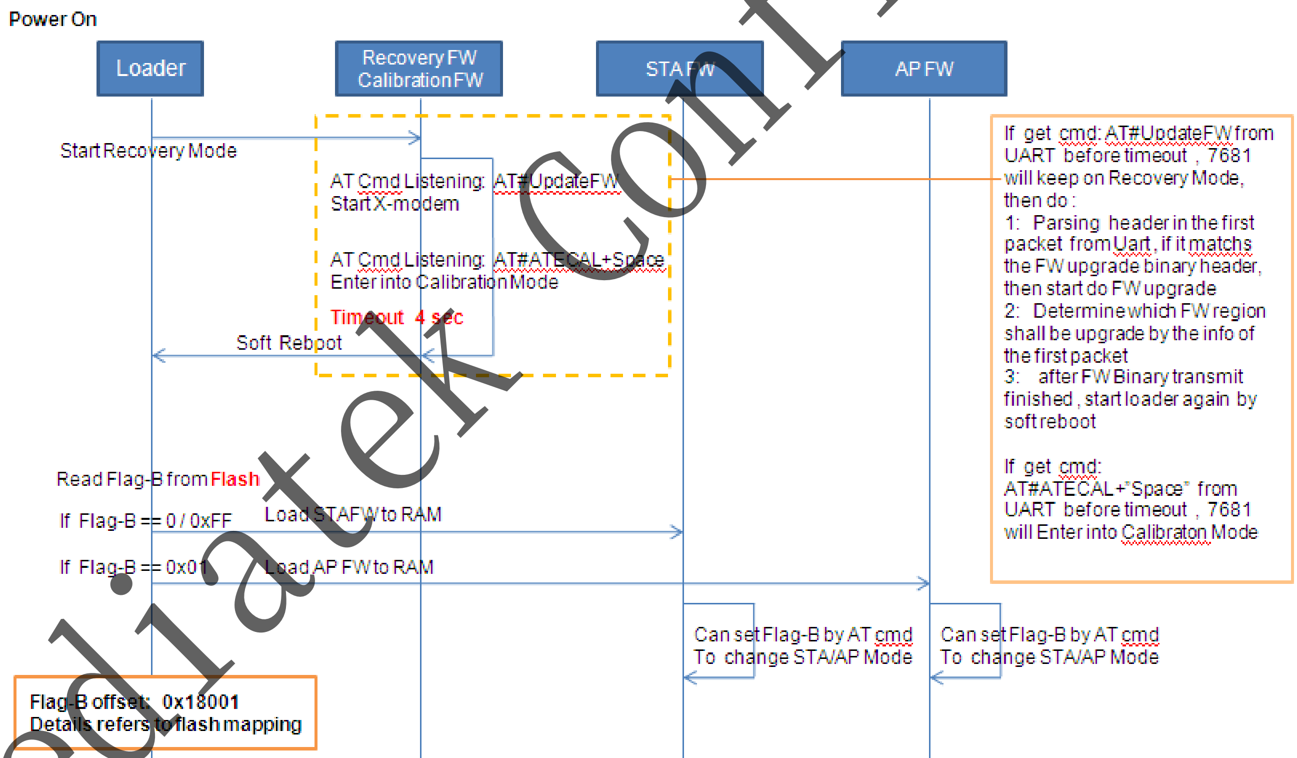


```
IoT_MT7681_PKG
  cust
    tcpip
      iot_at_cmd.c
      iot_at_cmd_tcpip.c
      iot_at_cmd_utility.c
      iot_custom.c
      iot_custom_uart2wifi.c
      iot_parse.c
      iot_parse_conn_mgmt.c
      iot_uart_rb.c
      main_pub.c
      rtmp_data_pub.c
      wifi_task_pub.c
  mak
    MT7681
      compiler.mk
      flags_ap.mk
      flags_recovery.mk
      flags_sta.mk
  obj
  out
  refbin
  src
    include
    flash_merge.sh
    header.sh
    libandesap.a
    libandesrecovery.a
    libandessta.a
    Makefile
    MT7681_all.bin
```

\cust\main_pub.c	, Main entry
\cust\wifi_task_pub.c	, Wifi Task Function
\cust\rtmp_data_pub.c	, Rx Handler
\cust\iot_customer.c	, The Initial Hook and setting load/storage
\cust\iot_at_cmd.c	, handle the AT command which received from UART
\cust\iot_at_cmd_utility.c	,the common api for AT command usage
\cust\iot_parse.c	,handle the Data command which received from Wi-Fi
\cust\tcpip\*.c	, the source code and sample code for TCP UDP
\mak\MT7681\	, store the configuration for compiler or linker
\mak\MT7681\flags_sta.mk	, store the macros for all station mode source code
\mak\MT7681\flags_ap.mk	, store the macros for all AP mode source code
\out\	, store the files which created by compiler
\out\build.log	, the compiling log
\out\MT7681.bin	, the target binary file
\src\include	, the header files
libandessta.a	, library for MT7681 Station Mode
libandesap.a	, library for MT7681 AP Mode
MT7681_all.bin	, The Image for FW upgrade by Flash writer (include Loader.bin, EEPROM.bin, RecoveryFW.bin,

StationFW.bin, APFW.bin)

4 FIRMWARE BOOT UP FLOW



If get cmd: AT#UpdateFW from UART before timeout , 7681 will keep on Recovery Mode, then do :
1: Parsing header in the first packet from Uart, if it matchs the FW upgrade binary header, then start do FW upgrade
2: Determine which FW region shall be upgrade by the info of the first packet
3: after FW Binary transmit finished , start loader again by soft reboot

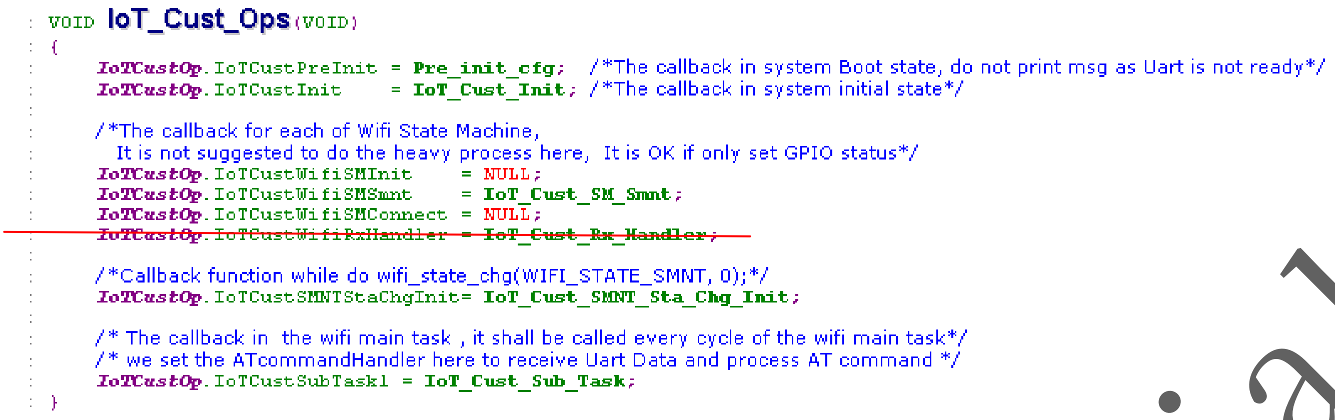
If get cmd: AT#ATECAL+"Space" from UART before timeout , 7681 will Enter into Calibraton Mode

