AN0400
Ameba-D Application Note

Realtek Semiconductor Corp.
No. 2, Innovation Road II, Hsinchu Science Park, Hsinchu 300, Taiwan
Tel.: +886-3-578-0211. Fax: +886-3-577-6047
www.realtek.com

Customers agree that it has the necessary expertise to select the product with the appropriate qualification designation for their applications and that proper product selection is at Customers' own risk. Customers are solely responsible for compliance with all legal and regulatory requirements in connection with such selection.

Customer will fully indemnify Realtek and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer's non-compliance with the terms and provisions of this Notice.

TRADEMARKS

Realtek is a trademark of Realtek Semiconductor Corporation. Other names mentioned in this document are trademarks/registered trademarks of their respective owners.

USING THIS DOCUMENT

Though every effort has been made to ensure that this document is current and accurate, more information may have become available subsequent to the production of this guide.

Contents

List of Tables

List of Figures

1 Build Environment Setup

1.1 Introduction

This chapter illustrates how to build Realtek Wi-Fi SDK under GCC environment. It focuses on both Windows platform and Linux distribution. For Windows, Windows 7 64-bit is used as platform. And for Linux distribution, Ubuntu 18.04 64-bit is used as platform. The build and download procedure are quite similar between Windows and Linux operating system.

1.2 Setup GCC Environment

1.2.1 Windows

On Windows, you can use Cygwin as the GCC environment. Cygwin is a large collection of GNU and open source tools which provide functionality similar to a Linux distribution on Windows.

Click http://cygwin.com and download the Cygwin package -setup-x86.exe for your Windows platform.

Note:

- Only 32-bit Cygwin is supported both for 32-bit Windows and 64-bit Windows.
- During the installation of Cygwin package, include 'Devel -> make' and 'Math -> bc' utilities on the **Select Packages** step, as Fig 1-1 and Fig 1-2 shows.

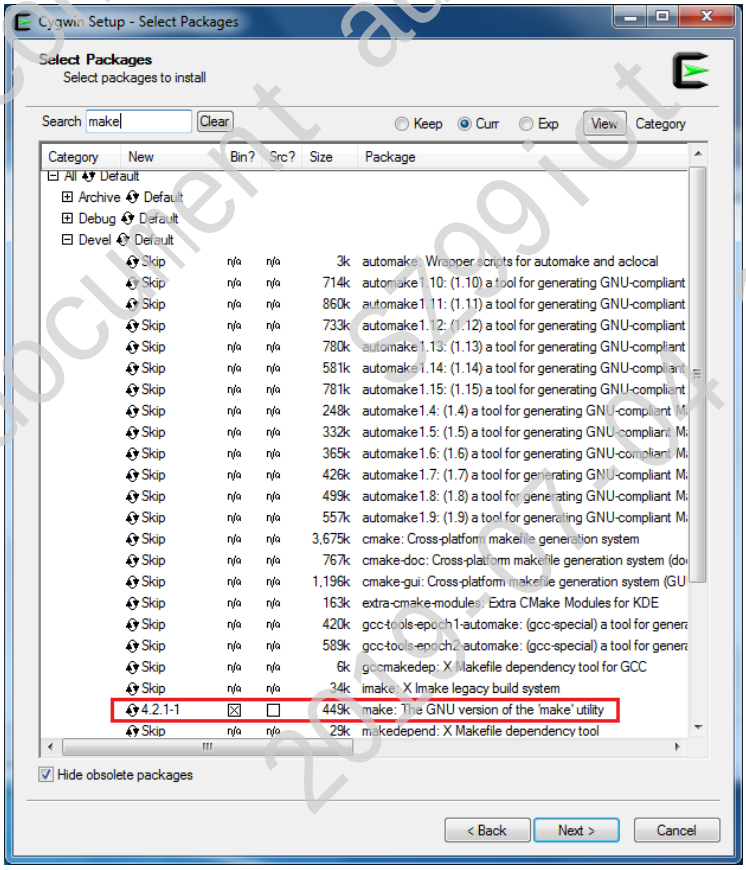


Fig 1-1 'Devel' setting during Cygwin installation

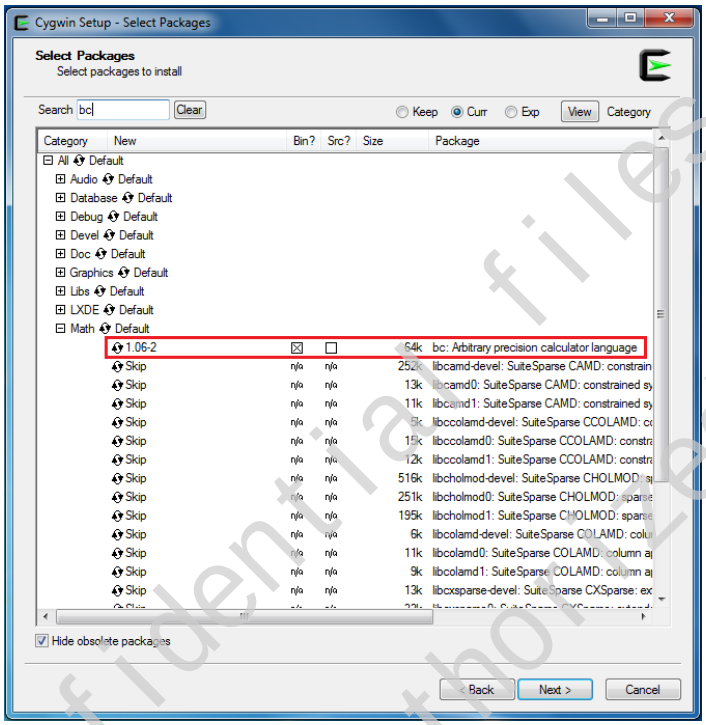


Fig 1-2 'Math' setting during Cygwin installation

1.2.2 Linux

On Linux, some packages must be installed for the GCC environment:

- **libc6-i386** (GNU C library for 64-bit platform. If you are using 32-bit platform, install **libc6** instead)
- **lib32ncurses5** (32-bit terminal handling for 64-bit platform. If you are using 32-bit platform, install **libncurses5** instead)
- **make**
- **bc**
- **gawk**
- **ncurses**

Some of these packages may have been pre-installed in your operating system. Use package manager to check and decide whether to install. For the last three packages, you can also type the corresponding version command on terminal like the following figures to check whether it exists. If not, make these packages installed.

- **$ make –v**

If **make** isn't installed, type **apt-get install make** to install **make**.

```
realtek@realtek:~$ make -v
GNU Make 4.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

- **$ bc –v**

If **bc** isn't installed, type **apt-get install bc** to install **bc**.

```
realtek@realtek:~$ bc -v
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software
 Foundation, Inc.
```

- **$ gawk --v**

If **gawk** isn't installed, type **apt-get install gawk** to install **gawk**.

```
@realtek:~$ gawk --v
GNU Awk 4.1.4, API: 1.1 (GNU MPFR 4.0.1, GNU MP 6.1.2)
Copyright (C) 1989, 1991-2016 Free Software Foundation.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see http://www.gnu.org/licenses/.
```

- **ncurses**

ncurses is needed if you want to use **make menuconfig** command. Type **apt-get install ncurses-dev** to install **ncurses.**

1.3 Build Code

This section illustrates how to build SDK. First, you need to switch to GCC project directory.

- For Windows, open Cygwin terminal and use $ **cd** command to change directory to KM0 or KM4 project directory of Ameba-D SDK.
  Note: You need to replace the *{path}* to your own SDK location, and add "cygdrive" prefix in front of the SDK location, so that Cygwin can access your file system.
  - $ cd /cygdrive/*{path}*/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp
  - $ cd /cygdrive/*{path}*/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp
- For Linux, open its own terminal and use **$ cd** command to change directory to KM0 or KM4 project directory of Ameba-D SDK.
  - $ cd /*{path}*/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp
  - $ cd /*{path}*/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp

1.3.1 Normal Image

To build SDK for normal image, simply use **$ make all** command under the corresponding project directories on Cygwin (Windows) or terminal (Linux).

1.3.1.1 KM0 Project

For KM0 project, if the terminal contains "km0_image2_all.bin" and "Image manipulating end" output message, means that the image has been built successfully.

```
/cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp
.0/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image/xip_im
age2.bin   __flash_text_start__   /cygdrive/d/sdk-amebad-beta_v6.0/project/realt
ek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image/target_img2.map
/cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_amebaD_va0_example/GCC-RELEAS
E/project_lp/asdk/gnu_utility/prepend_header.sh /cygdrive/d/sdk-amebad-beta_v6
.0/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image/ram_re
tention.bin   __retention_entry_func__   /cygdrive/d/sdk-amebad-beta_v6.0/projec
t/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image/target_img2.map
cat /cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_amebaD_va0_example/GCC-RE
LEASE/project_lp/asdk/image/xip_image2_prepend.bin /cygdrive/d/sdk-amebad-beta
_v6.0/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image/ram
_2_prepend.bin > /cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_amebaD_va0_e
xample/GCC-RELEASE/project_lp/asdk/image/km0_image2_all.bin
/cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_amebaD_va0_example/GCC-RELEAS
E/project_lp/asdk/gnu_utility/pad.sh /cygdrive/d/sdk-amebad-beta_v6.0/project/
realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image/km0_image2_all.bi
n
========== Image manipulating end ==========
make[1]: Leaving directory '/cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_a
mebaD_va0_example/GCC-RELEASE/project_lp/asdk'

[redacted] /cygdrive/d/sdk-amebad-beta_v6.0/project/realtek_amebaD_va0_
example/GCC-RELEASE/project_lp
$
```

Fig 1-3 KM0 project make all

If somehow it is built failed, type **$ make clean** to clean and then redo the make procedure. After successfully built, the image file is located in **project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/asdk/image**, as Fig 1-4 shows.

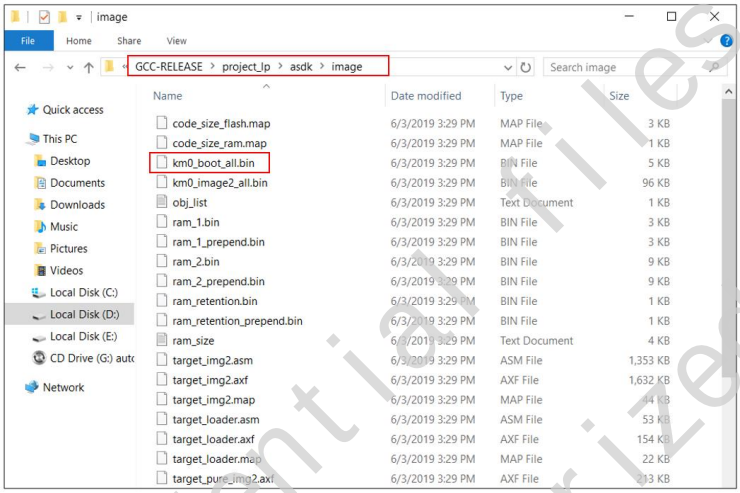


Fig 1-4 KM0 project bin generation

1.3.1.2 KM4 Project

For KM4 project, if the terminal contains “km4_image2_all.bin” and “Image manipulating end” output message, means that the image has been built successfully.

Fig 1-5 KM4 project make all

If somehow it built failed, type **$ make clean** to clean and then redo the make procedure. After built successfully, the image file is located in **project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/asdk/image**, as Fig 1-6 shows.

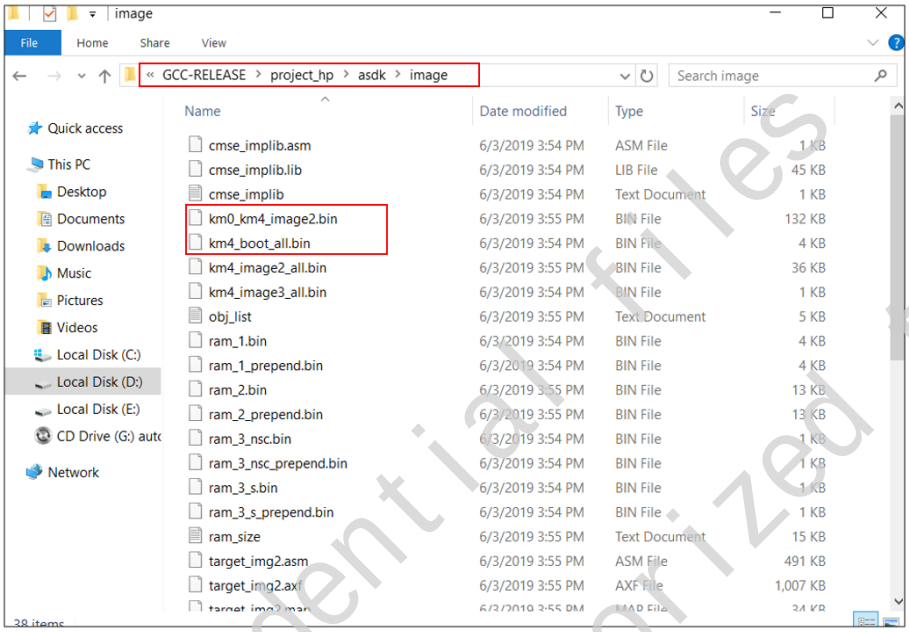


Fig 1-6 KM4 project bin generation

1.3.2 MP Image

Use **make mp** command under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp** to generate MP image. After successful compilation, you will find the generated images under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/asdk/image**, as shown in Fig 1-7.

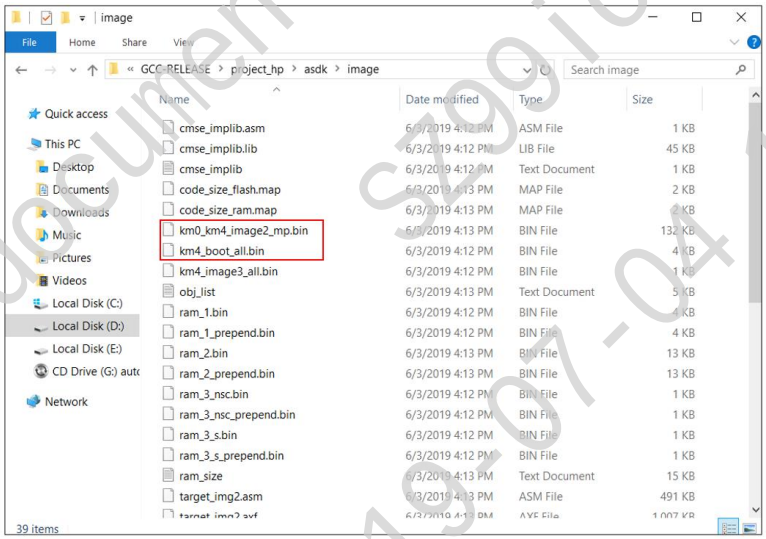


Fig 1-7 MP image generation

1.4 Debugger Setting

This section illustrates how to enter GDB debugger mode.

Ameba-D supports CMSIS-DAP and J-Link for code download and entering debugger mode. The settings are described below.

1.4.1 Probe

1.4.1.1 Windows

Ameba-D Device Board supports Probe debugger. We can use Probe to download the software and enter GBD debugger mode under GCC environment.

(1) Install Probe driver
Before using the Probe, its driver is required to be installed. Obtain the **RLX_Probe_Driver_2.3.11p18_Setup_190304.exe** under tools/probe, then install it correctly. Connect TCK pin of Probe to SWD CLK pin and TMS pin of Probe to SWD DATA pin. What's more, a common ground is needed between Probe Board and Ameba-D Device Board.

(2) Execute the **cm0_RTL_Probe.bat**
After the installation of the software pack, execute the **cm0_RTL_Probe.bat** under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/jlink_script**.
Note: The default path of Probe driver in **cm0_RTL_Probe.bat** file is **C:\RLX\PROBE\rlx_probe_driver.exe**, you may have to change the path according to your own settings.

The started Probe server looks like Fig 1-8. This window should NOT be closed if you want to download software or enter GDB debugger mode.

Fig 1-8 KM0 Probe server connection under windows

(3) On the Cygwin terminal, as Fig 1-9 shows, type **$ make setup GDB_SERVER=probe** command before using J-Link to download software or enter GDB debugger.

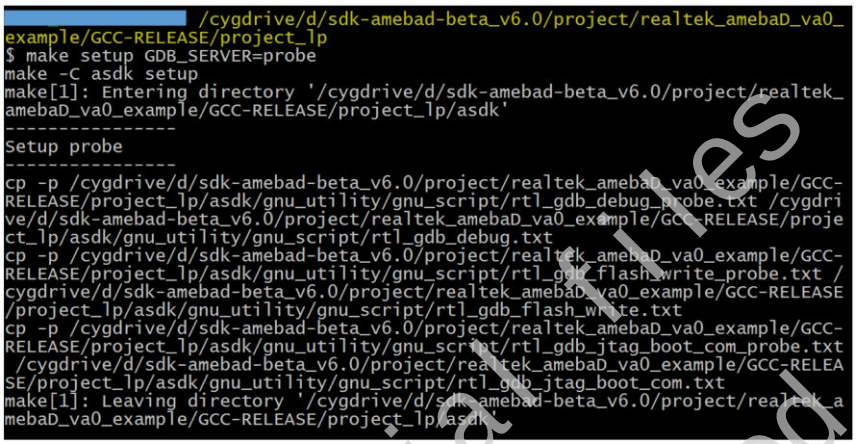


Fig 1-9 KM0 Probe setup

(4) Execute the **cm4_RTL_Probe.bat**

Execute the **cm4_RTL_Probe.bat** under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/jlink_script** the same as executing the **cm0_RTL_Probe.bat**. The started Probe server looks like Fig 1-10. This window should NOT be closed if you want to enter GDB debugger mode.

Fig 1-10 KM4 Probe server connection under Windows

Note: When **cm4_RTL_Probe.bat** is running, the connection built by running **cm0_RTL_Probe.bat** will be closed. If you want to connect the Probe to KM0 and KM4 simultaneously, follow the steps below:

a) Make a copy of **rlx_probe0.cfg** under the folder **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/jlink_script** and rename it as **rlx_probe1.cfg**.

b) Cut it and move it to **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/jlink_script.**
c) Connect the Probe to both KM0 and KM4 by executing **cm0_RTL_Probe.bat** under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/jlink_script**.

(5) Setup J-Link for KM4
On the Cygwin terminal, as Fig 1-11 shows, type **$ make setup GDB_SERVER=probe** command before using J-Link to download software or enter GDB debugger.

Fig 1-11 KM4 Probe setup

1.4.1.2 Linux

Before using Probe, its driver is required to be installed. Obtain the **RLX_Probe_Linux_Driver_v2.3.11p22** under tools/probe and install it correctly, to install the driver, the **read me.txt** under **RLX_Probe_Linux_Driver_v2.3.11p22** can be referenced. Then connect TCK pin of Probe to SWD CLK pin and TMS pin of Probe to SWD DATA pin. What's more, a common ground is needed between Probe Board and Ameba-D Device Board.

After the installation of the software pack, in both folder **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/jlink_script** and **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/jlink_script** , a soft link should be created to link the elf in **RLX_Probe_Linux_Driver_v2.3.11p22**.

Fig 1-12 Creating soft link to elf

Open a new terminal under the corresponding project folder and type the following command to start probe server. This terminal should NOT be closed if you want to download software or enter GDB debugger mode.

For KM0, open a new terminal under**/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/jlink_script**, and type **$ sudo ./rlx_probe_driver.elf.**

Fig 1-13 KM0 probe GDB server connection setting under Linux

If connection is successful, the log is shown as Fig 1-14.

Fig 1-14 KM0 probe GDB server connection success under Linux

Open a new terminal under the same project folder (**/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp**), type **$ make setup GDB_SERVER=probe** command before using probe to download software or enter GDB debugger.

For KM4, open a new terminal under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/jlink_script**, and type **$ sudo./rlx_probe_driver.elf.**

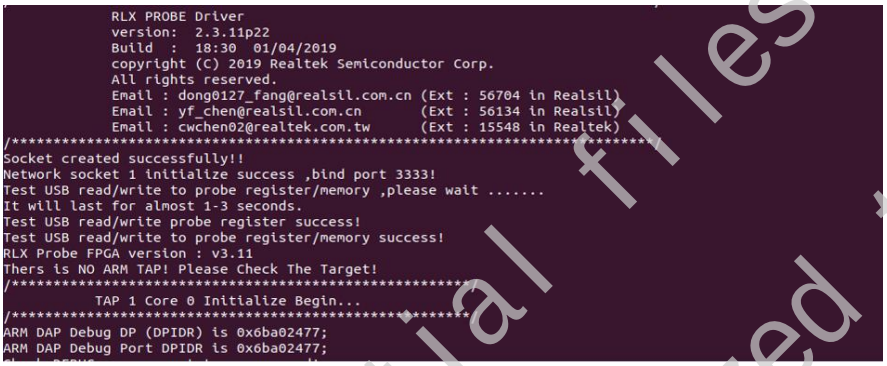


Fig 1-15 KM4 probe GDB server connection setting under Linux

If connection is successful, the log is shown as Fig 1-16.

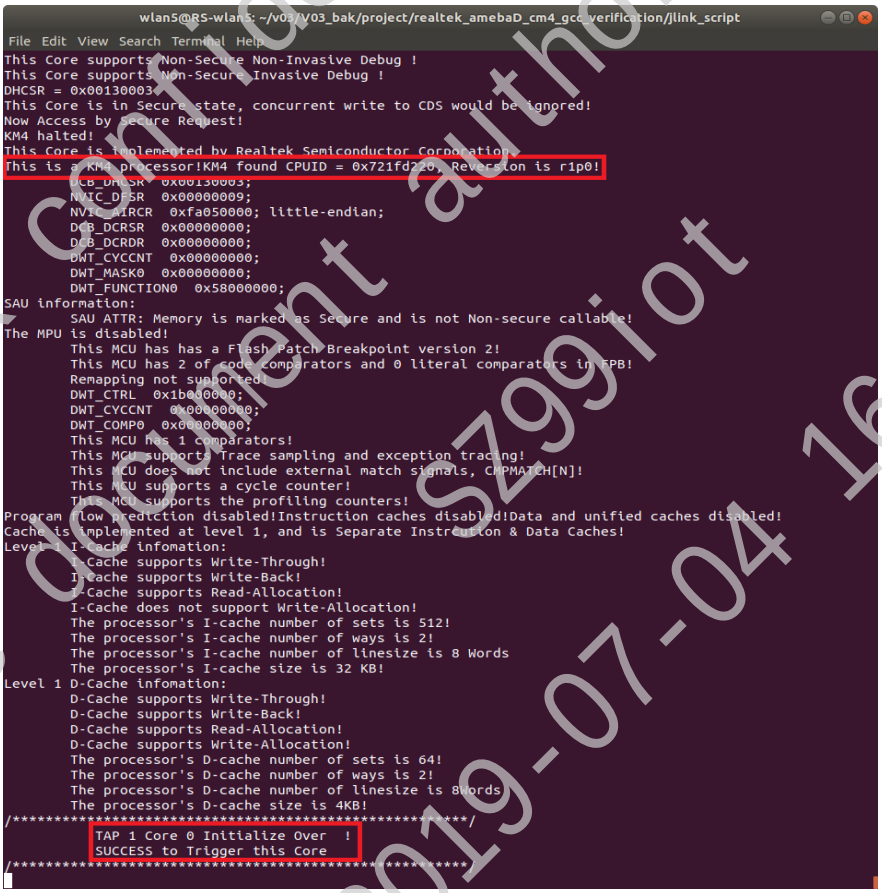


Fig 1-16 KM4 probe GDB server connection success under Linux

Open a new terminal under the same project folder (**/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp**), type **$ make setup GDB_SERVER=probe** command before using probe to enter GDB debugger.

1.4.2 J-Link

Ameba-D also supports J-Link debugger. You need to do some hardware configuration to use J-Link debugger, that is welding SWD connectors to HDK board and connecting with dupont line. The SWD pin definitions are listed on the bottom side. We recommend to weld the connector on the bottom side. After finishing these configurations, connect it to PC side.

Note: For Ameba-D CPU, the J-Link version must be v9 or higher. If Virtual Machine (VM) is used as your platform, make sure the USB connection setting between VM host and client is correct, so that the VM client can detect the device.

1.4.2.1 Windows

(1) Install J-Link GDB server
Besides the hardware configuration, J-Link GDB server is also required to install. For Windows, click https://www.segger.com/downloads/jlink and download the software in "**J-Link Software and Documentation Pack**", then install it correctly.
Note: The version of J-Link GDB server displayed in the pictures below is just an example, you can select the latest version to download.

(2) Execute the **cm0_jlink.bat**
After the installation of the software pack, execute the **cm0_jlink.bat** under **/project/realtek_amebaD_va0_example/GCCRELEASE/project_lp/jlink_script**.
Note: The default path of J-Link driver in **cm0_jlink.bat** file is **C:\Program Files (x86)\SEGGER\JLink_V634b\JLinkGDBServer.exe**, you may have to change the path according to your own settings.

The started J-Link GDB server looks like Fig 1-17. This window should NOT be closed if you want to download software or enter GDB debugger mode.

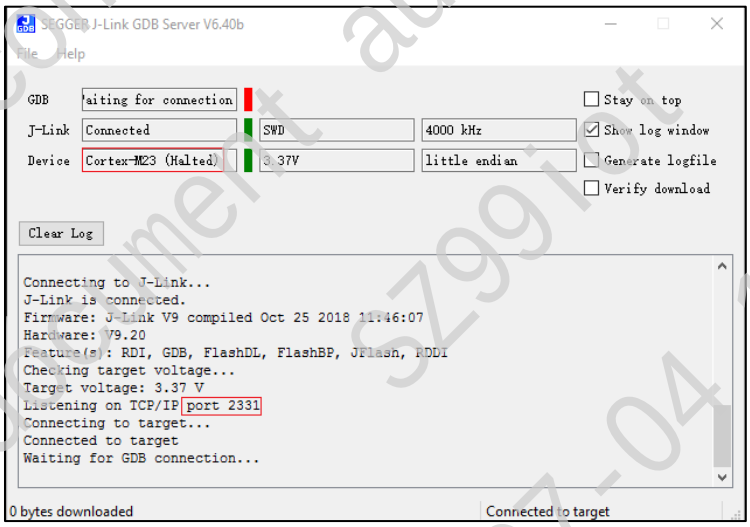


Fig 1-17 KM0 J-Link GDB server connection under windows

(3) Setup J-Link for KM0
On the Cygwin terminal, as Fig 1-18 shows, type **$ make setup GDB_SERVER=jlink** command before using J-Link to download software or enter GDB debugger.

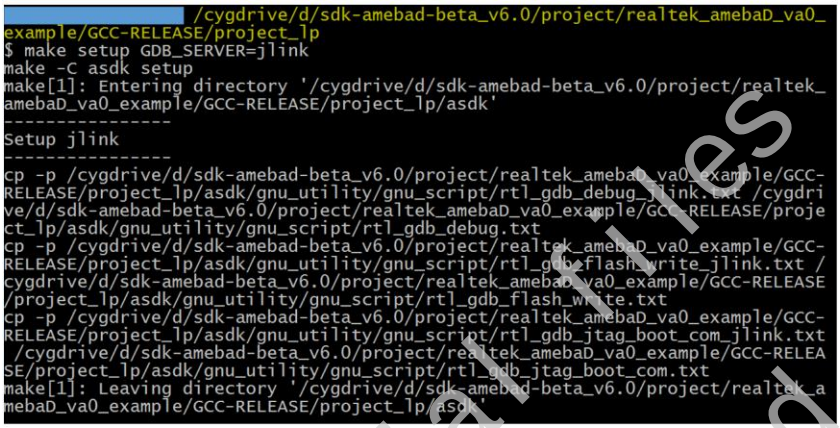


Fig 1-18 KM0 J-Link setup

(4) Execute the **cm4_jlink.bat**

Execute the **cm4_jlink.bat** under **/project/realtek_amebaD_va0_example/GCCRELEASE/project_lp/jlink_script** the same as executing the **cm0_jlink.bat**. The started J-Link GDB server looks like Fig 1-19. This window should NOT be closed if you want to download software or enter GDB debugger mode.

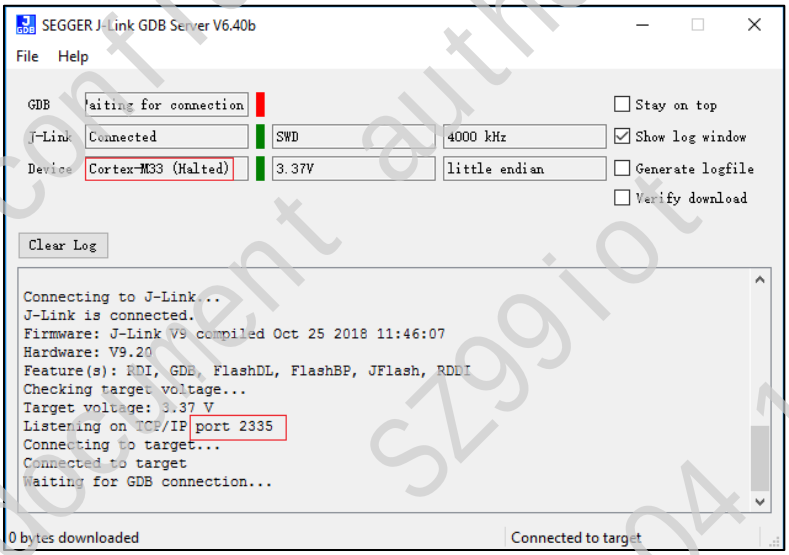


Fig 1-19 KM4 J-Link GDB server connection under Windows

(5) Setup J-Link for KM4

On the Cygwin terminal, as Fig 1-20 shows, type **$ make setup GDB_SERVER=jlink** command before using J-Link to download software or enter GDB debugger.

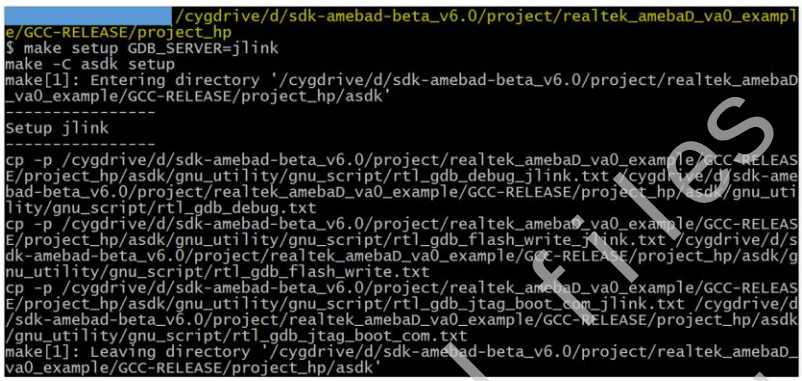


Fig 1-20 KM4 J-Link Setup

1.4.2.2 Linux

For J-Link GDB server, click https://www.segger.com/downloads/jlink and download the software in "J-Link Software and Documentation Pack". We suggest using Debian package manager to install the Debian version.

- **$ dpkg –i jlink_6.0.7_x86_64.deb**

After the installation of the software pack, there is a tool named "JLinkGDBServer" under JLink directory. Take Ubuntu 18.04 as an example, the JLinkGDBServer can be found at **/opt/SEGGER/JLink**. Open a new terminal under the corresponding project folder and type the following command to start GDB server. This terminal should NOT be closed if you want to download software or enter GDB debugger mode.

For KM0, open a new terminal under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_lp/jlink_script**, and type **$ /opt/SEGGER/JLink/JLinkGDBServer –device cortex-m23 -if SWD -scriptfile AP1_KM0.JLinkScript port 2331.**

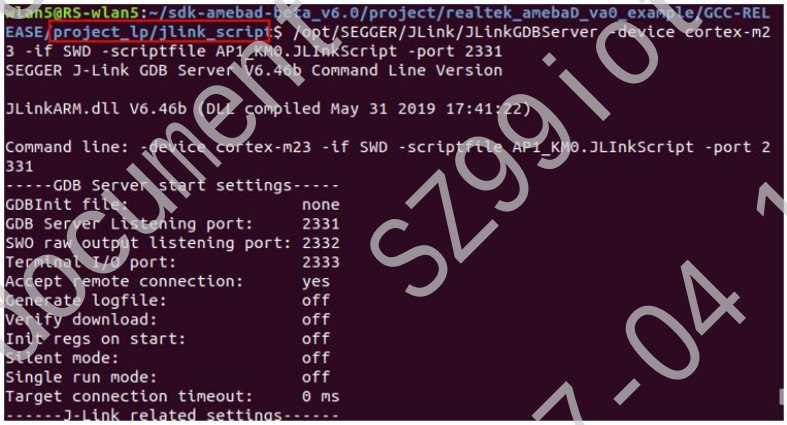


Fig 1-21 KM0 J-Link GDB server connection setting under Linux

If connection is successful, the log is shown as Fig 1-22.

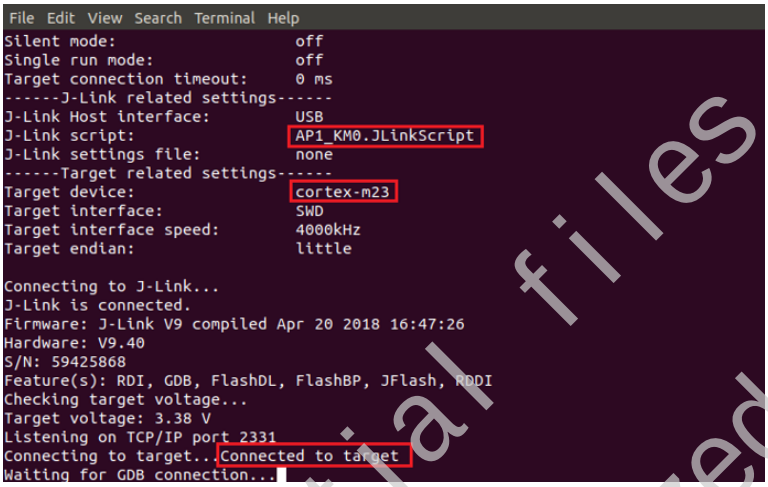


Fig 1-22 KM0 J-Link GDB server connection success under Linux

Open a new terminal under the same project folder, type **$ make setup GDB_SERVER=jlink** command before using J-Link to download software or enter GDB debugger.

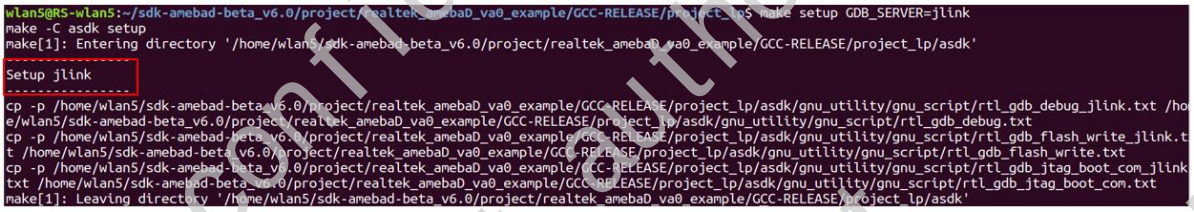


Fig 1-23 KM0 J-Link terminal setup under Linux

For KM4, open a new terminal under **/project/realtek_amebaD_va0_example/GCC-RELEASE/project_hp/jlink_script**, and type **$ /opt/SEGGER/JLink/JLinkGDBServer –device cortex-m33 -if SWD -scriptfile AP2_KM4.JLinkScript –port 2335.**

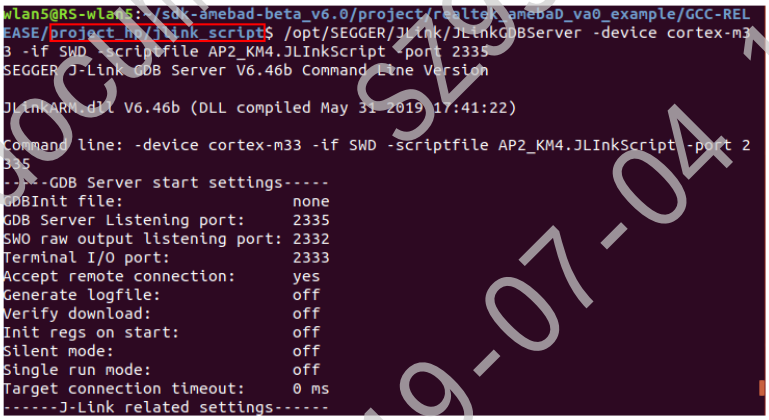


Fig 1-24 KM4 J-Link GDB server connection setting under Linux

If connection is successful, the log is shown as Fig 1-25.

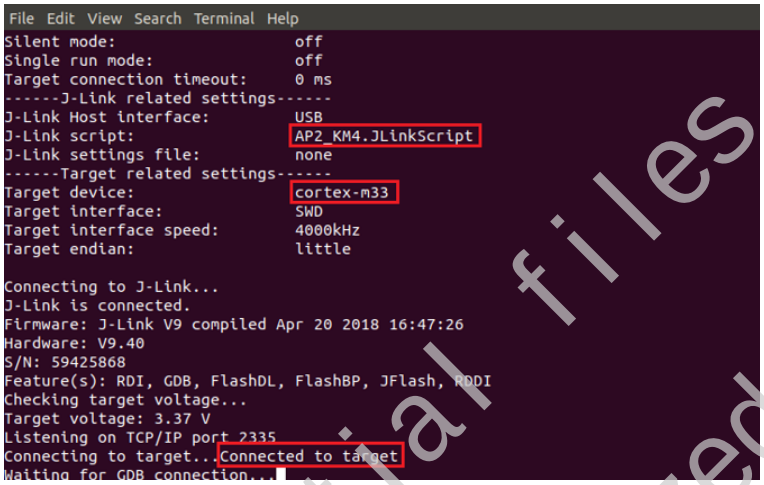


Fig 1-25 KM4 J-Link GDB server connection success under Linux

Open a new terminal under the same project folder, type **$ make setup GDB_SERVER=jlink** command before using J-Link to enter GDB debugger.

Fig 1-26 KM4 J-Link terminal setup under Linux

1.5 Download Code to Flash

This section illustrates how to download image to flash.

To download software into Ameba-D Device Board, make sure steps mentioned in section 1.3 to 1.4 are done and then type **$ make flash** command on Cygwin (Windows) or terminal (Linux).

Both KM0 and KM4 download codes by this command. This command downloads the software into flash and it will take a few seconds to finish. After downloaded successfully, as shown in Fig 1-27, press the **Reset** button and you can see the device is booted with new image.

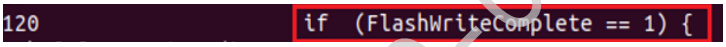


Fig 1-27 Download codes success log

Note: For Probe download,

- Make chip enter download mode before downloading code to flash
- Download KM4 project code also through **cm0_RTL_Probe.bat**
- Probe uses USB 1.0 interface, so its download rate is limited by the USB 1.0 protocol.

1.6 Enter GDB Debugger

To enter GDB debugger mode, make sure steps mentioned in section 1.3 to 1.5 are finished and then reset the device first. After resetting the chip, type **$ make debug** command on Cygwin (Windows) or terminal (Linux).

- For Windows, a popup window will display, as shown in Fig 1-28.

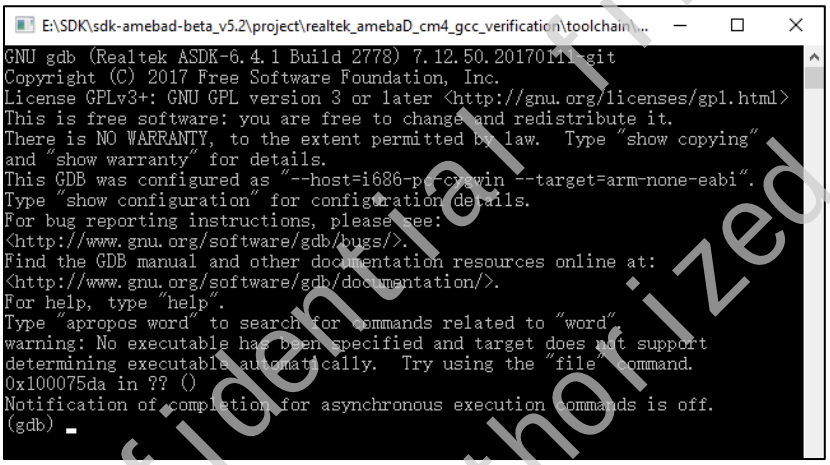


Fig 1-28 Debug window under Windows

- For Linux, the log is shown in Fig 1-29.

Fig 1-29 Debug window under Linux

1.7 Command List

The commands used above are listed in Table 1-1.

Table 1-1 Command list

Usage	Command	Description
all	$ make all	Compile project to generate ram_all.bin

setup	$ make setup GDB_SERVER= jlink	Setup GDB_SERVER
flash	$ make flash	Download ram_all.bin to flash
clean	$ make clean	Remove compile result (*.bin, *.o, …)
clean_all	$ make clean_all	Remove compile result and toolchains
debug	$ make debug	Enter GDB debugger mode

1.8 GDB Debugger Basic Usage

GDB, the GNU project debugger, allows you to examine the program while it executes, and it is helpful for catching bugs. Section 1.6 has described how to enter GDB debugger mode, this section illustrates some basic usage of GDB commands. For further information about GDB debugger and its commands, click https://www.gnu.org/software/gdb/ and https://sourceware.org/gdb/current/onlinedocs/gdb/.

Table 1-2 GDB debugger command list

Usage	Command	Description
Breakpoint	$ break	Breakpoints are set with the *break* command (abbreviated *b*). The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Set-Breaks.html
Watchpoint	$ watch	You can use a watchpoint to stop execution whenever the value of an expression changes. The related commands include *watch*, *rwatch*, and *awatch*. And the usage of these commands can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Set-Watchpoints.html **Note**: Keep the range of watchpoints less than 20 bytes.
Print breakpoints & watchpoints	$ info	To print a table of all breakpoints, watchpoints set and not deleted, use the *info* command. You can simply type *info* to know its usage.
Delete breakpoints	$ delete	To eliminate the breakpoints, use the *delete* command (abbreviated *d*). The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Delete-Breaks.html.
Continue	$ continue	To resume program execution, use the *continue* command (abbreviated *c*). The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Continuing-and-Stepping.html.
Step	$ step	To step into a function call, use the *step* command (abbreviated *s*). It will continue running your program until control reaches a different source line. The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Continuing-and-Stepping.html.
Next	$ next	To step through the program, use the *next* command (abbreviated *n*). The execution will stop when control reaches a different line of code at the original stack level. The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Continuing-and-Stepping.html.
Quit	$ quit	To exit GDB debugger, use the *quit* command (abbreviated *q*), or type an end-of-file character (usually *Ctrl-d*). The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Quitting-GDB.html.
Backtrace	$ backtrace	A backtrace is a summary of how your program got where it is. You can use *backtrace* command (abbreviated *bt*) to print a backtrace of the entire stack. The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Backtrace.html.
Print source lines	$ list	To print lines from a source file, use the *list* command (abbreviated *l*). The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/List.html.
Examine data	$ print	To examine data in your program, you can use *print* command (abbreviated *p*). It evaluates and prints the value of an expression. The usage can be found at https://sourceware.org/gdb/current/onlinedocs/gdb/Data.html.

1.9 Q & A

Q1 (EN): “Error 127” for Building Code.
Q1 (CN)：编译的时候产生“Error 127”的错误。

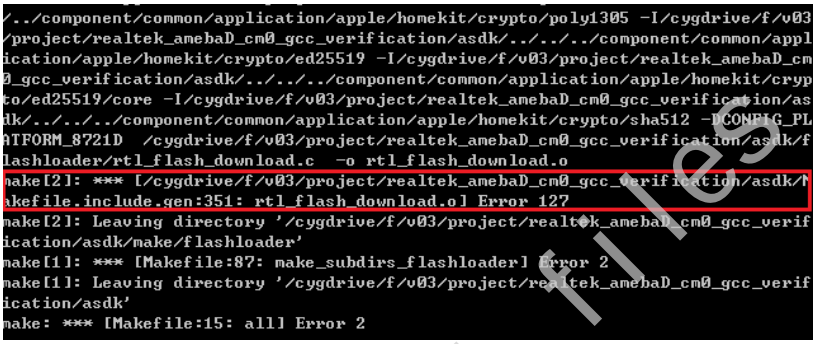


Fig 1-30 “Error 127” error message

A1 (EN): If you use **$ make all** command to build code but encounter “Error 127” error message like Fig 1-30, it’s caused by incorrect Cygwin package. Both 32-bit and 64-bit operating system need to install 32-bit installation Cygwin Package – setup-x86.exe.

A1 (CN)：如果在使用**$ make all** 命令编译 code 出现如图 Fig 1-30 所示的“Error 127”错误信息，这是由于 Cygwin 的安装包版本不正确导致的。无论 32 位还是 64 位的操作系统，都需要安装 32 位的 Cygwin 的安装包 – setup-x86.exe。

Q2 (EN): “Permission denied” Error under Linux.

Q2 (CN)：在 Linux 环境下编译产生“Permission denied”的错误。

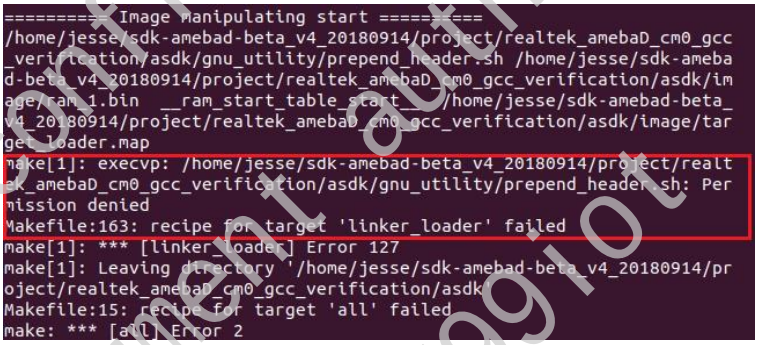


Fig 1-31 “Permission denied” error message under Linux

A2 (EN): If you use **$ make all** command to compile project but encounter “Permission denied” error message like Fig 1-31, it’s caused by file access permission. You can enter **chmod -R 777 $SDK_FILENAME** under the path of SDK first to change the permission of files in SDK, and then continue compile operation.

A2 (CN)：如果在使用**$ make all** 命令编译 code 出现如图 Fig 1-31 所示的“Permission denied”错误信息，这是文件访问权限受限所致。需要在 SDK 的路径下键入“**chmod -R 777 $SDK_FILENAME**”去修改 SDK 文件的权限，之后再进行编译即可。

Q3 (EN): How to reset KM0/KM4 under debug?

Q3 (CN)：如何在调试的时候让 KM0/KM4 重新启动？

A3 (EN): Steps to reset KM0 and KM4 are different.

- For KM0, you need to use the “monitor reset” instruction in corresponding rtl_gdb_debug.txt to reset it under debug. The default path of this file is under \project\realtek_amebaD_va0_example\GCC-RELEASE\project_lp\asdk\gnu_utility\gnu_script\rtl_gdb_debug.txt. Please change the default setting which is “#monitor reset 1” to “monitor reset”.
- For KM4, you need to use the “monitor reset” instruction in corresponding rtl_gdb_debug.txt first (this process is similar to the process of KM0). Then, set the bit[25] of memory address 0x4800_0378 to 1 because the boot process of KM4 is controlled by the KM0. Without this operation, the KM4 can't jump out of the boot function. Details of this operation are as follows: After executing “make debug” instruction, a debug window pops up. In this window. Set bit[25] to 1, then type “c”, the KM4 is reset. refer to Fig 1-32 for more information.
- Only resetting KM0 may cause KM4 to work in an abnormal way because reboot KM0 will change some settings of KM4. If you find KM4 doesn’t work after rebooting KM0, you should reset KM4.

A3 (CN)：KM0 和 KM4 的系统重置是分别控制的。

- 对于 KM0，需要在相应的 rtl_gdb_debug.txt 中使用“**monitor reset**”指令。该文件的默认地址是 \project\realtek_amebaD_va0_example\GCC-RELEASE\project_lp\asdk\gnu_utility\gnu_script\rtl_gdb_debug.txt。请将默认设定的“#monitor reset 1”改为“monitor reset”。
- 对于 KM4，先使用相应的 rtl_gdb_debug.txt 中的“**monitor reset**”指令（与 KM0 操作类似）。因为 KM4 的 boot 是由 KM0 控制的，对于 KM4 系统重置调试，需要在 reset 之前先将 0x4800_03f8 的 bit[25]置 1，否则 KM4 会在启动函数中跳不出来。具体操作：在执行 make debug 指令之后会弹出调试窗口，在该窗口执行 bit[25]置 1 操作之后键入“c”就可以实现 KM4 系统重置。请参考 Fig 1-32 进行操作。
- 因为执行 KM0 的复位操作时时会去更改 KM4 的一些配置，所以只复位 KM0 时可能导致 KM4 不正常工作。若复位 KM0 后 KM4 不能正常工作，请尝试复位 KM4。

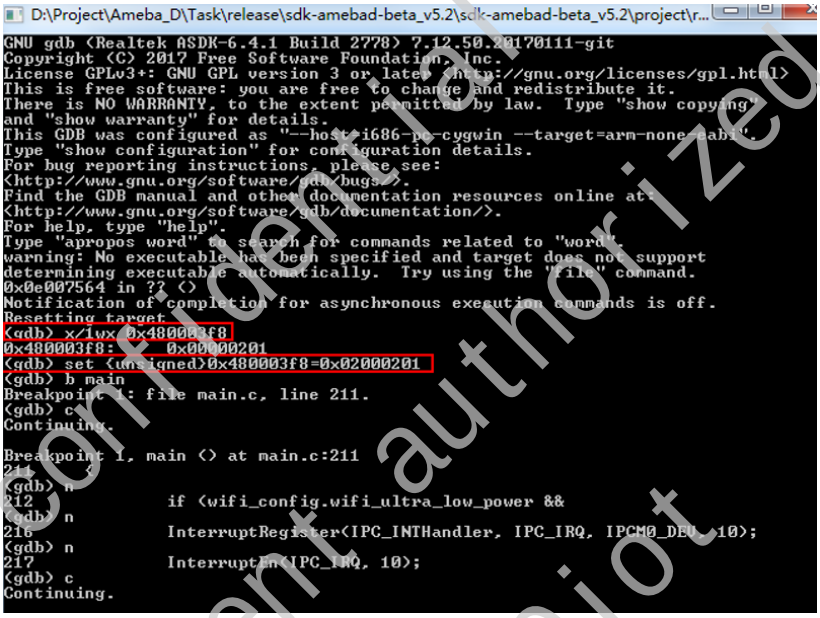


Fig 1-32 KM4 “monitor reset” setting

2 C++ Standards Supported in GCC

2.1 Introduction

This chapter mainly introduces how to build C++ codes in Ameba-D GCC project.

“iostream” is not available currently, and would be supported later.

2.2 How to Build C++ Codes

The following steps are necessary to support C++ in current GCC project.

(1) Modify Link script: **rlx8721d_img2_ns.ld**.

