

**MT7681 IoT Wi-Fi Firmware**

**Programming Guide**

Version: 0.08

Release date: 2014-5-30

© 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 |
| 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 |
| 05.30.2014 | v0.08 | Jinchuan | Add APIs for Rxfilter control  Add Set channel API  Add Rx Packet API: STARxDoneInterruptHandle  Add Tx Packet API: mt76xx\_dev\_send  Add APIs for MD5: RT\_MD5() … |
|  |  |  |  |
|  |  |  |  |

**Contents**

[1 Introduction 5](#_Toc390445661)

[1.1 Flow Chart Symbols 5](#_Toc390445662)

[1.2 Keywords 5](#_Toc390445663)

[2 SW Structure 6](#_Toc390445664)

[2.1 Flow chart 6](#_Toc390445665)

[3 File Structure 6](#_Toc390445666)

[4 FIRMWARE boot UP flow 7](#_Toc390445667)

[5 Customer Hook Function 7](#_Toc390445668)

[6 Customer UART-TO-WIFI Function 8](#_Toc390445669)

[7 Flash settings Load/Storage 9](#_Toc390445670)

[8 AT Command 12](#_Toc390445671)

[8.1 Flow chart: 12](#_Toc390445672)

[8.2 Function Description 12](#_Toc390445673)

[8.3 How to add a new AT command 13](#_Toc390445674)

[9 Data Command 13](#_Toc390445675)

[9.1 Flow chart: 13](#_Toc390445676)

[9.2 Function Description 14](#_Toc390445677)

[9.3 How to add a new Data command 14](#_Toc390445678)

[10 WIFI MAC APIS 15](#_Toc390445679)

[10.1 Rx Filter Control 15](#_Toc390445680)

[10.2 MAC Control 15](#_Toc390445681)

[11 Security APIS 16](#_Toc390445682)

[12 Timer APIs 18](#_Toc390445683)

[13 Interfae APIs 18](#_Toc390445684)

[13.1 Flash 18](#_Toc390445685)

[13.2 UART 19](#_Toc390445686)

[13.3 LED / PWM 19](#_Toc390445687)

[13.4 GPIO 21](#_Toc390445688)

[13.5 GPIO/Pin Mode Set 23](#_Toc390445689)

[14 Flash Partitions 24](#_Toc390445690)

[15 Compiler Setup 26](#_Toc390445691)

[16 AT Command Usage 27](#_Toc390445692)

[16.1 Display version 27](#_Toc390445693)

[16.2 Reboot the system 27](#_Toc390445694)

[16.3 Set Default 27](#_Toc390445695)

[16.4 Switch channel 27](#_Toc390445696)

[16.5 Configure UART interface 27](#_Toc390445697)

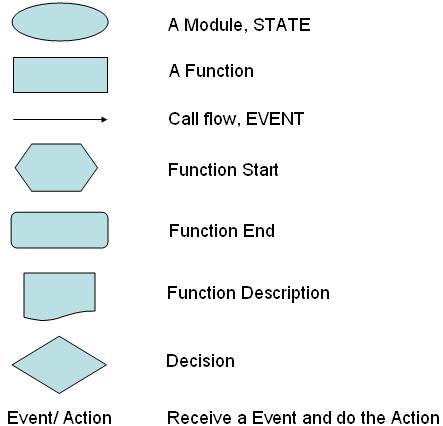
[16.6 Update Firmware from Uart 27](#_Toc390445698)