Flag-B offset: 0x18001
Details refers to flash mapping

The Boot up flow is: Loader→ Recovery Mode → Loader → STA/AP FW

5 CUSTOMER HOOK FUNCTION

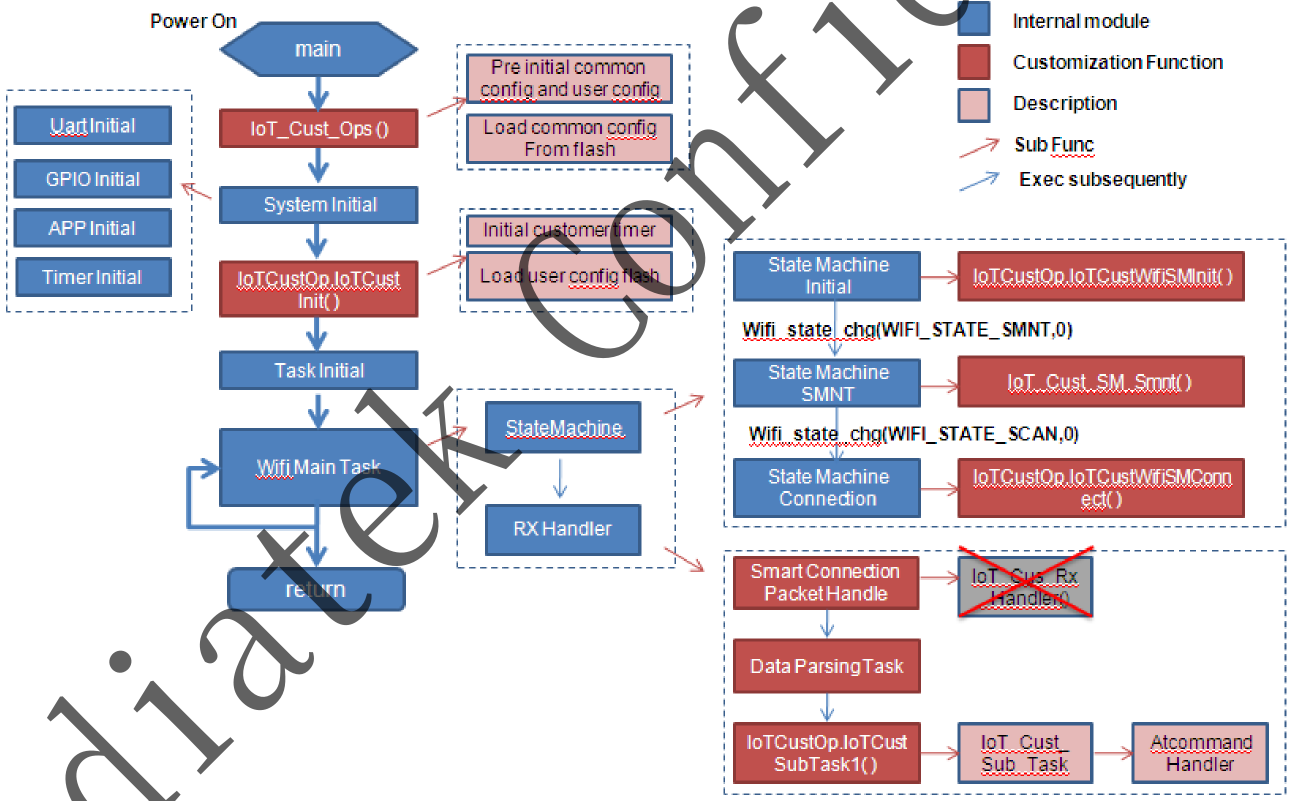
Iot_customer.c



```
VOID IoT_Cust_Ops(VOID)
{
    IoTCustOp.IoTCustPreInit = Pre_init_cfg;  /*The callback in system Boot state, do not print msg as Uart is not ready*/
    IoTCustOp.IoTCustInit    = IoT_Cust_Init; /*The callback in system initial state*/

    /*The callback for each of Wifi State Machine,
      It is not suggested to do the heavy process here,  It is OK if only set GPIO status*/
    IoTCustOp.IoTCustWifiSMInit    = NULL;
    IoTCustOp.IoTCustWifiSMSmnt    = IoT_Cust_SM_Smnt;
    IoTCustOp.IoTCustWifiSMConnect = NULL;
~~  IoTCustOp.IoTCustWifiRxHandler = IoT_Cust_Rx_Handler;~~

    /*Callback function while do wifi_state_chg(WIFI_STATE_SMNT, 0);*/
    IoTCustOp.IoTCustSMNTStaChgInit= IoT_Cust_SMNT_Sta_Chg_Init;

    /* The callback in  the wifi main task , it shall be called every cycle of the wifi main task*/
    /* we set the ATcommandHandler here to receive Uart Data and process AT command */
    IoTCustOp.IoTCustSubTask1 = IoT_Cust_Sub_Task;
}
```

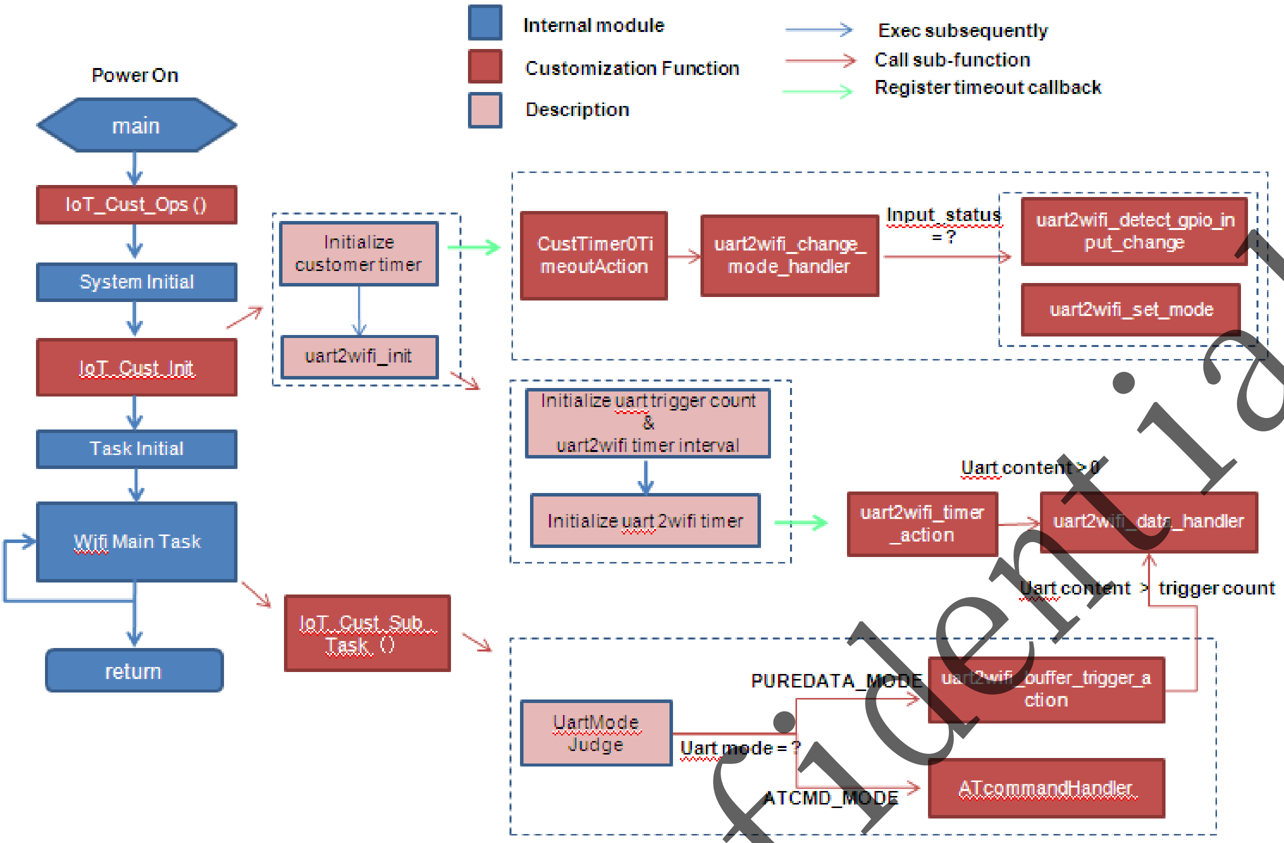
IoT_Cust_Ops() is used to register callback function , which could be called by Wifi main function

When and where this callback function will be used, please see next picture for the details



6 CUSTOMER UART-TO-WIFI FUNCTION

Iot_customer_uart2wifi.c



IoT_Cust_uart2wifi_data_handler() :

It is the key function for uart-to-wifi transmission.

It is the bridge between uart module and TCP/IP module of WiFi.

In the sample code, uip_send() is called to send data from uart to wifi.

IoT_Cust_uart2wifi_init() :

You can use it to configure uart-to-wifi timer interval and uart trigger count.

In the sample code, every 300 ms , or when the uart rx content is larger than 10,

a uart-to-wifi transmission judgment will be triggered.

IoT_Cust_uart2wifi_detect_gpio_input_change () :

You can use it to define your own input status change.

In the sample code, when the input of gpio2 is high, uart rx is switched to pure data mode;

otherwise, it is switched to AT cmd mode.