```
.xip_image2.text :
{
        __flash_text_start__ = .;

        *(.img2_custom_signature*)

        *(.text*)
        *(.rodata*)

        /* Add This for C++ support */
        . = ALIGN(4);
        __preinit_array_start = .;
        KEEP(*(.preinit_array))
        __preinit_array_end = .;
        . = ALIGN(4);
        __init_array_start = .;
        KEEP(*(SORT(.init_array.*)))
        KEEP(*(.init_array))
        __init_array_end = .;
        . = ALIGN(4);
        __fini_array_start = .;
        KEEP(*(SORT(.fini_array.*)))
        KEEP(*(.fini_array))
        __fini_array_end = .;
        /*----------------*/

        . = ALIGN (4);
        __cmd_table_start__ = .;
        KEEP(*(.cmd.table.data*))
        __cmd_table_end__ = .;

        __flash_text_end__ = .;

        . = ALIGN (16);
} > KM4_IMG2
```

```
/* Add This for C++ support */
.ARM.extab   :
{
        *(.ARM.extab* .gnu.linkonce.armextab.*)

} > KM4_IMG2

.ARM.exidx :
{
        __exidx_start = .;
        *(.ARM.exidx* .gnu.linkonce.armexidx.*)
        __exidx_end = .;
        end = .;
} > KM4_IMG2

__wrap_printf = 0x1010a3f5;
__wrap_sprintf = 0x1010a471;
__wrap_strcat = 0x10111635;
__wrap_strchr = 0x10111675;
__wrap_strcmp = 0x10111745;
__wrap_strncmp = 0x101118f9;
__wrap_strlen = 0x10111839;
__wrap_strnlen = 0x10111a05;
__wrap_strncat = 0x1011189d;
__wrap_strpbrk = 0x10111a39;
__wrap_strstr = 0x10111d25;
__wrap_strtok = 0x1011201d;
__wrap_strsep = 0x10111a65;
__wrap_strtoll = 0x10111ffd;
__wrap_strtoul = 0x101122e9;
__wrap_strtoull = 0x10111f3d;
__wrap_atoi = 0x101115e1;
__wrap_strcpy = 0x10111 7b9;
__wrap_strncpy = 0x1011199d;
__wrap_memset = 0x10110ea1;
__wrap_memcpy = 0x10110d2d;
__wrap_memcmp = 0x10110cc9;
__wrap_memmove = 0x10110dd9;
__wrap_snprintf = 0x1010a49d;
__wrap_malloc = pvPortMalloc;
__wrap_realloc = pvPortReAlloc;
__wrap_free = vPortFree;
/*----------------*/
```

(2) Modify startup code: **rtl8721dhp_app_start.c.**

```
#if defined (__GNUC__)
/* Add This for C++ support to avoid compile error */
void _init(void){}
#endif

// The Main App entry point
void app_start(void)
{
        simulation_stage_set(SIMULATION_KM4_CPUID, BIT_KM4_APP_ENTER);

        SOCPS_InitSYSIRQ_HP();

        __NVIC_SetVector(SVCall_IRQn, (VOID*)vPortSVCHandler);
        __NVIC_SetVector(PendSV_IRQn, (VOID*)xPortPendSVHandler);
        __NVIC_SetVector(SysTick_IRQn, (VOID*)xPortSysTickHandler);

#if defined ( __ICCARM__ )
        __iar_cstart_call_ctors(NULL);
#endif

#if defined (__GNUC__)
        /* Add This for C++ support */
        __libc_init_array();
#endif

        // Force SP align to 8 byte not 4 byte (initial SP is 4 byte align)
        __asm(
                "mov r0, sp\n"
                "bic r0, r0, #7\n"
                "mov sp, r0\n"
        );

        main();
}
```

(3) Modify makefile.

- project\realtek_amebaD_va0_example\GCC-RELEASE\project_hp\asdk\Makefile

```
linker_image2_ns:
        @echo "========= linker img2_ns start ========="
        $(LD) $(LD_ARG) target_img2.axf $(RAM_OBJS_LIST) $(LINK_ROM_LIB_NS) $(LINK_APP_LIB) $(IMAGE_TARGET_FOLDER)/cmse_implib.lib
-lm -lc -lnosys -lgcc -lstdc++
```

- project\realtek_amebaD_va0_example\GCC-RELEASE\project_hp\asdk\Makefile.include.gen

```
#GLOBAL_CFLAGS += -specs nosys.specs
#GLOBAL_CFLAGS += -fno-short-enums
GLOBAL_CFLAGS += -Wextra

GLOBAL_CFLAGS += $(IFLAGS)
GLOBAL_CFLAGS += -DCONFIG_PLATFORM_8721D
GLOBAL_CFLAGS += -DCONFIG_USE_MBEDTLS_ROM_ALG
GLOBAL_CFLAGS += -DCONFIG_FUNCION_00_OPTIMIZE
GLOBAL_CFLAGS += -DDM_ODM_SUPPORT_TYPE=32

CFLAGS = $(GLOBAL_CFLAGS)
CFLAGS += -Wstrict-prototypes

CPPFLAGS = $(GLOBAL_CFLAGS)
CPPFLAGS += -std=c++11 -fno-use-cxa-atexit
```

```
LD_ARG += -nostartfiles
LD_ARG += -specs nosys.specs
LD_ARG += -Wl,--gc-sections
LD_ARG += -Wl,--warn-section-align
LD_ARG += -Wl,-Map=text.map
LD_ARG += -Wl,--cref
LD_ARG += -Wl,--build-id=none
LD_ARG += -save-temps

LD_ARG += -Wl,-wrap,strcat  -Wl,-wrap,strchr   -Wl,-wrap,strcmp
LD_ARG += -Wl,-wrap,strncmp -Wl,-wrap,strcpy   #-Wl,-wrap,strnicmp
LD_ARG += -Wl,-wrap,strncpy -Wl,-wrap,strlen   #-Wl,-wrap,strlcpy
LD_ARG += -Wl,-wrap,strnlen -Wl,-wrap,strncat  -Wl,-wrap,strpbrk
LD_ARG += -Wl,-wrap,strstr  -Wl,-wrap,strtok   #-Wl,-wrap,strspn
LD_ARG += -Wl,-wrap,strsep  #-Wl,-wrap,strxfrm  -Wl,-wrap,strtod
LD_ARG += -Wl,-wrap,strtoll #-Wl,-wrap,strtof  -Wl,-wrap,strtold
LD_ARG += -Wl,-wrap,strtoul -Wl,-wrap,strtoull -Wl,-wrap,atoi
LD_ARG += #-Wl,-wrap,atoui   -Wl,-wrap,atol     -Wl,-wrap,atoul
LD_ARG += #-Wl,-wrap,atoull  -Wl,-wrap,atof
LD_ARG += -Wl,-wrap,malloc   -Wl,-wrap,free     -Wl,-wrap,realloc
LD_ARG += -Wl,-wrap,memcmp  -Wl,-wrap,memcpy
LD_ARG += -Wl,-wrap,memmove -Wl,-wrap,memset
LD_ARG += -Wl,-wrap,printf  -Wl,-wrap,sprintf
LD_ARG += -Wl,-wrap,snprintf  #-Wl,-wrap,vsnprintf
```

```
#*******************************************************************************#
#               RULES TO GENERATE OBJECT FILE FROM .CPP FILE                    #
#*******************************************************************************#
%.o:%.cpp
        $(CC) $(CPPFLAGS) $< -o $@
```

(4) Add C++ files into protect.

We have tested a small program and provide it as an example, which can be found in **main.cpp**.

(5) Use command **make** in console to rebuild project with C++ files.

3 SDK Architecture

The architecture of SDK is shown in Fig 1-1.

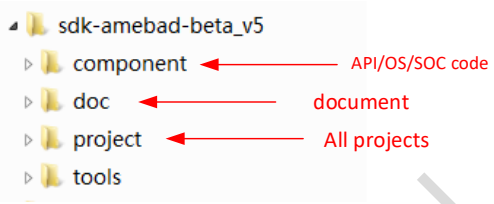


Fig 3-1 SDK architecture

3.1 Component

3.1.1 Common

Items	Description
api	● AT command ● Platform_stdlib.h: standard library header ● Wi-Fi driver interface ● Wi-Fi promisc mode interface ● Wi-Fi simple configuration
application	● DuerOS ● MQTT
bluetooth	● Internal BT driver
drivers	● alc5616/ alc5640/ alc5651/ alc5660/ alc5679/ alc5680/ sgtl5000 drivers ● IR NEC driver ● SDIO host driver ● Ameba-D internal codec rl6548 driver ● USB host and device drivers ● WLAN driver
example	● Utility examples: wlan_fast_connect/ssl_download/ fatfs/mdns/media_geo_mp4 …
file_system	● file_system
mbed	● mbed API source code
media	● multi-media framework
network	● coap ● dhcp ● http ● lwip ● mDNS ● rtsp ● sntp ● ssl (MBEDTLS) ● tftp ● websocket
utilities	● cJSON ● http_client ● ssl_client ● tcpecho ● udpecho ● webserver/xml

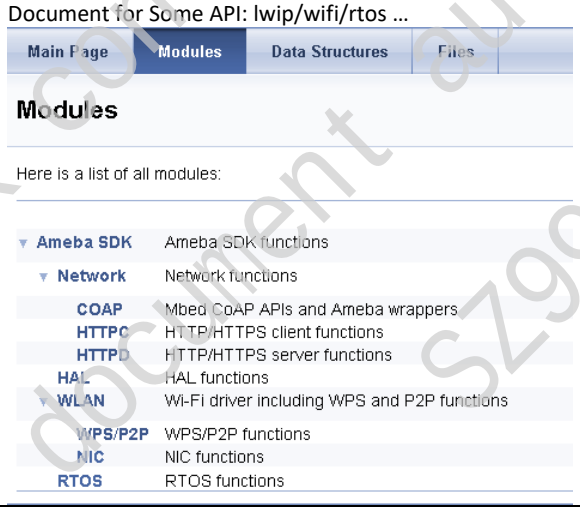
3.1.2 OS

Items	Description
freertos	FreeRTOS source code
os_dep	● **osdep_service.c**: Realtek encapsulating interface for FreeRTOS ● **osdep_service.h**: Realtek encapsulating interface header

3.1.3 SOC

Items	Description
app	Monitor and shell
bootloader	Boot loader
cmsis	ARM headers, include ARM CPU registers and operations
cmsis-dsp	ARM cmsis-dsp source code
fwlib	Low level drivers like: UART/I2C/SPI/Timer/PWM …
fwlib/usrcfg	User configuration files, maintained by user: bootcfg/trustzonecfg/sleepcfg/flashcfg/pinmapcfg …
swlib	Standard software library supported by ROM code, like: _memcpy/_memcmp …

3.2 Doc

Items	Description
api_doc&api_doc.htm	Document for Some API: lwip/wifi/rtos … 
AN0004	Realtek low power Wi-Fi MP user guide
AN0011	Realtek WLAN simple configuration
AN0012	Realtek secure socket layer (SSL)
AN0025	Realtek AT command
AN0043	Realtek mdns user guide
AN0075	Realtek Ameba-all at command v2.0
UM0150	Realtek Ameba CoAP User Guide.pdf

3.3 Tools

Items	Description
ImageTool	tools\AmebaZ\Image_Tool: image tool for Ameba-D & Ameba-Z
iperf.exe	iperf for Wi-Fi performance test

3.4 GCC Project for KM4

The architecture of KM4 project is shown in Fig 3-2.

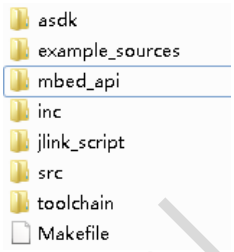


Fig 3-2 Architecture of project

Items	Description
asdk	● Menuconfig ● Link script ● Library: lib_wlan.a/lib_wps.a/rom_symbol … ● Make files
example_sources	Peripherals driver demo code: ADC/Audio/LCD/UART/I2C …
mbed_api	Mbed API headers
inc	● FreeRTOSConfig.h: config FreeRTOS ● Main.h ● platform_autoconf.h: generated by “make menuconfig” ● platform_opts.h: config some utilities like example code in component\common\example.
ld	link script for every image
src	main.c
toolchain	GCC & Cygwin toolchain
Makefile	Top makefile

3.5 Critical Header Files

Items	Description
basic_types.h	SUCCESS/FAIL/ TRUE/ FALSE/ NULL/u8/u16/u32/ BOOL/BIT(x) …
Platform_stdlib.h	Standard library API: memcmp/strcpy/atoll/strpbrk/sscanf/printf/sprint/snprintf/vsnprintf …
Section_config.h	Section definition used in link script: IMAGE2_RAM_TEXT_SECTION …
mbed api headers	Component/common/mbed/ … ● Peripheral header files for mbed_api ● If you use mbed api, you should include related headers
ameba_soc.h	● Peripheral header files for raw api ● If you use raw api, you should include this header ● Raw API have more features that mbed_api, mbed api just have basic features.

4 GCC Makefile

4.1 KM4 Makefile Architecture

The architecture of KM4 makefile is shown in Fig 4-1.

Fig 4-1 KM4 makefile architecture

4.2 How to Build Code into Flash

Makefile in project/xip is an example for how to build code into flash.

```
include $(MAKE_INCLUDE_GEN)

.PHONY: all clean

#*******************************************************************************#
#                                 VARIABLES                                     #
#*******************************************************************************#
#add your include path here
IFLAGS += #-I$(BASEDIR)/component/common/network/coap/include

#set your source code path here
CUSTOMER_DIR = $(ROOTDIR)/make/project/xip

#*******************************************************************************#
#                                 Source FILE LIST                              #
#*******************************************************************************#
#add your source code here
OBJS += \
                $(CUSTOMER_DIR)/xip_test.o\

#*******************************************************************************#
#                                 Include Dependency                            #
#*******************************************************************************#
-include $(OBJS:.o=.d)


#*******************************************************************************#
#                            RULES TO GENERATE TARGETS                          #
#*******************************************************************************#
all: CORE_TARGETS COPY_RAM_OBJS
#*******************************************************************************#
#                          GENERATE OBJECT FILE                                 #
#*******************************************************************************#
CORE_TARGETS: $(OBJS)

#*******************************************************************************#
#                          CLEAN GENERATED FILES                                #
#*******************************************************************************#
clean: CLEAN_OBJS
        $(REMOVE) *.o
        $(REMOVE) *.i
        $(REMOVE) *.s
        $(REMOVE) *.d

-include $(DEPS)
```

4.3 How to Build Code into SRAM

Makefile in project/SRAM is an example for how to build code into flash.

```
include $(MAKE_INCLUDE_GEN)


.PHONY: all clean

#*****************************************************************************#
#                                 VARIABLES                                   #
#*****************************************************************************#
#add your include path here
IFLAGS += #-I$(BASEDIR)/component/common/network/coap/include

#set your source code path here
CUSTOMER_DIR = $(ROOTDIR)/make/project/sram

#*****************************************************************************#
#                                 Source FILE LIST                            #
#*****************************************************************************#
#add your source code here
OBJS += \
                $(CUSTOMER_DIR)/ram_test.o\

#*****************************************************************************#
#                                 Include Dependency                          #
#*****************************************************************************#
-include $(OBJS:.o=.d)


#*****************************************************************************#
#                           RULES TO GENERATE TARGETS                         #
#*****************************************************************************#
all: CORE_TARGETS RENAME_CODE2SRAM COPY_RAM_OBJS
#*****************************************************************************#
#                 GENERATE OBJECT FILE
#*****************************************************************************#
CORE_TARGETS: $(OBJS)

%.o:%.c
        $(CC) $(GLOBAL_CFLAGS) $(MODULE_IFLAGS) $<  -o $@

#*****************************************************************************#
#                CLEAN GENERATED FILES                                        #
#*****************************************************************************#
clean: CLEAN_OBJS
        $(REMOVE) *.o
        $(REMOVE) *.i
        $(REMOVE) *.s
        $(REMOVE) *.d

-include $(DEPS)
```

4.4 How to Use Section Attribute

Because SRAM space is limited, we suggest you to build critical code/data into SRAM and other code/data left in flash, then you should use section attribute “IMAGE2_RAM_TEXT_SECTION” to locate some of your functions into SRAM.

```
#include <basic_types.h>
#include <section_config.h>

/* const is read only, it will build into flash */
const u32 xip_test_read_only_data = 0;

/* non read only data will build into SRAM */
u32 xip_test_data = 0;

/* code in in SRAM */
IMAGE2_RAM_TEXT_SECTION
void test_critical_code(void)
{

}

/* code in in Flash */
void test_non_critical_code(void)
{

}
```

Note:

- You should include “section_config.h”
- Const/read only data will build into flash
- Non read only data will build into SRAM (So if you need read only data build into SRAM to run faster, you should not use “const”)
- Functions limited by IMAGE2_RAM_TEXT_SECTION will build into SRAM
- Functions not limited by IMAGE2_RAM_TEXT_SECTION will build into FLASH

4.5 How to Build Library

Makefile in project/library is an example for build user library.

```
include $(MAKE_INCLUDE_GEN)

.PHONY: all clean

#*******************************************************************************#
#                                 VARIABLES                                     #
#*******************************************************************************#
#add your include path here
IFLAGS += #-I$(BASEDIR)/component/common/network/coap/include

#set your source code path here
CUSTOMER_DIR = $(ROOTDIR)/make/project/library

#*******************************************************************************#
#                                 Source FILE LIST                              #
#*******************************************************************************#
#add your source code here
OBJS += \
                $(CUSTOMER_DIR)/lib_user_test.o\

#*******************************************************************************#
#                                 Dependency                                    #
#*******************************************************************************#
-include $(OBJS:.o=.d)

#*******************************************************************************#
#                          RULES TO GENERATE TARGETS                            #
#*******************************************************************************#

# Define the Rules to build the core targets
all: CORE_TARGETS COPY_RAM_OBJS
        $(AR) rvs lib_user.a $(OBJS) $(OBJS_SRAM)
        $(MOVE) -f lib_user.a $(ROOTDIR)/lib/application

#*******************************************************************************#
#             GENERATE OBJECT FILE                                              #
#*******************************************************************************#
CORE_TARGETS:   $(OBJS)

#*******************************************************************************#
#                          RULES TO CLEAN TARGETS                               #
#*******************************************************************************#
clean: CLEAN_OBJS
        $(REMOVE) *.o
        $(REMOVE) *.i
        $(REMOVE) *.s
        $(REMOVE) *.d
```

4.6 How to Add Library

Open project_hp\asdk\Makefile and add lib_user.a into LINK_APP_LIB:

```
LINK_APP_LIB += $(ROOTDIR)/lib/application/lib_user.a
```

5 GCC Standard Library

5.1 Introduction

This chapter mainly introduces how to use GCC standard library in Ameba-D.

To save flash memory and improve efficiency, ROM code has implemented a standard software library, like _memcpy and _memcmp, and some functions are simplified version in ROM software library, such as printf and sprintf. Therefore, there are two methods when using standard library functions such as printf and sprintf in GCC.

(1) Using ROM software library.
(2) Using GCC standard library.

5.2 Default Use of Library Function in SDK

In current SDK, the default use of library function is illustrated in Fig 5-1 and Table 5-1

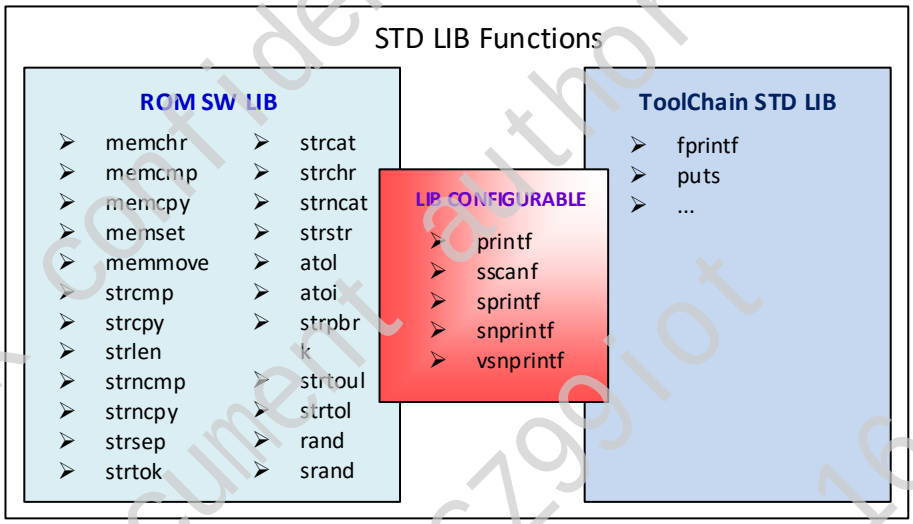


Fig 5-1 Library function guide

Table 5-1 Default use of Library function

Items	Functions
Using ROM software library	memchr/memcmp/memcpy/memset/memmove/strcmp/strcpy/strlen/strncmp/strncpy/strsep/strtok/strcat /strchr/strncat/strstr/atoll/atoi/strpbrk/strtoul/strtol/rand/srand
Using GCC standard library	fprintf/puts/…
Use library is configurable	printf/sprintf/snprintf/vsnprintf/sscanf **Note:** The default is to use ROM software library; you can switch to use GCC standard library by defining macro **STD_PRINTF.**

Because printf/sprintf/snprintf/vsnprintf/sscanf in ROM software library are a simplified version compared with these in GCC standard library, they only support a few formats. In order to remind users that unsupported format occurs when using these functions in ROM library, SDK wraps up these functions.

Table 5-2 Wrapper functions

Items	Wrapper functions
printf	_rtl_printf
sprintf	_rtl_sprintf

snprintf	_rtl_snprintf
vsnprintf	_rtl_vsnprintf
sscanf	_rtl_sscanf

When using printf/sprintf/snprintf/vsnprintf/sscanf in GCC standard library, the memory size will increase 40KB compared with using these functions in ROM software library.

5.3 How to Use Configurable Function in GCC Standard Library?

Compared to printf/sprintf/snprintf/vsnprintf/sscanf functions in GCC library, which only support a few formats in ROM library. The following content takes printf as an example.

Table 5-3 Printf supported format

Library	Printf Supported Format
ROM software lib	%s, %x, %X, %p, %P, %d, %c
GCC standard lib	%s, %x, %X, %p, %P, %d, %c, %f, %L, %l, %u, %U, %e, %E, …

For printf/sprint/snprintf/vsnprintf/sscanf, ROM library is linked by default. If "format not support!" log dumps out in trace tool, it means that unsupported format occurs, you should link these functions to GCC standard library.

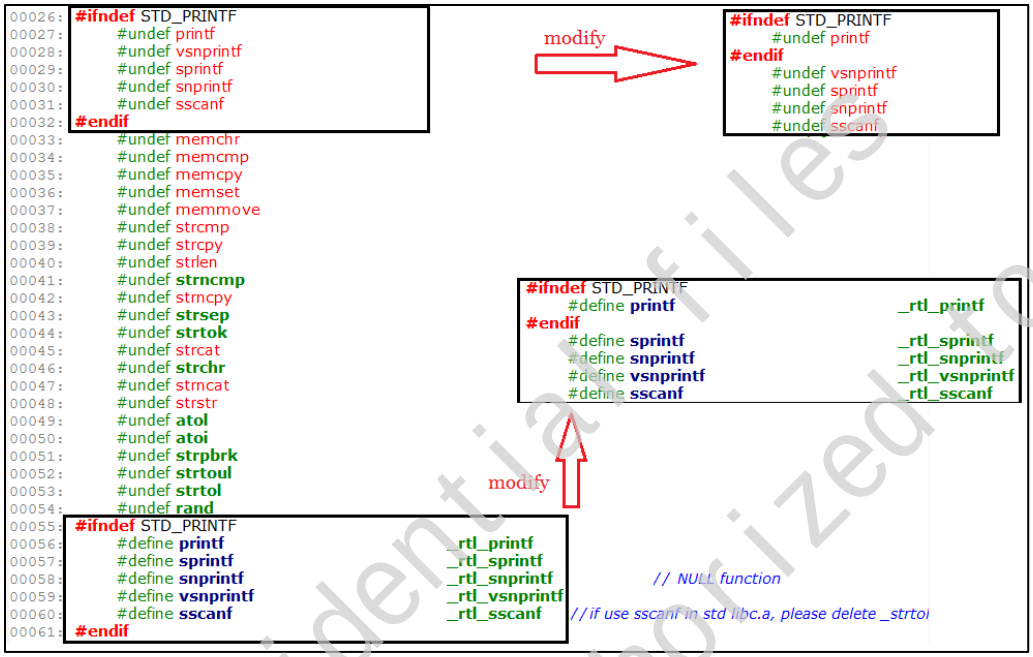
To use printf/sprint/snprintf/vsnprintf/sscanf in GCC standard library, follow these steps:
(1) Add **#define STD_PRINTF** in the front.
There are two cases:

Items	Description

case 1: add **#define STD_PRINTF** in the front of c files, and need to include "**platform_stdlib.h**"	Change printf/sprint/snprintf/vsnprintf/sscanf of some files link to GCC standard library.
	Example:
	`// main.c`
	`#define STD_PRINTF`
	`#include "main.h"`
	`#include "platform_stdlib.h"`
	`int main(void)`
	`{`
	`printf("%f\n", 1.2);`
	`printf("%u\n", 20);`
	`}`

case 2: add **#define STD_PRINTF** in the front of "**platform_stdlib_rtl8721d.h**"	Change printf/sprint/snprintf/vsnprintf/sscanf of all SDK link to GCC standard library.

(2) Modify "**platform_stdlib_rtl8721d.h**" file if you don't want to switch to use all the five functions in GCC standard library.
In SDK, macro **STD_PRINTF** controls printf/sprintf/snprintf/vsnprintf/sscanf at the same time. If you only want to configure some but not all, you need to modify "**platform_stdlib_rtl8721d.h**" file.
For example, if you only want to use printf in GCC standard library, and other four functions sprintf/snprintf/vsnprintf/sscanf maintain the default state, you need to modify "**platform_stdlib_rtl8721d.h**" file.



5.4 Tips

- If user wants to use GCC standard library, we recommend only link some user specified c files to GCC standard library instead of link all SDK to GCC standard library, because printf/sprint/snprintf/vsnprintf/sscanf in our SDK should link to ROM standard library.
- If use printf in GCC standard library libc.a, and there is no "\n" in the end, please use fflush(stdout) after printf to dump log in cache. For example:

  printf("hello");
  fflush(stdout);
- In current GCC project, KM0 does not support floating-point temporarily, so printf cannot dump %f in KM0 in default GCC project setting.
- if using sscanf in GCC standard library libc.a, delete _strtol_r symbol in rlx8721d_rom_symbol_acut.ld.

6 IAR Build Environment Setup

This chapter illustrates how to setup IAR develop environment for Realtek Ameba-D SDK, including building the project, downloading images and debugging.

6.1 Requirement

6.1.1 IAR workbench

IAR provides an IDE environment for code building, downloading, and debugging. Check “IAR Embedded Workbench” on http://www.iar.com/, and a trail version is available for 30 days.

Note: To support ARMv8-M with Security Extension (Ameba-D HS CPU, also called KM4), IAR version must be 8.30 or higher.

6.1.2 J-Link probe

If users need to download images or debug code for Ameba-D with IAR, then a J-Link adapter is necessary.

Note: For Ameba-D CPU, the J-Link probe version must be v9 or higher.

6.2 Hardware Configuration

The hardware block diagram of Ameba-D demo board is shown in Fig 6-1. The following modules are mentioned in subsequent chapters.

- USB TO UART: supply power and print logs, baud rate is 115200 bps.
- SWD: J-Link SWD interface, used for image downloading and debugging with IAR.
- Reset button: reset Ameba-D to run firmware after IAR completes downloading.

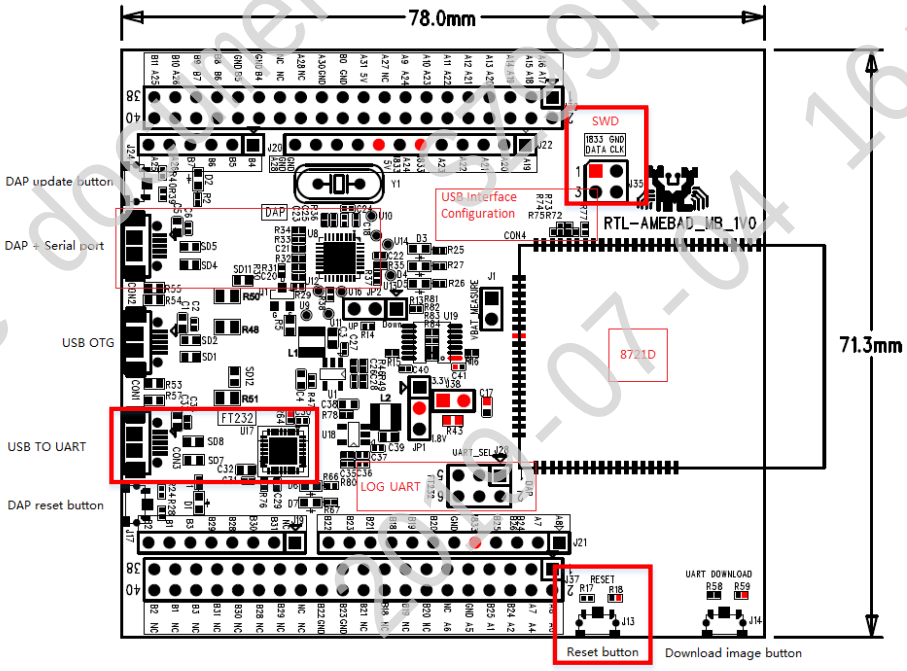


Fig 6-1 Ameba-D demo board

The Ameba-D SWD interface should be connected with J-Link probe as follows:

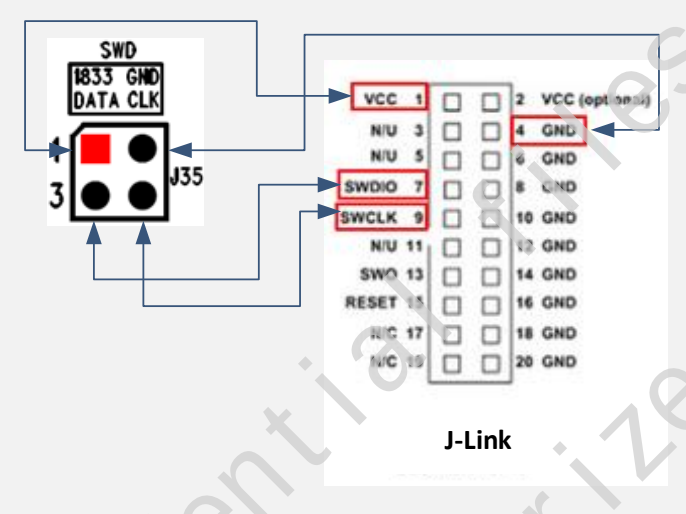


Fig 6-2 J-Link SWD connection diagram

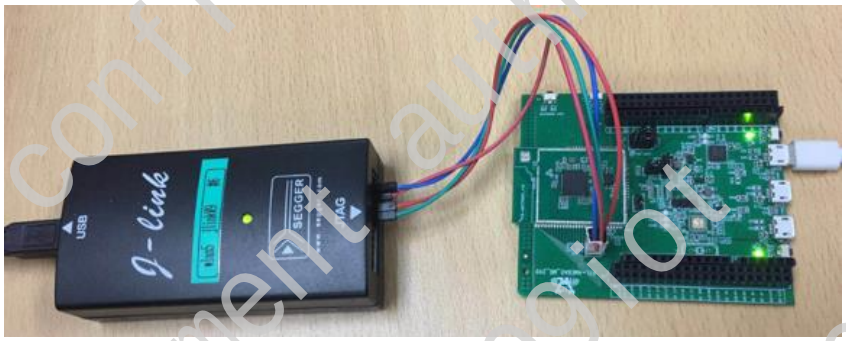


Fig 6-3 J-Link and Ameba-D SWD connection

6.3 How to Use IAR?

6.3.1 IAR Build

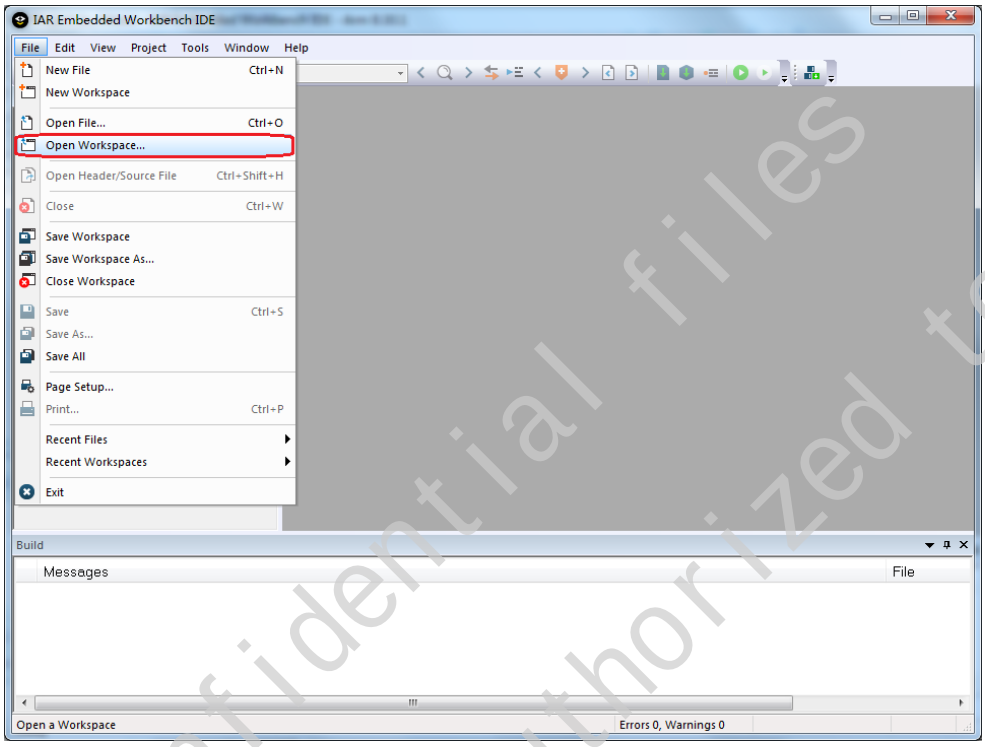
Because Ameba-D is dual-core CPUs platform, two workspaces are provided to build for each core in **project\realtek_amebaD_va0_example\EWARM-RELEASE**.

- Project_lp_release.eww (KM0 workspace) contains the following sub-projects:
  - km0_bootloader
  - km0_application
- Project_hp_release.eww (KM4 workspace) contains the following sub-projects:
  - km4_bootloader
  - km4_application
  - km4_secure: project of Trustzone-protected code, also known as RDP. It's not necessary if RDP is not enabled in your system.

The following steps show how to build Ameba-D project:

(1) Open IAR Workbench.

(2) Click File > Open Workspace… to open project.



(3) Select Project_lp_release.eww (km0 project) or Project_hp_release.eww (km4 project).
Click Project > Options, General Options > Target > Processor Variant > Core, verify the CPU configurations according to Table 6-1.

Note: When building SDK for the first time, users should build BOTH km0 project and km4 project. Other times, users only need to make the modified project.

Table 6-1 CPU Configuration

Workspace	Project	Core	FPU	DSP Extension	TrustZone
KM0	km0_bootloader	Cortex-M23	-	-	-
	km0_application				
KM4	km4_bootloader	Cortex-M33	VFPv5 single precision	Yes	Secure
	km4_application				Non-secure
	km4_secure				Secure

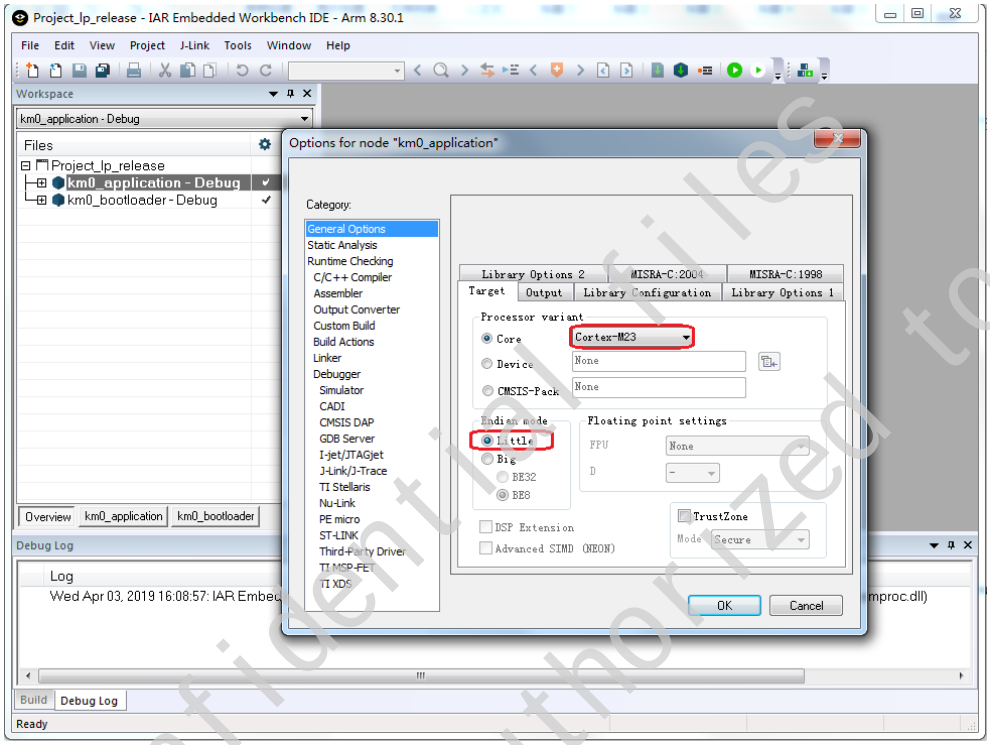


Fig 6-4 KM0 processor options

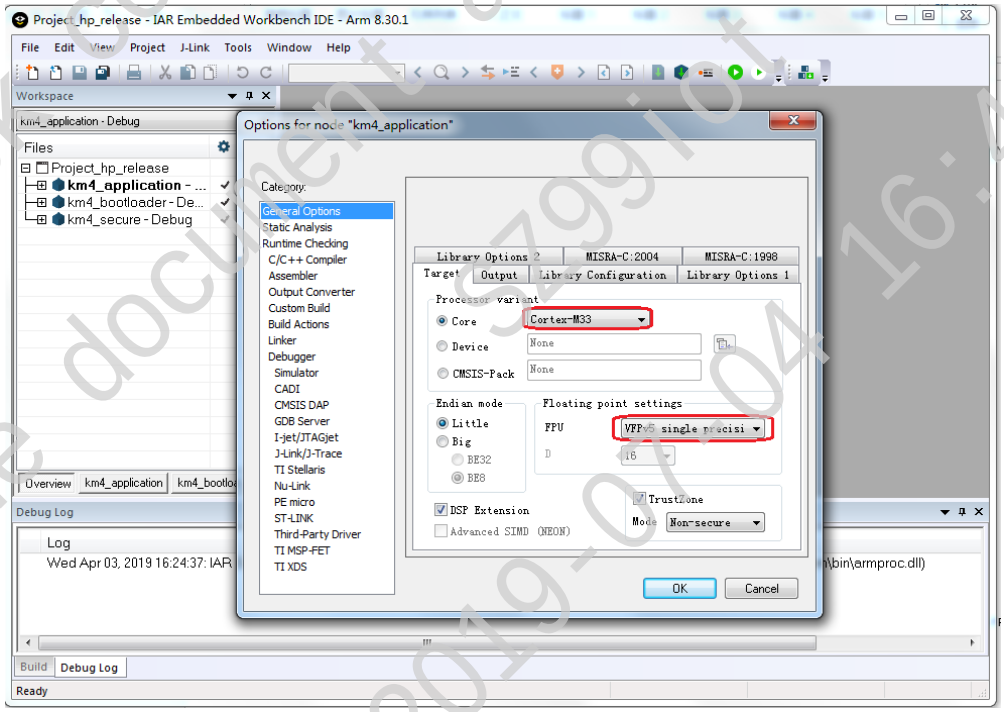


Fig 6-5 KM4 processor options

(4) Add your own files to the project if customized development is required.
Click Project > Add Files…/Add Group…, then select target files.

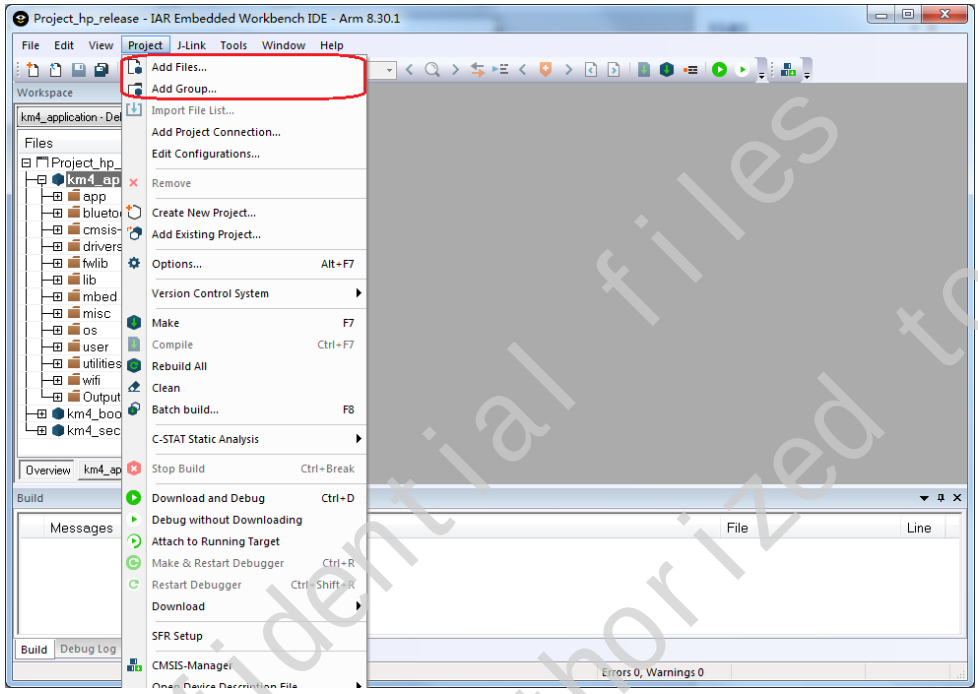


Fig 6-6 Add Files

Note: By default, the code in new files would be placed in Flash and execute in place (XIP). If you would like to execute in SRAM, right click the target group or files, click Project > Options > C/C++ Compiler > Output, set “Code section name” as “.image2.ram.text”.

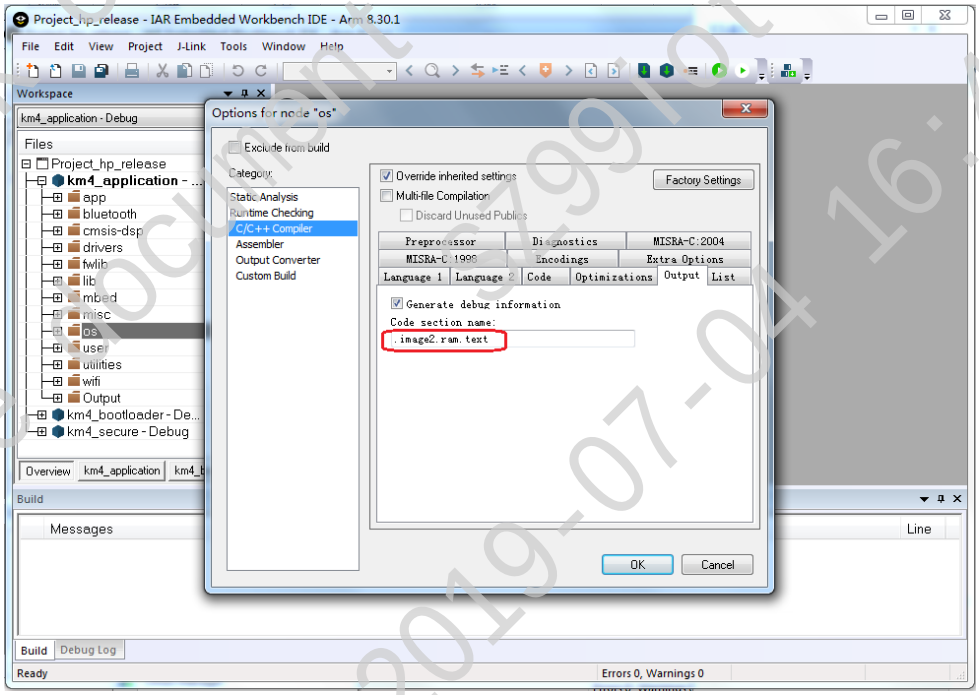


Fig 6-7 Designate files to SRAM

(5) To build project, click Project > Rebuild All/Make.

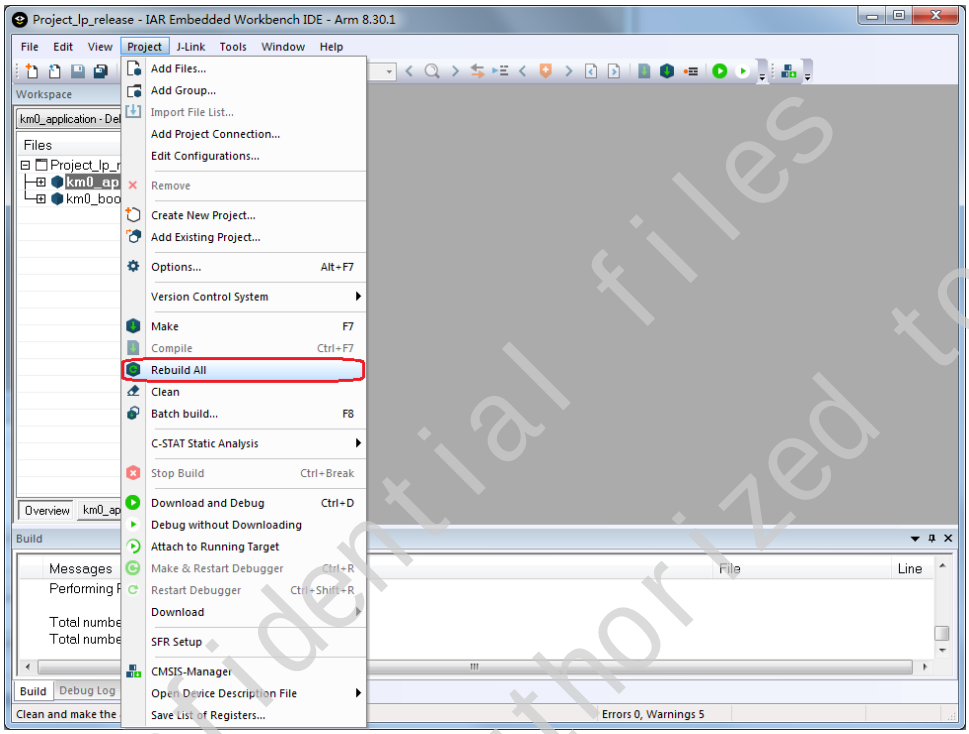


Fig 6-8 Build project

Note: After building, IAR would execute Post-build action to generate images from executable files. This may takes several seconds. As a result, users should wait to see and make sure the new images are generated before downloading. Otherwise, the old images would download instead.

Finally, you would get images in **project\realtek_ameba_va0_example \EWARM-RELEASE \Debug\Exe**.

Table 6-2 Generated images

Workspace	Project	Generated Images	Download Image
KM0	km0_bootloader	km0_boot_all.bin	km0_boot_all.bin
	km0_application	km0_image2_all.bin km0_km4_image2.bin (Combined IMG2)	km0_km4_image2.bin
KM4	km4_bootloader	km4_boot_all.bin	km4_boot_all.bin
	km4_application	km4_image2_all.bin km0_km4_image2.bin (Combined IMG2)	km0_km4_image2.bin
	km4_secure	km4_image3_all.bin	-

The generated images can be downloaded in two ways:

- IAR J-Link SWD, and this would be introduced in next section.
- Ameba-D Image Tool, please refer to Chapter Image Tool for more information.

6.3.2 IAR Download

The Ameba-D demo board supports JLINK SWD download and debug.

Image of each project can be download individually as Table 6-2 shows. As km0 and km4 application image are combined in memory layout, so they would be download together. “km4_image3_all.bin” needs to be encrypted before download, so it can’t download directly. You can refer to RDP chapter to see how to manipulate it.

Note: Considering KM4 is power-on by KM0, users should make sure that KM0 has already boot up before download images to KM4. Otherwise, J-Link can't connect to KM4 and show the error message as Fig 6-9 shows.

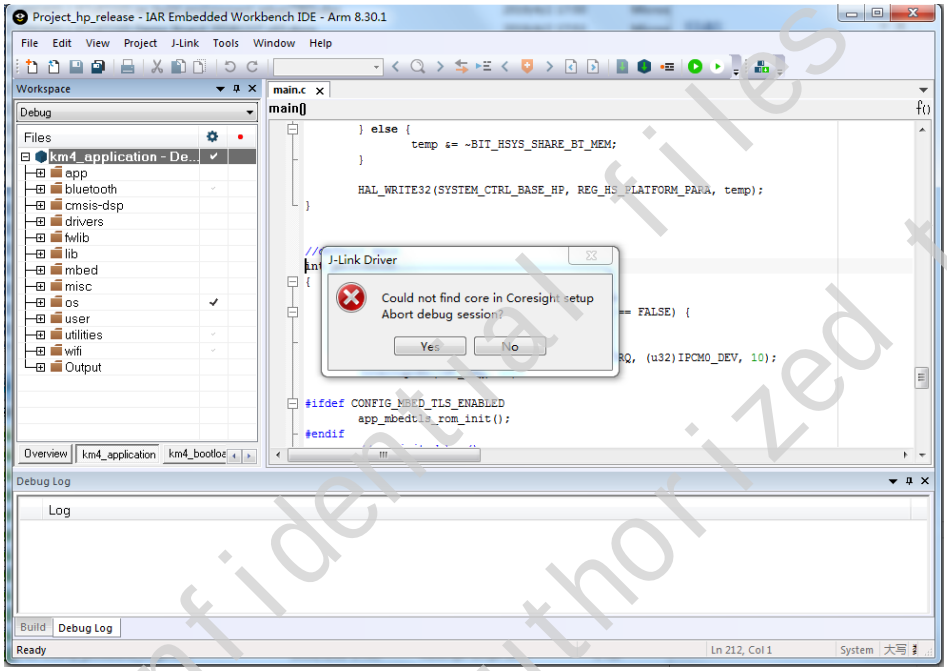


Fig 6-9 J-Link can't find KM4

As a result, if the flash memory is empty, the sequence to download images is:
(1) Download for km0_bootloader and km0_application projects
(2) Click Reset button on demo board to make KM0 boots up
(3) Download for km4_bootloader and km4_application projects

During development, if flash memory is not empty and km0 can boot up successfully, then users can download updated images to KM4 directly, and there is no need to re-download for km0.

The following steps show how to download image for target project with IAR.
(1) Set the target project as active project in the workspace.
- Workspace > Overview, right click the target project > Set as Active, as Fig 6-10 shows.
- Or switch to the target project view, as Fig 6-11 shows.

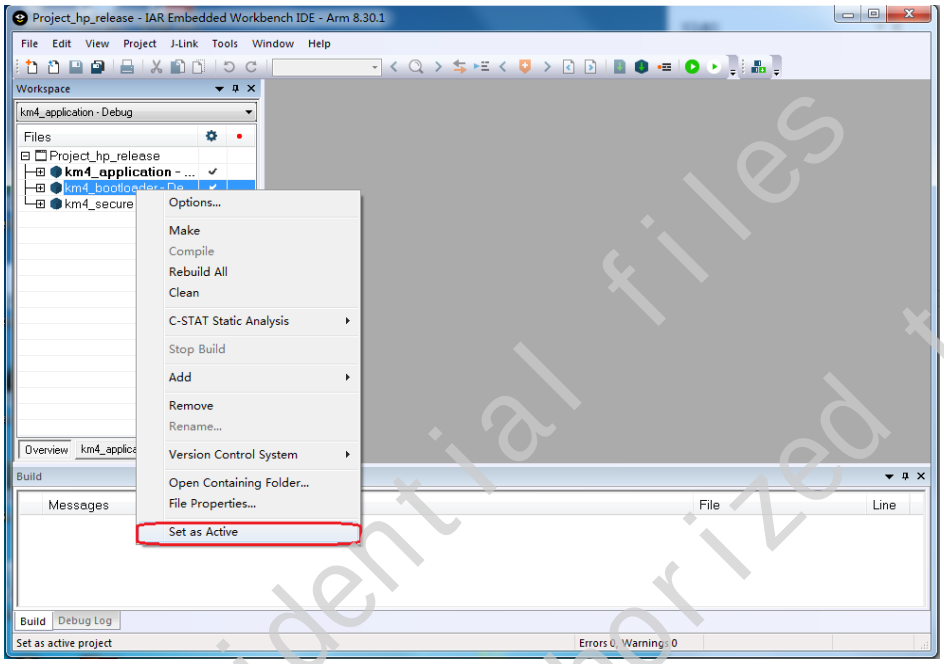


Fig 6-10 Setting Target Project as Active

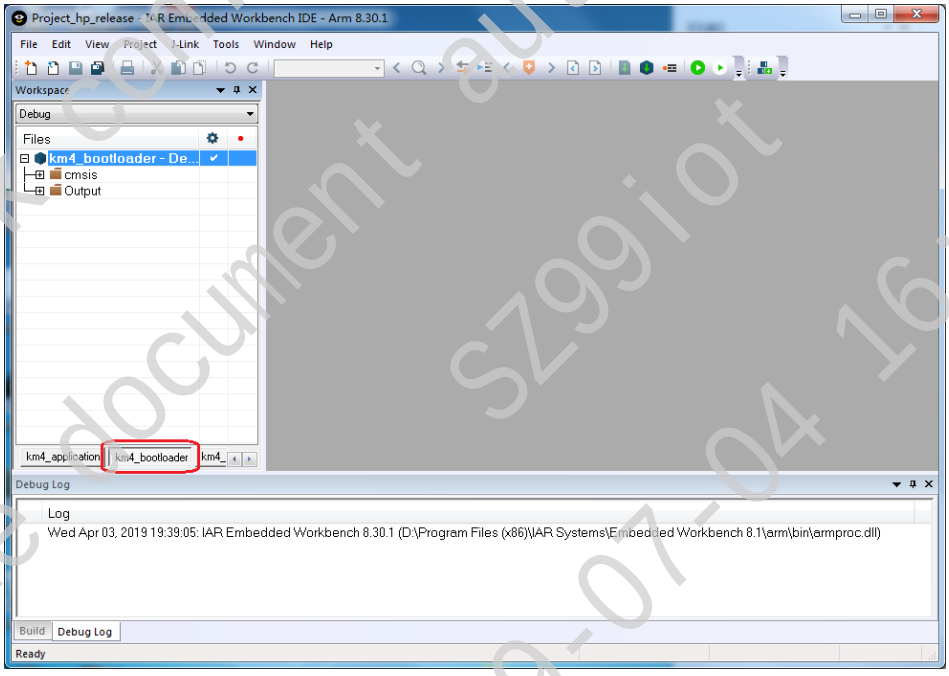


Fig 6-11 Switch to Target Project View

(2) Check whether the J-link debugger setting is correct.

a) Click Project > Options > Debugger > Setup > Driver, and choose “J-Link/J-Trace”.

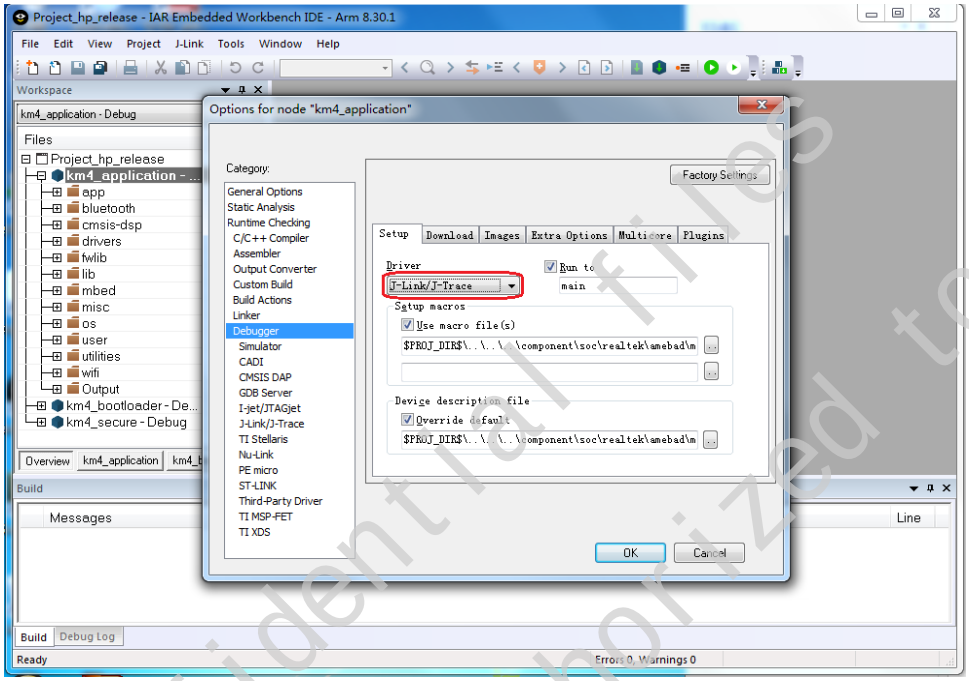


Fig 6-12 Debugger setup

b) Click Debugger > J-Link/J-Trace > Connection > Interface, and choose “SWD”.

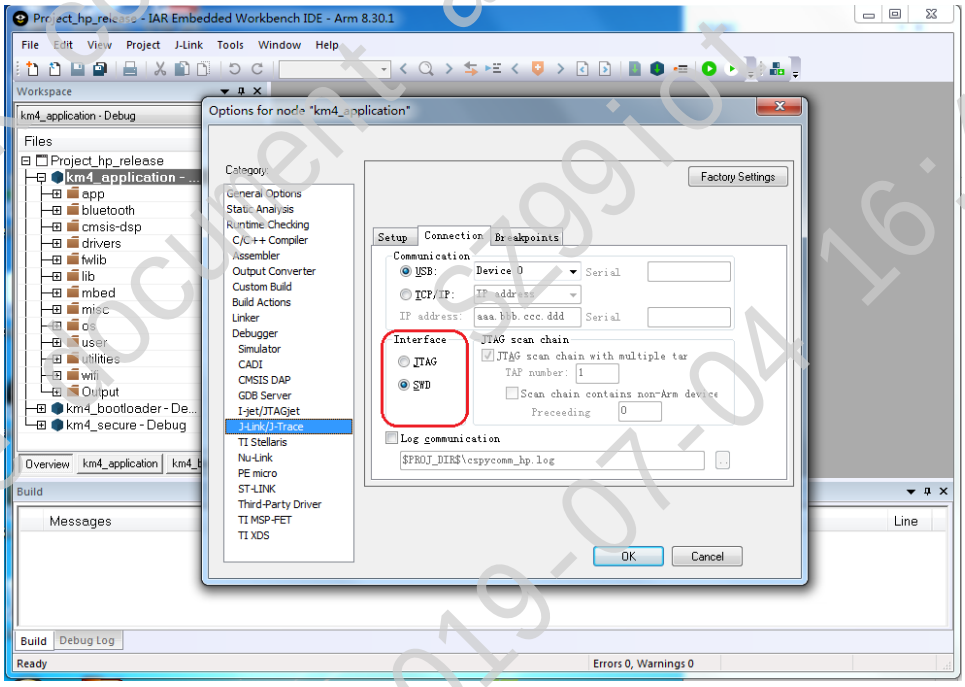


Fig 6-13 J-Link interface setup

(3) Click Project > Download > Download active application, image downloading starts.

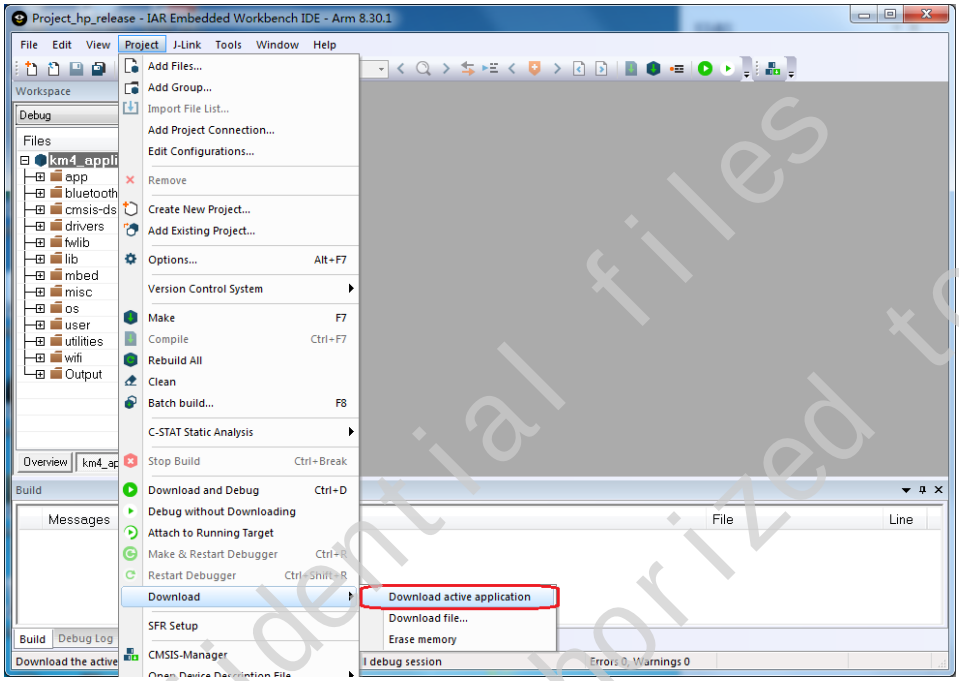


Fig 6-14 Download Active Application

(4) When downloading, Ameba-D prints the log. Users can check the log to see if download is successful.