# 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

## Flow Chart Symbols



## Keywords

BBP: Base Band Processor

SEC: Security Engine

PBF: Packet Buffer

PDMA: Programmable Direct Memory Access

FCE: Frame Control Engine

# SW Structure

## 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

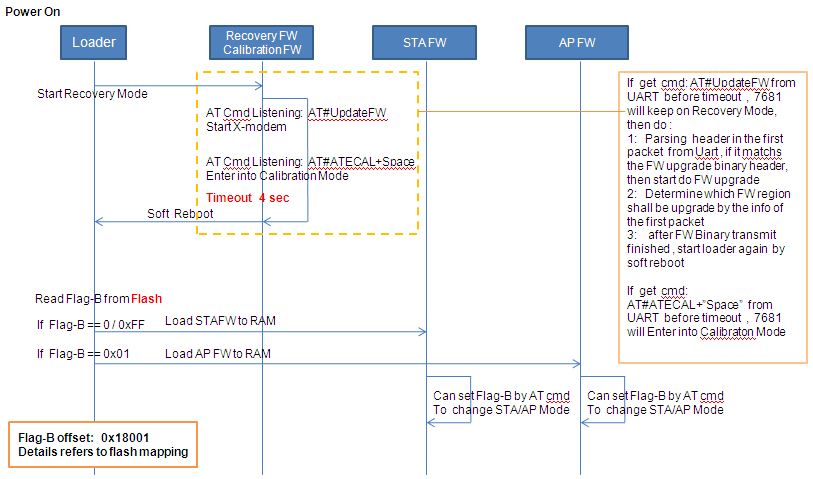
Data Command handler: handle the Data command which received from Wi-Fi

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

# File Structure

|  |  |
| --- | --- |
|  | \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) |