Note:

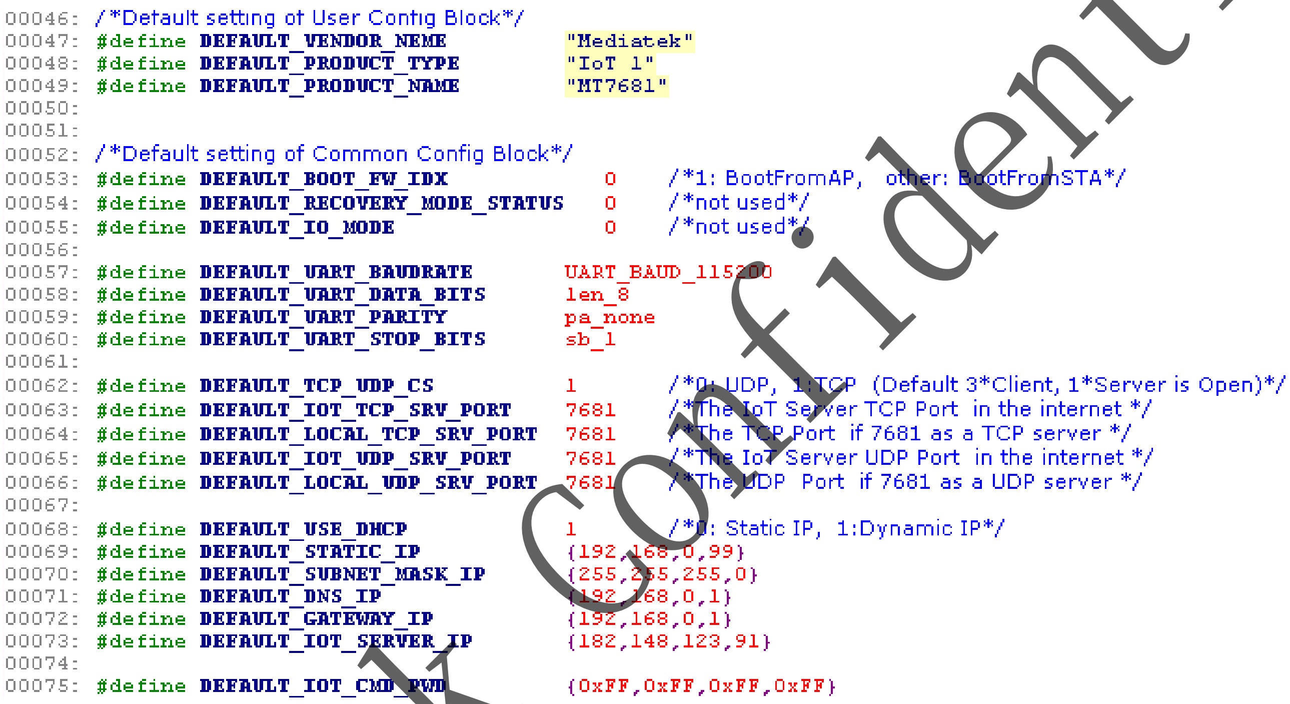
Uart-to-wifi function collides with data parser uart rx function, so you must set DATAPARSING_UARTRX_SUPPORT to 0 first.

When and where these functions will be used, please see next picture for the details

7 FLASH SETTINGS LOAD/STORAGE

Iot_customer.c

The default settings on User/Common config block of the flash , if there is no content or the content is invalid, system shall used these default settings , the detail implementation is on the iot_customer.c



```
/*Default setting of User Config Block*/
#define DEFAULT_VENDOR_NEME             "Mediatek"
#define DEFAULT_PRODUCT_TYPE            "IoT 1"
#define DEFAULT_PRODUCT_NAME            "MT7681"


/*Default setting of Common Config Block*/
#define DEFAULT_BOOT_FW_IDX               0     /*1: BootFromAP,  other: BootFromSTA*/
#define DEFAULT_RECOVERY_MODE_STATUS      0     /*not used*/
#define DEFAULT_IO_MODE                   0     /*not used*/

#define DEFAULT_UART_BAUDRATE           UART_BAUD_115200
#define DEFAULT_UART_DATA_BITS          len_8
#define DEFAULT_UART_PARITY             pa_none
#define DEFAULT_UART_STOP_BITS          sb_1

#define DEFAULT_TCP_UDP_CS              1       /*0: UDP,  1:TCP  (Default 3*Client, 1*Server is Open)*/
#define DEFAULT_IOT_TCP_SRV_PORT        7681    /*The IoT Server TCP Port  in the internet */
#define DEFAULT_LOCAL_TCP_SRV_PORT      7681    /*The TCP Port  if 7681 as a TCP server */
#define DEFAULT_IOT_UDP_SRV_PORT        7681    /*The IoT Server UDP Port  in the internet */
#define DEFAULT_LOCAL_UDP_SRV_PORT      7681    /*The UDP  Port  if 7681 as a UDP server */

#define DEFAULT_USE_DHCP                1       /*0: Static IP,  1:Dynamic IP*/
#define DEFAULT_STATIC_IP               {192,168,0,99}
#define DEFAULT_SUBNET_MASK_IP          {255,255,255,0}
#define DEFAULT_DNS_IP                  {192,168,0,1}
#define DEFAULT_GATEWAY_IP              {192,168,0,1}
#define DEFAULT_IOT_SERVER_IP           {182,148,123,91}

#define DEFAULT_IOT_CMD_PWD             {0xFF,0xFF,0xFF,0xFF}
```

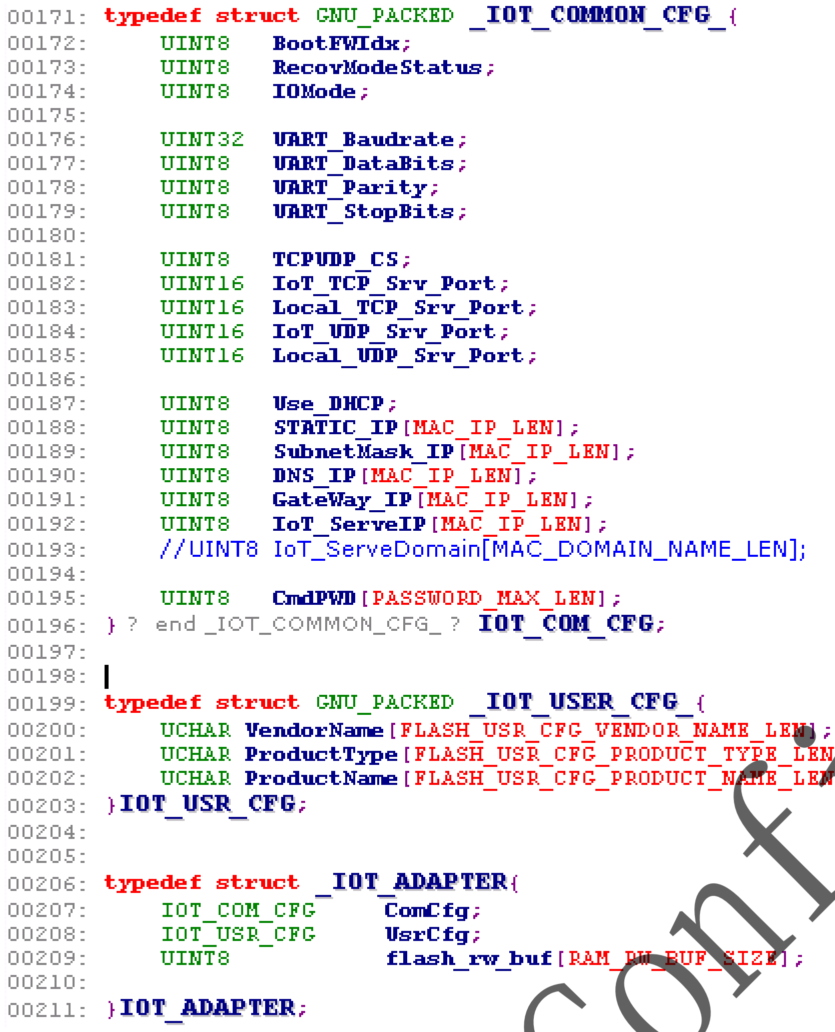
There are two structures: IOT_COM_CFG, IOT_USR_CFG

IOT_COM_CFG: Please **do not** modify this structure, because the Wi-Fi main task / TCP IP will use this structure for module initialization or operation

IOT_USR_CFG: Can be customized, because only iot_parser.c, iot_custom.c will use this structure

Notice: Both of above two structures are mapping with Flash Layout, and the settings load/reset is optimized for code size slim by macro “FLASH_STRUCT_MAPPING 1” with this condition.

If the structure is not mapping with Flash Layout, “FLASH_STRUCT_MAPPING” should be set as 0



```
typedef struct GNU_PACKED _IOT_COMMON_CFG_{
    UINT8   BootFWIdx;
    UINT8   RecovModeStatus;
    UINT8   IOMode;

    UINT32  UART_Baudrate;
    UINT8   UART_DataBits;
    UINT8   UART_Parity;
    UINT8   UART_StopBits;

    UINT8   TCPUDP_CS;
    UINT16  IoT_TCP_Srv_Port;
    UINT16  Local_TCP_Srv_Port;
    UINT16  IoT_UDP_Srv_Port;
    UINT16  Local_UDP_Srv_Port;

    UINT8   Use_DHCP;
    UINT8   STATIC_IP[MAC_IP_LEN];
    UINT8   SubnetMask_IP[MAC_IP_LEN];
    UINT8   DNS_IP[MAC_IP_LEN];
    UINT8   GateWay_IP[MAC_IP_LEN];
    UINT8   IoT_ServeIP[MAC_IP_LEN];
    //UINT8 IoT_ServeDomain[MAC_DOMAIN_NAME_LEN];

    UINT8   CmdPWD[PASSWORD_MAX_LEN];
} ? end _IOT_COMMON_CFG_ ? IOT_COM_CFG;


typedef struct GNU_PACKED _IOT_USER_CFG_{
    UCHAR VendorName[FLASH_USR_CFG_VENDOR_NAME_LEN];
    UCHAR ProductType[FLASH_USR_CFG_PRODUCT_TYPE_LEN];
    UCHAR ProductName[FLASH_USR_CFG_PRODUCT_NAME_LEN];
}IOT_USR_CFG;


typedef struct _IOT_ADAPTER{
    IOT_COM_CFG     ComCfg;
    IOT_USR_CFG     UsrCfg;
    UINT8           flash_rw_buf[RAM_RW_BUF_SIZE];

}IOT_ADAPTER;
```

