MT7681 IoT Wi-Fi Firmware

Programming Guide

Version: 0.03

Release date: 2014-1-24

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

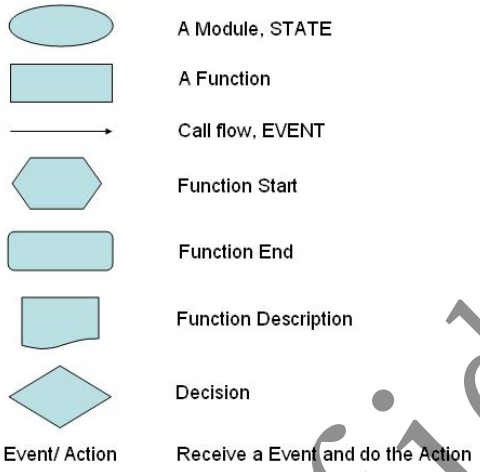
Contents

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

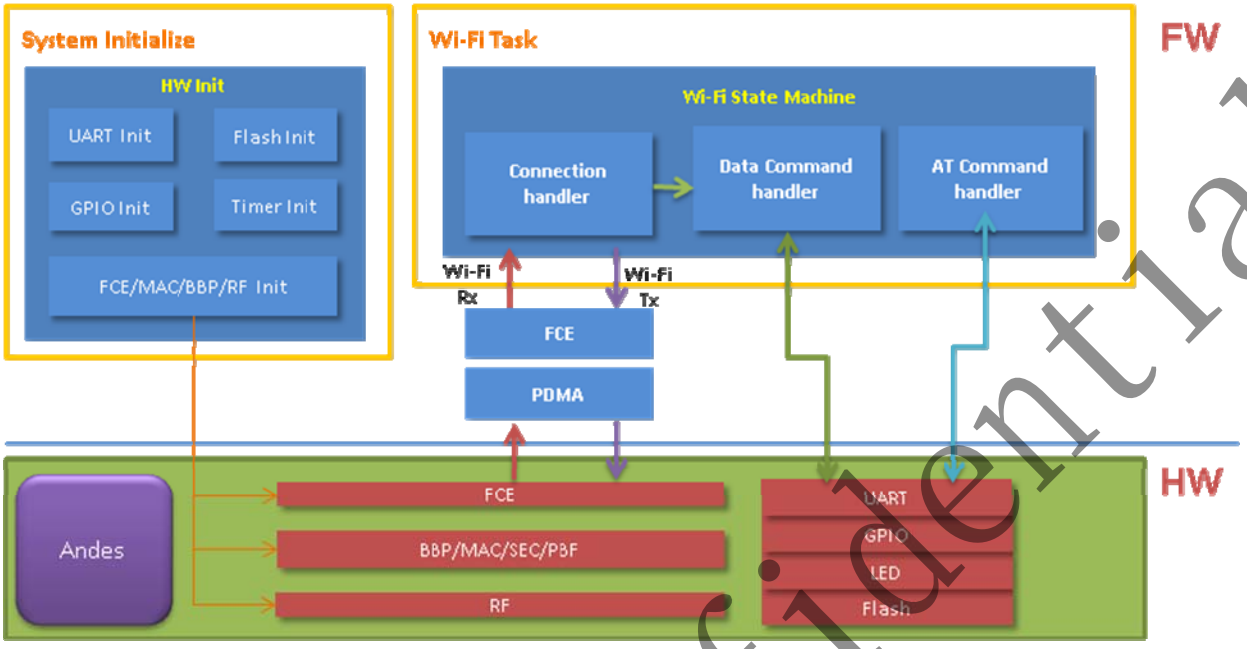
PBF: Packet Buffer

PDMA: Programmable Direct Memory Access

FCE: Frame Control Engine

2 SW STRUCTURE

2.1 Flow chart



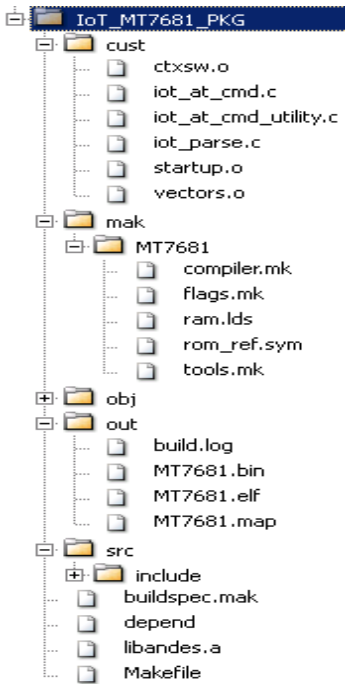
HW init: initialize the HW registers to enable HW interfaces and Wi-Fi function