```
[FlashInit] image_size:f48, link_address:8000000, flags:0
[FlashErase] block_start:0, block_size:1000
[FlashWrite] block_start:0, offset_into_block:0, count:f48
[FlashInit] image_size:82000, link_address:8006000, flags:0
[FlashErase] block_start:6000, block_size:1000
[FlashErase] block_start:7000, block_size:1000
[FlashErase] block_start:8000, block_size:1000
[FlashErase] block_start:9000, block_size:1000
[FlashErase] block_start:a000, block_size:1000
[FlashErase] block_start:b000, block_size:1000
[FlashErase] block_start:c000, block_size:1000
[FlashErase] block_start:d000, block_size:1000
[FlashErase] block_start:e000, block_size:1000
[FlashErase] block_start:f000, block_size:1000
[FlashErase] block_start:10000, block_size:1000
[FlashErase] block_start:11000, block_size:1000
[FlashWrite] block_start:6000, offset_into_block:0, count:c000
[FlashErase] block_start:12000, block_size:1000
[FlashErase] block_start:13000, block_size:1000
[FlashErase] block_start:14000, block_size:1000
[FlashErase] block_start:15000, block_size:1000
[FlashErase] block_start:16000, block_size:1000
[FlashErase] block_start:17000, block_size:1000
[FlashErase] block_start:18000, block_size:1000
[FlashErase] block_start:19000, block_size:1000
[FlashErase] block_start:1a000, block_size:1000
[FlashErase] block_start:1b000, block_size:1000
[FlashErase] block_start:1c000, block_size:1000
[FlashErase] block_start:1d000, block_size:1000
[FlashWrite] block_start:12000, offset_into_block:0, count:c000
[FlashErase] block_start:1e000, block_size:1000
[FlashErase] block_start:1f000, block_size:1000
[FlashErase] block_start:20000, block_size:1000
[FlashErase] block_start:21000, block_size:1000
[FlashErase] block_start:22000, block_size:1000
[FlashErase] block_start:23000, block_size:1000
[FlashErase] block_start:24000, block_size:1000
```

Fig 6-15 Downloading log

(5) You can also erase all parts of the flash memory if necessary.

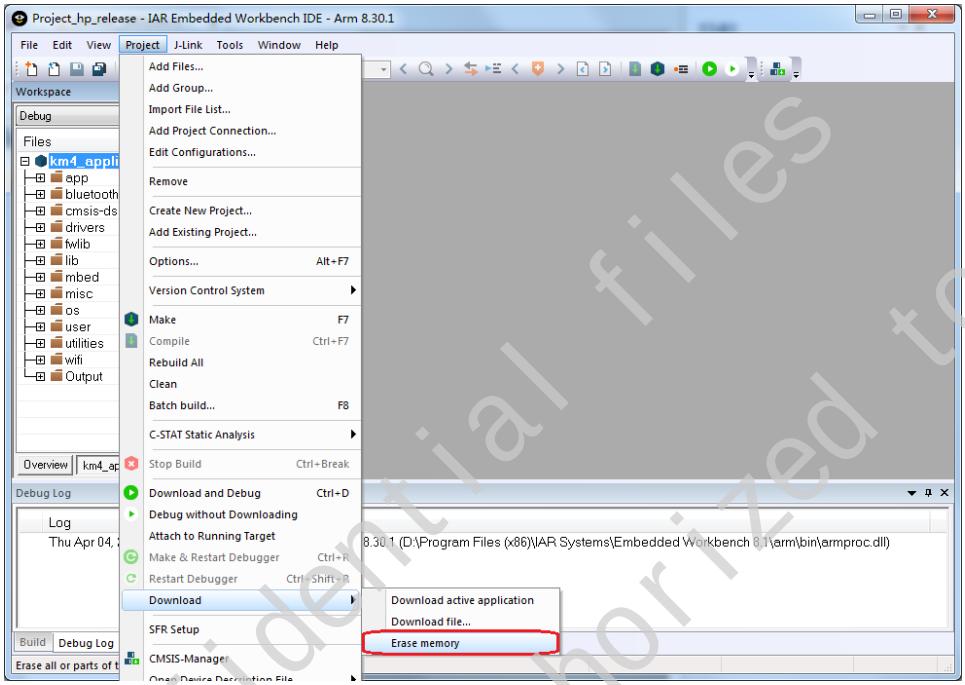


Fig 6-16 Erase Flash memory

6.3.3 IAR Debug

Users can debug or trace KM0 and KM4 system individually with J-Link SWD.

Note: Considering KM4 is power-on by KM0, users should make sure that KM0 has already boot up before debug KM4. For KM0, there is no such requirement because KM0 is power-on immediately after reset.

Users should follow the steps to debug and trace code of target project with IAR:

(1) Set the target project as active project and verify the debugger configurations as 0 step (1) and step (2).

(2) Click Project > "Download and Debug" or "Debug without Downloading"

- Download and Debug: downloads the application and debug the project object file. If necessary, a make will be performed before download to ensure the project is up to date.
- Debug without Downloading: debug the project object file

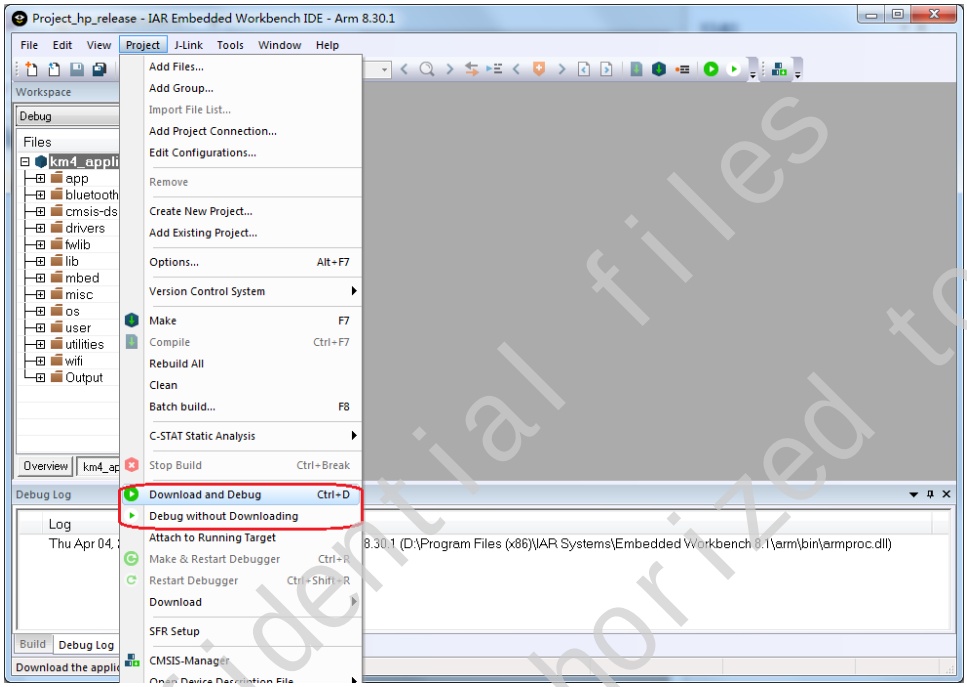


Fig 6-17 Debug options

(3) When starting IAR C-SPY to debug, it will firstly reset the target CPU and run to main function.

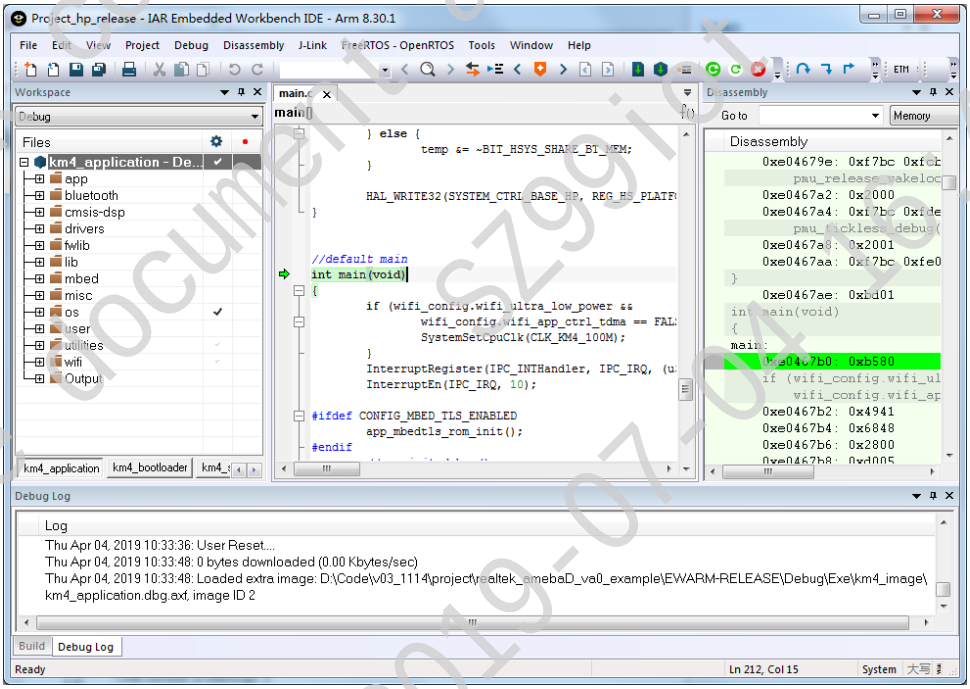


Fig 6-18 Run to main when debugging

(4) Toggles a breakpoint at the statement or instruction that contains or is located near the cursor in the source window. The “Toggle Breakpoint” button is on the debug toolbar.

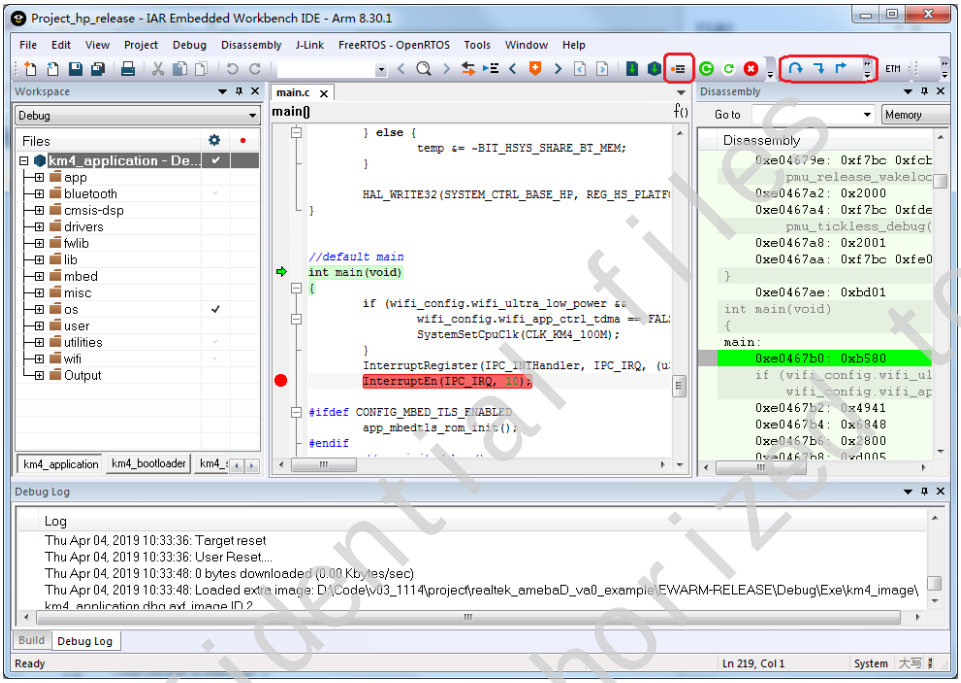


Fig 6-19 Toggle breakpoint

(5) You can trace code step by step with “Step Into” or “Go” until triggering a breakpoint. These function buttons are available on toolbar.

6.4 How to Build Sample Code?

The example source code is located in “project\realtek_amebaD_va0_example\example_sources”. To build sample code, you should copy the “main.c” file in the target example to “project\realtek_amebaD_va0_example\src\src_hp\” and replace the original one.

For example, to use i2c_int_mode example code, you can copy “main.c” from “project\realtek_amebaD_va0_example\example_sources\I2C\mbed\i2c_int_mode\src\”.

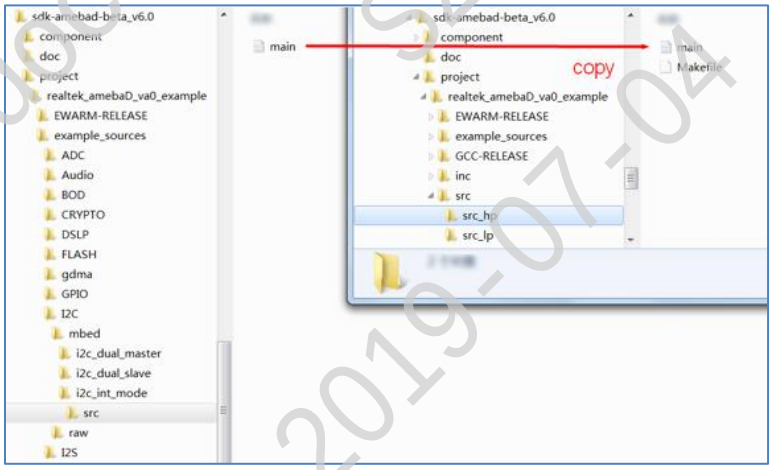


Fig 6-20 Building sample code

7 Demo Board

7.1 PCB Layout Overview

The PCB layout overview is shown in Fig 7-1.

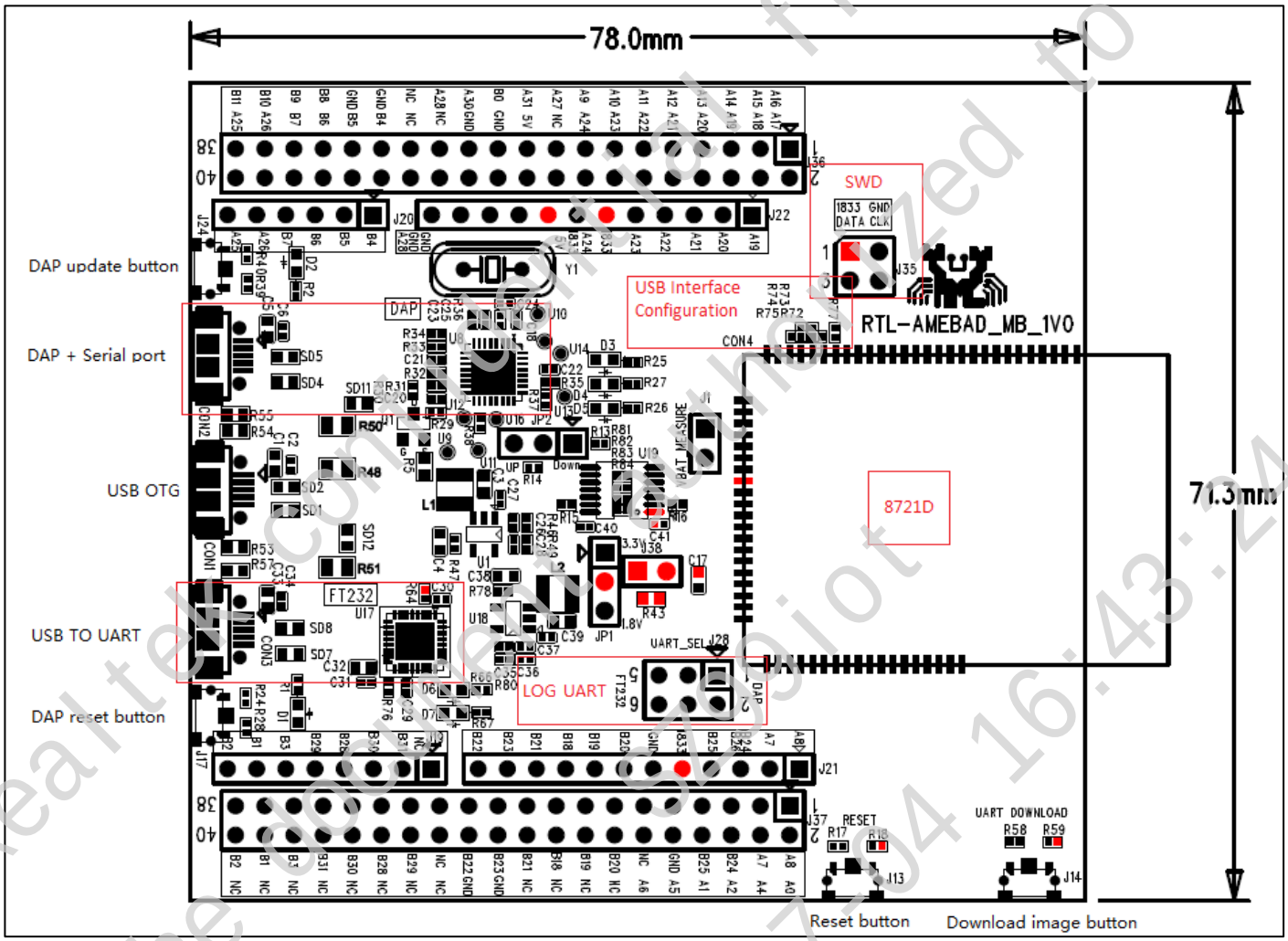


Fig 7-1 Demo board – PCB layout overview

7.2 Pin Out

The pin out board is shown in Fig 7-2.

- Blue Box is for Arduino REF
- Red Box is all the GPIO pins

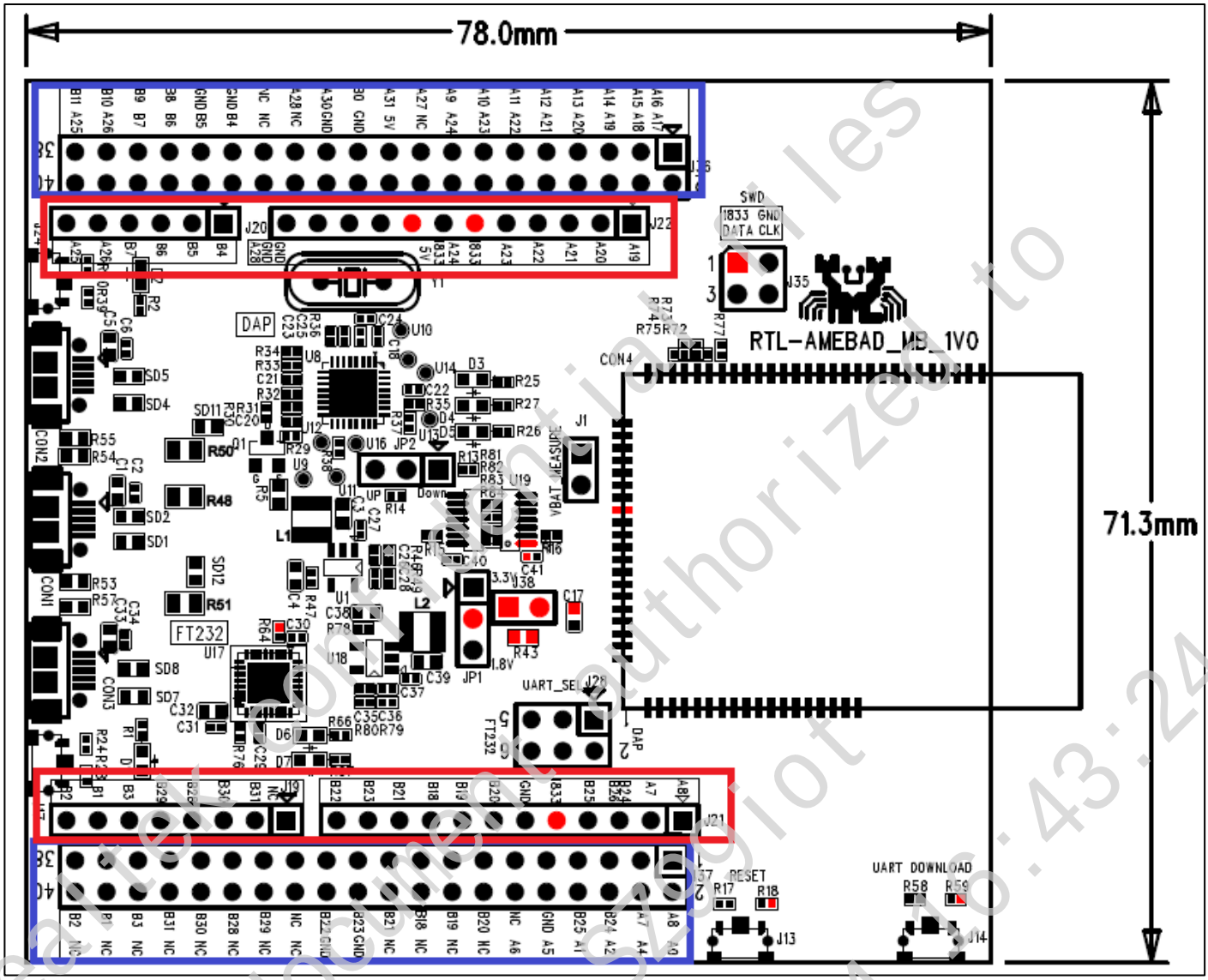


Fig 7-2 Demo board – pin out

7.3 DC Power Supply

The 3.3V/1.8V power supply board is shown in Fig 7-3.

- Jump JP1 is used to select 3.3V or 1.8V power supply
- Jump J38 is for current test. You can test the current power after taking off the R43.

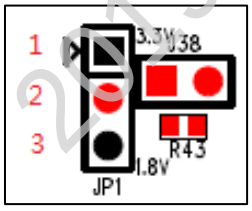


Fig 7-3 Demo board – 3.3V/1.8V power supply

When you select power supply, please refer to Table 7-1.

Table 7-1 3.3V/1.8V power supply selection

Power Supply Select	JP1
3.3V	1-2 connected
1.8V	2-3 connected

7.4 USB Interface Configuration

The USB interface configuration board is shown in Fig 7-4 and Fig 7-5. For normal GPIO usage by default, R72/R75/R77 on mother board will part on with 0ohm resistors, R5 on module board needs to take off. For USB usage, you need to take off R77, part on R73&R74 with 0ohm resistors on mother board and part on R5 on module board with a 12kohm 1% precision resistor.

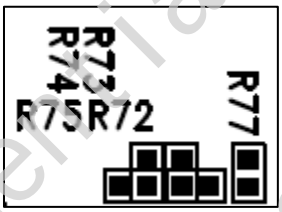


Fig 7-4 Mother board – USB interface configuration

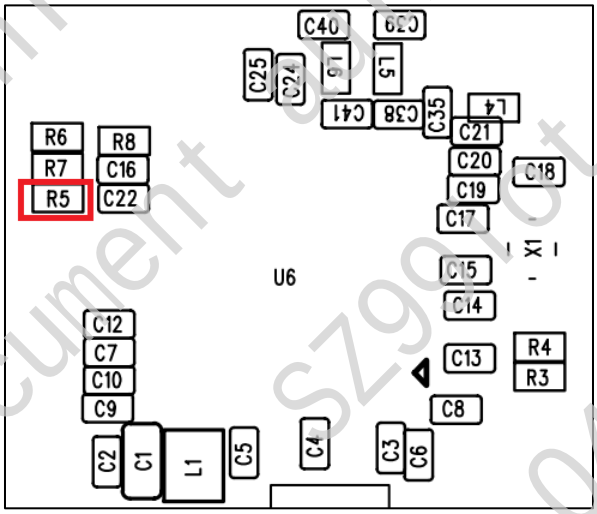


Fig 7-5 Module board – USB interface configuration

7.5 LOGUART

The LOGUART board is shown in Fig 7-6. When you select LOGUART, please refer to Table 7-2.

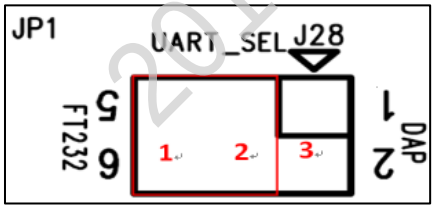


Fig 7-6 Demo board – LOGUART

Table 7-2 LOGUART selection

LOGUART Select	JP1
FT232	1-2 connected
DAP	2-3 connected

7.6 SWD

The SWD board is shown in Fig 7-7.

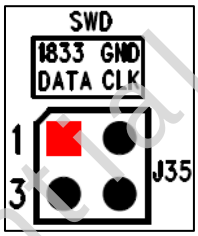


Fig 7-7 Demo board – SWD

Note: For 1V0 board, there is a bug, you should use CLK as DATA, and use DATA as CLK.

7.7 VBAT ADC

The VBAT ADC board is shown in Fig 7-8. J1 is used to test VBAT ADC.

Fig 7-8 Demo board – VBAT ADC

7.8 CMSIS-DAP & LOGUART (TBD)

Demo board supports CMSIS-DAP debugger. It requires installing serial to USB driver at first. It can be found in **tools\serial_to_usb\mbedWinSerial_16466**.

Connect board to the PC with micro-USB cable.

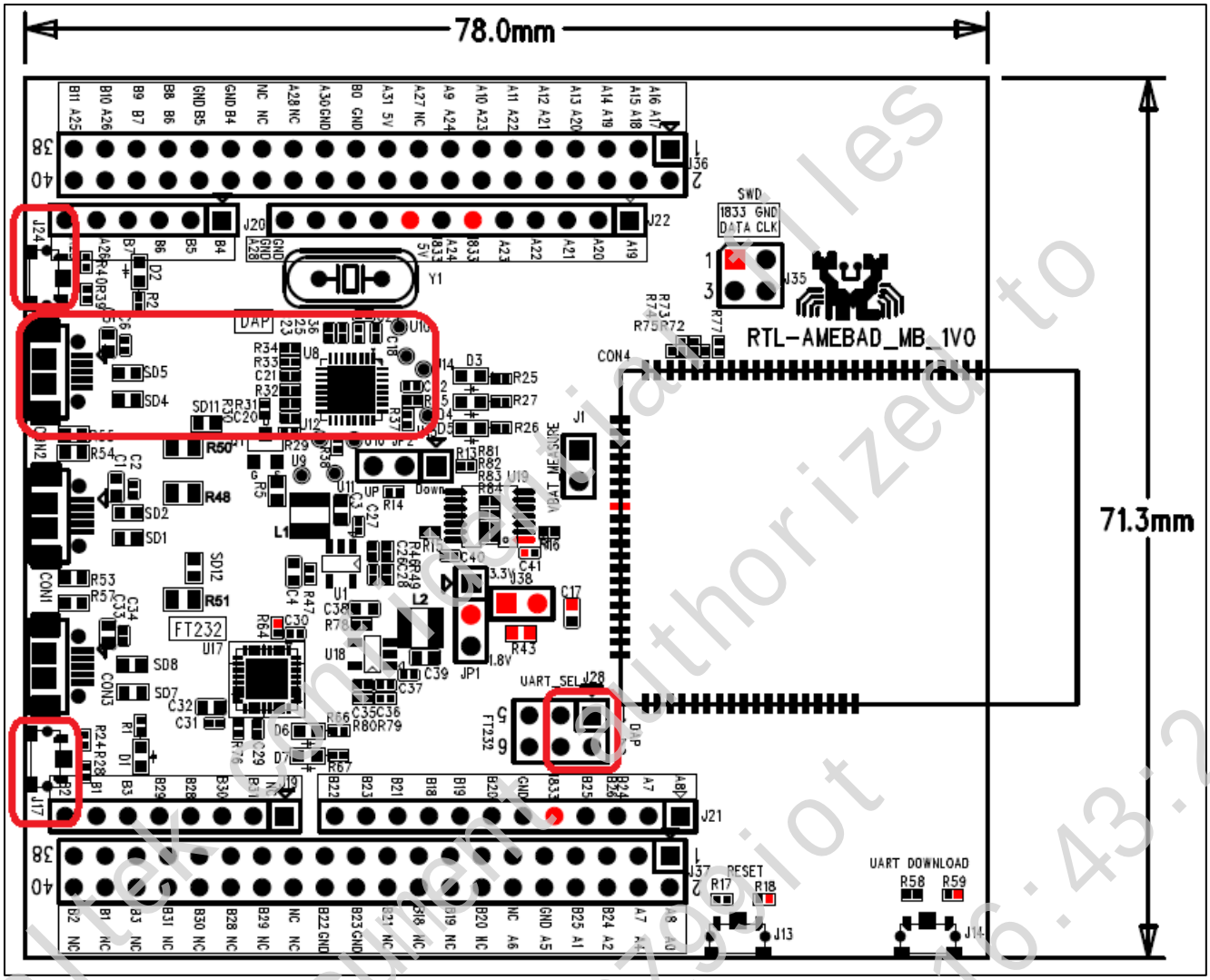


Fig 7-9 Demo board – CMSIS-DAP & LOGUART

7.9 DAP Firmware Update (TBD)

In DAP mode, the DAP firmware can be updated. Hold **TGT_NRESET** button (J24, red-circled), then press **nRESET** button (J17, blur-circled), the DAP mode window will show up.

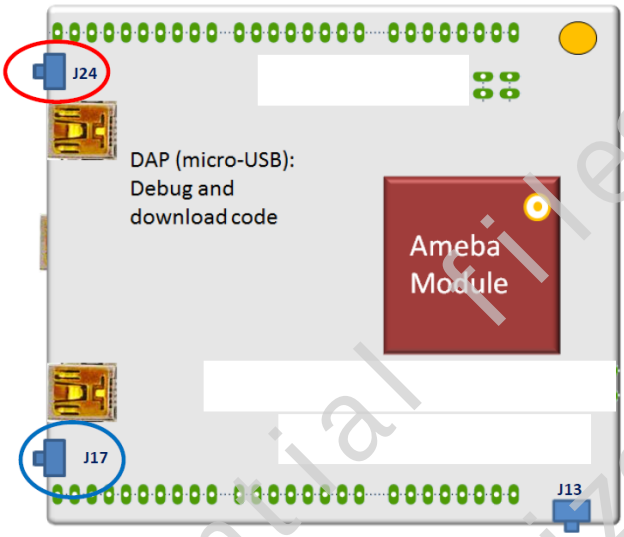


Fig 7-10 Demo board – CMSIS-DAP firmware update

8 ImageTool

8.1 Introduction

This chapter introduces how to use ImageTool to encrypt, generate and download images. As show in Fig 8-1, ImageTool has four tabpages.

- Download: used as image download server to transmit images to Ameba through UART.
- Generate: contact individual images and generate a composite image.
- Encrypt: encrypt images which are used for firmware protection.
- Security: generate key pair and signature for secure bootloader and save as new image.

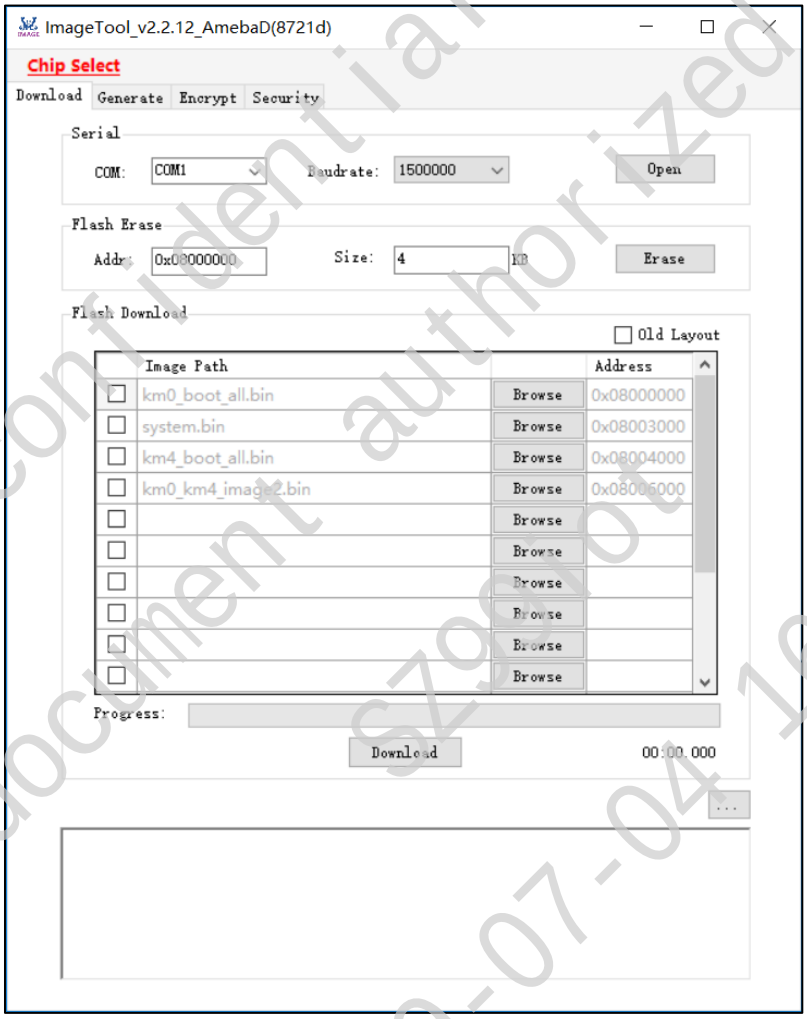


Fig 8-1 ImageTool UI

8.2 Environment Setup

8.2.1 Hardware Setup

The hardware setup is shown in Fig 8-2.

Note: If using external UART to download images, FT232 USB to UART dongle must be used.

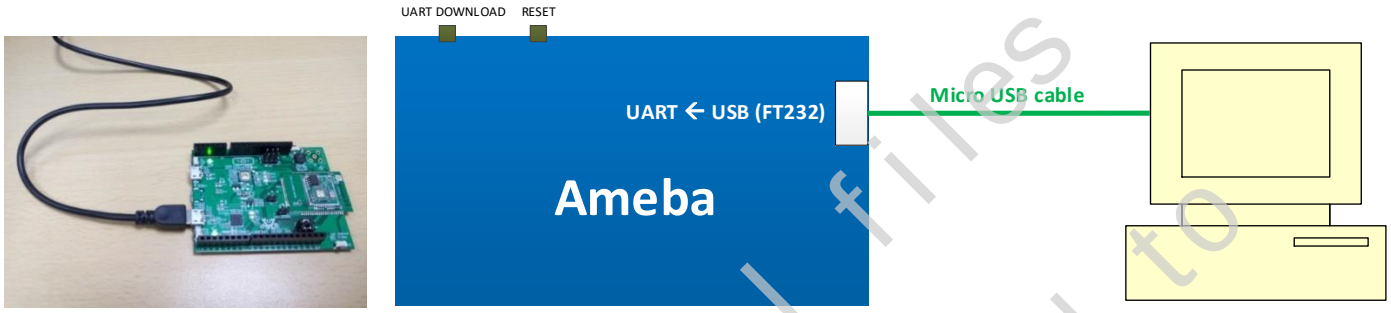
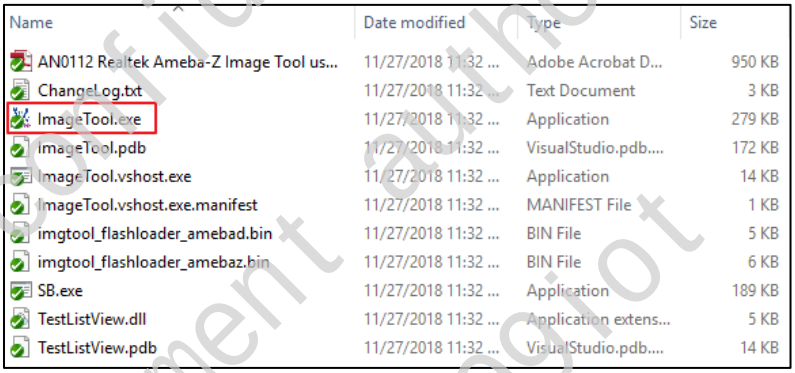


Fig 8-2 Hardware setup

8.2.2 Software Setup

- Environment Requirements: EX. WinXP, Win 7 Above, Microsoft .NET Framework 3.5
- ImageTool.exe Location: **\tools\AmebaZ\Image_Tool\ImageTool.exe**



8.3 Download

8.3.1 Image Download

Assuming that the ImageTool on PC is a server, which sends images files to Ameba (client) through UART. The steps to download images to board is as following:

(1) Ameba enters into UART_DOWNLOAD mode.
   a) Push the **UART DOWNLOAD** button and keep it pressed.
   b) Re-power on the board or press the **Reset** button.
   c) Release the **UART DOWNLOAD** button. Now Ameba gets into UART DOWNLOAD mode and is ready to receive data.

(2) Click **Chip Select** (in red) on UI and select chip (AmebaD or AmebaZ).

(3) Select the corresponding serial port and transmission baud rate. The default baud rate is 1.5Mbps (recommended). Then click **Open** button.

(4) Click the **Browse** button to select the images (km0_boot_all.bin/km4_boot_all.bin/km0_km4_image2.bin) to be programmed and input addresses.
   - The image path is located in ***{path}*\project\realtek_amebaD_va0_example\GCC-RELEASE\project_hp\asdk\image** and ***{path}*\project\realtek_amebaD_va0_example\GCC-RELEASE\project_hp\asdk\image**, where ***{path}*** is the location of the project on your own computer.
   - The default target address is the SDK default image address, you can use it directly.

(5) Click **Download** button to start. The progress bar will show the transmit progress of each image. You can also get the message of operation successfully or errors from the log window.

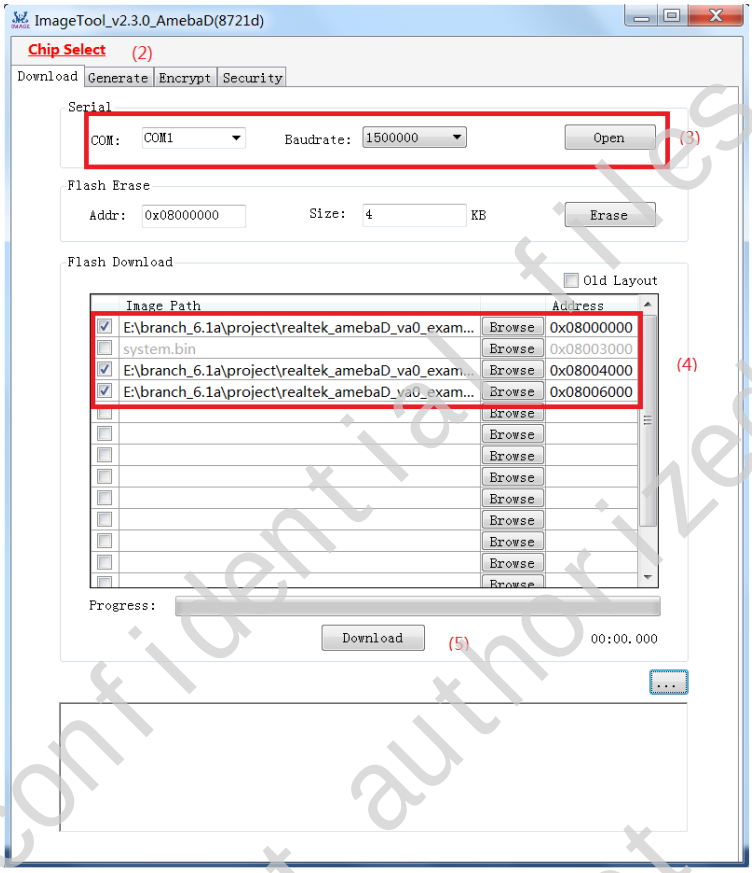


Fig 8-3 ImageTool 'Download' tabpage setting

8.3.2 Flash Erase

Steps to erase flash are as following:

(1) Ameba enters into UART_DOWNLOAD mode as introduced in section 8.3.1.
(2) Select the corresponding serial port and open it.
(3) Input erase start address which has to be 4-byte aligned.
(4) Input erase size which would be cast to a multiple of 4KB.
(5) Click **Erase** button, and erase operation begins. You would get the operation result from the log window.

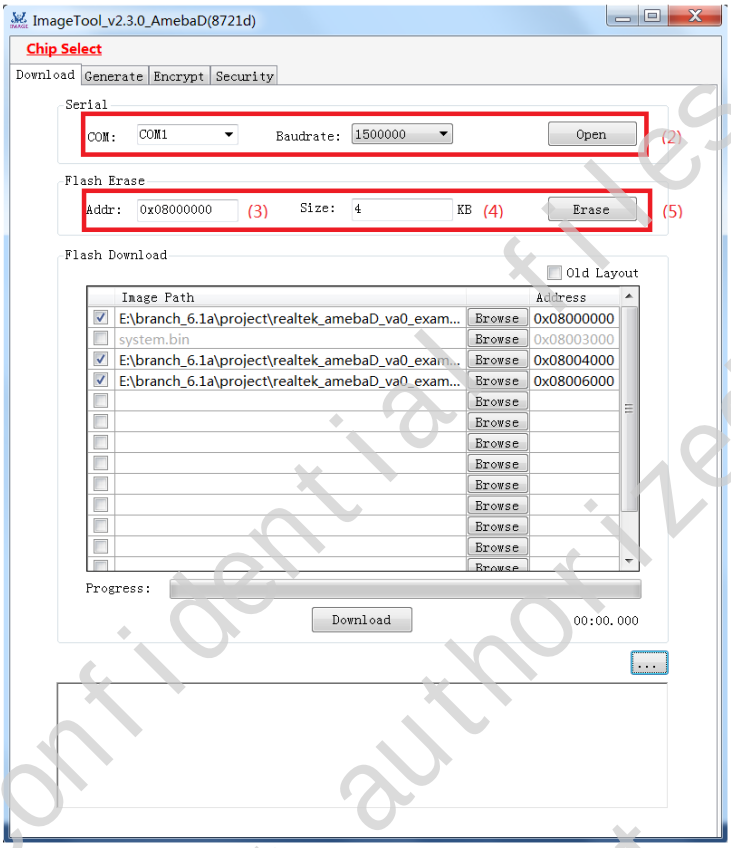


Fig 8-4 Flash erase operation

Note: Users don't need to erase flash manually before download images into flash. Image Download operation would erase flash automatically according to the image and address to be programmed.

8.3.3 Flash Status Operation

After some abnormal operations such as power lost when flash is programming, the flash would enter into Write Protection (WP) to protect some area against program and erase.

To release flash from WP state, the BPs in status register should be cleared. ImageTool provides the function of setting and getting flash status register value. To use it, the following steps are necessary.

(1) Check the Read/Write Status Register Command and Sequence according to the flash datasheet. Notice that different flash IC would have different sequence.
(2) Get Ameba into UART_DOWNLOAD mode as introduced in section 8.3.1.
(3) Select the corresponding serial port and open it.
(4) Click **Advanced Settings** button in Fig 8-5, and the **Advanced Settings** window shown in Fig 8-6 popups.

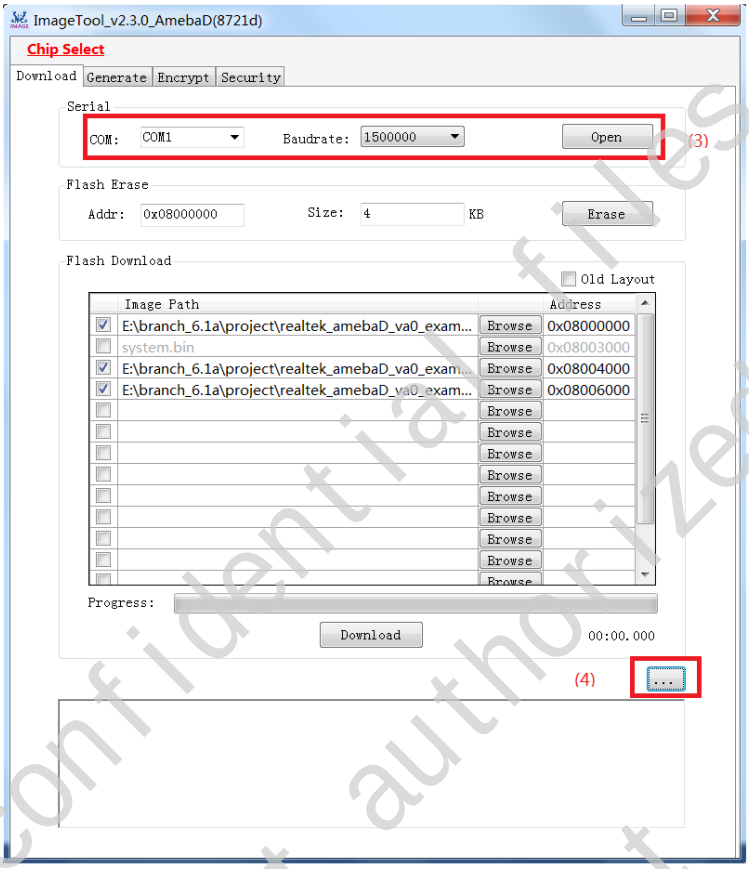


Fig 8-5 'Advanced Settings' window

(5) Select Read/Write Command and Length according to flash datasheet. If write status to flash, remember to input value.
(6) Click **Read** button to read status register value, or **Write** button to write status value to flash.

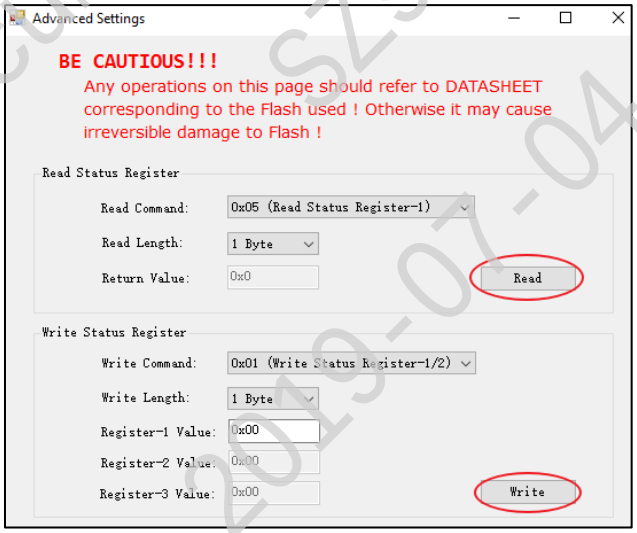


Fig 8-6 Flash status operation

In case of MXIC flash chip, the status register is shown in Table 8-1.

- To clear BP bits, read status register configuration is as below:

- Read Command: 0x05
- Read Length: 1 Byte

- Write status register configuration is as below:
  - Write Command: 0x01
  - Write Length: 1 Byte
  - Register-1 Value: 0x40

Table 8-1 MXIC flash status register

Bit	Name	Description	Volatile Bit
7	SRWD	Status register write protect. ● 1: Status register write disable ● 0: Status register write enable	No
6	QE	Quad enable. ● 1: Quad enable ● 0: Not Quad enable	No
5	BP3	Level of protected block.	No
4	BP2	Level of protected block.	No
3	BP1	Level of protected block.	No
2	BP0	Level of protected block.	No
1	WEL	Write enable latch. ● 1: Write enable ● 0: Not write enable	Yes
0	WIP	Write in progress bit. ● 1: Write operation ● 0: Not in write operation	Yes

Note: For other flash IC, please refer to corresponding datasheet. Otherwise, wrong operation would cause irreversible damage to flash.

8.4 Generate

The 'Generate' tabpage has two functions:

- Merge individual images and generate a composite image named **Image_All.bin**
- Generate Cloud OTA image named **OTA_All.bin**

8.4.1 Merge Images

Steps to generate a composite image are as following:

(1) Select **Image_All** as Generate Target type (in red).

(2) Click **Browse** button to select images to be contacted and input corresponding relative address. The **Memory Layout** bar will show the relative positions of the selected images. If the contiguous images overlap, the overlapped area is in red color for warning.

(3) Click **Generate** button. Specify the output file name and path yourself in the popup '**Save As**' window, the final image (**Image_All.bin**) is generated.

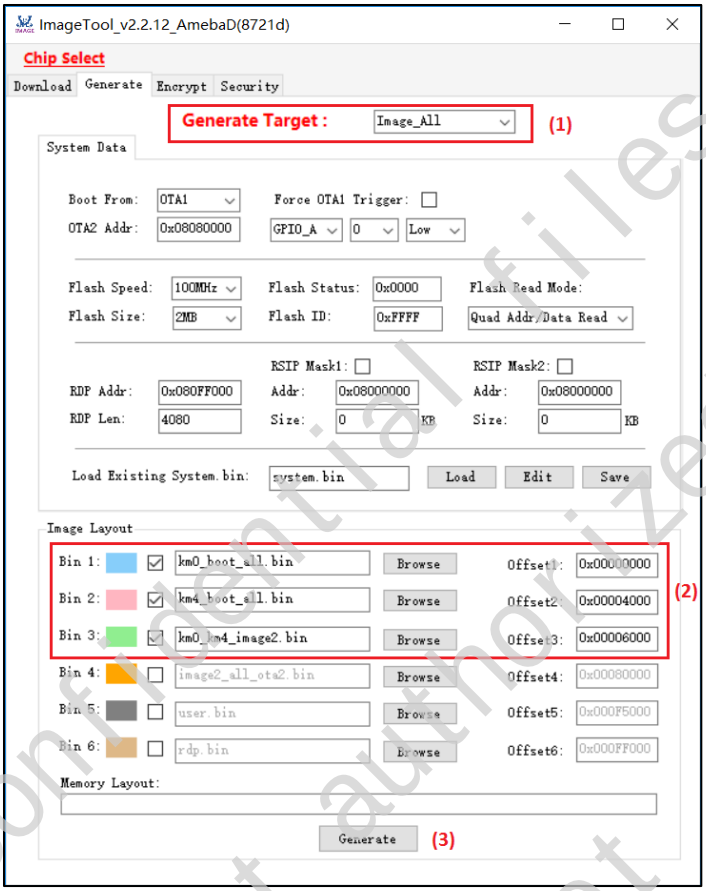


Fig 8-7 Image_All.bin generation

8.4.2 Generate Cloud OTA Image

Steps to generate a Cloud OTA image are as following:

(1) Select **OTA_All** as Generate Target type (in red).

(2) Input Image Version, the default value is 0xFFFFFFFF.

(3) Click Browse button to select images. The address can be ignored. The Memory Layout bar will show the relative positions of the two images. If they overlap, the overlapped area is in red color for warning.

(4) Click **Generate** button to specify output file name and path. After the operation is done, the cloud image (**OTA_All.bin)** is generated.

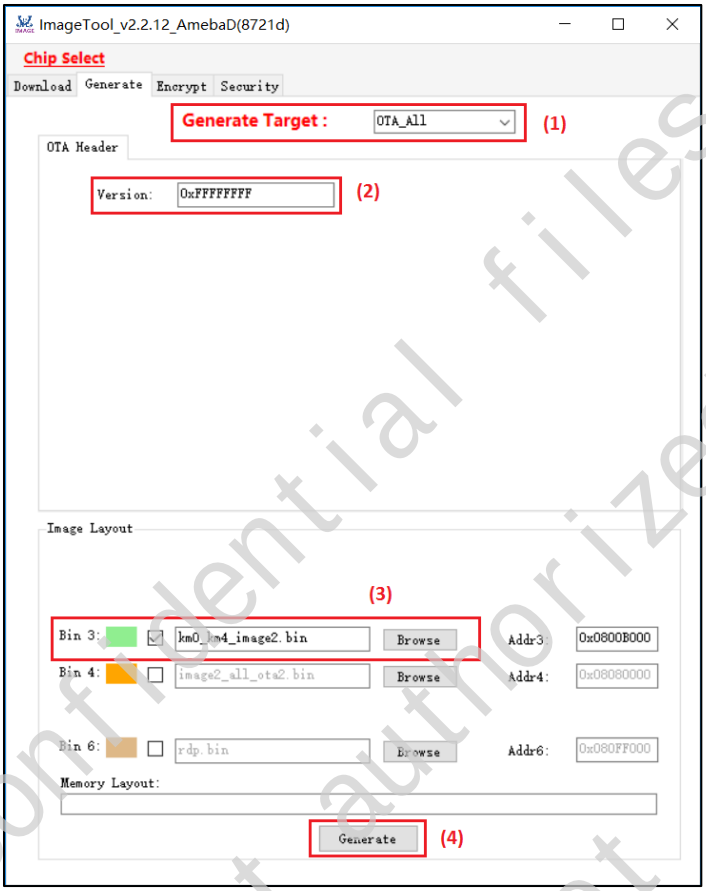


Fig 8-8 OTA_All generation

8.5 Encrypt

The 'Encrypt' tabpage is for encrypting images which is used in firmware protection. Steps to encrypt images are as following:

(1) Select **Encryption Type.**

(2) Input **Key** (16 bytes)**.**

(3) Select Images that need to be encrypted and input corresponding address.

(4) Click **Encrypt** button. You will get the message of encryption from the log window. The encrypted images are named with "-en" after the original file, which located in the same directory.

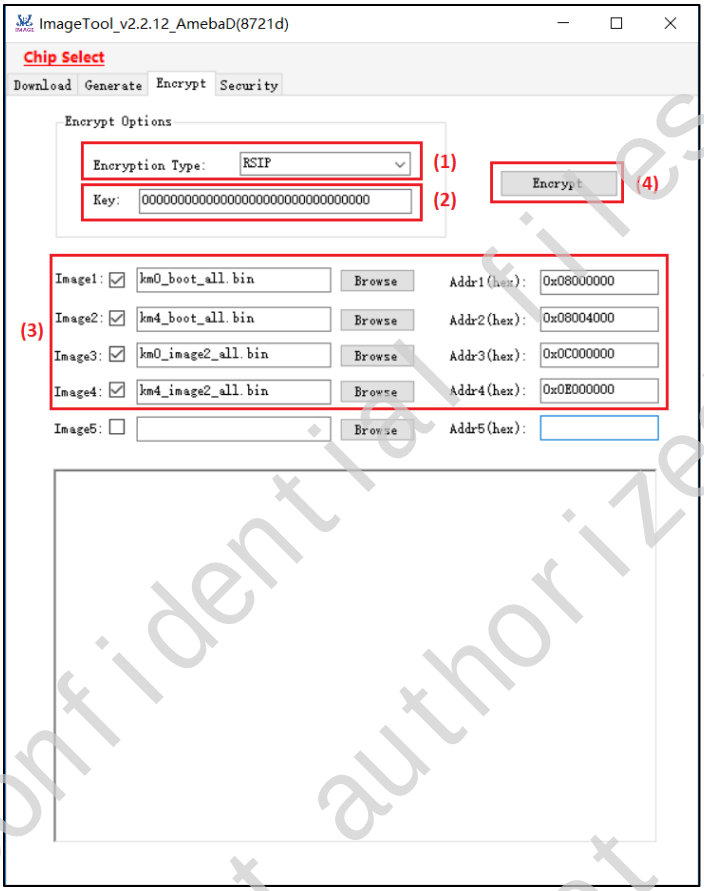


Fig 8-9 ImageTool 'Encrypt' tabpage setting

8.6 Security

The 'Security' tabpage is used to generate secure boot image. Steps to generate Secure Boot Image are as following:

(1) Input **Seed** (32 bytes), which is used to generate key pair.
(2) Click **Gen Keypair** button to generate Public Key and Private Key.
(3) Click **Browse** button to select the bootloader image, and input the flash address of this image.
(4) Click **Gen Signature** button to calculate signature for the image.
(5) Click **Save Image** button to generate new image which contains Secure Boot information and signature.

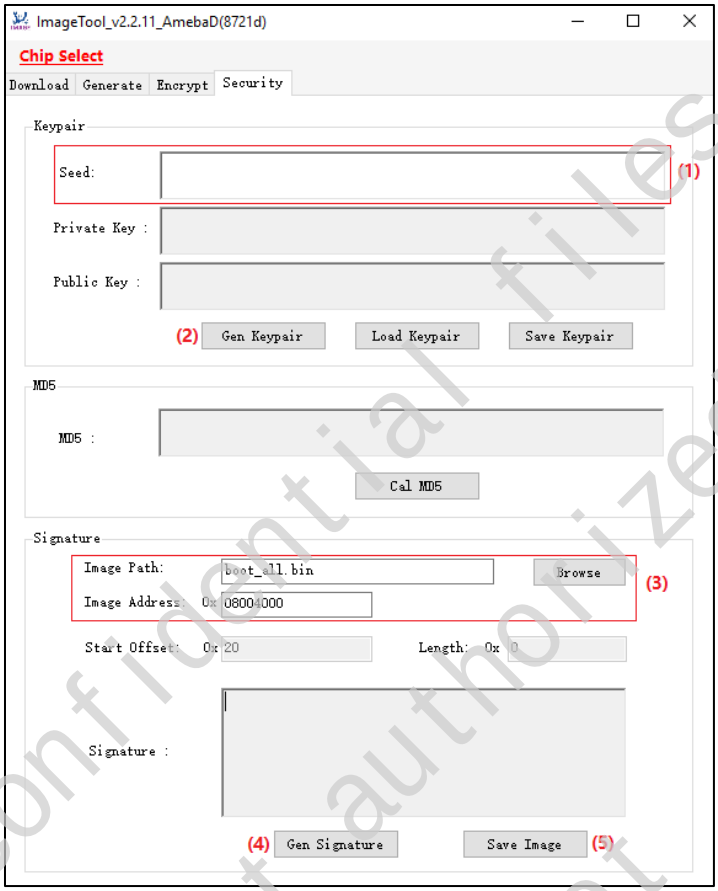


Fig 8-10 ImageTool 'Security' tabpage setting

9 Memory Layout

9.1 Flash Program Layout

The default flash layout used in SDK is illustrated in Fig 9-1 and Table 9-1. Because bootloader will be called by ROM code, the address of bootloader cannot be changed. Other addresses except bootloader can all be changed by users, but before changing these addresses, users must make sure that they are familiar with Ameba-D system and SDK.

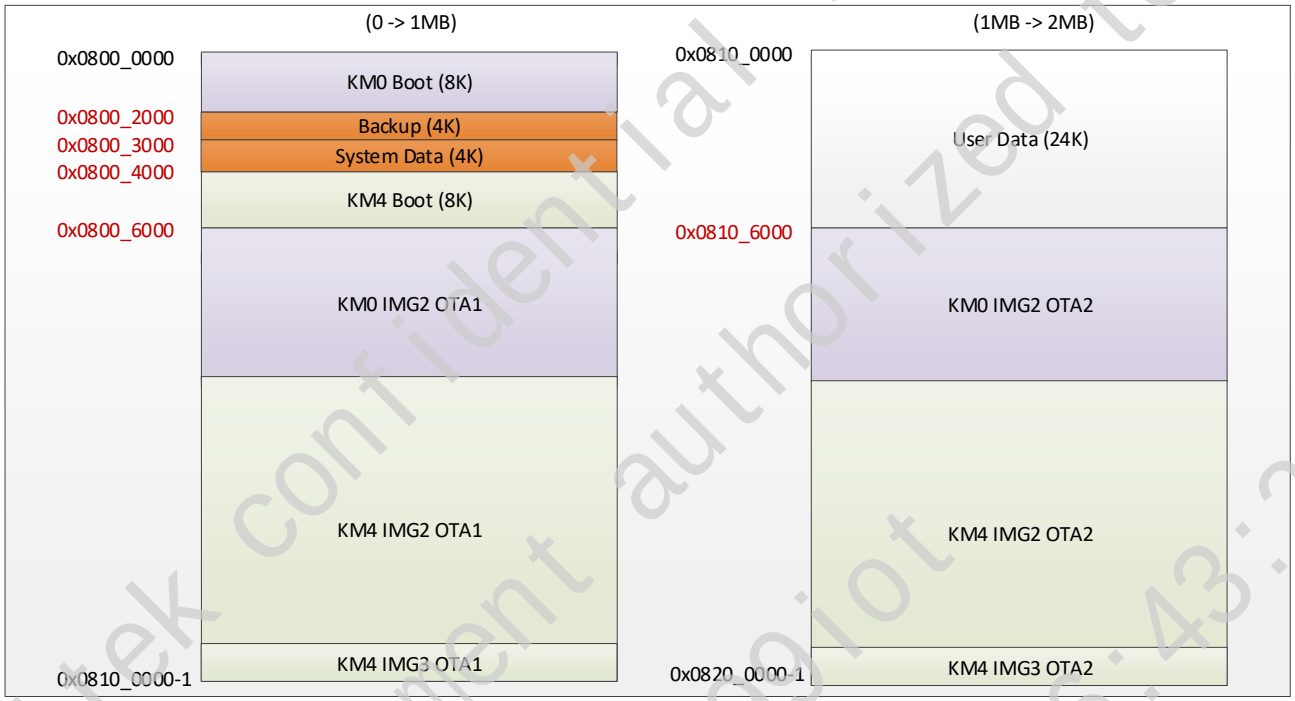


Fig 9-1 1M/2M flash program layout

Table 9-1 1M/2M flash program layout

Items	Start Address	Limited Address	Size	Description	Mandatory
KM0 Boot	0x0800_0000	0x0800_2000-1	8K	KM0 bootloader code/data	Yes
Backup	0x0800_2000	0x0800_3000-1	4K	Backup flash area, reserved for system use. When program system data, this area will be used as backup area.	Yes
System Data	0x0800_3000	0x0800_4000-1	4K	System Data, user should program this area carefully.	Yes
KM4 Boot	0x0800_4000	0x0800_6000-1	8K	KM4 bootloader code/data	Yes
KM0 IMG2 OTA1	0x0800_6000	0x0810_0000-1	1000K	KM0 image2 OTA1 code/data	No
KM4 IMG2 OTA1				KM4 image2 OTA1 code/data	No
KM4 IMG3 OTA1				RDP image OTA1 code/data	No
User Data	0x0810_0000	0x0810_2000-1	8K	Reserved for user data	No
	0x0810_2000	0x0810_5000-1	12K	BT FTL physical map	No
	0x0810_5000	0x0810_6000-1	4K	WIFI Fast Connect profile	No
KM0 IMG2 OTA2	0x0810_6000	0x0820_0000-1	1000K	KM0 image2 OTA2 code/data	No
KM4 IMG2 OTA2				KM4 image2 OTA2 code/data	No
KM4 IMG3 OTA2				RDP image OTA2 code/data	No

There is an image header for every image; it helps the bootloader to load code or data to the correct destination address, or it's used to realize ECC secure boot.

9.1.1 Image Header

Normal bootloader doesn't include secure boot signature. It's just used for loading code or data to the correct destination, as shown in see Fig 9-2 and Table 9-2.

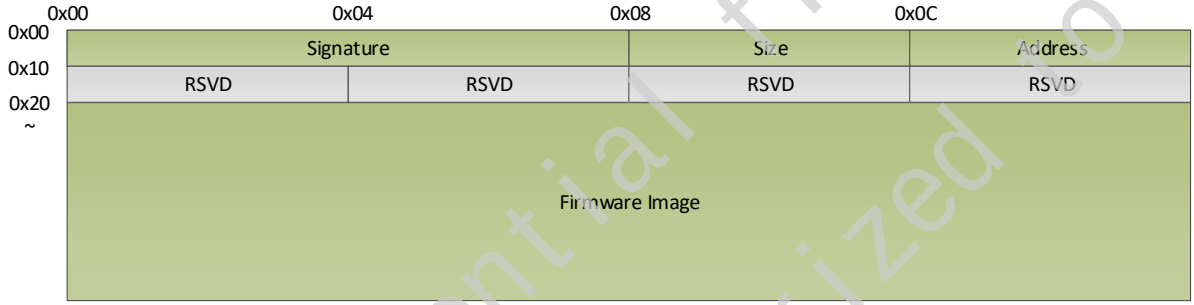


Fig 9-2 Normal image header

Table 9-2 Image header without secure boot

Items	Comment
Signature (8 bytes)	Bootloader Signature for flash calibration
Size (4 bytes)	The size of Firmware Image
Address (4 bytes)	Code executes start address after boot

Image header with secure includes secure boot signature. It is used to realize ECC secure boot. For more information, refer to Image Header with Secure Boot.

9.1.2 System Data (4K)

The System data layout is illustrated in Fig 9-3 and Table 9-3.

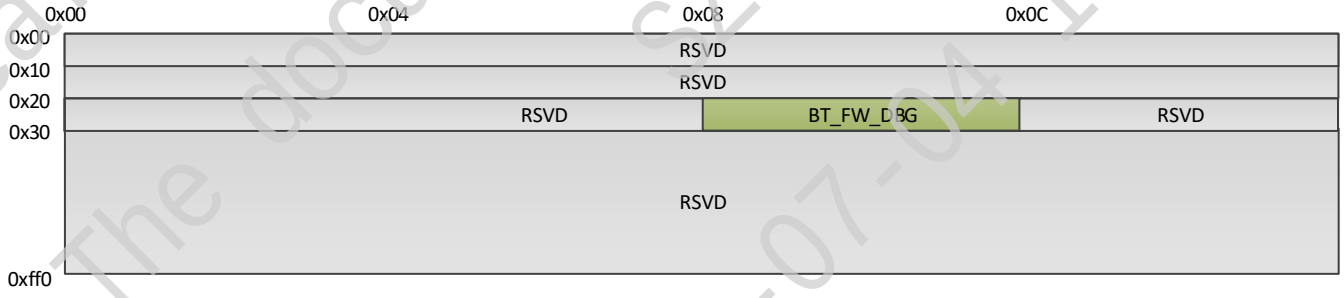


Fig 9-3 System data layout

Table 9-3 System Data Layout

Items	Default	Description
BT_FW_DBG	0xFFFFFFFF	Reserved for FW debug.

9.1.3 User Data

There are totally 24KB for user data. Parts of this region are defined by default SDK.

Table 9-4 User data layout

Items	Range	Description
Reserved	0x0810_0000~0x0810_2000-1	Reserved for user data
BT FTL	0x0810_2000~0x0810_5000-1	Bluetooth FTL physical map. The start address is defined by **ftl_phy_page_start_addr** variable (**rtl8721dhp_intfcfg.c**). If Bluetooth is not enabled in your system, this can be used by users.
Wi-Fi Fast Connect Profile	0x0810_5000~0x0810_6000-1	Used to store Wi-Fi fast connect profile. Defined by **FAST_RECONNECT_DATA** macro (**platform_opts.h**). If Wi-Fi fast connect function is not enabled in your system, this area can be used by users.

Warning: If flash memory layout is modified and the 0x0800_2000~0x0800_5FFF is covered, it is important to modify the FTL physical map or Wi-Fi fast connect area address.

9.1.4 4M Flash Usage Example

For 4M flash, an example of flash layout is illustrated in Fig 9-4. In this example, 2024K flash size can be used for KM0 & KM2 image, and you can modify the layout according to your actual conditions. Because bootloader will be called by ROM code, the address of bootloader cannot be changed. Other addresses except bootloader can all be changed by user, but before you change these addresses, make sure that you are familiar with Ameba-D system and SDK.

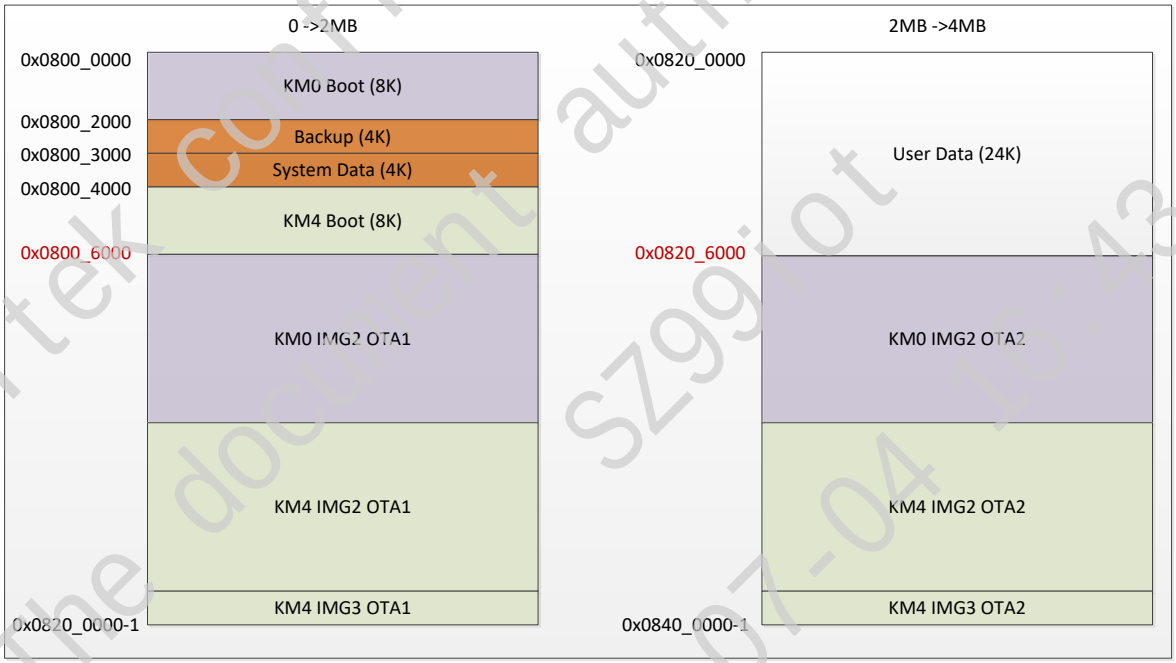


Fig 9-4 4M flash program layout example

Table 9-5 4M flash program layout example

Items	Start Address	Limited Address	Size	Description	Mandatory
KM0 Boot	0x0800_0000	0x0800_2000-1	8K	KM0 bootloader code/data	Yes
Backup	0x0800_2000	0x0800_3000-1	4K	Backup flash area, reserved for system use. When program system data, this area will be used as backup area.	Yes
System Data	0x0800_3000	0x0800_4000-1	4K	System Data, user should program this area carefully.	Yes
KM4 Boot	0x0800_4000	0x0800_6000-1	8K	KM4 bootloader code/data	Yes

KM0 IMG2 OTA1	0x0800_6000	0x0820_0000-1	2024K	KM0 image2 OTA1 code/data	No
KM4 IMG2 OTA1				KM4 image2 OTA1 code/data	No
KM4 IMG3 OTA1				RDP image OTA1 code/data	No
User Data	0x0820_0000	0x0820_2000-1	8K	Reserved for user data	No
	0x0820_2000	0x0820_5000-1	12K	BT FTL physical map	No
	0x0820_5000	0x0820_6000-1	4K	Wi-Fi Fast Connect profile	No
KM0 IMG2 OTA2	0x0820_6000	0x0840_0000-1	2024K	KM0 image2 OTA2 code/data	No
KM4 IMG2 OTA2				KM4 image2 OTA2 code/data	No
KM4 IMG3 OTA2				RDP image OTA2 code/data	No

For 4M flash, some configurations need to be modified in default SDK.

- OTA start address

The default OTA start address in SDK is set for 2M flash, thus OTA address in rtl8721d_bootcfg.h and rtl8721d_ota.h needs to be modified when using 4M flash.

(1) Modify **OTA_Region** in rtl8721d_bootcfg.h, change OTA2 region start address according to your actual flash layout. When using 4M flash program layout example as shown in Fig 9-4, OTA region should be modified as below:

```
/*
* @brif   OTA start address. Because KM0 & KM4 IMG2 are combined, users only need to set the start address
*    of KM0 IMG2.
*/
BOOT_RAM_DATA_SECTION
u32 OTA_Region[2] = {
    0x08006000,      /* OTA1 region start address */
    0x08206000,      /* OTA2 region start address */
};
```

(2) Modify OTA address definition in rtl8721d_ota.h, redefine **LS_IMG2_OTA2_ADDR** macro according to your actual flash layout. When using 4M flash program layout example as shown in Fig 9-4, OTA address should be modified as below:

```
#define LS_IMG2_OTA1_ADDR  0x08006000          /* KM0 OTA1 start address*/
#define LS_IMG2_OTA2_ADDR  0x08206000          /* KM0 OTA2 start address*/
```

- User data flash address

In default SDK, 24KB user data is defined in flash address 0x0800_2000~0x0800_5FFF, which is now used for BT FTL and Wi-Fi fast connect profile. For 4M flash, if Bluetooth and Wi-Fi fast connect function are enabled in your system, and flash address 0x0800_2000~0x0800_5FFF is covered, you need to modify BT FTL address and Wi-Fi fast connect profile address according to your actual flash layout.

(1) For BT FTL start address, re-define **ftl_phy_page_start_addr** variable in rtl8721dhp_intfcfg.c, note that SPI FLASH address base 0x0800_0000 is subtracted when calculating ftl_phy_page_start_addr variable, and ftl_phy_page will consume 3*4=12KB flash starting from **ftl_phy_page_start_addr**. When using 4M flash program layout example as shown in Fig 9-4, BT FTL start address should be modified as below:

```
const u32 ftl_phy_page_start_addr = 0x00202000;
```

(2) For Wi-Fi fast connect profile address, redefine **FAST_RECONNECT_DATA** macro in platform_opts.h, and SPI FLASH address base 0x0800_0000 is subtracted when calculating FAST_RECONNECT_DATA macro. Note that Wi-Fi fast connect profile address should not recover the 12KB flash starting from **ftl_phy_page_start_addr**, as this region is reserved for BT FTL. When using 4M flash program layout example as shown in Fig 9-4, Wi-Fi fast connect profile address should be set to 0x202000+0x3000=0x205000.

```
#define FAST_RECONNECT_DATA    0x205000
```

After the above modification, rebuild KM0 project and KM4 project, then download km0_boot_all.bin, km4_boot_all.bin and km0_km4_image2.bin.

9.2 SRAM Layout

9.2.1 KM4 SRAM Layout

The start address of KM4 SRAM is 0x1000_0000. The size is 512KB. The memory layout is illustrated in Fig 9-5 and Table 9-6.

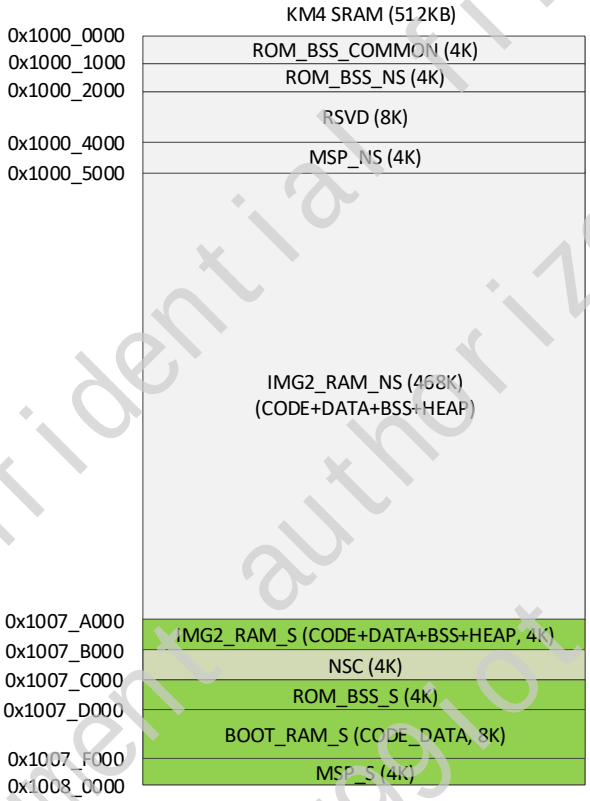


Fig 9-5 KM4 RAM program layout

Table 9-6 KM4 RAM program layout

Items	Start Address	End Address	Size	Description	Mandatory
ROM_BSS_COMMON	0x1000_0000	0x1000_1000-1	4K	ROM Secure and Non-secure common .bss	Yes
ROM_BSS_NS	0x1000_1000	0x1000_2000-1	4K	ROM Non-secure common .bss	Yes
RSVD	0x1000_2000	0x1000_4000-1	8K	Reserved for Non-secure MSP & Non-secure ROM .bss	No
MSP_NS	0x1000_4000	0x1000_5000-1	4K	Non-secure Main Stack Pointer	Yes
IMG2_RAM_NS	0x1000_5000	0x1007_A000-1	468K	Image2 Non-secure RAM includes CODE, DATA, BSS and HEAP	No
IMG2_RAM_S	0x1007_A000	0x1007_B000-1	4K	Image2 Secure RAM includes CODE, DATA, BSS and HEAP	No
NSC	0x1007_B000	0x1007_C000-1	4K	Images Secure function call entries	No
ROM_BSS_S	0x1007_C000	0x1007_D000-1	4K	ROM Secure only .bss	Yes
BOOT_RAM_S	0x1007_D000	0x1007_F000-1	8K	KM4 Secure bootloader, includes CODE and DATA	Yes
MSP_S	0x1007_F000	0x1008_0000-1	4K	Secure Main Stack Pointer	Yes

9.2.2 KM0 SRAM Layout

The start address of KM0 SRAM is 0x0008_0000. The size is 64KB. The memory layout is illustrated in Fig 9-6 and Table 9-7.

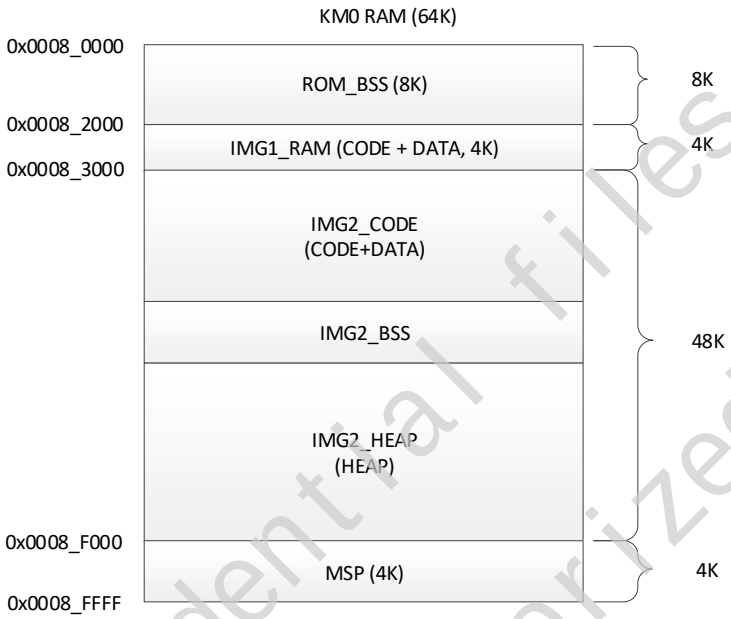


Fig 9-6 KM0 RAM program layout

Table 9-7 KM0 RAM program layout

Items	Start Address	End Address	Size	Description	Mandatory
ROM_BSS	0x0008_0000	0x0008_2000-1	8K	ROM .bss	Yes
IMG1_RAM	0x0008_2000	0x0008_3000-1	4K	KM0 bootloader SRAM, including CODE and DATA	Yes
IMG2_RAM	0x0008_3000	0x0008_F000-1	48K	KM0 image2 SRAM, including CODE, DATA, BSS and HEAP	No
MSP	0x0008_F000	0x0009_0000-1	4K	Main Stack Pointer	Yes

9.3 Retention SRAM

Retention SRAM would not be power off even when system enters into deepsleep mode. As a result, it is designed to retain data or code which would be lost in SRAM for some cases, such as deepsleep.

9.3.1 Retention SRAM Layout

The start address of Retention SRAM is 0x000C_0000. The size is 1KB. The memory layout is illustrated in Table 9-8.

Table 9-8 Retention SRAM layout

Items	Start Address	End Address	Size	Description	Mandatory
System Area	0x000C_0000	0x000C_0080-1	128B	Used by Realtek. The size is indicated by the second dword (0x000C_0004).	Yes
User Area	0x000C_0080	0x000C_0130-1	176B	Refer to RRAM_TypeDef, and RRAM_USER_RSVD [user_size] is reserved for customer use.	No
Wi-Fi Area	0x000C_0130	0x000C_0400-1	720B	Realtek Wi-Fi FW use	Yes

9.3.2 User Area

RRAM_USER_RSVD is reserved for customer use. The size for user area is 124B now, it may be changed latter.