Example A: Modify IOT Server IP to {172.133.125.12}

Method1: Change #define DEFAULT_IOT_SERVER_IP {182.148.129.91} to {172.133.125.12}

Method2: not modify DEFAULT_IOT_SERVER_IP, but use FLASH API to write the new value to related position of the FLASH, thus, while MT7681 reboot or power on again, the new settings on flash will be load

Example B: Add a Uart2Wifi Length parameter to User Config Block

Step1: add new macro in flash_mapping.h to indicate Uart2Wifi position on the flash,

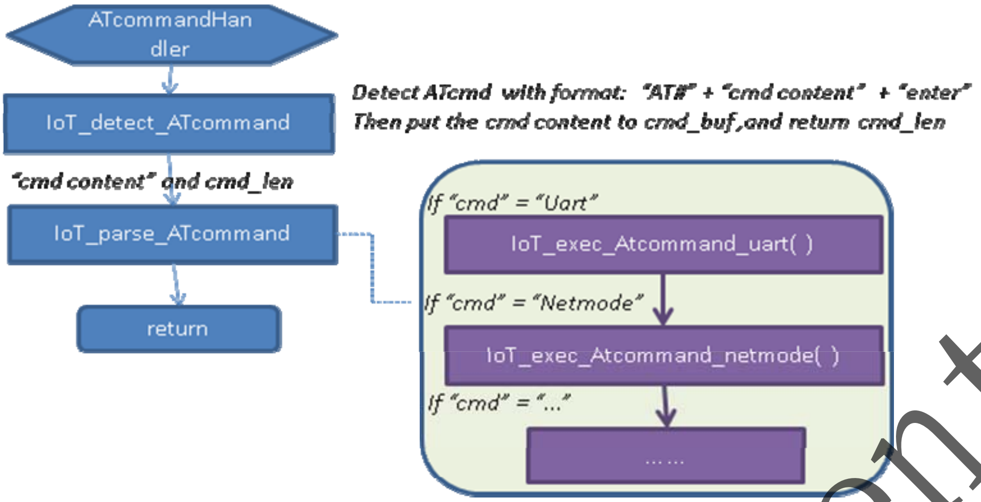
Step2: Add new member on IOT_USR_CFG structure

Step3: Add a default value macro, just like: #define DEFAULT_IOT_SERVER_IP {182.148.129.91}

Step4: Add load/reset implementation for Uart2Wifi on load_usr_cfg(), rest_usr_cfg()

8 AT COMMAND

8.1 Flow chart:


ATcommandHandler
IoT_detect_ATcommand
Detect ATcmd with format: "AT#" + "cmd content" + "enter"
Then put the cmd content to cmd_buf,and return cmd_len
"cmd content" and cmd_len
IoT_parse_ATcommand
return
If "cmd" = "Uart"
IoT_exec_Atcommand_uart( )
If "cmd" = "Netmode"
IoT_exec_Atcommand_netmode( )
If "cmd" = "..."
……

8.2 Function Description

- **INT32 IoT_parse_ATcommand (PUCHAR cmd_buf, INT32 at_cmd_len);**

  Description: This function parses AT command from the Uart port. It classifies the commands and call respective functions to parse the commands.

  Paramters:

  [IN]: cmd_buf ---- Pointer to AT command buffer

  [IN]: at_cmd_len ---- Length of the AT command.

  Return Values: Return negative if error occurs. Return zero, otherwise.

  Remarks: The command header "AT#" is removed before entering this function.

- **INT32 IoT_exec_ATcommand_uart (PUCHAR cmd_buf, INT32 at_cmd_len)**

  Description: This function parses uart AT command.

  Paramters:

  [IN]: cmd_buf ---- Pointer to uart AT command buffer

  [IN]: at_cmd_len ---- Length of the uart AT command.

  Return Value : Return negative if error occurs. Return zero, otherwise.

  Remark : None.

- **INT32 IoT_exec_ATcommand_netmode (PUCHAR cmd_buf, INT32 at_cmd_len)**

  Description: This function parses netmode AT command.

  Paramters:

  [IN]: cmd_buf ---- Pointer to netmode AT command buffer

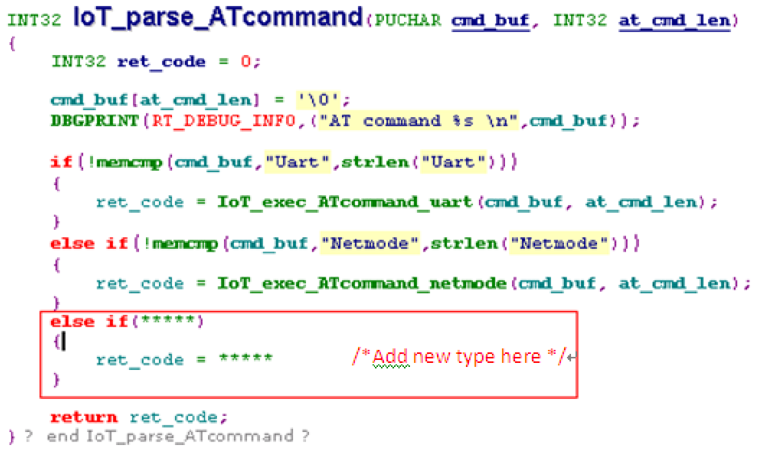
  [IN]: at_cmd_len ---- Length of the netmode AT command.

  Return Value : Return negative if error occurs. Return zero, otherwise.

  Remark: None.

8.3 How to add a new AT command

1) Add a new else/if branch for the new AT command type in the function IoT_parse_ATcommand.

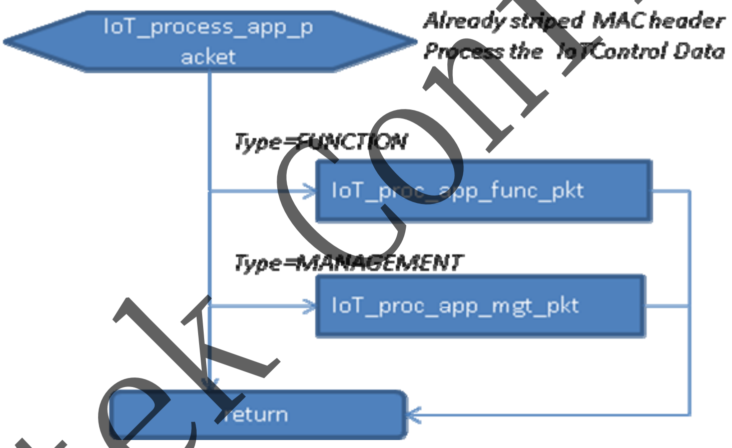


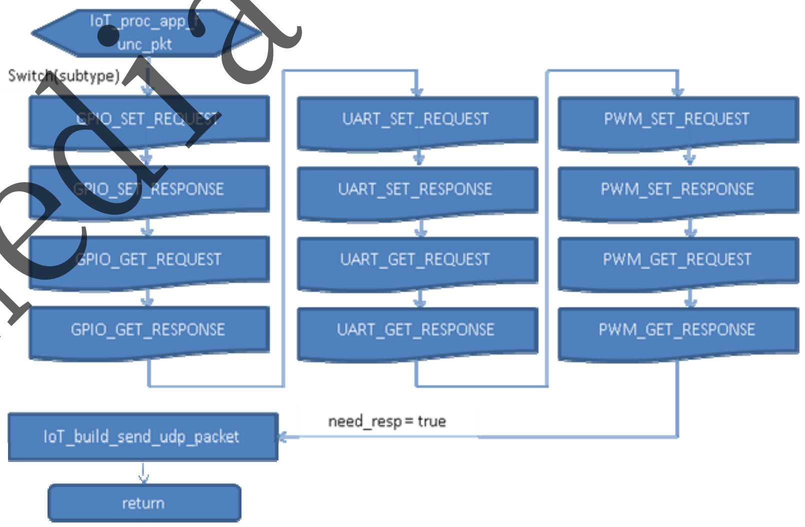
```
INT32 IoT_parse_ATcommand(PUCHAR cmd_buf, INT32 at_cmd_len)
{
    INT32 ret_code = 0;

    cmd_buf[at_cmd_len] = '\0';
    DBGPRINT(RT_DEBUG_INFO,("AT command %s \n",cmd_buf));

    if(!memcmp(cmd_buf,"Uart",strlen("Uart")))
    {
        ret_code = IoT_exec_ATcommand_uart(cmd_buf, at_cmd_len);
    }
    else if(!memcmp(cmd_buf,"Netmode",strlen("Netmode")))
    {
        ret_code = IoT_exec_ATcommand_netmode(cmd_buf, at_cmd_len);
    }
    else if(*****)
    {
        ret_code = *****          /*Add new type here */
    }

    return ret_code;
} ? end IoT_parse_ATcommand ?
```

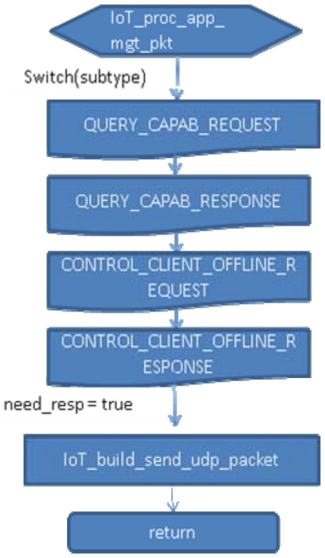
2) Add a new parsing function for the new type. IoT_exec_ATcommand_netmode can be a template.