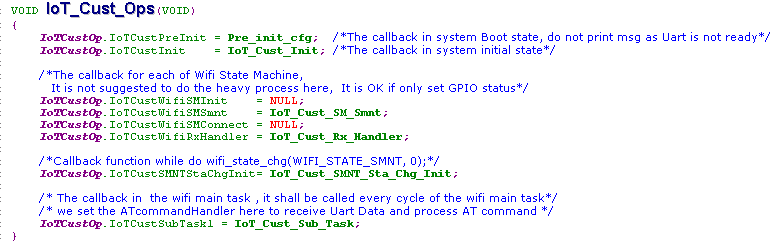
# FIRMWARE boot UP flow



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

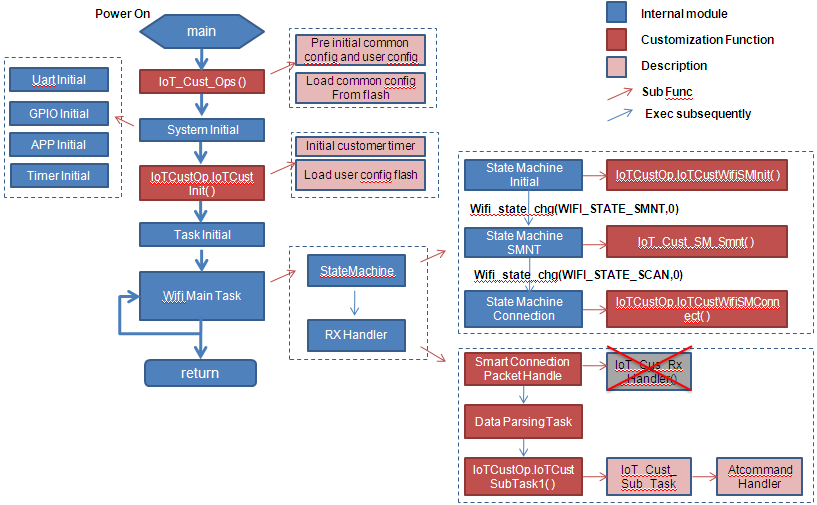
# Customer Hook Function

Iot\_customer.c



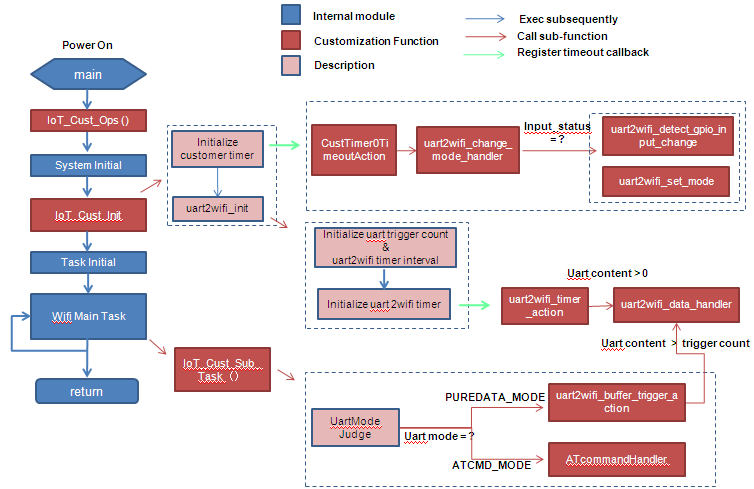
IoT\_Cust\_Ops() is used to register callback function , which could be called by Wifi main function

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



# 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,

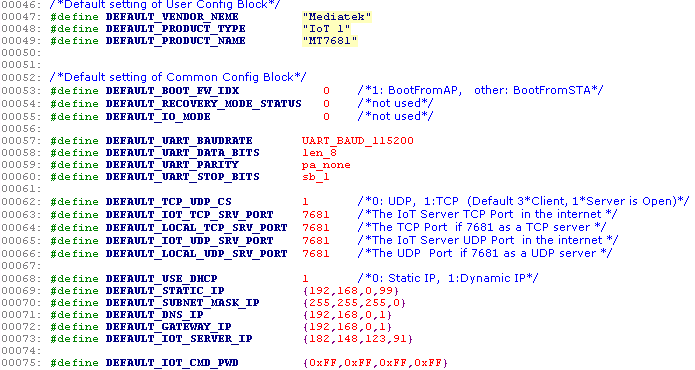
so you must set DATAPARSING\_UARTRX\_SUPPORT to 0 first.

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

# 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



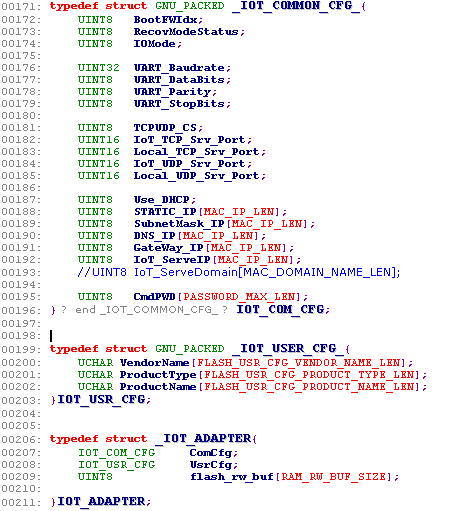
There are two structures: IOT\_COM\_CFG, IOT\_USR\_CFG

**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



**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()

# AT Command

## Flow chart:

## 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 netmodeAT command buffer

[IN]： at\_cmd\_len ---- Length of the netmodeAT command.

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

Remark： None.

## 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.



2) Add a new parsing function for the new type. IoT\_exec\_ATcommand\_netmode can be a template.

# Data Command

## Flow chart:

## 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** **Io****T\_****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.

## 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

1. **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

1. **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

# WIFI MAC APIS

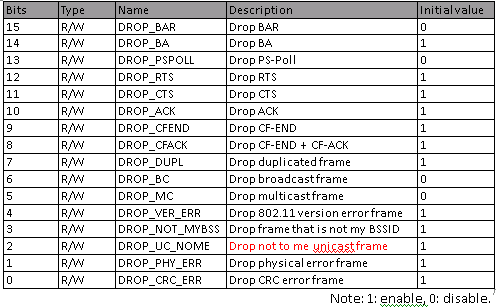
## Rx Filter Control

* **UINT16 IoT\_Get\_RxFilter(VOID)**

Description: Get Rx filter about frame receive

Parameters None

Return value：Rx filter value, the definition of the setting as blow table



* **UINT16 IoT\_Set\_RxFilter(UINT16 Value)**

Description: Set Rx filter about frame receive

Parameters

[IN]： Value ---- The Rx Filter value , the settings refers to above table

Return value：the value actually write to RxFilter, it should same as Input Parameter

Notes: when bit6=1, both BC/MC packet shall be dropped, even if bit5=0.