```
/******************************************************************************//**
 * @defgroup AMEBAD_RRAM
 * @{
 * @brief AMEBAD_RRAM Declaration
 * @ the Max space for RRAM_TypeDef is 0xB0, user can alloc space from RRAM_USER_RSVD
 *****************************************************************************/
typedef struct {
    uint8_t RRAM_WIFI_STATUS; /* RETENTION_RAM_SYS_OFFSET 0x80 */
    uint8_t RRAM_WIFI_P_SECURITY;
    uint8_t RRAM_WIFI_G_SECURITY;
    uint8_t RRAM_RSVD1;

    uint32_t RRAM_NET_IP;
    uint32_t RRAM_NET_GW;
    uint32_t RRAM_NET_GW_MASK;

    uint32_t FLASH_ID2;             /*offset 0x90*/
    uint32_t FLASH_cur_cmd;
    uint32_t FALSH_quad_valid_cmd;
    uint32_t FALSH_dual_valid_cmd;
    uint32_t FLASH_QuadEn_bit;      /*offset 0xA0*/
    uint32_t FLASH_StructInit;

    uint8_t FLASH_phase_shift_idx;
    uint8_t FLASH_rd_sample_phase_cal;
    uint8_t FLASH_class;
    uint8_t FLASH_cur_bitmode;

    uint8_t FLASH_ClockDiv;
    uint8_t FLASH_ReadMode;
    uint8_t FLASH_pseudo_prm_en;
    uint8_t FLASH_addr_phase_len;

    uint8_t FLASH_cmd_wr_status2;   /*offset 0xB0*/
    uint8_t FLASH_rd_dummy_cyle0;
    uint8_t FLASH_rd_dummy_cyle1;
    uint8_t FLASH_rd_dummy_cyle2;

    uint8_t RRAM_USER_RSVD[124];          /*usr can alloc from this RSVD space*/

} RRAM_TypeDef;
/** @} */
```

9.4 OTA Layout

9.4.1 OTA Program Layout

The OTA program layout is shown in Fig 9-7.

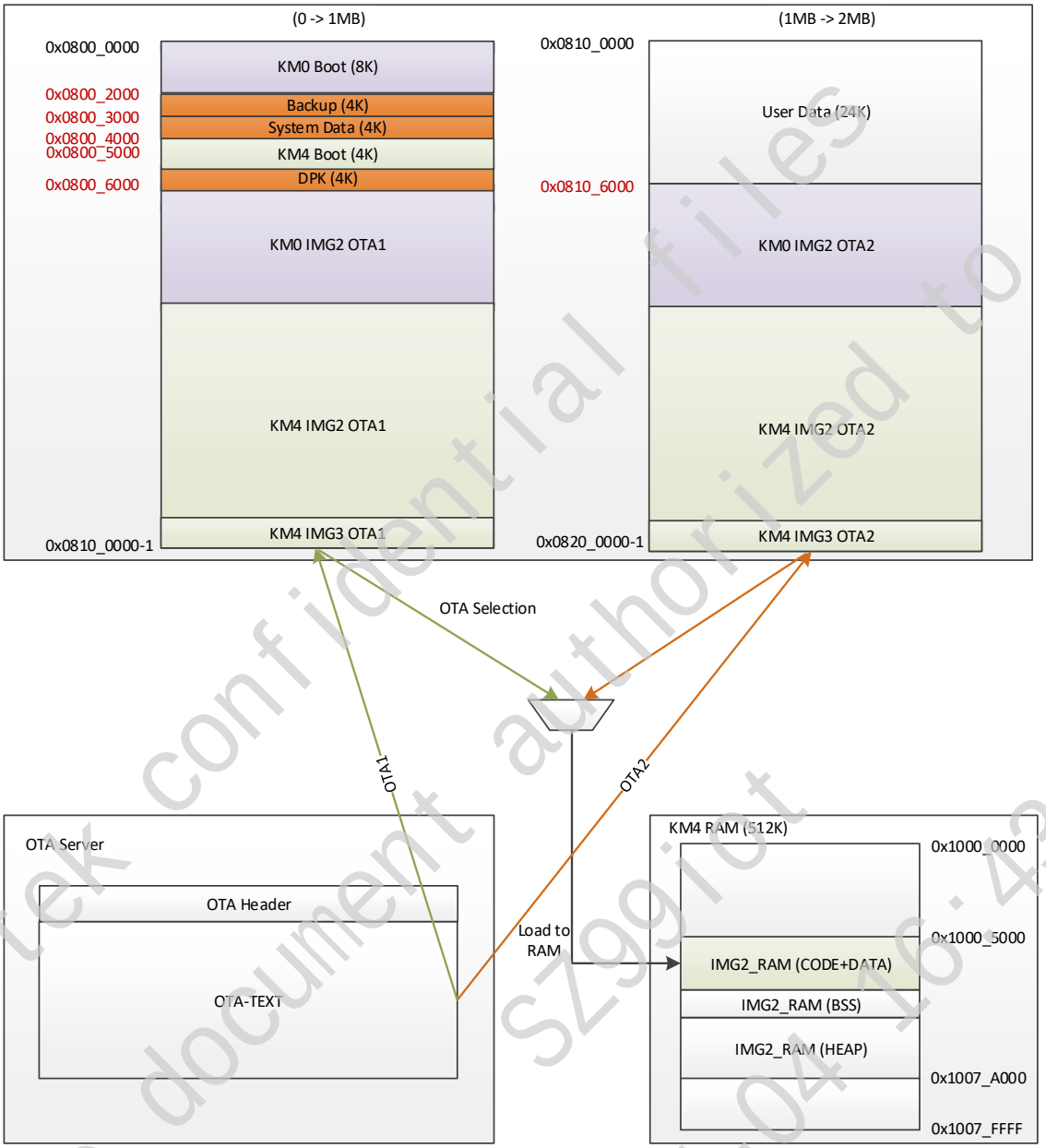


Fig 9-7 OTA program layout

9.4.2 OTA Header Layout

The OTA header layout is given in Fig 9-8 and Table 9-9.

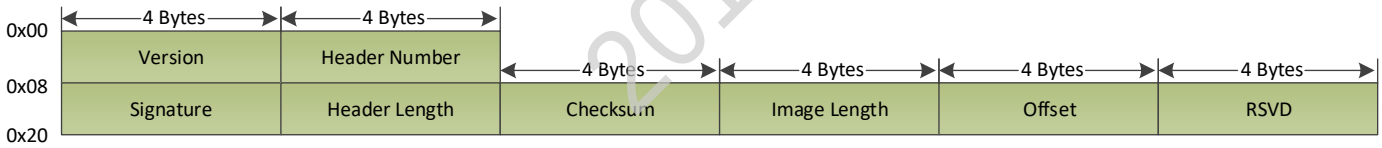


Fig 9-8 OTA header layout

Table 9-9 OTA header layout

Items	Address Offset	Size	Description
Version	0x00	4 bytes	The version of OTA Header, default 0xFFFFFFFF
Header Number	0x04	4 bytes	The number of OTA Header. For Ameba-D, this value is 0x01
Signature	0x08	4 bytes	OTA Signature is string. For Ameba-D, this value is “OTA”
Header Length	0x0C	4 bytes	The length of OTA header. For Ameba-D, this value is 0x18
Checksum	0x10	4 bytes	The checksum of OTA image
Image Length	0x14	4 bytes	The size of OTA image
Offset	0x18	4 bytes	The start position of OTA image in current image
RSVD	0x1C	4 bytes	Reserved

9.5 TrustZone Memory Layout

All addresses are Secure when boot, you can set some area to Non-secure using SAU & IDAU. Fig 9-9 is the default secure setting of Ameba-D SDK.

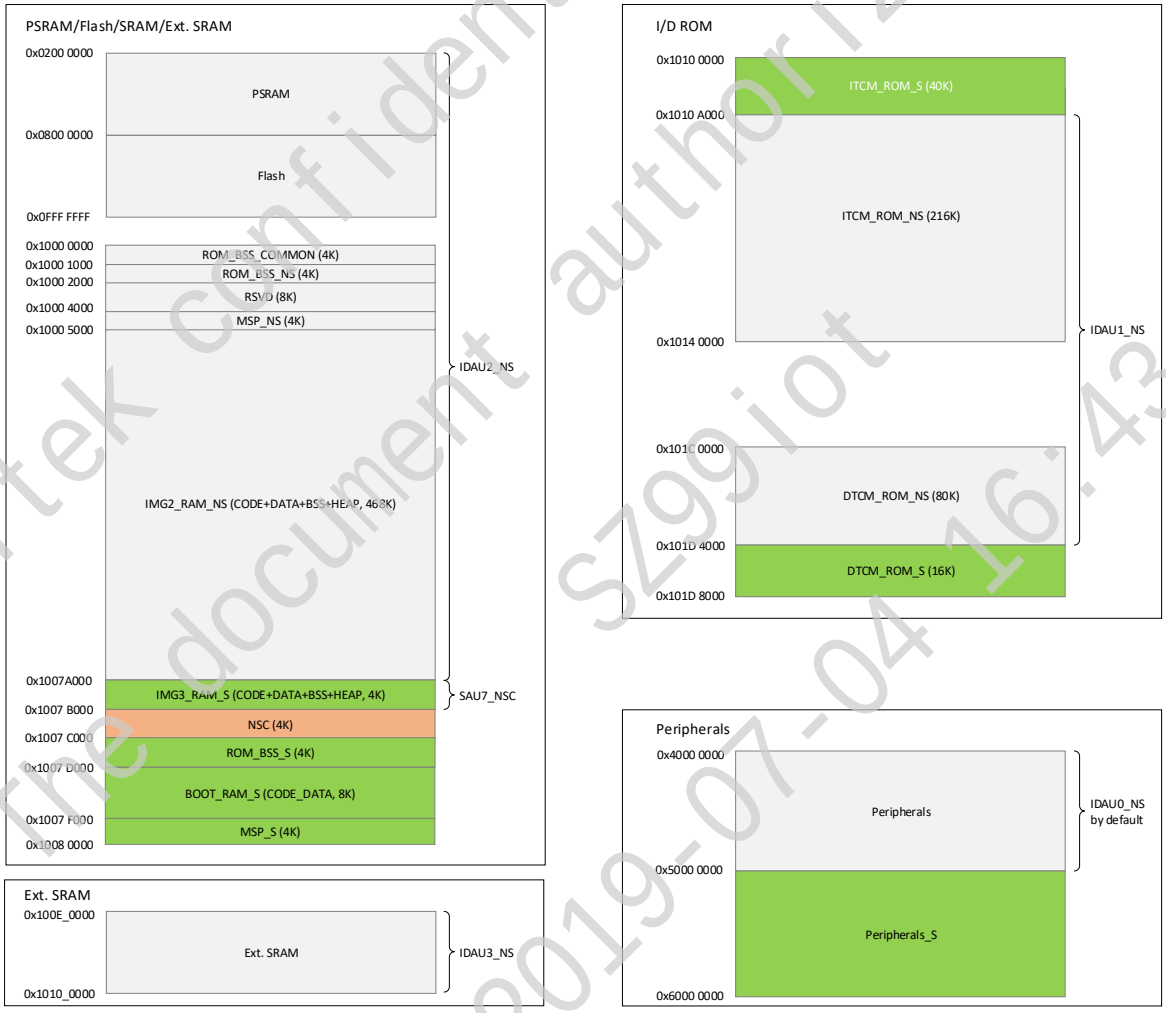


Fig 9-9 TrustZone layout

If you don’t want to use TrustZone, you can set IMG3_RAM_S and NSC to Non-secure, other secure area used by ROM code are mandatory.

10 Boot Process

10.1 Features

- On-chip boot ROM
- Contains the bootloader with In-System Programming (ISP) facility
- Secure boot process use Error Correcting Code (ECC)
- Suspend resume process

10.2 Boot Address

After reset, CPU will boot from vector table start address, these address is fixed by hardware. KM0 and KM4 boot from different addresses as Table 10-1 lists.

Table 10-1 Boot ROM address

CPU	Address	Type
KM4	0x1010_0000	KM4 ITCM ROM
KM0	0x0000_0000	KM0 ITCM ROM

10.3 Pin Description

The parts support ISP via LOGUART (PA7 & PA8). The ISP mode, given in Table 10-2, is determined by the state of the PA7 pin at boot time.

Table 10-2 ISP mode

Boot Mode	PA7 (UART_DOWNLOAD)	Description
No ISP	HIGH	ISP bypassed. Part attempts to boot from flash.
ISP	LOW	Part enters ISP via LOGUART.

10.4 Boot Flow

The boot flow of Ameba-D is shown in Fig 10-1. After a power-up or hardware reset, hardware will boot KM0 at clock 26MHz or 20MHz based on XTAL clock set in eFuse. The boot process is handled by the on-chip boot ROM (0x0000_0000) and is always executed by the KM0 core. After the KM0 bootloader code, the KM0 will set up the environment for the KM4.

(1) Power on PLL
(2) Flash calibration @100MHz
(3) Power on KM4
(4) After that the KM4 can be taken out of reset by the KM0.
(5) KM4 will boot from ROM code 0x1010_0000.

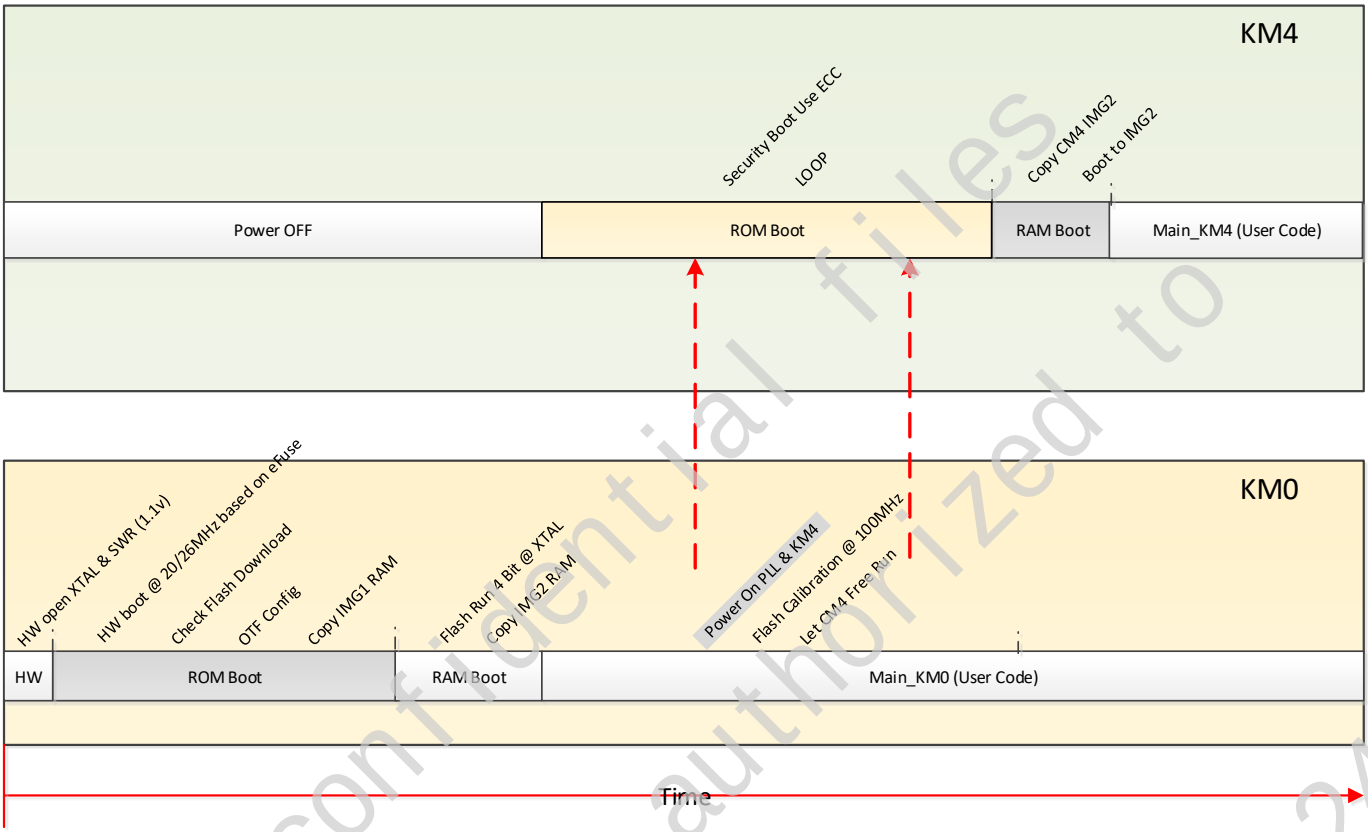


Fig 10-1 Boot flow

10.5 Boot Time

Boot time was tested when this document was written, the result may be changed as the changing of SDK.

Table 10-3 Boot Time

CPU	Normal BOOT (ms)	Deepsleep WAKEUP (ms)	Comment
KM0 BOOT	30	18	
KM4 BOOT	6+x	3+x	x=image2 size/flash rate If image2 size = 100KB, Flash rate = 80Mbps & 2bit mode: x=100*1024*8/160=5.1ms (100KB is SDK current image2 size)
WIFI ON	40	40	

10.6 General Description

The bootloader controls initial operation after reset or powered on and also provides the means to program the flash memory. This could be initial programming of a blank device, erasure and re-programming of a previously programmed device.

Assuming that power supply pins are at their nominal levels when the rising edge on RESET pin is generated, then boot pins are sampled and the decision whether to continue with user code or ISP handler is made. If the boot pins are sampled LOW, the external hardware request to start the ISP command handler is ignored. If there is no request for the ISP command handler execution, a search is made for a valid user program. If a valid user program is found, then the execution control is transferred to it. If a valid user program is not found, the dead loop is invoked.

10.6.1 KM0 ROM Boot

The KM0 ROM boot process is shown in Fig 10-2.

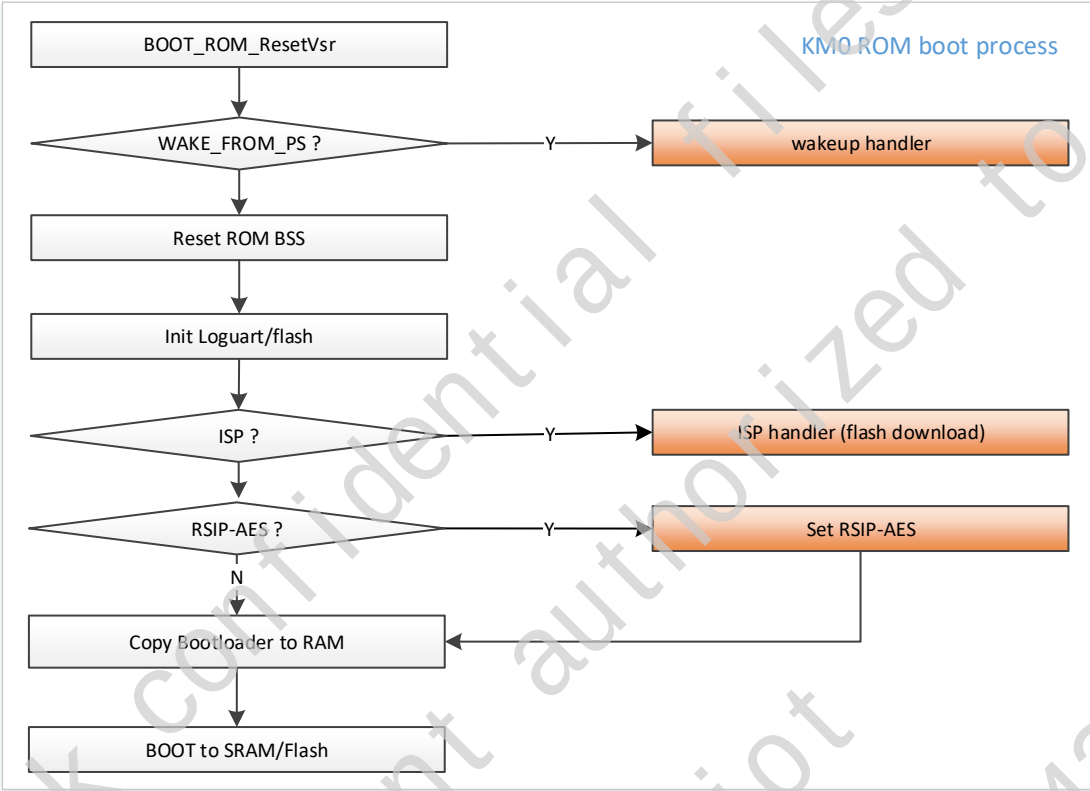


Fig 10-2 KM0 ROM boot process

10.6.2 KM4 ROM Boot

The KM4 ROM boot process is shown in Fig 10-3.

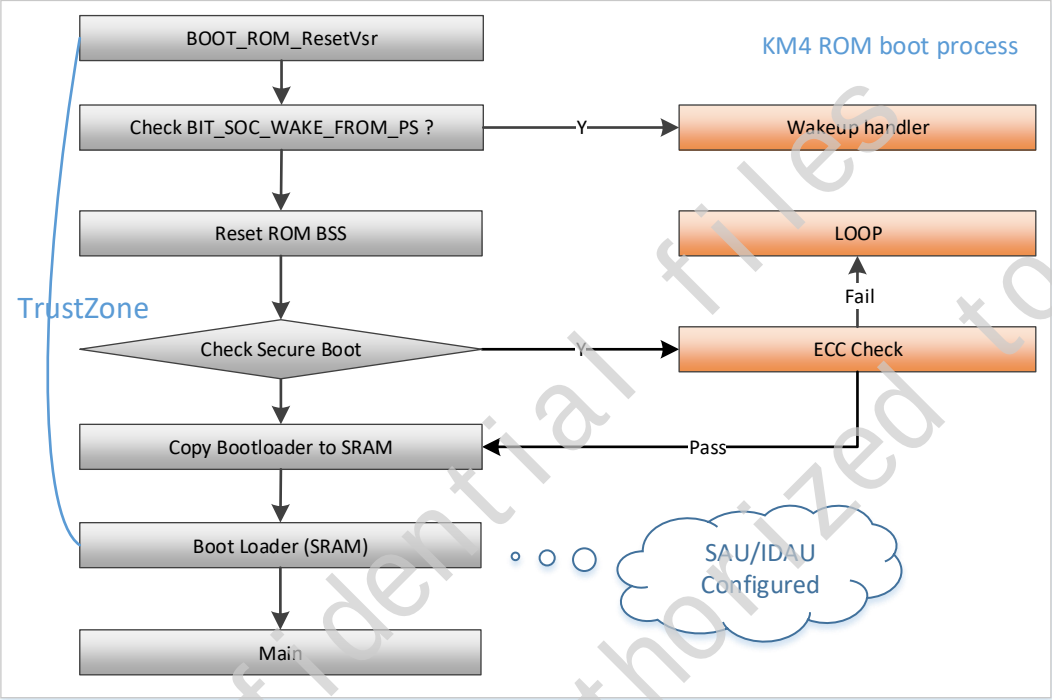


Fig 10-3 KM4 ROM boot process

10.7 Boot Reason APIs

Source code: **rtl8721d_backup_reg.c**

API	Introduction
< BOOT_Reason >	Get boot reason

Table 10-4 Boot reason definition

Bit	Comment
BIT_BOOT_BOD_RESET_HAPPEN	BOD Reset
BIT_BOOT_DSLP_RESET_HAPPEN	Wakeup form deep sleep
BIT_BOOT_KM4WDG_RESET_HAPPEN	KM4 watchdog reset
BIT_BOOT_KM4SYS_RESET_HAPPEN	KM4 system reset
BIT_BOOT_WDG_RESET_HAPPEN	KM0 watchdog reset
BIT_BOOT_SYS_RESET_HAPPEN	KM0 system reset

10.8 Security Boot

10.8.1 Diagram

Secure boot aims at firmware protection, which prevents attackers from modifying or replacing firmware maliciously. When the chip is power on, the ROM security boot executes to check the validation of image signature.

If the signature is valid, authentication will be successful, which means the firmware is safe and the subsequent operations can be continued. Otherwise, the SoC goes into infinite loop.

Users do not need to implement the security boot code themselves, which are already contained in the ROM code. Users only need to program the bootloader with signature information into Flash (0x0800_0000).

To generate Public Key and signature of bootloader, we provide Image Tool. With the Image Tool, users can calculate and append signature-related information conveniently, please reference application note for more information of realization.

The security boot diagram of Ameba-D is shown in Fig 10-4.

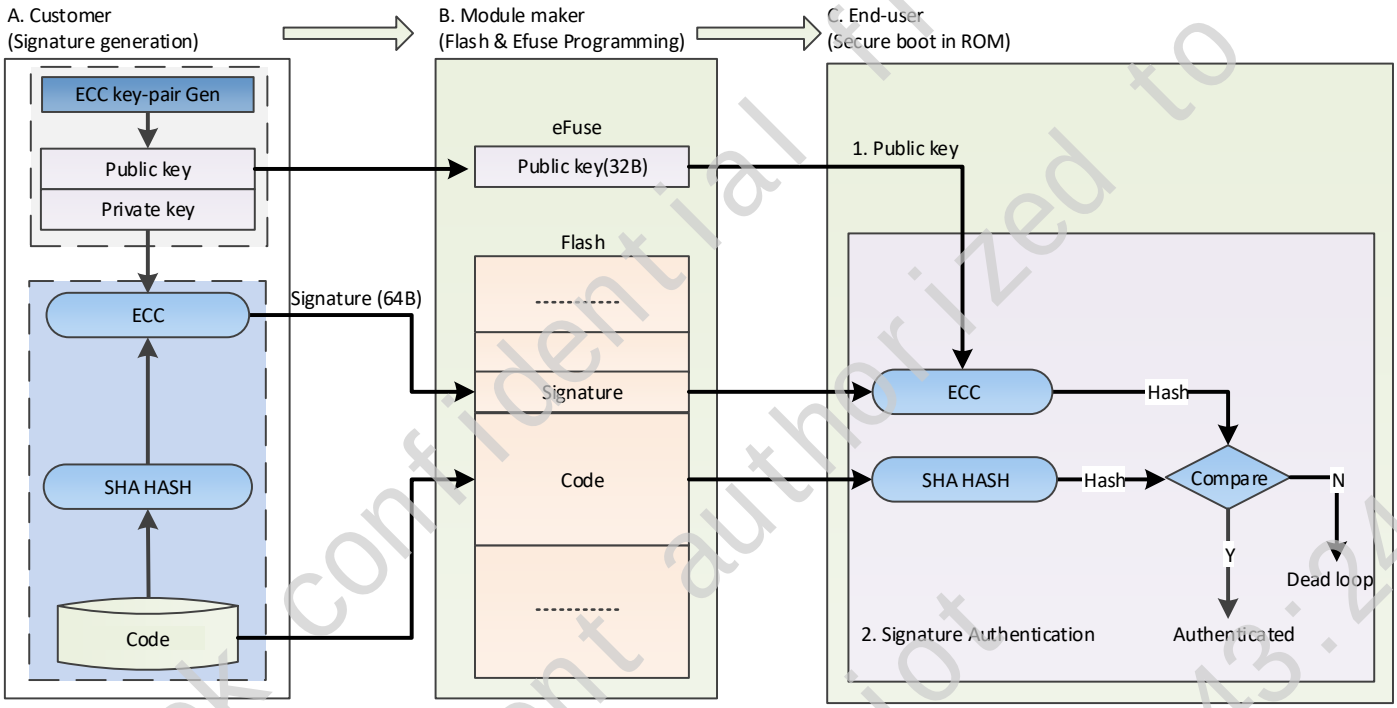


Fig 10-4 Ameba-D security boot diagram

10.8.2 Image Header with Secure Boot

Image header with secure includes secure boot signature. It is used to realize ECC secure boot, as shown in Fig 10-5 and Table 10-5.

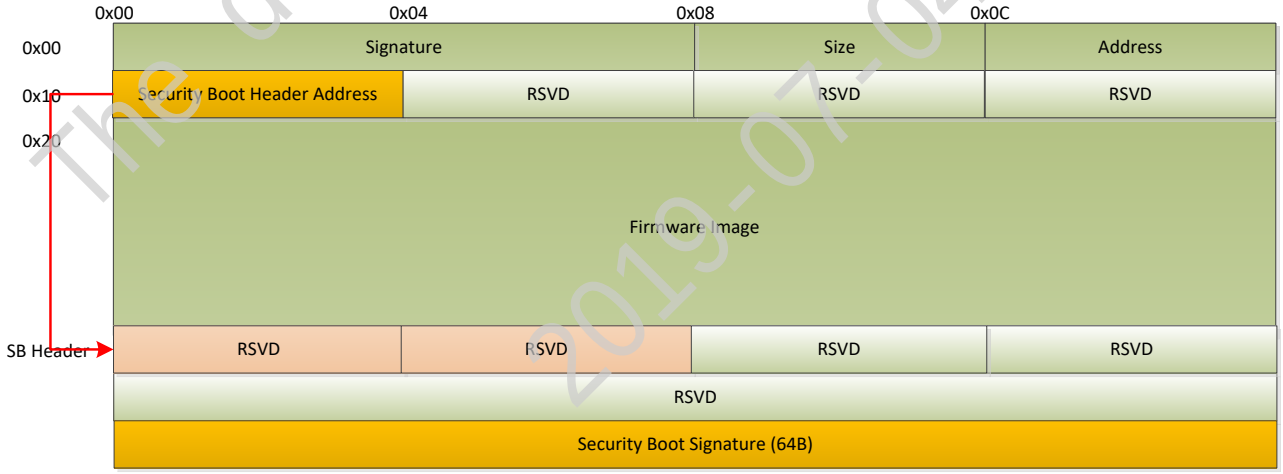


Fig 10-5 Security boot image header

Table 10-5 Image header with secure boot

Items	Comment
Signature (8 bytes)	Image1 Signature for Flash calibration: “96969999” Image2 Signature is string: “81958711”
Size (4 bytes)	The size of Firmware Image
Address (4 bytes)	Code executes start address after boot
Security Boot Header Address (4 bytes)	Indicate where the SB Header locates. Generally, the SB Header is appended to the end of the image
Security Boot Signature	The generated signature, which is to be verified when boot.

Note:

- Secure Boot should be rechecked after the SoC wakeup from deepsleep.
  Cause: We can’t detect whether the flash chip is swapped out by another flash with non-secure code or not during deep sleep.
- The start address of image data from which we start to calculate signature should be fixed.
  Cause: There are security risks when hacker attacks in the following way:
  (1) Copy the original image to another flash address
  (2) Overwrite the original image with non-secure image
  (3) Modify the start address in SB Header to the new flash address in step (1)
  (4) Restart, then the non-secure image would execute
- The length of image data that involved in signature calculation should be nonconfigurable.
  Cause: The length should be appropriate. It can’t be too small, otherwise the image can’t be protected well. And the signature is independent of image if the length is 0, which enables hacker to replace the firmware easily. For Ameba-D, the length is set to the image size, and the header length is not included.

10.8.3 Secure Boot Image Generation

ImageTool is provided to generate secure boot image. The Security function of ImageTool is used to append Secure Boot Header to normal image.

As Fig 10-6 shows, steps to generate Secure Boot Image are as follows:
(1) Input **Seed** (32 bytes), which is used to generate key pair.
(2) Click **Gen Keypair** button to generate Public Key and Private Key.
(3) Click **Browse** button to select the bootloader image, and input the flash address of this image.
(4) Click **Gen Signature** button to calculate signature for the image.
(5) Click **Save Image** button to generate new image which contains Secure Boot information and signature.

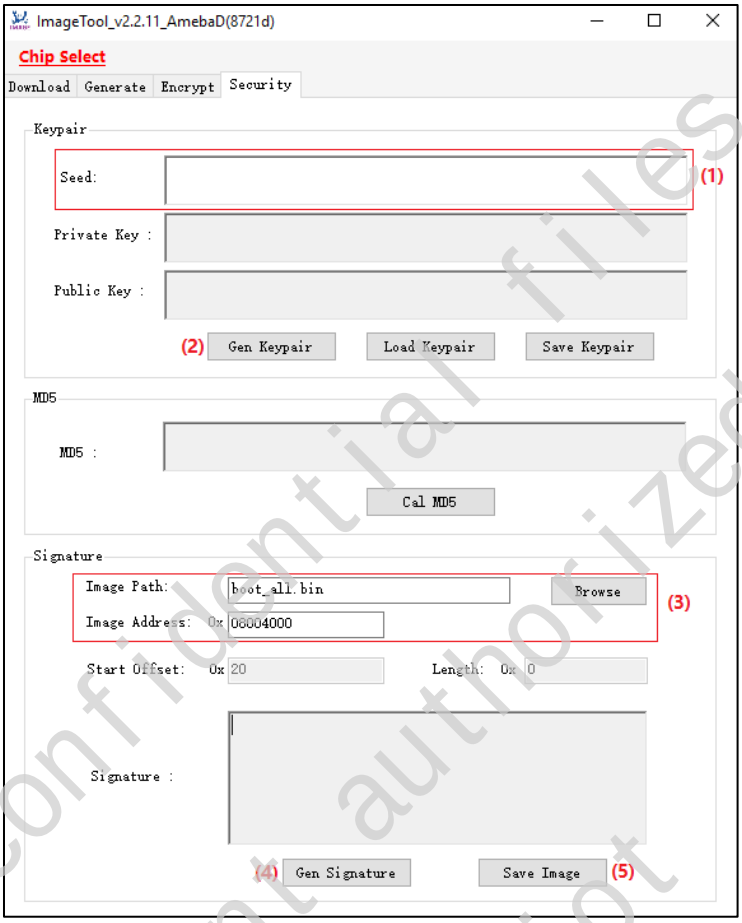


Fig 10-6 ImageTool 'Security' function

11 OTA Firmware Update

11.1 Introduction

Over-the-air (OTA) programming provides a methodology of updating device firmware remotely via TCP/IP network.

RTL8721d provides solutions to implement OTA firmware upgrade from local server or cloud. Next, we will introduce the design principles and usage of OTA from local server. It has well transportability to porting to OTA applications from cloud.

11.2 RSIP-MMU

The flash MMU diagram is shown in Fig 11-1.

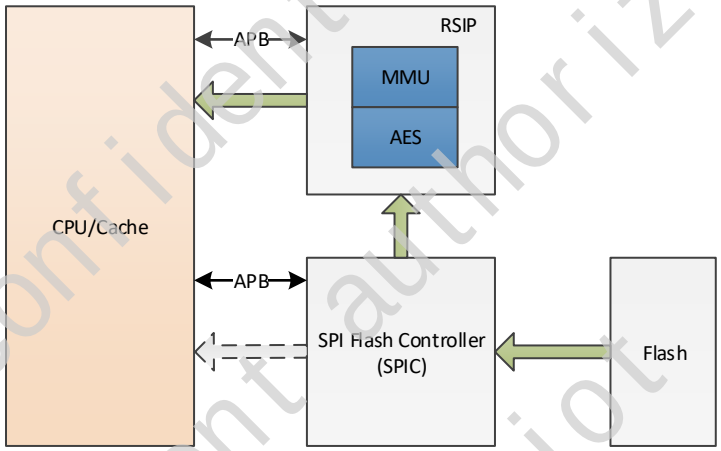


Fig 11-1 Flash MMU

The RSIP-MMU can perform virtual-to-physical memory address translation. This can be used to map an external flash memory to a certain virtual memory area. For example, If you want to access physical page 4 ~ 7 through virtual page 0 ~ 3, you can use a MMU entry to map virtual page 0 ~ 3 to physical page 4 ~ 7 like Fig 11-2.

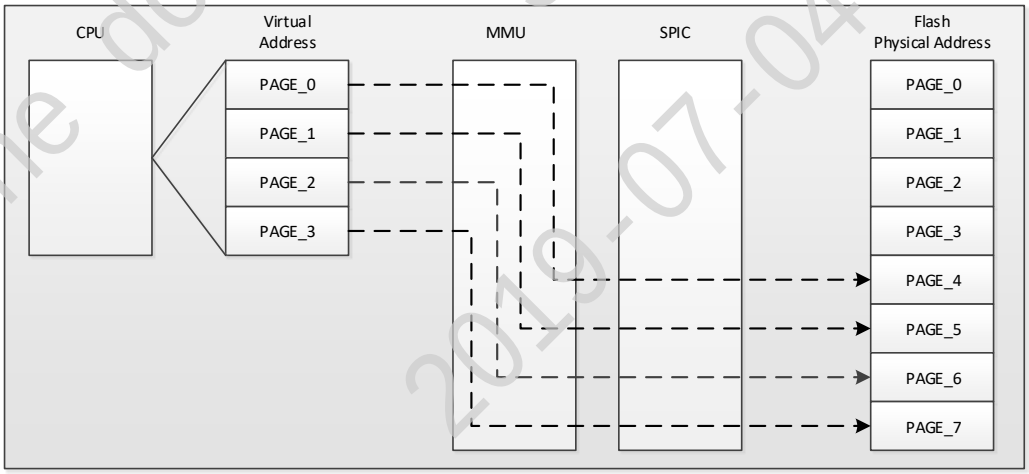


Fig 11-2 MMU virtual address to physical address

Ameba-D provides 8 MMU Entries. If virtual address is not included in the MMU entry, use virtual address as physical address. If virtual address is included in the MMU entry, physical address should be VAddress +/- MMU_ENTRYx_OFFSET.

MMU is implemented to facilitate OTA update procedure as Fig 11-3 shows. Fig 11-3 is an example for 2M flash OTA, we need 2 MMU entries in this example.

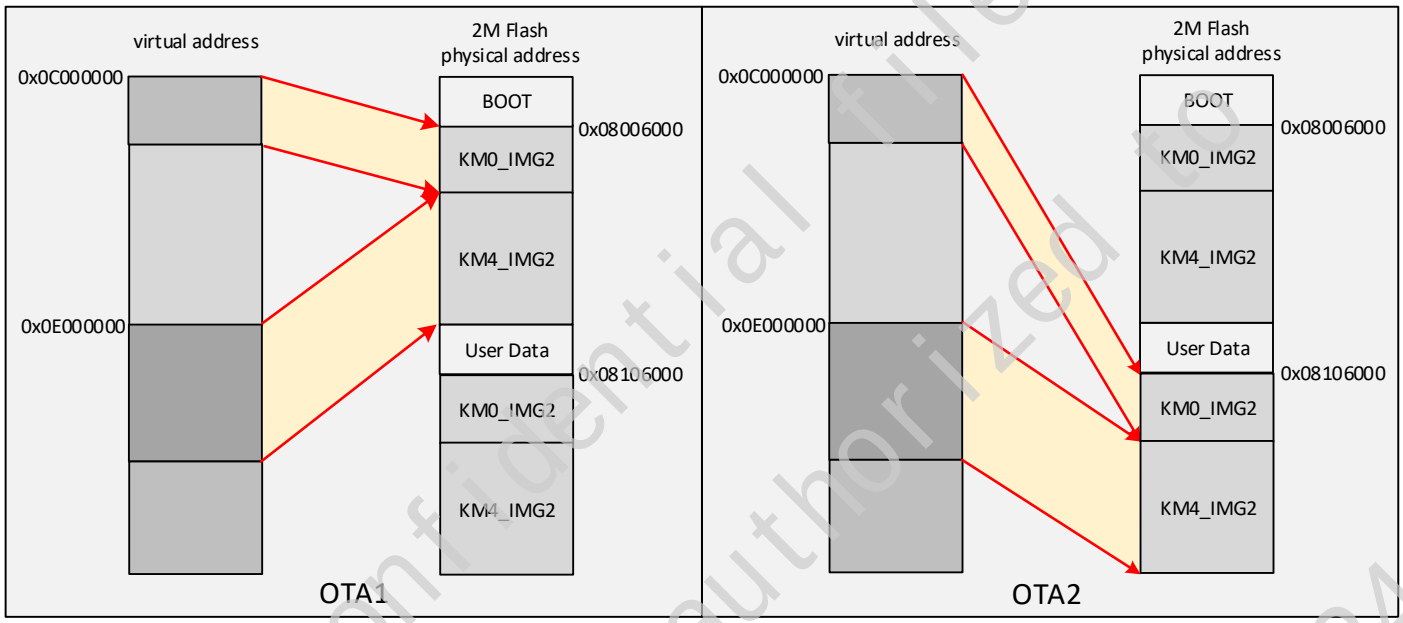


Fig 11-3 2M Flash OTA MMU

MMU entry should be set as Table 11-1.

Table 11-1 OTA under MMU

Register Type	Register	OTA1	OTA2
MMU Entry0	MMU_ENTRY0_STRADDR	0x0C00_0000	0x0C00_0000
	MMU_ENTRY0_ENDADDR	0x0C00_0000 + KM0 IMG2 size	0x0C00_0000 + KM0 IMG2 size
	MMU_ENTRY0_OFFSET	0x0C00_0000 – 0x0800_6000	0x0C00_0000 – 0x0810_6000
	+/-	-	-
MMU Entry1	MMU_ENTRY1_STRADDR	0x0E00_0000	0x0E00_0000
	MMU_ENTRY1_ENDADDR	0x0E00_0000 + KM4 IMG2 size	0x0E00_0000 + KM4 IMG2 size
	MMU_ENTRY1_OFFSET	0x0E00_0000 – KM4 start physical address	0x0E00_0000 – KM4 start physical address
	+/-	-	-

Note: Considering KM4 IMG2 is appended to the tail of KM0 IMG2, the KM4 start physical address is determined by KM0 start physical address and KM0 IMG2 size.

11.3 OTA Upgrade from Local Server

The OTA from local server shows how device updates image from a local download server. The local download server sends image to device based on network socket, as Fig 11-4 shows.

Make sure both device and PC are connecting to the same local network.

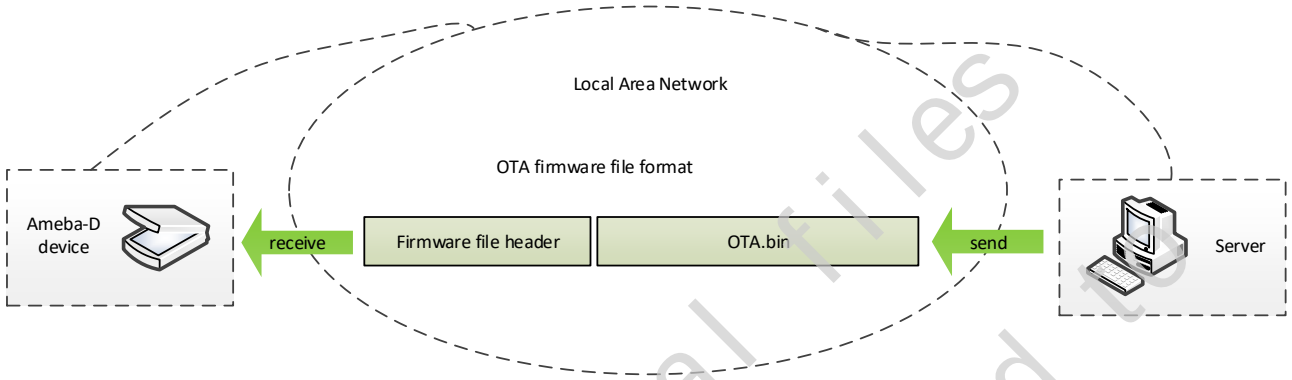


Fig 11-4 OTA update diagram

11.3.1 Firmware Format

The firmware format is illustrated in Fig 11-5.

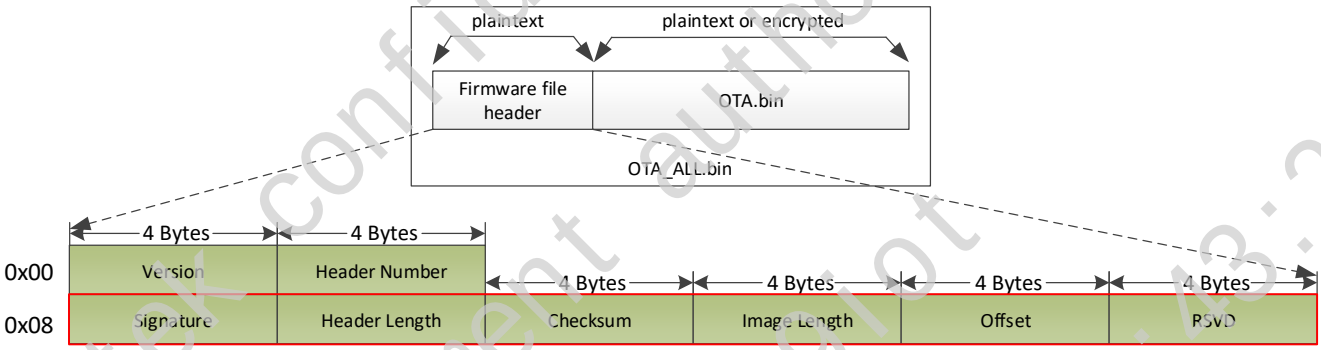


Fig 11-5 Firmware format

Table 11-2 Firmware header

Items	Address Offset	Size	Description
Version	0x00	4 bytes	The version of OTA Header, default 0xFFFFFFFF
Header Number	0x04	4 bytes	The number of OTA Header. For Ameba-D, this value is 0x01
Signature	0x08	4 bytes	OTA Signature is string. For Ameba-D, this value is “OTA”
Header Length	0x0C	4 bytes	The length of OTA header. For Ameba-D, this value is 0x18
Checksum	0x10	4 bytes	The checksum of OTA image
Image Length	0x14	4 bytes	The size of OTA image
Offset	0x18	4 bytes	The start position of OTA image in current image
RSVD	0x1C	4 bytes	Reserved

11.3.2 OTA Flow

The OTA demo is provided in **rtl8721d_ota.c**. The image upgrading is implemented in the following steps:

(1) Connect to local server with socket. The IP address and port are needed.
(2) Acquire the older firmware address to be upgraded according to MMU setting. If address is re-mapping to OTA1 space by MMU, the OTA2 address would be selected to upgrade. Otherwise OTA1 address would be selected.
(3) Receive firmware file header to get the target OTA image information, such as image length and destination address.
(4) Erase flash space for new firmware

(5) Download new firmware from server and write it to flash
(6) Verify checksum. If checksum error, OTA fail.
(7) If checksum is ok, write signature to the upgraded firmware region and change another signature to all zero to indicate boot from new firmware next time.
(8) OTA finish and reset the device. Then it would boot from new firmware.

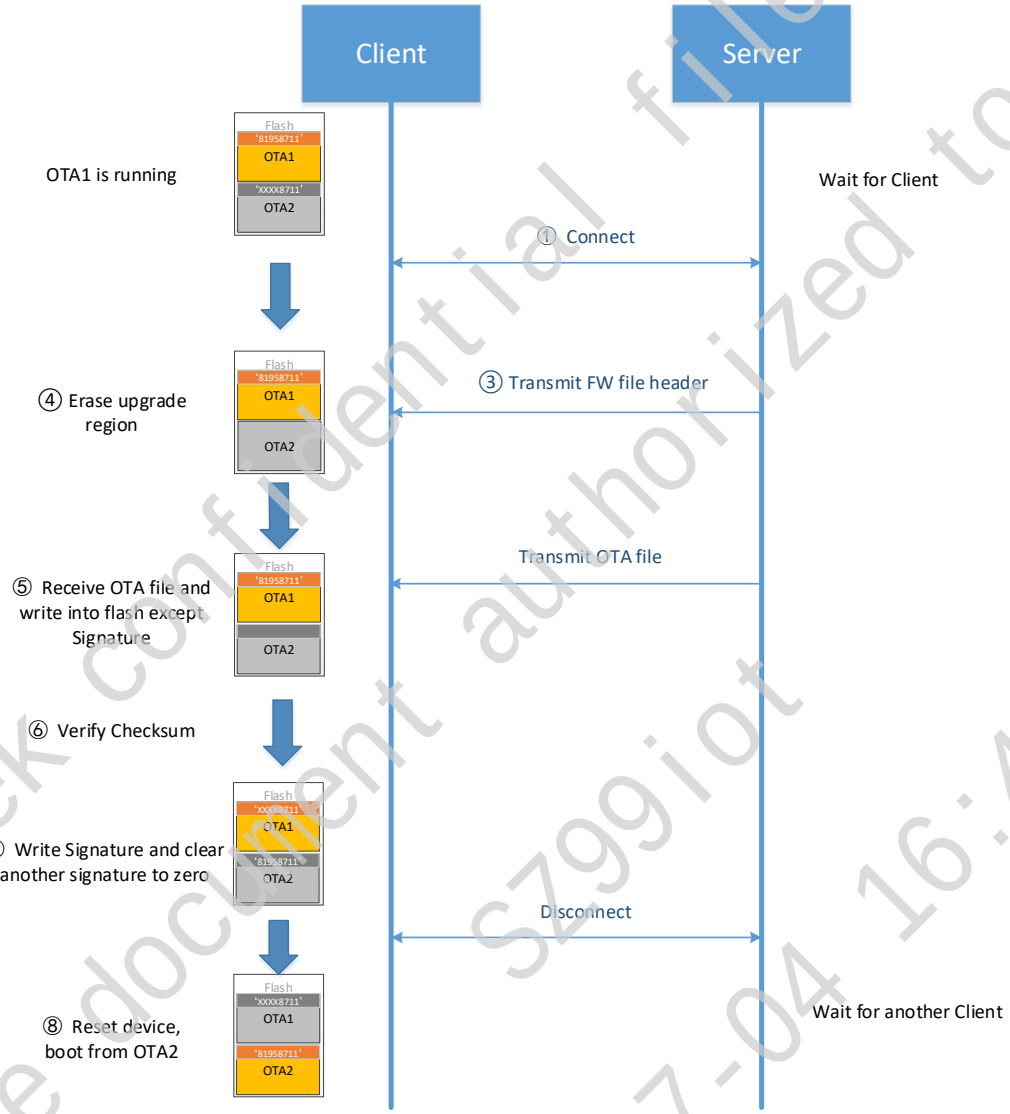


Fig 11-6 OTA operation flow

11.3.3 Boot Option

When reboot after OTA finished, device would check firmware to determine to boot from OTA1 or OTA2. The following items must be checked for each firmware:

- Signature. If it is "81958711", the signature is valid. Otherwise, the signature is invalid.
- Hash/checksum if needed. Users can define FwCheckCallback in rtl8721d_bootcfg.

The boot flow is as following:
(1) Check Signature and hash (if need) of OTA1 and OTA2 firmware.
(2) If both images are invalid, boot fail
(3) If only one image is valid, boot from this image

(4) If both images are valid, boot from OTA2

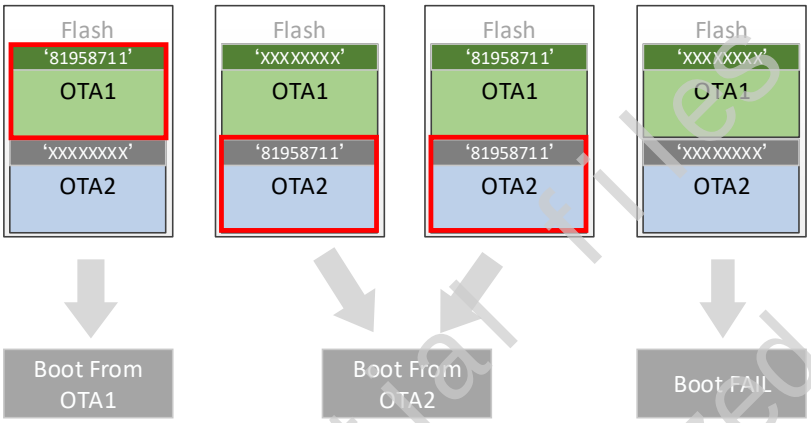


Fig 11-7 OTA select diagram

11.3.4 Address Remapping

In bootloader, MMU entry0 and entry1 would be set as Table 11-1 to remap memory. Here, the OTA space includes these images: KM0 IMG2, KM4 IMG2, KM4 IMG3 if need. These images all boot from OTA1 or OTA2.

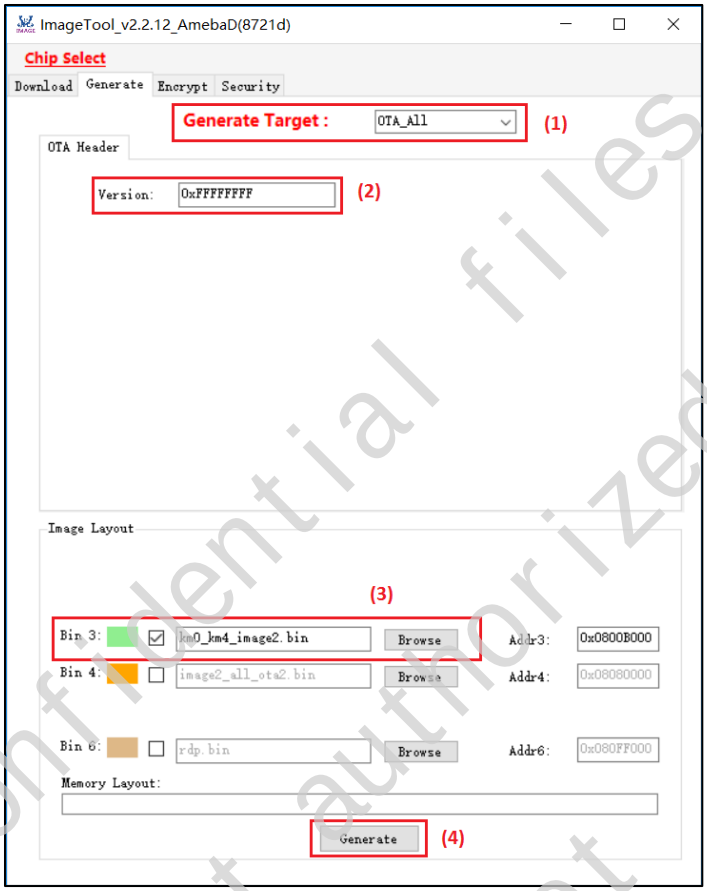
As Fig 11-3 illustrates, when boot from OTA, CPU may access different physical address.

- When boot from OTA1, MMU entry0 and entry1 would be set to remap virtual memory to OTA1 address space. That is, if CPU wants to read code or data from address 0x0C00_0000, it actually accesses physical address 0x0800_6000.
- When boot from OTA2, MMU entry0 and entry1 would be set to remap virtual memory to OTA2 address space. That is, if CPU wants to read code or data from address 0x0C00_0000, it actually accesses physical address 0x0810_6000.

11.3.5 How to Use OTA Demo?

These steps can be followed to run the OTA demo:

(1) Define CONFIG_OTA_UPDATE macro to 1 in **platform.opts.h** to enable OTA function. Rebuild the project.
(2) Download images to device.
(3) Generate OTA upgrade image file, named **OTA_All.bin**, with Image Tool and new firmware. This operation perpends firmware header information to new firmware, which is necessary for OTA.



(4) Copy **OTA_All.bin** into the **DownloadServer** folder.

- Tool path in SDK: **tools\DownloadServer**

(5) Edit **tools\DownloadServer\start.bat**.

- Port = 8082
- File name =OTA_All.bin