9 DATA COMMAND

9.1 Flow chart:







9.2 Function Description

- **INT32 IoT_proc_app_packet(UCHAR sock_num, PUCHAR packet , UINT16 rawpacketlength);**

  Description: This function parses control protocol packet in the application layer.

  It removes protocol header and call respective functions to parse the data header and data content.

  Parameters

  [OUT]: sock_num ---- socket number of the current TCP/UDP transmission

  [OUT]: packet ---- Pointer to protocol header

  [OUT]: rawpacketlength ---- Length of the packet

  Return Value: Return zero.

  Remark: sock_num is used to distinguish different TCP/UDP transmission

- **INT32 IoT_proc_app_func_pkt (DataHeader* DataHeader, UINT16 FuncType, IoTPacketInfo *PacketInfo);**

  Description: This function parses control protocol packet of the function type.

  Parameters

  [OUT]: DataHeader ---- Pointer to data header

  [OUT]: FuncType ---- the function command type

  [OUT] : PacketInfo ---- packet information that is used when sending the response.

  Return Value: Return zero.

  Remark: None.

- **INT32 IoT_proc_app_mgt_pkt(DataHeader* Dataheader, UINT16 MgtType, IoTPacketInfo *PacketInfo);**

  Description: This function parses control protocol packet of the management type.

  Parameters

  [OUT] : Dataheader ---- Pointer to data header

  [OUT] : MgtType ---- the management command type

  [OUT]: PacketInfo ---- packet information that is used when sending the response.

  Return Value: Return zero.

  Remark: None.

9.3 How to add a new Data command

1. function related command

1) Add new command in the structure t_FunctionCommand.
2) Add a new select/case branch in the function IoT_proc_app_func_pkt

2. management related command

1) Add new command in the structure t_ManagementCommand.
2) Add a new select/case branch in the function IoT_proc_app_mgt_pkt

3. command of other class

1) Add new type in the structure t_CommandType.
2) Add new parsing function for the new type. IoT_proc_app_func_pkt can be a template.

3) Add a new type and its parsing function in the function IoT_proc_app_pkt

10 SECURITY APIS

- **VOID RT_ATE_Decrypt(UINT8 CipherBlock[]. UINT CipherBlockSize, UINT8 Key[],**
  UINT KeyLen, UINT8 PlainBlock[], UINT *PlainBlockSize);

  Description: This function is used to Decrypt data with ATE algorithm

  Parameters

  [IN]： CipherBlock[] ---- The block of cipher text, 16Bytes(128bit) each block
  [IN]：CipherBlockSize ---- The length of block of cipher text in bytes
  [IN]：Key[] ---- Cipher key , it maybe 16,24 or 32bytes
  [IN]：KeyLen ---- The length cipher key in bytes
  [IN] : PlanBlockSize ---- The length of allocated plain block in bytes

  [OUT] : PlanBlock[] ---- Plain block to store plain text
  [OUT] : PlanBlockSize ---- The length of real used plain block in bytes
  Return value：None

- **VOID RT_ATE_Encrypt(UINT8 PlainBlock[], UINT *PlainBlockSize, UINT8 Key[],**
  UINT KeyLen, UINT8 CipherBlock[], UINT CipherBlockSize);

  Description: This function is used to Decrypt data with ATE algorithm

  Parameters

  [IN] : PlanBlock[] ---- The block of Plain text, 16bytes(128bit) each block
  [IN] : PlanBlockSize ---- The length of block of plain text in bytes
  [IN]：Key[] ---- Cipher key , it maybe 16,24 or 32bytes
  [IN]：KeyLen ---- The length cipher key in bytes
  [IN]：CipherBlockSize ---- The length of allocated cipher block in bytes

  [OUT]：CipherBlock[] ---- cipter text
  [OUT]：CipherBlockSize ---- The length of real used cipher block in bytes
  Return value：None

- **INT32 RtmpPasswordHash(PSTRING password, PUCHAR ssid, INT32 ssid_len PUCHAR output);**

  Description: This function is used to calculate the PMK

  Parameters

  [IN] : password ---- ASCLL string up to 63 characters in length
  [IN] : ssid ---- octect string up to 32 octects
  [IN]：ssid_len ---- length of ssid in octects
  [OUT]：output ---- must be 40 octects in length and 0~32octects(256bits) is the key
  Return value：None

11 TIMER APIS

- **VOID cnmTimerInitTimer(IN P_TIMER_T prTimer,**
  IN PFN_MGMT_TIMEOUT_FUNC pfFunc,
  IN UINT_32 u4Data
  IN UINT_32 u4Data2)

  Description: This function is used to initialize a timer

Parameters

[IN]：prTimer ---- Pointer to a timer structure

[IN]：pfFunc ---- Pointer to the call back function

[IN]：u4Data ---- parameter for call back function

[IN]：u4Data2 ---- parameter for call back function

Return value：None

- **VOID cnmTimerStartTimer (IN P_TIMER_T prTimer, IN UINT_32 u4TimeoutMs)**

  Description: This function is used to start a timer

  Parameters

  [IN]：prTimer ---- Pointer to a timer structure

  [IN]：u4TimeoutMs ---- Timeout to issue the timer and callback function (unit:ms)

  Return value：None

- **VOID cnmTimerStopTimer(IN P_TIMER_T prTimer)**

  Description: This function is used to stop a timer

  Parameters

  [IN]：prTimer ---- Pointer to a timer structure

  Return value：None

There is a example on IoT_customer.c

- **UINT32 GetMsTimer(VOID)**

  Description: Get the time from system start (Unit: 1ms)

  Parameters

  Return value：the counter value

12 INTERFAE APIS

12.1 Flash

- **int32 spi_flash_read(uint32 addr, uint8 *data, uint16 len)**

  Description: This function is used to read specified data from flash

  Parameters

  [IN]：addr ---- The offset which the reading data stored on the flash

  [IN]：len ---- The data length need to read

  [OUT]：data ---- The pointer indicate the reading data

  Return value：0 means successful, non-zero means fail

- **int32 spi_flash_write(uint32 addr, uint8 *data, uint16 len)**

  Description: This function is used to write specified data to flash

  Parameters

  [IN]：addr ---- The offset which the data will be write on the flash

  [IN]：len ---- The data length need to write

  [IN]：data ---- The pointer indicate the writing data

  Return value: 0 means successful, non-zero means fail

  Notes：As the RAM limitation, the **len** must <= FLASH_OFFESET_WRITE_BUF (4KB)

  This API will erase Sector → store original data → merge the modified data → write back to sector

Thus, if you want to write some data to flash, please do not call spi_flash_erase_SE() or spi_flash_erase_BE() to erase flash again, but just call spi_flash_write().

- **void spi_flash_erase_SE(uint32 address)**

  Description: This function is used to erase the sector in which the address specifies.

  Parameters

  [IN]：addr ---- the address in flash to be erased

  Return value: None

- **void spi_flash_erase_BE(uint32 address)**

  Description: This function is used to erase the block in which the address specifies.

  Parameters

  [IN]：addr ---- the address in flash to be erased

  Return value: None

Note: 1. Due to the characteristic of flash, erase the sector/block where data is to be written is mandatory before write anything to flash.

2. The size of sector/block of one flash is different. Please check the datasheet of using flash.

3. above two APIs will erase a sector or a block, please consider if there are some data should not be erased in one sector/block before using those two APIs

12.2 UART

- **INT32 IoT_uart_input(UINT_8 *msg, INT32 count);**

  Description: This function reads a given length of data from the uart port.

  Parameters

  [IN] ：msg ---- Pointer to a uart rx buffer

  [OUT] ：count ---- Length of data to read

  Return Value： Return zero.

  Remark：None.

- **INT32 IoT_uart_output(UINT_8 *msg, INT32 count);**

  Description：This function writes a given length of data to the uart port.

  Parameters

  [OUT] ： msg ---- Pointer to a uart tx buffer

  [OUT] ： count ---- Length of data to write

  Return Value：Return zero.

  Remark：None.

12.3 LED / PWM

- **INT32 IoT_led_pwm (INT32 led_num, INT32 brightness);**

  Description：This function configures the brightness of a led.

  Parameters

  [OUT] ：led_num ---- In hardware pwm mode, led_num is led controller number, range (1~ 3)..