## MAC Control

* **IoT\_Cmd\_Set\_Channel(UINT8 Channel)**

Description: switch Current Channel

Parameters

[IN]： Channel ---- The new channel shall be switched to, Channel Range [0~13]

Return value：None

* **INT rtmp\_bbp\_set\_bw(UINT8 bw)**

Description: switch Current Bandwidth

Parameters

[IN]： bw ---- The new bandwidth shall be switched to, bandwidth Range [0:BW20, 1:BW40]

Return value： Always TRUE

* **BOOLEAN STARxDoneInterruptHandle (pBD\_t pBufDesc)**

Description: Wifi MAC Rx packet handler

Parameters

[IN]： pBufDesc ---- The received Rx descriptor, include Rx MAC content and Rx Info

Return value： FALSE if the Rx packet is invalid, TRUE if the Rx packet is valid

* **VOID mt76xx\_dev\_send(void)**

Description: Send uip Tx packet

Parameters None

Return value： None

Notes: This function will copy data from “**uip\_buf**” to Tx Descriptor and send it to wireless

# 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

* **VOID \_\_romtext RT\_MD5 ( const UINT8 Message[], UINT MessageLen, UINT8 DigestMessage[])**

Description: MD5 algorithm

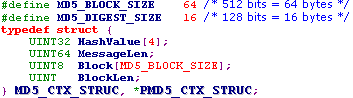
Parameters

[IN]：message ---- Message context

[IN]：messageLen ---- The length of message in bytes

[OUT]： digestMessage ---- Digest message

Return Value: None

* **VOID \_\_romtext RT\_MD5\_Init (MD5\_CTX\_STRUC \*pMD5\_CTX)**

Description: Initial Md5\_CTX\_STRUC

Parameters

[IN]：pMD5\_CTX Pointer to Md5\_CTX\_STRUC

Return Value: None

* **VOID \_\_romtext RT\_MD5\_Append (MD5\_CTX\_STRUC \*pMD5\_CTX,**

**const UINT8 Message[],**

**UINT MessageLen)**

Description: The message is appended to block. If block size > 64 bytes, the MD5\_Hash will be called.

Parameters:

pMD5\_CTX Pointer to MD5\_CTX\_STRUC

message Message context

messageLen The length of message in bytes

Return Value: None

* **VOID \_\_romtext RT\_MD5\_End (MD5\_CTX\_STRUC \*pMD5\_CTX, UINT8 DigestMessage[])**

Description:

1. Append bit 1 to end of the message

2. Append the length of message in rightmost 64 bits

3. Transform the Hash Value to digest message

Parameters

[IN] pMD5\_CTX Pointer to MD5\_CTX\_STRUC

[OUT] digestMessage Digest message

Return Value: None

# 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

# Interfae APIs

## 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

## 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.

## 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.

1. 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.

1. Software pwm mode consumes more CPU resources.

However, it has high frequency and more brightness levels.

1. 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

## 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, UINT32 \*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.

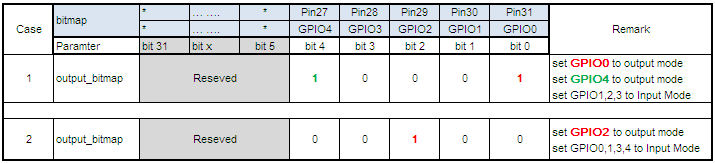
Bit(i) stands for gpio(i). Should be ranged from 00000B to 11111B

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



* **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

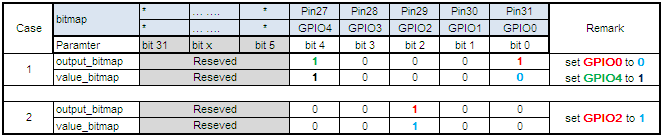
Remarks:

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



* **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

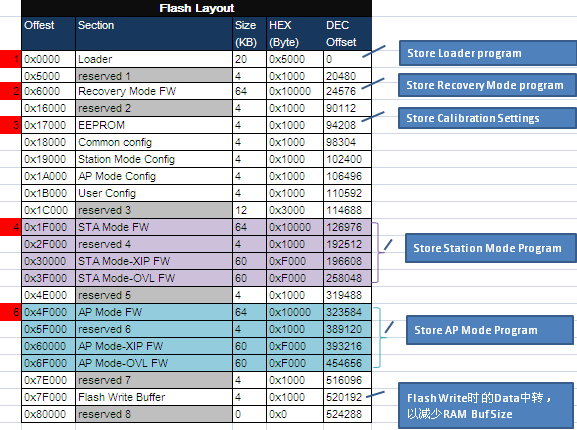
Return Values: 0- Success, 1-invalid input

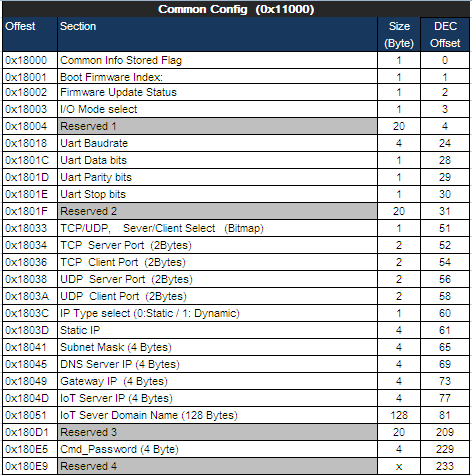
## GPIO/Pin Mode Set

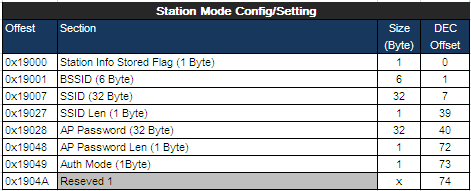
Current, We use IOT\_PWM\_TYPE=2, The GPIO list as below:

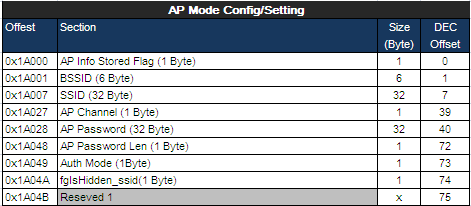
|  |  |  |  |
| --- | --- | --- | --- |
|  | **IOT\_PWM\_TYPE** | | |
| **CHIP Pin** | **0** | **1 (HW PWM Mode)** | **2 (SW PWM Mode)** |
| Pin 31 | GPIO 0 | PWM2 | GPIO0 / PWM1 |
| Pin 30 | GPIO 1 | PWM3 | GPIO1 / PWM2 |
| Pin 29 | GPIO 2 | GPIO 2 | GPIO2 / PWM3 |
| Pin 28 | GPIO 3 | GPIO 3 | GPIO3 / 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 |

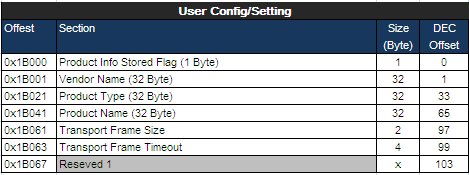
# Flash Partitions

****

****

****

****

****

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.

# 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>

# AT Command Usage

## Display version

Command: **Ver**

Argument Descriptions: None

Example: AT#Ver+enter

## Reboot the system

Command: **Reboot**

Argument Descriptions: None

Example: AT#Reboot+enter

## Set Default

Command: **Default**

Argument Descriptions: -s <channel number>

Example: AT#Default+enter

## Switch channel

Command: **Channel**

Argument Descriptions:  -s <channel number>

Example: AT#Channel -s 6+enter

## 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

## 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