```
@echo off
DownloadServer 8082 OTA_All.bin
set /p DUMMY=Press Enter to Continue ...
```

(6) Click the **start.bat**, start to download server program.

(7) Reboot the DUT and connect the device to the AP which the OTA Server in.
(8) Enter command: **ATWO=*IP*[*PORT*]**.

- IP: IP of the OTA Server.
- Port: 8082, the same with start.bat

```
# ATWO=192.168.0.20[8082]
[ATWO]: _AT_WLAN_OTA_UPDATE_

[MEM] After do cmd, available heap 28400


#
[ota_update_local_task] Update task start
```

OTA upgrade procedure is started between DUT and server. Here is the local download server success message.

```
Listening on port (8082)

Waiting for client ...
Accept client connection from 192.168.1.105
sending OTA_All.bin now...
Sending file...
.....
.....
.....
.....
.....
.....
.....
.....
.....
Total send 672024 bytes
```

(9) Reboot DUT to execute the new firmware after finishing image download.

11.3.6 OTA Firmware Swap

Fig 11-8 shows the firmware swap procedure after OTA upgrade.

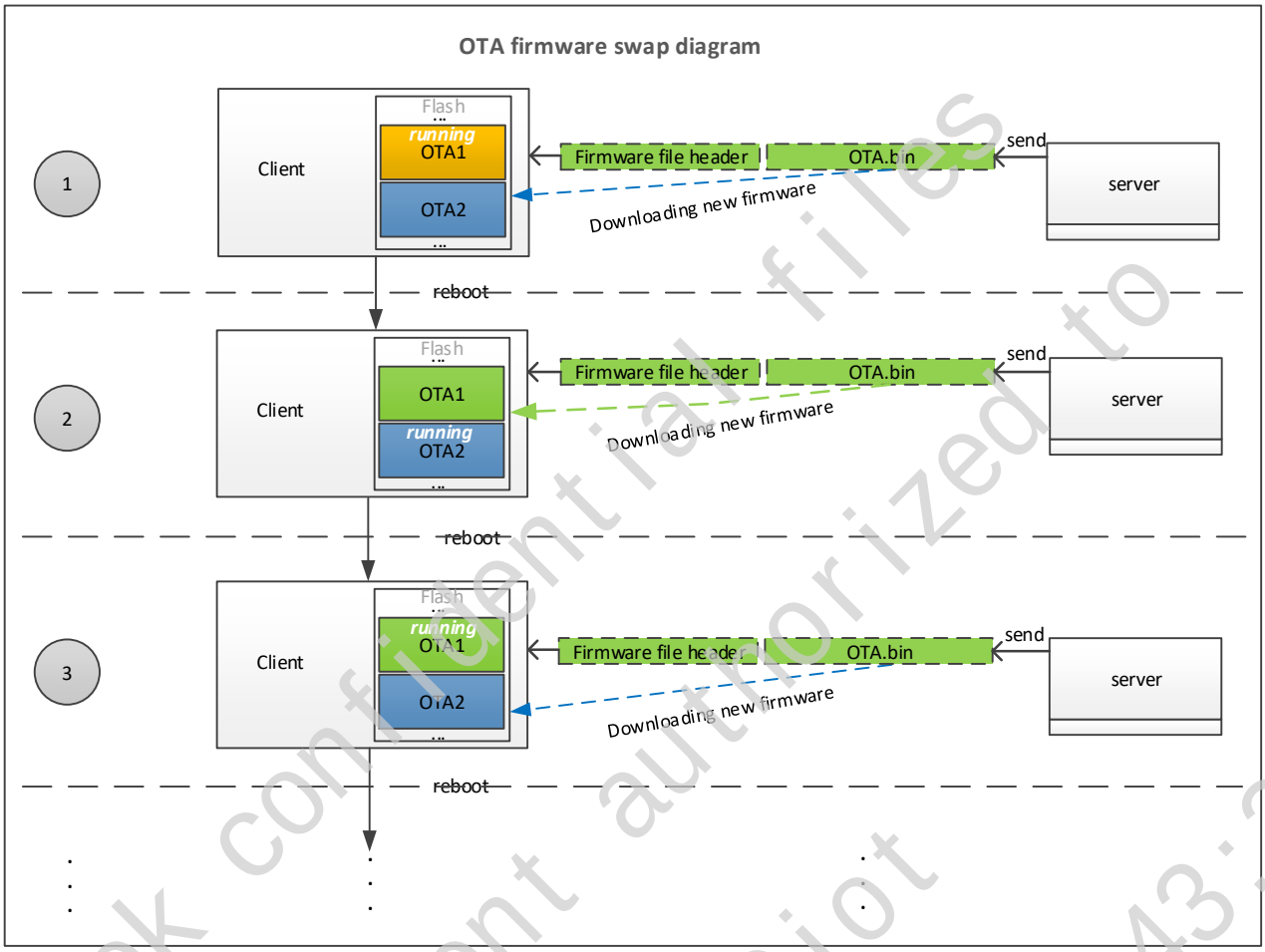


Fig 11-8 OTA firmware swap procedure

11.4 User Configuration

Users can find the following configure items in **rtl8721d_bootcfg.c**.

- OTA start address

```
/*
* @brief OTA start address. Because KM0 & KM4 IMG2 are combined, users only need to set the start address
*        of KM0 IMG2.
*/
BOOT_RAM_DATA_SECTION
u32 OTA_Region[2] = {
        0x08006000,             /* OTA1 region start address */
        0x08106000,             /* OTA2 region start address */
};
```

- User-defined MMU configure

```
/*
* @brif MMU Configuration.
*       There are 8 MMU entries totally. Entry 0 & Entry 1 are already used by OTA, Entry 2~7 can be used by Users.
*/
BOOT_RAM_DATA_SECTION
MMU_ConfDef Flash_MMU_Config[] = {
        /*VAddrStart,   VAddrEnd,                   PAddrStart,           PAddrEnd*/
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},    //Entry 2
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},    //Entry 3
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},    //Entry 4
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},    //Entry 5
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},    //Entry 6
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},    //Entry 7
        {0xFFFFFFFF,    0xFFFFFFFF,                 0xFFFFFFFF,           0xFFFFFFFF},
};
```

- Firmware check handler configure

```
/**
* @brif  Firmware verify callback handler.
*       If users need to verify checksum/hash for image2, implement the function and assign the address
*       to this function pointer.
*/
BOOT_RAM_DATA_SECTION
FwCheckFunc FwCheckCallback = NULL;
```

12 Power Save

12.1 Hardware Power Save Modes

12.1.1 Power Mode

Ameba-D supports two low power modes which are deepsleep mode and sleep mode. Deepsleep mode turns off more power domain than sleep mode.

RTC/AON Timer/AON Pin (low power pin)/Key-Scan/Cap-Touch can be used to wakeup both sleep and deepsleep mode (refer to 12.2.4).

Table 12-1 Power domain comparison

Item	Sleep (WoWLAN)	Deepsleep
Main digital power	✓	✓
SWR	Δ[1]	x
KM4 core	x	x
KM0 core	Δ	x
System PLL	x	x
KM4 Timer	x	x
KM0 Timer	Δ	x
XTAL	Δ	x
OSC4M	Δ	x
OSC2M	Δ	x
OSC131K/32K	✓	✓
SRAM	✓	x
Retention SRAM	✓	✓
AON register	✓	✓
AON Timer (32.768KHz, Max. 546min)	✓	✓
Key-Scan	✓	✓
Cap-Touch	✓	✓
RTC	✓	✓
Wi-Fi	Δ	x
UART	Δ	x
GPIO wakup	Δ	x
AON pin (low power pin)	✓	x
ADC	✓	x
Comparator	✓	x

1. Users can choose to turn on/off this power domain in sleep mode by themselves.

Note:

- All GPIOs can wake up system from sleep, which are listed in UM0402 pinmux table
- There are some special GPIOs, for example, PA[7] and PA[8] are used for UART download, while PA[27] and PB[3] are used for SWD. If they are used for other purposes, they can't be used as a general GPIO to wake up system.

Depending on QFN package, there are at most 12 AON_Wakepin, refer to UM0402 pinmux table (aon pinmux sheet) for more information. All AON_Wakepin can wake up system from deepsleep, as shown in Table 12-2.

Table 12-2 AON_Wakepin

Item	PINMUX_S0	PINMUX_S1	PINMUX_S2
wakepin 0	PA[12]	PA[16]	PA[20]
wakepin 1	PA[13]	PA[17]	PA[21]
wakepin 2	PA[14]	PA[18]	PA[25]
wakepin 3	PA[15]	PA[19]	PA]26]

12.1.2 Power Consumption Summary

Table 12-3 and Table 12-4 list the power consumption under 3.3V/1.8V power supply.

Table 12-3 Power consumption under 3.3V/1.8V

Operation Mode		Condition	Current		Unit
Power Mode	Scenario		3.3V	1.8V	
Deepsleep	Deepsleep	RTC timer 1KB retention RAM	7~8	7-8	uA
	Deepsleep with Key-Scan	RTC timer 1KB retention RAM Key-Scan	12~13	12~13	uA
	Deepsleep with Cap-Touch (average current)	RTC timer 1KB retention RAM Cap-Touch	20	16	uA
Sleep	WoWLAN sleep power	KM4 power gate KM0 clock gate All RAM retained Wi-Fi retained	30~50	30~50	uA
Active	Wi-Fi Rx Idle	HT20 MCS0~7 normal mode KM4 in active mode Rx idle	52	81	mA
		HT20 MCS0~7 ultra-low power mode KM4 in active mode Rx idle	35	60	mA
	Wi-Fi Rx UDP	HT20 MCS0~7 ultra-low power mode KM4 in active mode UDP Rx @ 8Mbps	39	67	mA
WoWLAN	WoWLAN Rx Beacon	Rx beacon mode @ normal mode KM4 in sleep mode	28	45	mA
		Rx beacon mode @ ultra-low power mode KM4 in sleep mode	23	39	mA
	WoWLAN DTIM=1 (Average)	KM4 in sleep mode All SRAM retained Wi-Fi retained Shielding room	700~800	1100~1200	uA
		KM4 in sleep mode All SRAM retained Wi-Fi retained Open space	1~2	1.1~2	mA

Note: Ultra-low power mode side effect:

- OFDM: Rx Sensitivity Degree 2~4dBm
- CCK: Rx Sensitivity Degree 1~2dBm

Table 12-4 Tx power consumption under 3.3V/1.8V

Channel	Power	Operation Mode	2.4G	5G	Unit
2442MHz@2.4G 5500MHz@5G 20M 5510MHz@5G 40M	1.8V	1T-MCS7/BW40M (9dBm@2.4G, 8dBm@5G)	181	216	mA
		1T-MCS7/BW40M (12dBm@2.4G, 11dBm@5G)	195	237	
		1T-MCS7/BW20M (9dBm@2.4G, 8dBm@5G)	180	214	
		1T-MCS7/BW20M (12dBm@2.4G, 11dBm@5G)	193	234	
		1T-Legacy_OFDM54M (10dBm@2.4G, 9dBm@5G)	184	219	
		1T-Legacy_OFDM54M (13dBm@2.4G, 12dBm@5G)	214	245	
		1T_CCK11M (12dBm)	203	-	
		1T_CCK11M (15dBm)	224	-	
		1R-Idle/BW40	89	90	
		1R-MCS7/BW40M (Pin= -60dBm)	98	103	

		1R-MCS7/BW20M (Pin= -60dBm)	102	106
		1R-Legacy_OFDM54M (Pin= -60dBm)	98	103
		1R-CCK11M (Pin= -60dBm)	81	-
		RF Standby	58	55
		RF Disable	43	42
	3.3V	1T-MCS7/BW40M (15dBm)	206	286
		1T-MCS7/BW40M (18dBm@2.4G, 17dBm@5G)	247	310
		1T-MCS7/BW20M (15dBm)	204	286
		1T-MCS7/BW20M (18dBm)	248	308
		1T-Legacy_OFDM54M (16dBm)	214	296
		1T-Legacy_OFDM54M (19dBm@2.4 18dBm@5G)	262	323
		1T_CCK11M (18dBm)	257	-
		1T_CCK11M (21dBm)	312	-
		1R-Idle/BW40	52	53
		1R-MCS7/BW40M (Pin= -60dBm)	61	64
		1R-MCS7/BW20M (Pin= -60dBm)	62	63
		1R-Legacy_OFDM54M (Pin= -60dBm)	61	62
		1R-CCK11M (Pin= -60dBm)	52	-
		RF Standby	24	23
		RF Disable	24	23

Note:

- The values above are tested with the Ameba-D QFN68 board.
- Tx: Continue Tx DPK on
- Rx: Set Rx Packets idle interval as short as possible
- Load W11N_11M0 Idle Waveform (8us idle interval) for 1R-CCK11M test.

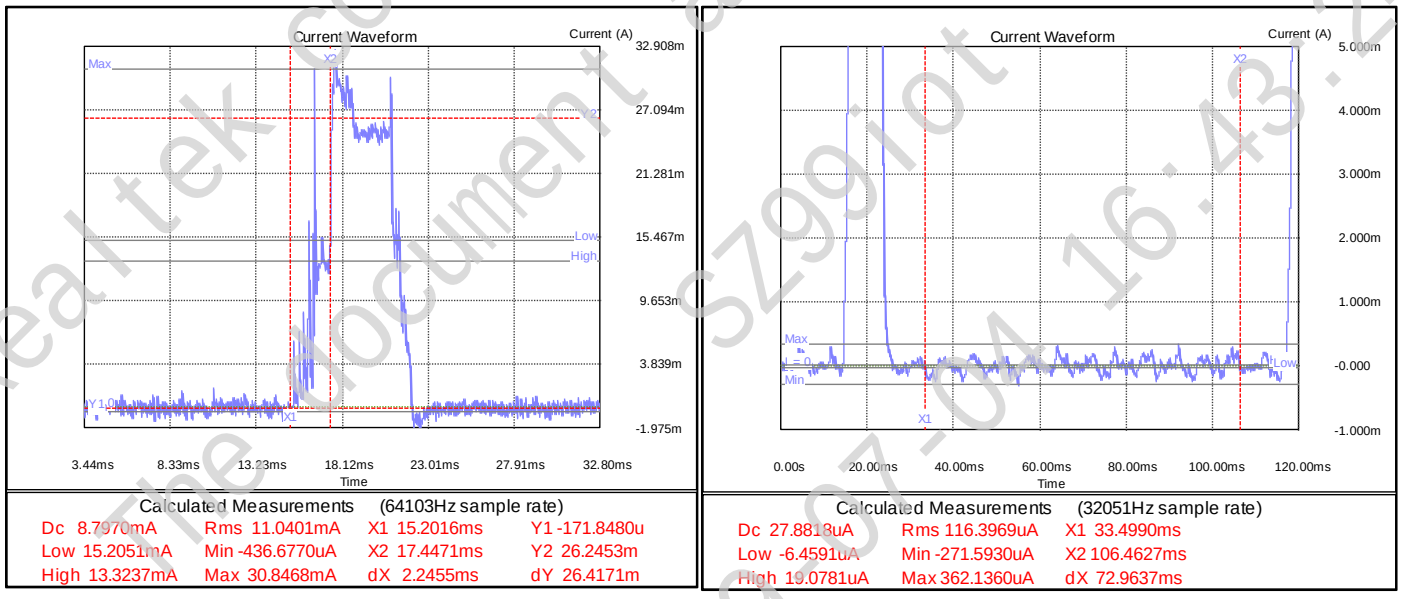


Fig 12-1 WoWLAN (3.3V normal mode)

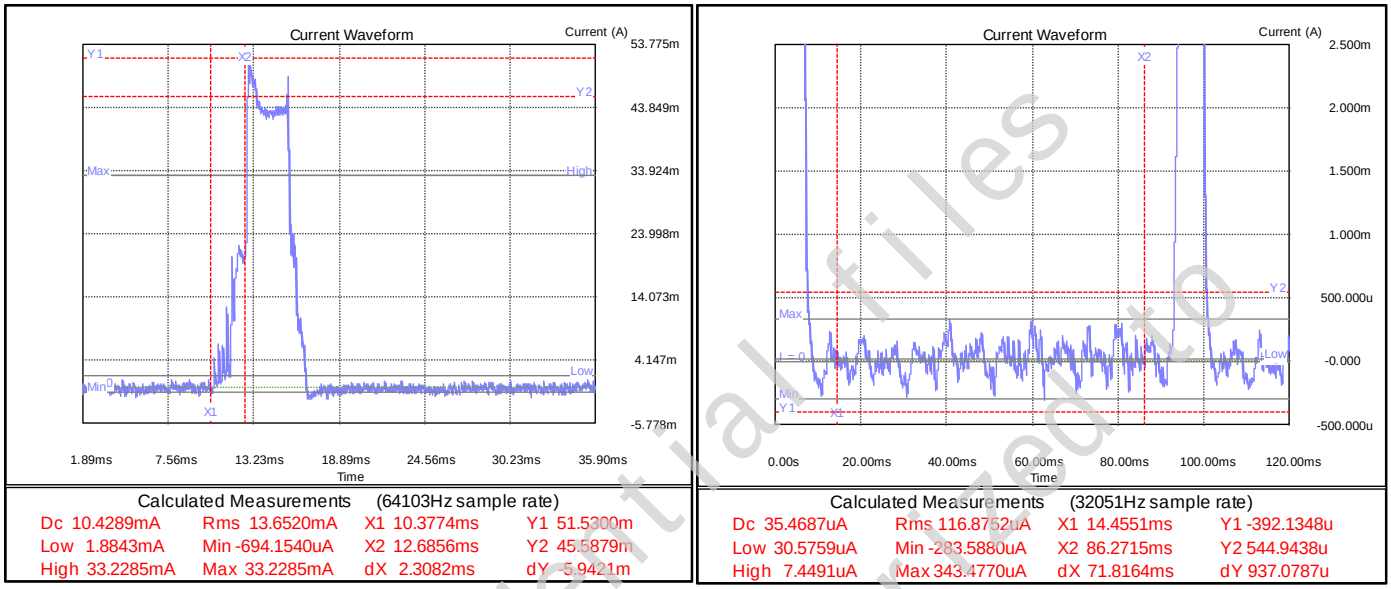


Fig 12-2 WoWLAN (1.8V normal mode)

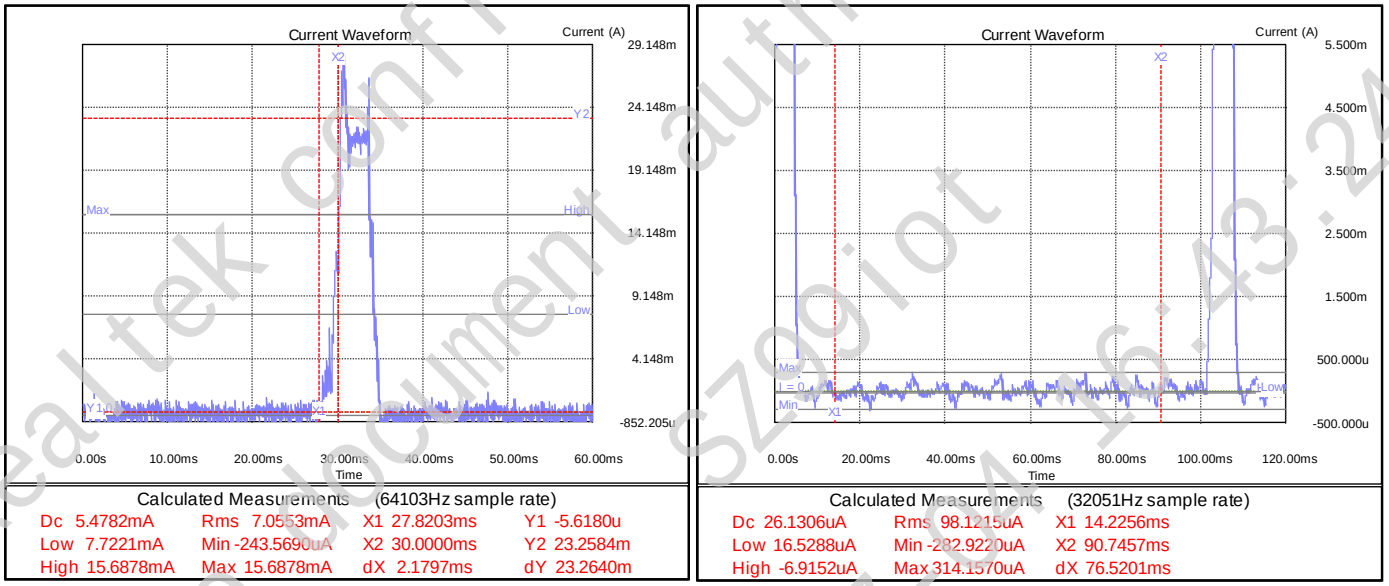


Fig 12-3 WoWLAN (3.3V ultra-low power mode)

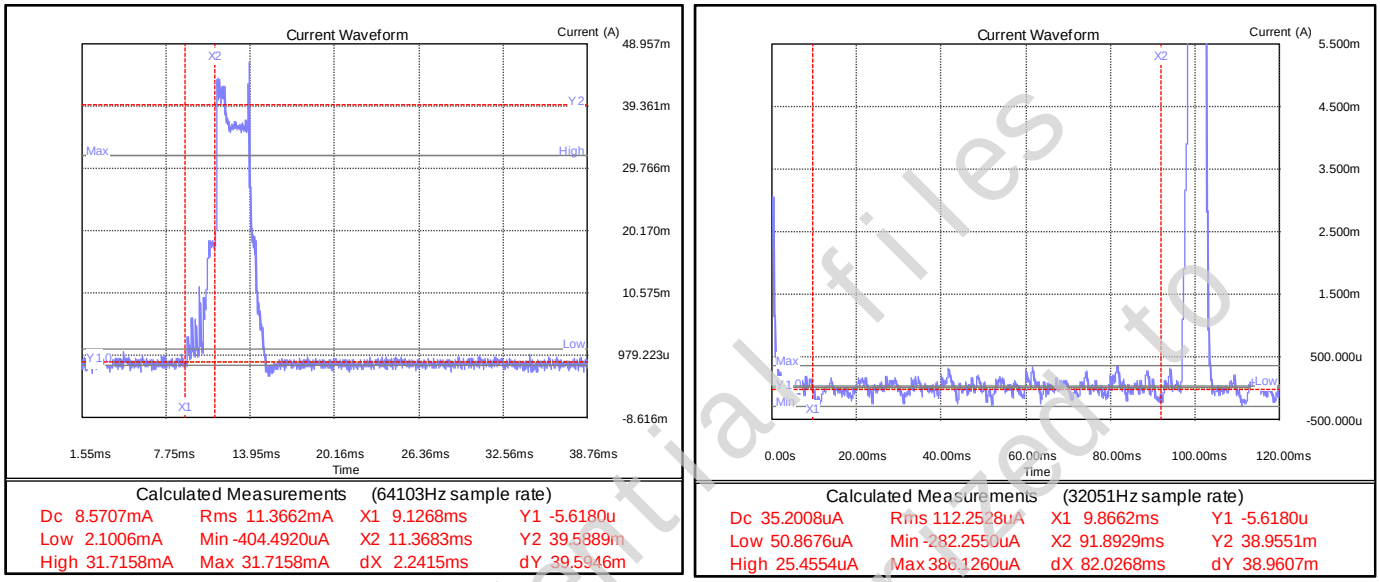


Fig 12-4 WoWLAN (1.8V ultra-low power mode)

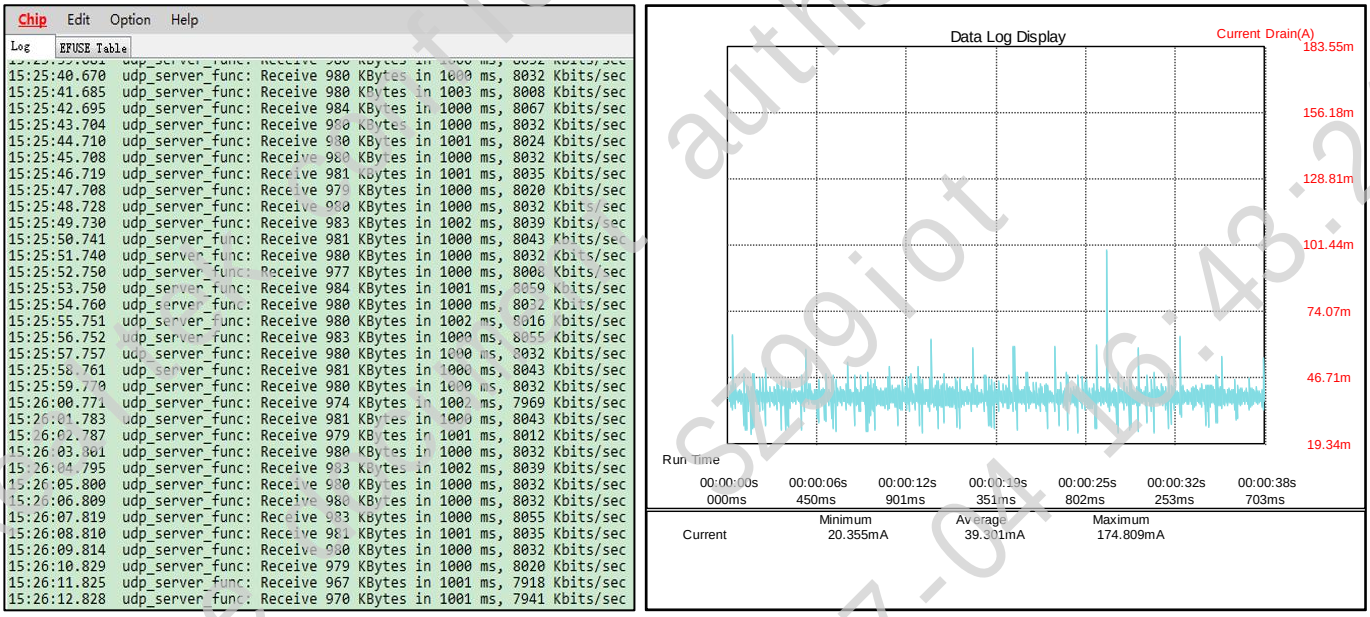


Fig 12-5 UDP performance (3.3V ultra-low power mode)

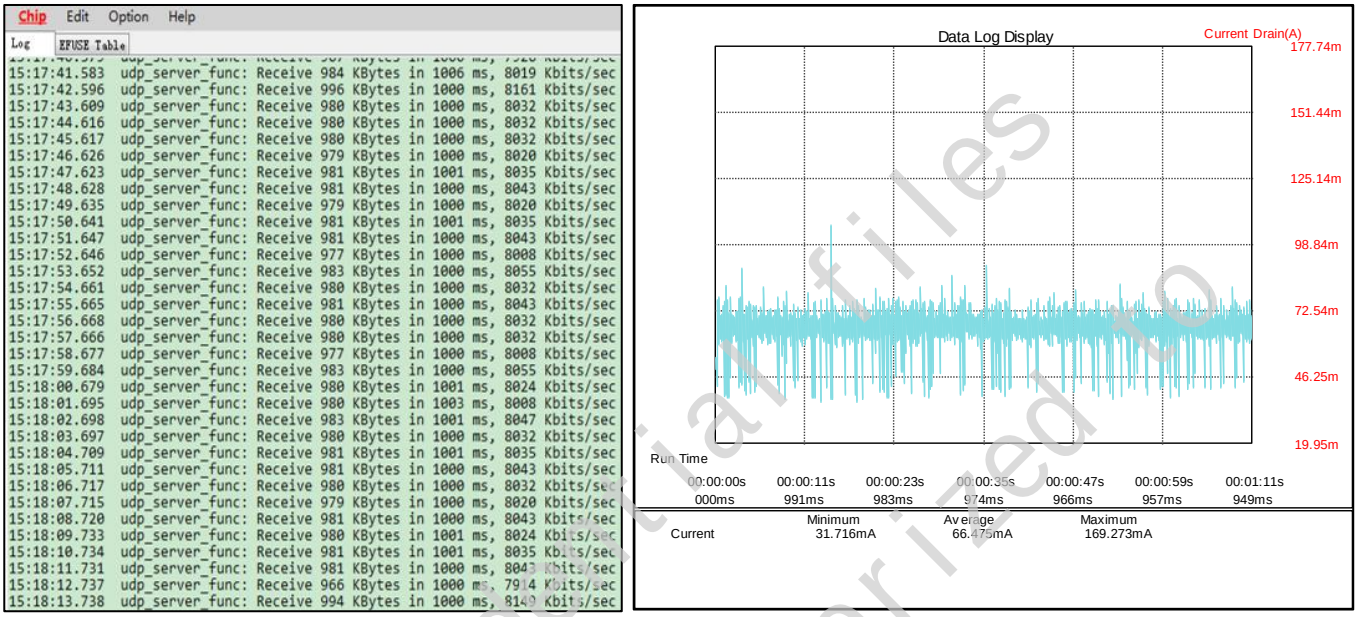


Fig 12-6 UDP performance (1.8V ultra-low power mode)

12.2 Software Power Management

12.2.1 FreeRTOS Tickles

FreeRTOS supports a low power feature called tickles. It is implemented in idle task which has the lowest priority. That is, it is invoked when there is no other task under running.

FreeRTOS places the microcontroller into a low power state, if it expects there will be no event in near future. It calculates expected idle time by looking timer task list. Then it performs suspend action by using ARM instruction "WFI" (Wait for Interrupt) which makes the processor suspend execution (Clock is stopped) until interrupt happened.

The interrupts include timer, which is set by FreeRTOS with value equal to the expected idle time. So when idle task is invoked every time, FreeRTOS calculates the expected idle time, sets timer, and puts process into suspend.

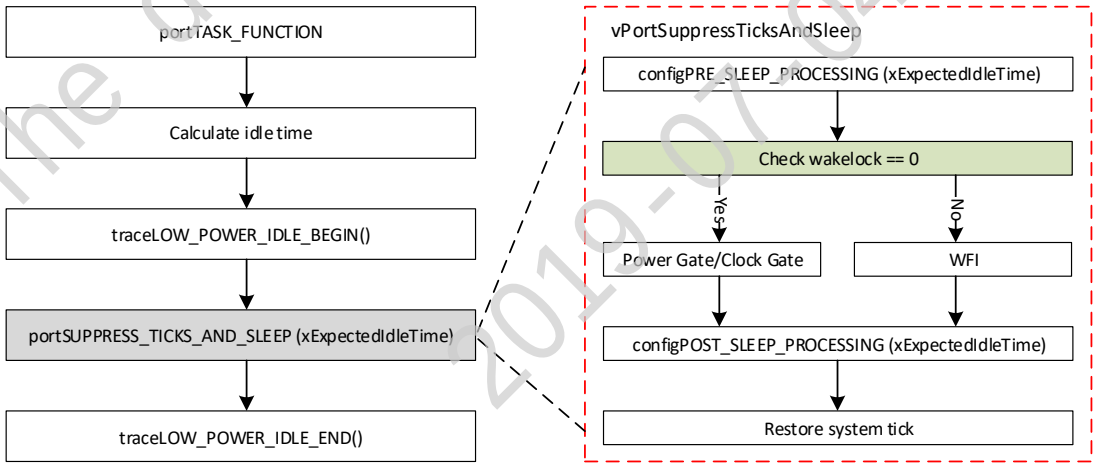


Fig 12-7 FreeRTOS tickles in idle task

12.2.2 Wakelock

In some situations, system is needed to keep awake to receive certain events. Otherwise, event may be missed when system is under sleep. An idea of wakelock is introduced that system cannot sleep if some module holding wakelock.

A wakelock bit map is for storing the wakelock status. Each module has its own bit in wakelock bit map (see enum. PMU_DEVICE). If the wakelock bit map equals to zero, it means there is no module holding wakelock. If the wakelock bit map larger than zero, it means there is some module holding wakelock.

If there is no one holding wakelock, then system is permit to sleep. If there are one or more modules holding wakelock, then we abort sleep.

12.2.3 Wi-Fi Power Save

In IEEE 802.11 power save management, it allows station enter their own sleep state. It defines station need keep awake in certain timestamp and enter sleep state otherwise.

WLAN driver acquire wakelock to avoid system enter sleep when WLAN need keep awake. And it releases wakelock when it is permitted to enter sleep state.

IEEE 802.11 power management allows station enter power saving mode. Station cannot receive any frames during power saving. Thus AP need buffers these frames and requires station periodically wakeup to check beacon which has information of buffered frames.

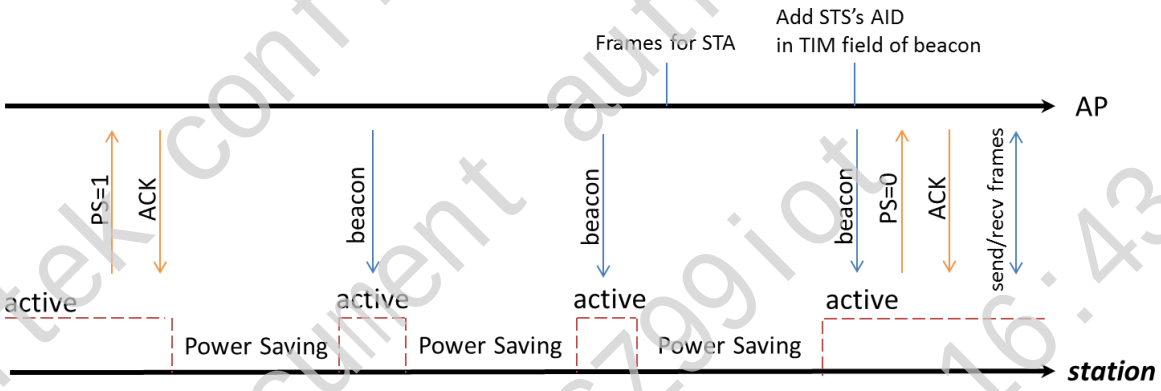


Fig 12-8 Timeline of power saving

In SDK IEEE 802.11 power management is called LPS, and if KM4 enter power save mode when Wi-Fi is in LPS mode, we call it WoWLAN mode.

Except LPS and WoWLAN we also have IPS, which can be used when Wi-Fi is disconnected. Table 12-5 lists all three power save mode for Wi-Fi.

Table 12-5 Wi-Fi power save mode

Items	Wi-Fi Status	Comment
IPS	Not associated	Driver automatically turns off Wi-Fi to save power.
LPS	Associated	LPS is used to implement IEEE 802.11 power management. KM0 will control RF ON/OFF based on TSF and TIM IE in beacon.
WoWLAN	Associated	LPS + Tickles KM0 will wakeup KM4 when receive data packet.

12.2.4 Software Configuration

Type	Items	User Configuration
	km0_pwrmgt_config	KM0 power management control: system use

Sleep (WoWLAN)	sleep_wevent_config	● BIT_LP_WEVT_HS_MSK: KM4 peripheral wakeup open ● BIT_LP_WEVT_AON_MSK: AON wake event wakeup sleep mode ● BIT_LP_WEVT_SGPIO_MSK: SGPIO wakeup event ● BIT_LP_WEVT_COMP_MSK: ADC comparator wakeup event ● BIT_LP_WEVT_ADC_MSK: ADC wakeup event ● BIT_LP_WEVT_I2C_MSK: I2C slave wakeup event ● BIT_LP_WEVT_UART_MSK: LP UART wakeup event ● BIT_LP_WEVT_WLAN_MSK: WLAN wake up event ● BIT_LP_WEVT_GPIO_MSK: GPIO wakeup event ● BIT_LP_WEVT_TIMER_MSK: LP TIM0~5 wakeup event
	sleep_hsram_config	System use
	sleep_lsram_config	System use
	sleep_aon_wevent_config	● BIT_CAPTOUCH_WAKE_STS: Cap-Touch wake event ● BIT_KEYSCAN_WAKE_STS1: Key-Scan wake event ● BIT_RTC_WAKE_STS1: RTC wake event ● BIT_AON_WAKE_TIM0_STS1: AON timer wake event ● BIT_GPIO_WAKE_STS1: AON GPIO wake, see aon_wakepin & dsleep_wakepin_config
Deepsleep	dsleep_aon_wevent_config	● BIT_CAPTOUCH_WAKE_STS: Cap-Touch wake event ● BIT_KEYSCAN_WAKE_STS1: Key-Scan wake event ● BIT_RTC_WAKE_STS1: RTC wake event ● BIT_AON_WAKE_TIM0_STS1: AON timer wake event ● BIT_GPIO_WAKE_STS1: AON GPIO wake, see aon_wakepin & dsleep_wakepin_config
	dsleep_lsram_config	System use
Sleep or Deepsleep	sleep_wakepin_config	Configure AON wake pin pinmux and enable
	ps_config	System use

12.2.5 GPIO Pull Control

It needs doing I/O pull control when enter deep sleep and sleep mode. Otherwise it results power leakage. For example, UART voltage level is high. If we pull down UART pin or not pull, then power leakage happens. So we need make sure each pin has proper pull control.

In SDK you should set GPIO function pull control and sleep pull control in pmap_func based on you PCB board, and SDK will set pull control based on your setting between suspend and resume.