  [Ex: Led_num=1 , use Pin26 as Led/PWM

  Led_num=2, use pin31 as Led/PWM

  Led_num=3, use pin30 as Led/PWM]

  In software pwm mode, led_num is gpio number , range (0~ 4).

[Ex: Led_num=0 , use Pin31 as Led/PWM
Led_num=1, use pin30 as Led/PWM
Led_num=2, use pin29 as Led/PWM
Led_num=3, use pin28 as Led/PWM
Led_num=4, use pin27 as Led/PWM]

[OUT] ：brightness --- Brightness level of led.
In hardware pwm mode, range (0 ~ 5)
In software pwm mode, range (0 ~ 20).

Return Value: Return -1 if led_num is invalid. Return 0, otherwise.

Remark

1) Two pwm mode is supported.
If IOT_PWM_TYPE==1, hardware pwm mode is used.
If IOT_PWM_TYPE==2(default type), software pwm mode is used.
2) Level 0 is off. Level 5 is the brightest in hardware pwm mode.
Level 0 is off. Level 20 is the brightest in software pwm mode.
3) Software pwm mode consumes more CPU resources.
However, it has high frequency and more brightness levels.
4) In Hardware PWM mode,
if you want to cancel PWM mode for pin26, 31, 30 and set them as GPIO mode
need call **IoT_gpio_output(5 , 0), IoT_gpio_output(0 , 0), IoT_gpio_output(1 , 0)**
The pin and GPIO relationship, please refer to section:“GPIO/Pin Mode Set”

- **VOID IoT_software_pwm_addset (INT32 led_num, INT32 brightness)**

Description：This function configures a gpio pin to software pwm mode and set the brightness level.
It absolute same as **IoT_led_pwm()** in soft PWM mode

Parameters

[OUT] ：led_num ---- Specify the gpio number which is to be configured to software pwm mode.
Should be ranged from 0 to 4

[OUT] ：brightness --- Brightness level of led.
Available only In software pwm mode, should be ranged from 0 to 20.

Return Value: Return -1 if led_num is invalid. Return 0, otherwise.

Remark

1) This API is available. only if software pwm mode is used
2) Level 0 is off. Level 20 is the brightest in software pwm mode.

- **INT32 IoT_software_pwm_del (INT32 led_num)**

Description：This function changes a gpio pin from software pwm mode back to gpio mode

Parameters

[OUT] ：led_num ---- Specify the gpio number which is to be changed. Should be ranged from 0 to 4

Return Value: Return -1 if led_num is invalid. Return 0, otherwise.

Remark

1) This API is available. only if software pwm mode is used

12.4 GPIO

- **INT32 IoT_gpio_read (INT32 gpio_num, UINT8 *pVal, UINT8 *pPolarity);**

  Description：This function set the GPIO as input mode, and read it's input value

  Parameters

  [IN]: gpio_num ---- Specify the gpio number. Should be ranged from 0 to 6.

  [OUT]: pPolarity ---- read the gpio polarity, 0=Output Mode, 1=Input Mode

  [OUT]: pVal ---- read the gpio status, 0=low, 1=High

  Return Value: none

  Remarks:

  The GPIO/Pin Mode Set please refer to section: "GPIO/Pin Mode Set"

We can set one specific GPIO's mode/value at one time with following APIs

- **INT32 IoT_gpio_input(INT32 gpio_num, INT32 *input);**

  Description：This function set the GPIO as input mode, and read it's input value

  Parameters

  [IN]: gpio_num ---- Specify the gpio number. Should be ranged from 0 to 6

  [OUT] : input ---- the input status of the given gpio number. 0 is low. 1 is high.

  Return Value: Return -1 if gpio_num is invalid.
  Return -2 if input is invalid.
  Return zero, Otherwise.

  Remarks:

  The GPIO/Pin Mode Set please refer to section: "GPIO/Pin Mode Set"

- **INT32 IoT_gpio_output(INT32 gpio_num, INT32 output);**

  Description: This function configures the output status of a gpio.

  Parameters

  [IN] : gpio_num ---- Specify the gpio number. Should be ranged from 0 to 6

  [OUT]: output ---- the output status of the given gpio number. 0 is low. 1 is high.

  Return Values: Return -1 if gpio_num is invalid. Return -2 if output is invalid. Return 0, otherwise.

  Remarks:

  The GPIO/Pin Mode Set please refer to section: "GPIO/Pin Mode Set"

We can set several GPIOs mode/value at one time with following APIs

- **INT32 IoT_gpio_batch_modify_mode(INT32 output_bitmap);**

  Description: This function configures a batch of gpio pins to output mode

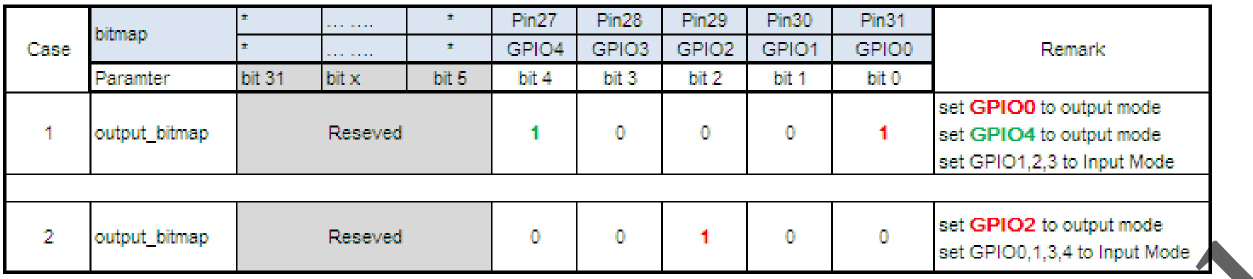
  Parameters

  [OUT] : output_bitmap ---- Specify the gpio output mode bitmap.
  Bit(i) stands for gpio(i). Should be ranged from 00000B to 11111B

  Return Values: Return 0

  Remarks:

  1. The GPIO/Pin Mode Set please refer to section: "GPIO/Pin Mode Set"
  2. If output_bitmap is 10001B, gpio0 and gpio4 will be set to output mode



Case	bitmap	*	… ….	*	Pin27	Pin28	Pin29	Pin30	Pin31	Remark
		*	… ….	*	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0	
	Paramter	bit 31	bit x	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
1	output_bitmap	Reseved			1	0	0	0	1	set GPIO0 to output mode set GPIO4 to output mode set GPIO1,2,3 to Input Mode
2	output_bitmap	Reseved			0	0	1	0	0	set GPIO2 to output mode set GPIO0,1,3,4 to Input Mode

- **INT32 IoT_gpio_batch_modify_output_value(INT32 output_bitmap, INT32 value_bitmap);**

  Description: This function configures a batch of gpio pins to output high.

  Parameters

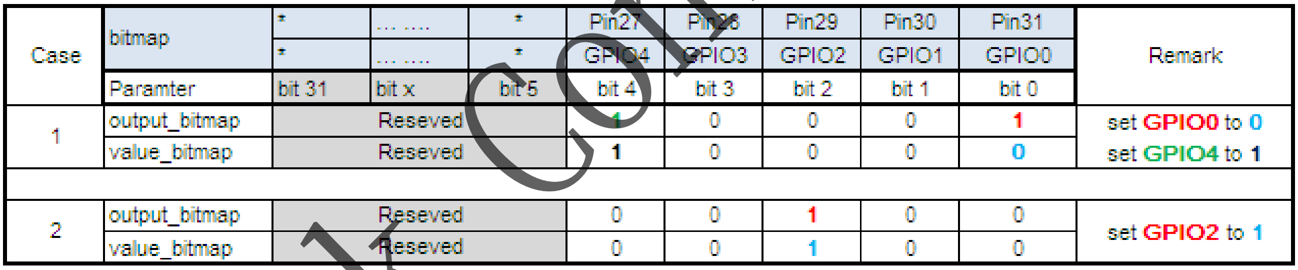
  [OUT] : output_bitmap ---- Specify the gpio output mode bitmap.
  Bit(i) stands for gpio(i). Should be ranged from 00000B to 11111B.

  [OUT]: value_bitmap ---- Specify the gpio output status bitmap.
  Bit(i) stands for gpio(i). Should be ranged from 00000B to 11111B.

  Return Values: Return 0

  Remarks:

  1. The GPIO/Pin Mode Set please refer to section: “GPIO/Pin Mode Set”
  2. This function does not change gpio pins to output mode. It modifies the output value only
  3. If output_bitmap is **10001**B, and value_bitmap is **1**000**0**B,
     gpio0 will be set to low, and **gpio4** will be set to high



Case	bitmap	*	… ….	*	Pin27	Pin28	Pin29	Pin30	Pin31	Remark
		*	… ….	*	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0	
	Paramter	bit 31	bit x	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
1	output_bitmap	Reseved			1	0	0	0	1	set GPIO0 to 0 set GPIO4 to 1
	value_bitmap	Reseved			1	0	0	0	0	
2	output_bitmap	Reseved			0	0	1	0	0	set GPIO2 to 1
	value_bitmap	Reseved			0	0	1	0	0	

- **UINT8 IoT_Cust_Set_GPIINT_MODE(IN UINT8 GPIO_Num, IN UINT8 Val)**

  Description: Set GPIO interrupt mode

  Parameters

  [IN] : GPIO_Num ---- [0~6].