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



\mak\MT7681\ .store the configuration for compiler or linker

\mak\MT7681\flags.mk ,store the macro which shall be used in all source files

\out\ , store the files which created by compiler

\out\build.log , the compiling log

\out\MT7681.bin , the target binary file

\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

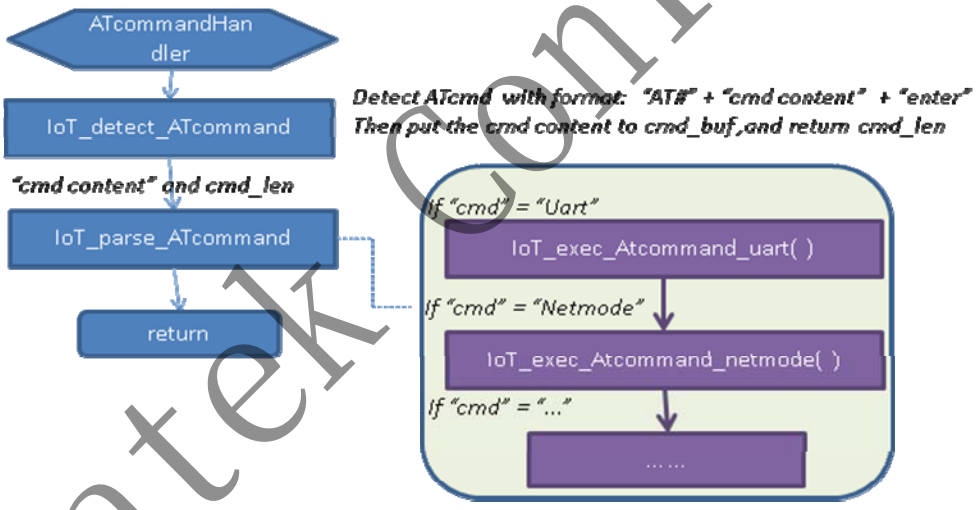
\cust\iot_parse.c ,handle the Data command which received from Wi-Fi

\src\include , the header files

libandes.a , library for MT7681

4 AT COMMAND

4.1 Flow chart:



4.2 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.

4.3 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.

5 DATA COMMAND

5.1 Flow chart:

5.2 Function Description

- **INT32 IoT_proc_app_packet(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]: packet ---- Pointer to protocol header
  [OUT]: rawpacketlength ---- Length of the packet
  Return Value: Return zero.
  Remark: None.

- **INT32 IoT_proc_app_func_pkt (struct t_DataHeader* DataHeader, UINT_8 FuncType, struct t_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(struct t_DataHeader* DataHeader, UINT_8 MgtType, struct t_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.

5.3 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

6 INTERFAE APIS

6.1 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

6.2 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.

6.3 LED

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

  Description：This function configures the brightness of a led.

  Parameters

  [OUT] ： led_num ---- Specify the led controller. Should be ranged from 1 to 3

  [OUT] ： brightness --- Brightness level of led. Should be ranged from 0 to 5.

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

Remark : Level 0 is off. Level 5 is the brightest.

6.4 GPIO

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

Description：This function reads the input status of a gpio

Parameters

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

[IN] : 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: Gpio 0 collides with led 2.

Gpio 1 collides with led 3.

Gpio 5 collides with led 1 and uart tx.

Gpio 6 collides with uart rx

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

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

Parameters

[OUT] : 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: Gpio 0 collides with led 2.

Gpio 1 collides with led 3.

Gpio 5 collides with led 1 and uart tx.

Gpio 6 collides with uart rx

7 FLASH PARTITIONS

Offest	Section	Size
0x00000	loader	4KB
0x01000	reserved	4KB
0x02000	STA Mode FW	64KB
0x12000	reserved	4KB
0x13000	AP Mode FW	64KB
0x23000	reserved	4KB
0x24000	Calibration FW	64KB
0x34000	reserved	4KB
0x35000	updated Mode FW	64KB
0x45000	reserved	4KB
0x46000	Common config	4KB
0x47000	Station Mode Config	4KB
0x48000	AP Mode Config	4KB
0x49000	reserved	4KB
0x4A000	EEPROM	4KB
0x4B000	reserved	4KB
0x4C000	Flash Write Buffer	4KB
0x4D000	reserved	12KB
0x50000	User config	4KB
0x51000	reserved	

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Common config								
0x46000						mode select	ap mode status	station mode status
0x46001						update FW type: 00 station mode 01 ap mode 10 calibra		FW update status

Station mode config/setting	
0x47000	BSSID (6 byte)
0x47006	SSID (32 byte)
0x47026	Password (64 byte)
0x47066	auth mode

AP mode config/setting	
0x48000	BSSID (6 byte)
0x48006	SSID (32 byte)
0x48026	Password (64 byte)
0x48066	auth mode
0x48067	fglshidden_ssid

Note: 1. the space between 0x00000 and 0x53000 is specified for WIFI function. Please do not modify it.

2. The space of 4KB from 0x56000 for user configuration is optional.

3. Please download loader.bin to 0x00 and MT7681.bin of station mode to 0x02000.

4. As the limitation of RAM size, only 1KB of data can be read from FLASH to RAM, and then rewrite the data to corresponding place after being modified.

8 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

9 AT COMMAND USAGE

9.1 Display version

Command: **Ver**

Argument Descriptions: None

Example: AT#Ver+enter

9.2 Reboot the system

Command: **Reboot**

Argument Descriptions: None

Example: AT#Reboot+enter

9.3 Switch channel

Command: **Channel**

Argument Descriptions: -s <channel number>

Example: AT#Channel -s 6+enter

9.4 Configure UART interface

Command: **Uart**

Argument Descriptions:

-s <baud rate>

-w <data bits>

-p <parity>

-s <stop bits>

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

9.5 Send a TCP connection request to a TCP server

Command: **Tcp_Connect**

Argument Descriptions:

-t <protocol type> (type 6 is for IPv4)

-a <ip address>

-p <port number>

Example: AT#Tcp_Connect -t 6 -a 192.168.1.131 -p 1234 +enter

9.6 Send data via a TCP connection

Command: **Tcp_Send**

Argument Descriptions:

-s <socket index> (start from 0)

-d <data content>

Example: AT#Tcp_Send -s 0 -d test data +enter

9.7 Listen to tcp connection request from the network

Command: **Tcp_Listen**

Argument Descriptions: -p <port number>

Example: AT#Tcp_Listen -p 7682 +enter

9.8 Remove a TCP connection

Command: **Tcp_Disconnect**

Argument Descriptions: -s <socket index>

Example: AT#Tcp_Disconnect -s 0 +enter