  [IN]: Val ---- [0~4]

  0: no trigger, 1: falling edge trigger

  2: rising edge trigger 3:both falling adn rising edge trigger

  Return Values: 0- Success, 1-invalid input

- **UINT8 IoT_Cust_Get_GPIINT_MODE(OUT UINT16* pGPI_INT_MODE)**

  Description: Set GPIO interrupt mode

  Parameters

  [OUT] : GPI_STS

  [1:0]: GPIO1 Interrupt mode

  [3:2]: GPIO0 Interrupt mode

  [5:4]: GPIO2 Interrupt mode

  [7:6]: GPIO3 Interrupt mode

[9:8]: GPIO4 Interrupt mode
[11:10]: GPIO5 Interrupt mode
[13:12]: GPIO6 Interrupt mode
For each GPIO's interrupt mode
0: no trigger, 1: falling edge trigger
2: rising edge trigger 3:both falling adn rising edge trigger
Return Values: None

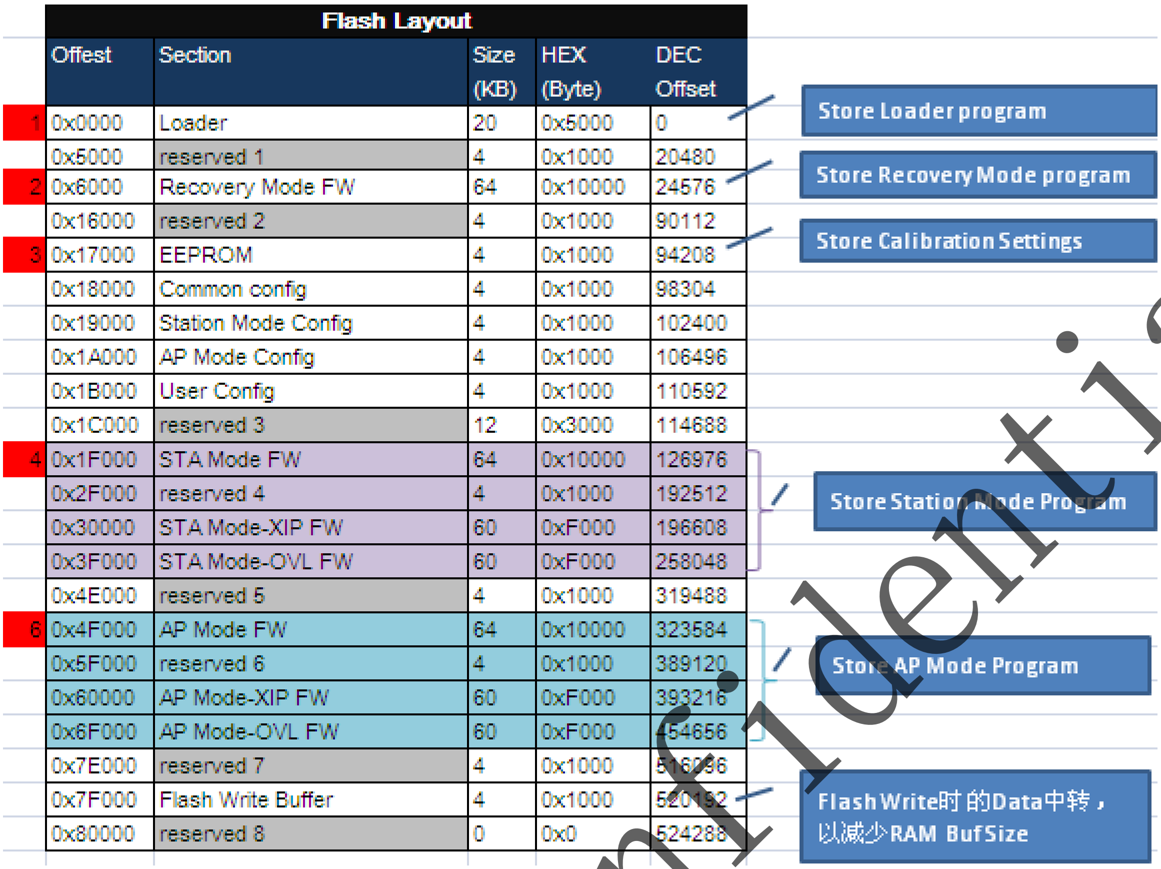
- **VOID IoT_Cust_GPIINT_Hdlr(IN UINT8 GPI_STS);**
  Description: This Handler shall be called as any GPIO Interrput be triggered
  Parameters
  [IN] : GPIO_Num ---- [0~6]. GPIO0~6 Interrupt status
  Return Values: 0- Success, 1-invalid input

12.5 GPIO/Pin Mode Set

Current, We use IOT_PWM_TYPE=2, The GPIO list as below:

	IOT_PWM_TYPE		
CHIP Pin	**0**	**1** **(HW PWM Mode)**	**2** **(SW PWM Mode)**
Pin 31	GPIO 0	PWM2	GPIO0 / PWM1
Pin 30	GPIO 1	PWM3	GPIO1 / PWM2
Pin 29	GPIO 2	GPIO 2	GPIO2 / PWM3
Pin 28	GPIO 3	GPIO 3	GPIO3 / PWM4
Pin 27	GPIO 4	GPIO 4	GPIO4 / PWM5
Pin 26	Uart Tx	PWM1 / Uart Tx	Uart Tx
Pin 25	Uart Rx	Uart Rx	Uart Rx
Remark		PWM: 20Hz, Level(0~5) Level 0 =off Level 5= brightest	PWM: 50Hz, Level(0~20) Level 0 =off Level 20= brightest

13 FLASH PARTITIONS



Flash Layout

	Offest	Section	Size (KB)	HEX (Byte)	DEC Offset
1	0x0000	Loader	20	0x5000	0
	0x5000	reserved 1	4	0x1000	20480
2	0x6000	Recovery Mode FW	64	0x10000	24576
	0x16000	reserved 2	4	0x1000	90112
3	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
4	0x1F000	STA Mode FW	64	0x10000	126976
	0x2F000	reserved 4	4	0x1000	192512
	0x30000	STA Mode-XIP FW	60	0xF000	196608
	0x3F000	STA Mode-OVL FW	60	0xF000	258048
	0x4E000	reserved 5	4	0x1000	319488
6	0x4F000	AP Mode FW	64	0x10000	323584
	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
	0x80000	reserved 8	0	0x0	524288

Store Loader program

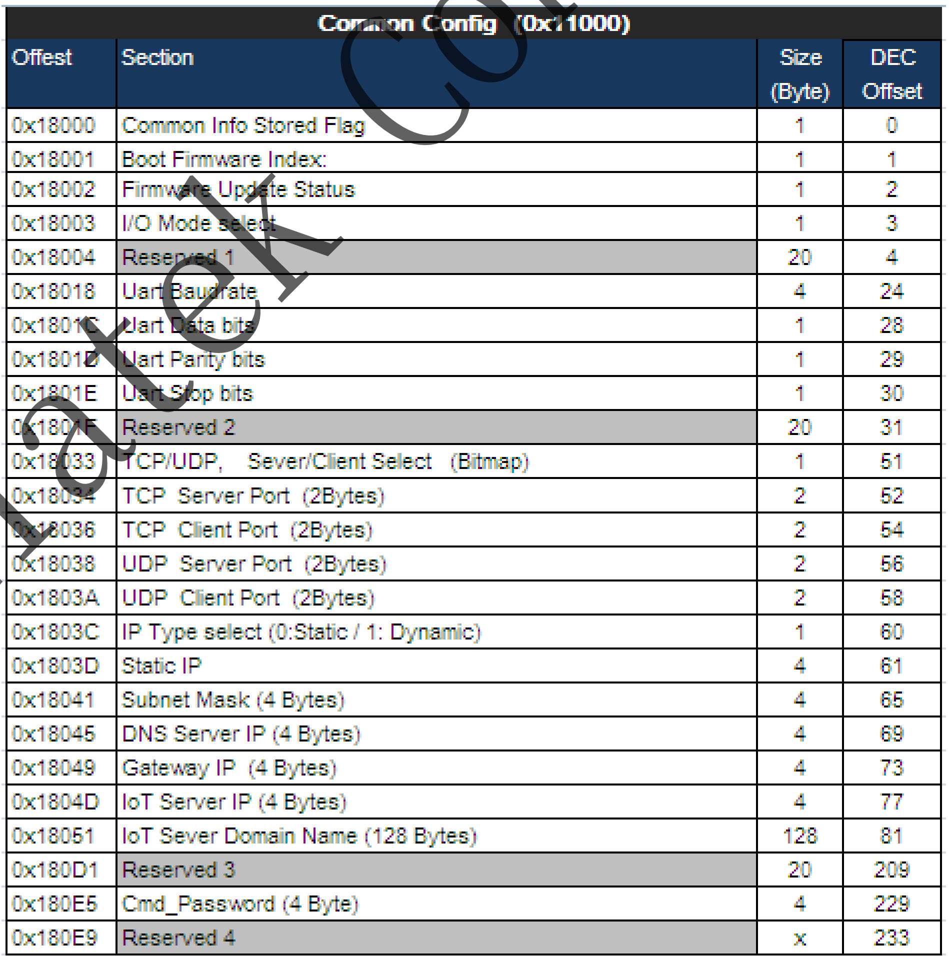
Store Recovery Mode program

Store Calibration Settings

Store Station Mode Program

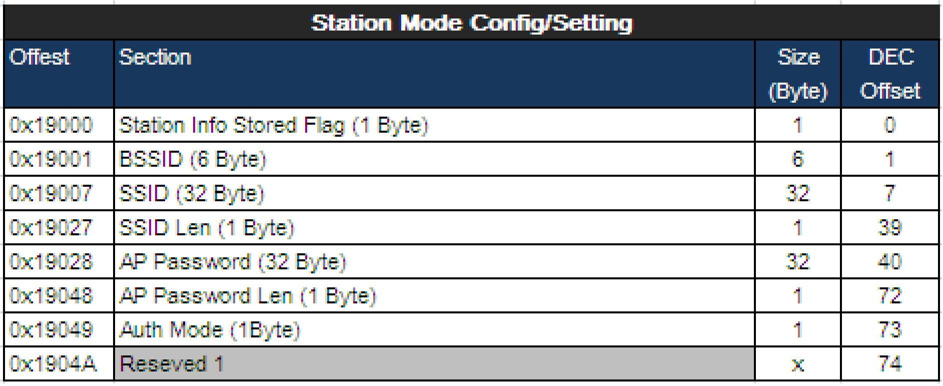
Store AP Mode Program

Flash Write时的Data中转，以减少RAM Buf Size

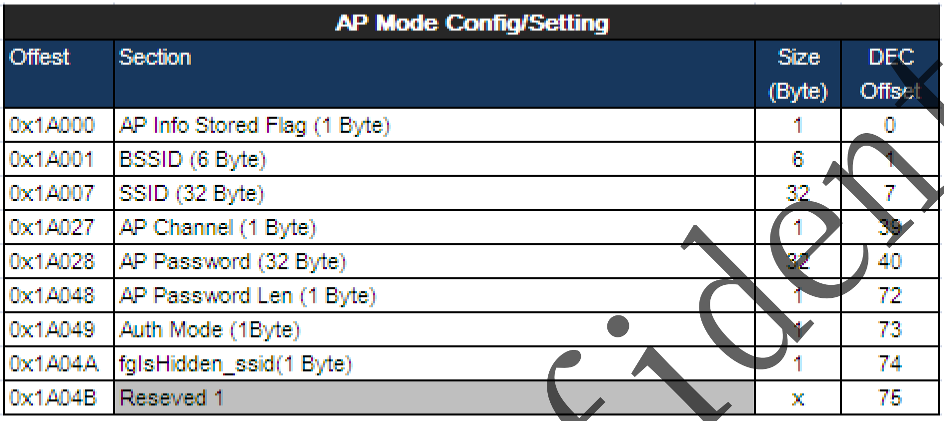


Common Config (0x11000)

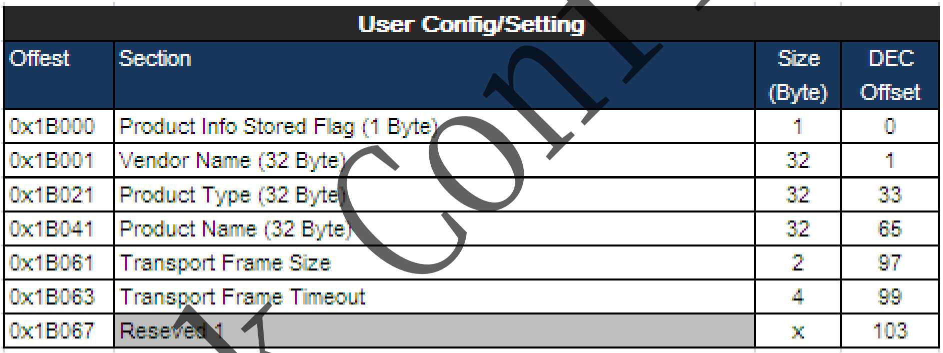
Offest	Section	Size (Byte)	DEC Offset
0x18000	Common Info Stored Flag	1	0
0x18001	Boot Firmware Index:	1	1
0x18002	Firmware Update Status	1	2
0x18003	I/O Mode select	1	3
0x18004	Reserved 1	20	4
0x18018	Uart Baudrate	4	24
0x1801C	Uart Data bits	1	28
0x1801D	Uart Parity bits	1	29
0x1801E	Uart Stop bits	1	30
0x1801F	Reserved 2	20	31
0x18033	TCP/UDP, Sever/Client Select (Bitmap)	1	51
0x18034	TCP Server Port (2Bytes)	2	52
0x18036	TCP Client Port (2Bytes)	2	54
0x18038	UDP Server Port (2Bytes)	2	56
0x1803A	UDP Client Port (2Bytes)	2	58
0x1803C	IP Type select (0:Static / 1: Dynamic)	1	60
0x1803D	Static IP	4	61
0x18041	Subnet Mask (4 Bytes)	4	65
0x18045	DNS Server IP (4 Bytes)	4	69
0x18049	Gateway IP (4 Bytes)	4	73
0x1804D	IoT Server IP (4 Bytes)	4	77
0x18051	IoT Sever Domain Name (128 Bytes)	128	81
0x180D1	Reserved 3	20	209
0x180E5	Cmd_Password (4 Byte)	4	229
0x180E9	Reserved 4	x	233



Station Mode Config/Setting			
Offest	Section	Size (Byte)	DEC Offset
0x19000	Station Info Stored Flag (1 Byte)	1	0
0x19001	BSSID (6 Byte)	6	1
0x19007	SSID (32 Byte)	32	7
0x19027	SSID Len (1 Byte)	1	39
0x19028	AP Password (32 Byte)	32	40
0x19048	AP Password Len (1 Byte)	1	72
0x19049	Auth Mode (1Byte)	1	73
0x1904A	Reseved 1	x	74



AP Mode Config/Setting			
Offest	Section	Size (Byte)	DEC Offset
0x1A000	AP Info Stored Flag (1 Byte)	1	0
0x1A001	BSSID (6 Byte)	6	1
0x1A007	SSID (32 Byte)	32	7
0x1A027	AP Channel (1 Byte)	1	39
0x1A028	AP Password (32 Byte)	32	40
0x1A048	AP Password Len (1 Byte)	1	72
0x1A049	Auth Mode (1Byte)	1	73
0x1A04A	fglsHidden_ssid(1 Byte)	1	74
0x1A04B	Reseved 1	x	75



User Config/Setting			
Offest	Section	Size (Byte)	DEC Offset
0x1B000	Product Info Stored Flag (1 Byte)	1	0
0x1B001	Vendor Name (32 Byte)	32	1
0x1B021	Product Type (32 Byte)	32	33
0x1B041	Product Name (32 Byte)	32	65
0x1B061	Transport Frame Size	2	97
0x1B063	Transport Frame Timeout	4	99
0x1B067	Reseved 1	x	103

Note: 1. As the limitation of RAM size, while do flash read/write at a time,
only **256B** of data can be read from FLASH to RAM, (use IoTpAd.flash_rw_buf[256])
Then rewrite the data to corresponding place after being modified.

14 COMPILER SETUP

Please refer to description on the Andes web

http://forum.andestech.com/viewtopic.php?f=23&t=576&p=672

http://forum.andestech.com/viewtopic.php?f=23&t=587

15 AT COMMAND USAGE

15.1 Display version

Command: **Ver**

Argument Descriptions: None

Example: AT#Ver+enter

15.2 Reboot the system

Command: **Reboot**

Argument Descriptions: None

Example: AT#Reboot+enter

15.3 Set Default

Command: **Default**

Argument Descriptions: -s <channel number>

Example: AT#Default+enter

15.4 Switch channel

Command: **Channel**

Argument Descriptions: -s <channel number>

Example: AT#Channel -s 6+enter

15.5 Configure UART interface

Command: **Uart**

Argument Descriptions:

-b <baud rate> (57600, 115200, 230400 , …)
-w <data bits> (5, 6, 7, 8)
-p <parity> (0 for no parity, 1 for odd, 2 for even)
-s <stop bits> (1 for 1bit, 2 for 2bits, 3 for 1.5bits)

Example: AT#Uart -b 57600 -w 7 -p 1 -s 1 +enter

Remarks: dlr= round(systemclock/(16* baudrate), 0)
actual baudrate = **systemclock/(16* dlr)**
You can find more supported baudrate for your system according the formula and experiment

15.6 Update Firmware from Uart

Command: **UpdateFW**

Argument Descriptions: -t <flash area type>

Example: AT# UpdateFW +enter

Remarks: should be enabled on Recovery mode, X-modem shall be start up after implement this command