```
const PMAP_TypeDef pmap_func[]=
{
//      Pin Name        Func PU/PD          Slp PU/PD          DSlp PU/PD              LowPowerPin
    {_PA_0,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_1,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_2,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_3,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_4,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_5,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_6,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_7,      GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //UART_LOG_TXD
    {_PA_8,      GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //UART_LOG_RXD
    {_PA_9,      GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_10,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_11,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_12,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_13,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_14,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_15,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_16,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_17,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_18,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_19,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_20,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_21,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_22,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_23,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_24,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_25,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_26,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_27,         GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SWD_DATA or nc
    {_PA_28,         GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_29,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_30,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_31,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_0,      GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_1,      GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_2,      GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_3,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SWD_CLK
    {_PB_4,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch0
    {_PB_5,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch1
    {_PB_6,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch2
    //{_PB_6,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE},
    {_PB_7,      GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch3
    {_PB_8,      GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_9,      GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_10,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_11,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_12,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA3
    {_PB_13,         GPIO_PuPd_UP,       GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     FALSE}, //SPI_CLK
    {_PB_14,         GPIO_PuPd_UP,       GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA0
    {_PB_15,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA2
    {_PB_16,         GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SPI_CS
    {_PB_17,         GPIO_PuPd_UP,       GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA1
    {_PB_18,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_19,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_20,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_21,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_22,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //IR pin should no-pull
    //{_PB_22,       GPIO_PuPd_NOPULL,   GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE},
    {_PB_23,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_24,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_25,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_26,         GPIO_PuPd_SHUTDOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, // this is a pcb boa
    {_PB_27,         GPIO_PuPd_UP,    GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_28,         GPIO_PuPd_UP,    GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_29,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_30,         GPIO_PuPd_UP,    GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_31,         GPIO_PuPd_DOWN,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PNC,       GPIO_PuPd_KEEP,  GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //table end
};
```

12.3 Suspend Resume APIs

Tickless API	Introduction
<pmu_register_sleep_callback>	Register suspend/resume call back function for one module
<pmu_unregister_sleep_callback>	Unregister suspend/resume call back function
<pmu_acquire_wakelock>	Acquire wake lock for one module
<pmu_release_wakelock>	Release wake lock
<pmu_set_sysactive_time>	Acquire wake lock, and release wake lock automatically after timeout
<pmu_set_max_sleep_time>	Set max sleep time

12.3.1 pmu_register_sleep_callback

Register suspend/resume call back function for <nDeviceId>, suspend callback function will be called by PMU before system enter sleep mode, and resume callback function will be called after system resume.

Note:

- Yield OS is not permitted in suspend/resume callback function like: taskYIELD, vTaskDelay, mutex, sema and so on.
- pmu_set_sysactive_time is not permitted in suspend callback function.
- pmu_set_sysactive_time is permitted in resume call back function.

Parameter	Type	Introduction
<nDeviceId>	uint32_t	Device ID need suspend/resume callback typedef enum { PMU_OS = 0, … PMU_MAX = 31 } PMU_DEVICE;
<sleep_hook_fun>	PSM_HOOK_FUN	Suspend call back function
<sleep_param_ptr>	void*	Suspend call back function parameter
<wakeup_hook_fun>	PSM_HOOK_FUN	Resume call back function
<wakeup_param_ptr>	void*	Resume call back function parameter

12.3.2 pmu_unregister_sleep_callback

Unregister suspend/resume call back function for <nDeviceId>.

Parameter	Type	Introduction
<nDeviceId>	uint32_t	The same as pmu_register_sleep_callback

12.3.3 pmu_acquire_wakelock

Wakelock is a 32-bit map. Each module owns 1 bit in this bit map. FreeRTOS tickless reference the wakelock and decide that if it can or cannot enter sleep state.

If any module acquires and hold a bit in wakelock, then the whole system won't enter sleep state.

Parameter	Type	Introduction
<nDeviceId>	uint32_t	The same as pmu_register_sleep_callback

12.3.4 pmu_release_wakelock

Release bit[nDeviceId] of wakelock bit map, if wakelock equals to 0, then the system may enter sleep state after system idle.

Parameter	Type	Introduction
<nDeviceId>	uint32_t	The same as pmu_register_sleep_callback

12.3.5 pmu_set_sysactive_time

This function sets system active time, system cannot sleep before timeout.

Note:

- pmu_set_sysactive_time is not permitted in suspend callback function as it is ineffective.

- pmu_set_sysactive_time is permitted in resume call back function.

Parameter	Type	Introduction
<timeout>	uint32_t	system cannot sleep before timeout. Unit is ms.

12.3.6 pmu_set_max_sleep_time

This function sets system max sleep time

Note:
- system maybe waked by other wake event before this timer timeout
- This setting only works once, the timer will be clear after wake up

Parameter	Type	Introduction
<timeout>	uint32_t	system max sleep timeout. Unit is ms.

12.4 Power Consumption Measurement

12.4.1 Test Command

We provide command in KM4 for tickles testing. Below is the description.

Items	Comment
tickps r debug	Release OS wakelock to open tickles function, KM4 will enter sleep mode when idle. In program you can use pmu_release_wakelock (PMU_OS) to open tickles function (ref. 12.3.4).
tickps a	Get OS wakelock to close tickles function.
tickps get	Get current wakelock, it prints current wakelock bit map. You can use this command to debug why system doesn't enter sleep.

12.4.2 Hardware Preparation

In Ameba-D reference board, there are other components that consume power. For example, DAP, LEDs, FT232, and capacitances. To measure power consumptions only for Ameba-D, you need remove resistance at R43.

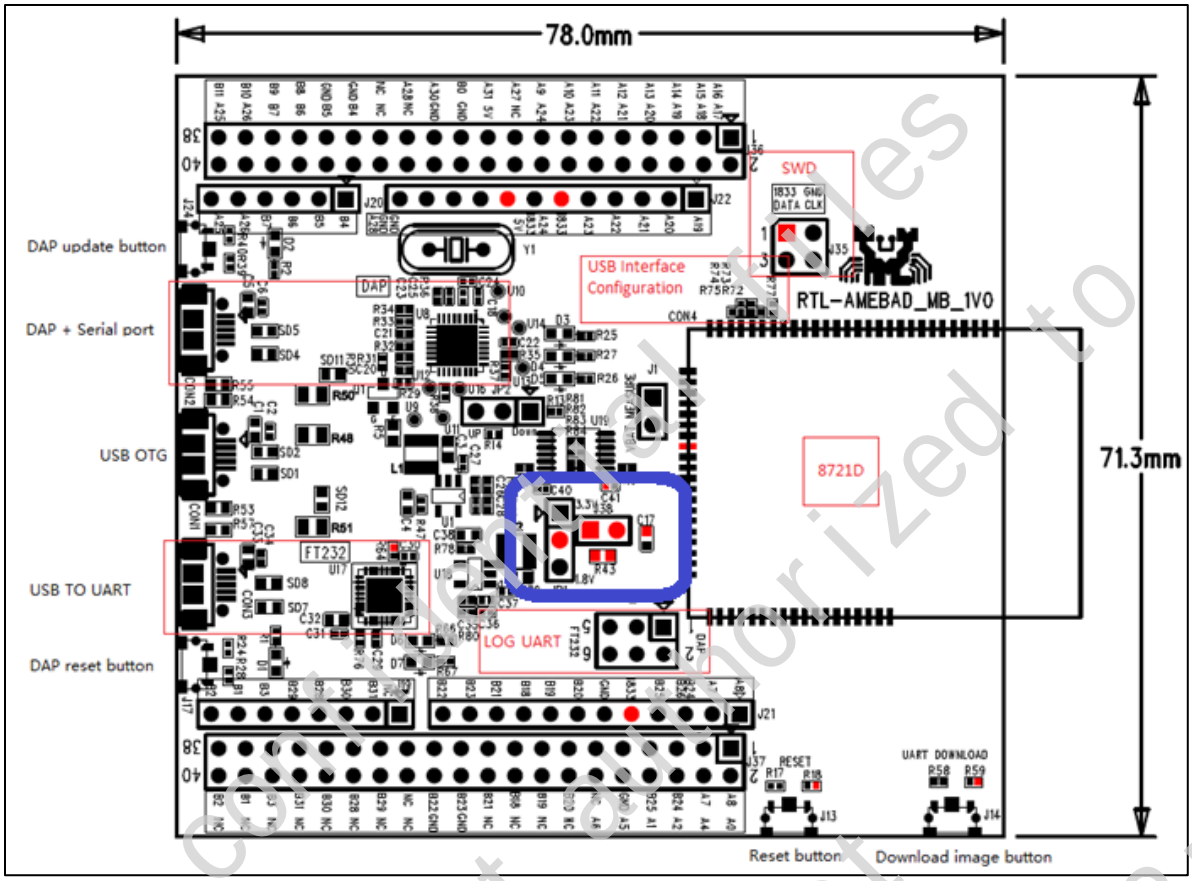


Fig 12-9 Power consumption measurement

You can use micro USB to supply power for the board, and link current meter use J38 like Fig 12-10.

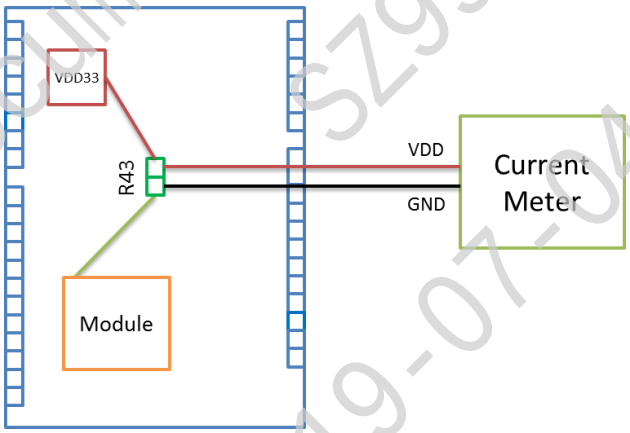


Fig 12-10 Measure power consumption from micro USB

13 User Configuration

Some functions have complex parameters, it's hard for SDK to give standard API. For Example, different users have different parameters for BOOT. So we give some user configuration files which will be called by SDK and SDK will have different activities based on user configurations.

These configuration files should be maintained by users.

13.1 Configuration File List

The user configuration files are listed in Table 13-1.

Table 13-1 User configuration file

File	Configuration Items	Image
rtl8721dhp_boot_trustzonecfg.c	Used to configure KM4 TrustZone no-secure areas.	KM4 bootloader
rtl8721dlp_flashcfg.c	Used to configure flash settings: ● Flash_Speed ● Flash_ReadMode ● Flash_AVL You can also configure your flash use flash_init_userdef, if your flash is not in Flash_AVL.	KM0 image2
Rtl8721dlp_pinmapcfg.c	Used to reduce I/O power leakage: ● per pin function configuration ● per pin function pull up & pull down ● per pin sleep pull up & pull down	KM0 image2
Rtl8721dlp_sleepcfg.c	● Sleep/deepsleep power management ● Sleep/deepsleep wakeup event ● Sleep/deepsleep wake pin	KM0 image2
rtl8721d_bootcfg.c	Used to configure boot loader: ● Flash_MMU_Config ● RSIP_Mask_Config ● Force_OTA1_GPIO ● Boot_Log_En	● KM0 bootloader ● KM4 bootloader
rtl8721d_ipccfg.c	Used to define your IPC channels	● KM0 image2 ● KM4 image2
rtl8721dlp_intfcfg	Enable/Enable low power RX	● KM0 image2
rtl8721dhp_intfcfg.c	psram configuration: ● enable psram ● enable psram calibration function ● enable psram retention ● config psram heap start address, should be 8 bytes aligned ● config psram heap size, should be 8 bytes aligned SDIO host configuration: ● SDIO Host bus speed ● SDIO Host bus width ● Card Detect pin ● Write Protection pin SDIO host configuration: ● The number of physical map pages ● The offset of flash sectors which is allocated to FTL physical map FTL parameter configuration.	● KM4 image2

13.2 boot_trustzonecfg

The whole address space will be security when boot up, SAU is used to configure non-secure area for some special address space. SAU should be configured in secure bootloader, and it can't be changed again after setting it, it is protected by hardware.

There are 8x SAU entries in Ameba-D, entry0 to entry4 has been used by system, and entry5 to entry7 are left for user use. You can configure your own TrustZone non-secure area.

```
BOOT_RAM_DATA_SECTION
const TZ_CFG_TypeDef tz_config[]=
{
//  Start                   End                     NSC
    {0x40000000,            0x50000000-1,           0}, /* entry0: Peripherals NS */
    {0x1010A000,            0x101D4000-1,           0}, /* entry1: IROM & DROM NS */
    {0x00080000,            __ram_image3_start__ -1,0}, /* entry2: KM0 SRAM, Retention SRAM, PSRAM & FLASH & SRAM NS */
    {__ram_image3_end__,    0x1007C000-1,           1}, /* entry3: NSC 4K */
    {0x100E0000,            0x10100000-1,           0}, /* entry4: BT/WIFI Extention SRAM */
    {0xFFFFFFFF,            0xFFFFFFFF,             0}, /* entry5: TODO */
    {0xFFFFFFFF,            0xFFFFFFFF,             0}, /* entry6: TODO */
    {0xFFFFFFFF,            0xFFFFFFFF,             0}, /* entry7: TODO */
    {0xFFFFFFFF,            0xFFFFFFFF,             0},
};
```

13.3 flashcfg

The user flash configuration items are listed in Table 13-2.

Table 13-2 User flash configuration

Items	Comment	Default
Flash_Speed	Indicates the flash baud rate. It can be one of the following value: ● 0xFFFF: 80MHz ● 0x7FFF: 100MHz ● 0x3FFF: 67MHz ● 0x1FFF: 57MHz ● 0x0FFF: 50MHz	0xFFFF
Flash_ReadMode	Indicates the flash read I/O mode. It can be one of the following value: ● 0xFFFF: Read quad I/O, Address & Data 4 bits mode ● 0x7FFF: Read quad O, just data 4 bits mode ● 0x3FFF: Read dual I/O, Address & Data 2 bits mode ● 0x1FFF: Read dual O, just data 2 bits mode ● 0x0FFF: 1 bit mode If the configured read mode is not supported, other modes would be searched until find the appropriate mode.	0xFFFF
Flash_AVL	Maintains the flash IC supported by SDK, refer to Fig 13-1 for detailed information.	
flash_init_userdef		

13.3.1 Flash Classification

Ameba-D support flash chips of multi-vendor, such Winbond, MXIC, Gigadevice, ESMT etc. The flash chips which have been verified on Ameba platform can be found in "Realtek Ameba Flash AVL.doc". We divide the supported flash into 6 species as Table 13-3 shows according to their characteristics and they can be used on Ameba-D directly.

Table 13-3 Flash species

Flash Classification	Manufacture	Flash ID
FlashClass1	Winbond	0xEF
	FM	0xA1
	XTX	0x0B
	XTX (FT)	0x0E

FlashClass2	GD (GD normal)	0xC8
FlashClass3	MXIC (MXIC normal)	0xC2
	Hua Hong	0x68
	GD (GD MD serial)	0x51
FlashClass4	ESMT	0x1C
FlashClass5	Micron	0x20
FlashClass6	MXIC (MXIC wide-range VCC)	0x28C2

For a new flash chip that is not available in AVL, users should follow these steps to check if it can be supported by Ameba-D.

(1) Review flash datasheet to collect essential information
- Flash ID
- Number of Flash status register and the definition (QE, BUSY, WEL, BP etc.)
- Flash Commands (Write/Read status register, Read, Read Dual/Quad Output/IO, Page Program, Sector/Block/Chip Erase, Enter/Release from Power-down etc.)
- Dummy cycle number of Read Dual/Quad Output/IO mode

(2) Check if these parameters match with those of the existing species
- a) If the flash belongs to one of the Realtek defined classes, it is already supported by SDK and you can use it directly.
- b) If all the parameters except Flash ID match with those of one species, users should add the flash ID into Flash_AVL table in SDK before use it as Fig 13-1 shows.
- c) If both the flash parameters and Flash ID mismatch with the existing species, users should define the parameters in function flash_init_userdef.

```
const FlashInfo_TypeDef Flash_AVL[] = {
        /*flash_id,     flash_id_mask,  flash_class,        status_mask     FlashInitHandler */

        /* case1: Realtek defined class, any modification is not allowed */
        {0xEF,          0x000000FF,     FlashClass1,        0x000043FC,     NULL},  /* Winbond: MANUFACTURER_ID_WINBOND */
        {0xA1,          0x000000FF,     FlashClass1,        0x0000FFFC,     NULL},  /* Fudan Micro: MANUFACTURER_ID_FM */
        {0x0B,          0x000000FF,     FlashClass1,        0x000043FC,     NULL},  /* XTX */
        {0x0E,          0x000000FF,     FlashClass1,        0x000043FC,     NULL},  /* XTX(FT) */
        {0xC8,          0x000000FF,     FlashClass2,        0x000043FC,     NULL},  /* GD normal: MANUFACTURER_ID_GD */
        {0x28C2,        0x0000FFFF,     FlashClass6,        0x000200FC,     NULL},  /* MXIC wide-range VCC: MANUFACTURER_ID_MXIC */
        {0xC2,          0x000000FF,     FlashClass3,        0x000000FC,     NULL},  /* MXIC normal: MANUFACTURER_ID_BOHONG */
        {0x68,          0x000000FF,     FlashClass3,        0x000000FC,     NULL},  /* Hua Hong */
        {0x51,          0x000000FF,     FlashClass3,        0x000000FC,     NULL},  /* GD MD serial */
        {0x1C,          0x000000FF,     FlashClass4,        0x000000FC,     NULL},  /* ESMT: MANUFACTURER_ID_EON */
        {0x20,          0x000000FF,     FlashClass5,        0x000000FC,     NULL},  /* Microm: MANUFACTURER_ID_MICRON */

        /* case2: new flash, ID is not included in case1 list, but specification is compatible with FlashClass1~FlashClass6 */
        //{0xXX,          0x0000XXXX,     FlashClassX,        0x0000XXXX,     NULL},

        /* case3: new flash, ID is not included in case1 list, and specification is not compatible with FlashClass1~FlashClass6 */
        {0x00,          0x000000FF,     FlashClassUser,     0xFFFFFFFF,     &flash_init_userdef},

        /* End */
        {0xFF,          0xFFFFFFFF,     FlashClassNone,     0xFFFFFFFF,     NULL},
};
```

Fig 13-1 Flash_AVL

13.3.1.1 Dummy Cycles

When reading flash, host sends command and address to flash. After that host needs to send dummy cycles to flash before it sends data back. The number of dummy cycles is various for different read modes.

The following section shows you how to find dummy cycles from flash datasheet.

Taking Winbond W25Q16JV for an example, there are 6 dummy cycles in Read Quad I/O mode as Fig 13-2 shows.

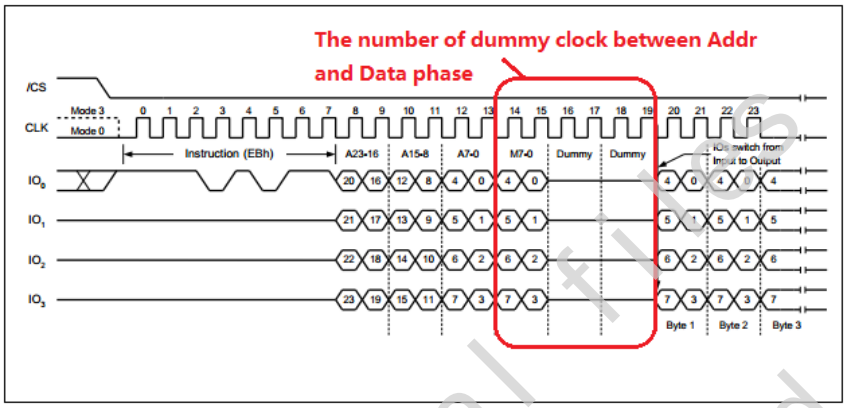


Fig 13-2 Fast read quad I/O instruction sequence

In Read Dual I/O mode, there are 4 dummy cycles. The fast read dual I/O instruction sequence is shown in Fig 13-3.

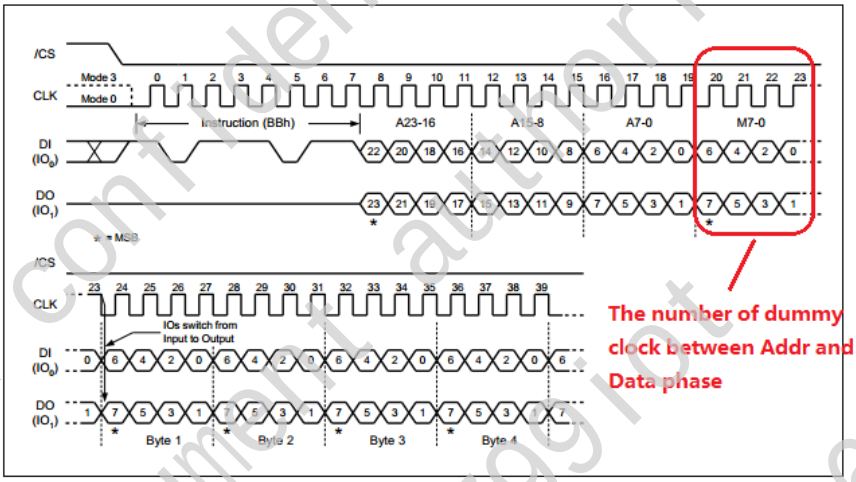


Fig 13-3 Fast read dual I/O instruction sequence

Users can find the dummy cycle of Read Quad Output and Read Dual Output mode in the same way.

13.3.1.2 Other Parameters

The parameters comparison of these species are listed in Table 13-4.

Table 13-4 Dummy cycles with different read mode

Items	Parameters	FlashClass 1	FlashClass 2	FlashClass 3	FlashClass 4	FlashClass 5	FlashClass 6
Status Register	Status Register Number	2	2 or 3	1	1	1	1
	QE bit	Bit[9]	Bit[9]	Bit[6]	No exist	No exist	Bit[6]
	BUSY bit	Bit[0]					
	WEL bit	Bit[1]					
Dummy Cycle	READ	0	0	0	0	0	0
	DREAD (1I/2O Read)	8	8	8	8	3	8
	2READ (2*I/O Read)	4	4	4	4	5	4
	QREAD (1I/4O Read)	8	8	8	8	5	8
	4READ (4*I/O Read)	6	6	6	6	9	6
Command	Write Enable	0x06	0x06	0x06	0x06	0x06	0x06
	Read JEDEC ID	0x9F	0x9F	0x9F	0x9F	0x9F	0x9F

	Read Status Register 1	0x05	0x05	0x05	0x05	0x05	0x05
	Read Status Register 2	0x35	0x35	-	-	-	-
	Write Status Register 1	0x01	0x01	0x01	0x01	0x01	0x01
	Write Status Register 2[1]	-	-/0x31	-	-	-	-
	Chip Erase	0x60	0x60	0x60	0x60	0xC7	0x60
	Block Erase	0xD8	0xD8	0xD8	0xD8	0xD8	0xD8
	Sector Erase	0x20	0x20	0x20	0x20	0x20	0x20
	Power-down	0xB9	0xB9	0xB9	0xB9	0xB9	0xB9
	Release from Power-down	0xAB	0xAB	0xAB	0xAB	0xAB	0xAB
	READ	0x03	0x03	0x03	0x03	0x03	0x03
	DREAD (1I/2O Read)	0x3B	0x3B	0x3B	0x3B	0x3B	0x3B
	2READ (2*I/O Read)	0xBB	0xBB	0xBB	0xBB	0xBB	0xBB
	QREAD (1I/4O Read)	0x6B	0x6B	0x6B	0x6B	0x6B	0x6B
	4READ (4*I/O Read)	0xEB	0xEB	0xEB	0xEB	0xEB	0xEB
	Read Configuration Register[2]	-	-	-	-	-	0x15
	Write Configuration Register[3]	-	-	-	-	-	-

Note 1:

- For FlashClass1, the Write Status Register 2 is written together with the Write Status Register 1 using 0x01 as Fig 13-4 shows.
- For FlashClass2, when GD density is less than 2MB, the Write Status Register 2 is written together with the Write Status Register 1 using 0x01 as Fig 13-4 shows. When GD density is larger than 2MB, the Write Status Register 2 is written individually using 0x31 as Fig 13-5 shows.

Note 2: For MXIC wide-range VCC, the Read Configuration Register would be checked if chip is in High performance mode by default once power-on as Fig 13-6 shows. If not, we would switch it to High performance mode.

Note 3: For MXIC wide-range VCC, the Write Configuration Register is written together with Write Status Register using 0x01 as Fig 13-7 shows.

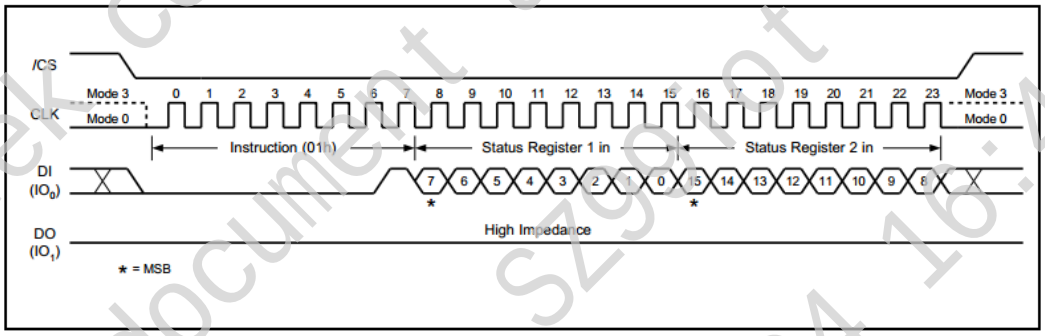


Fig 13-4 Write Status Register 1 & 2 sequence (FlashClass1, FlashClass2 when density < 2M)

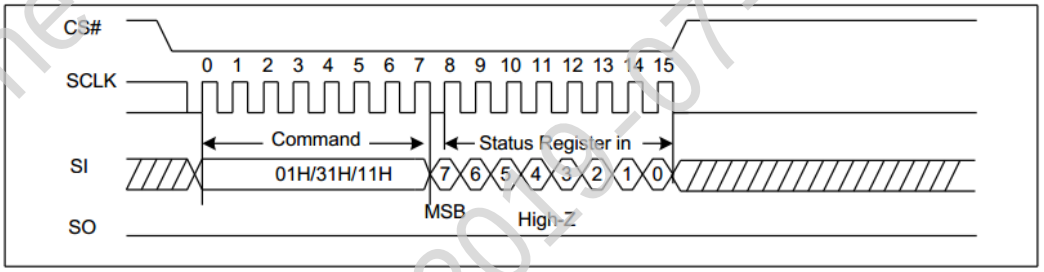


Fig 13-5 Write Status Register 1/2 sequence (FlashClass2 when density > 2MB)

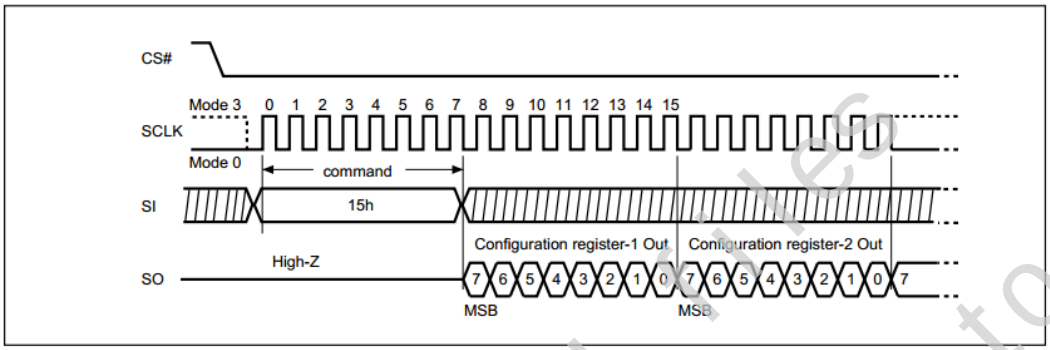


Fig 13-6 Read Configuration Register sequence (FlashClass6)

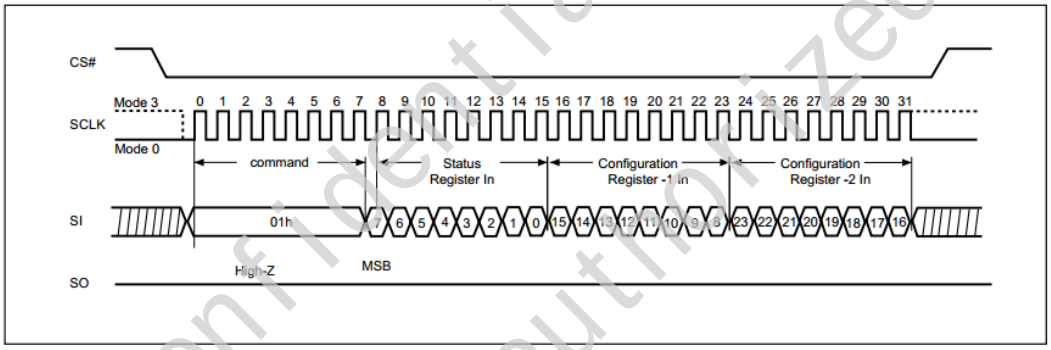


Fig 13-7 Write Status Register and Write Configuration Register sequence (FlashClass6)

13.3.2 FlashClass1

Table 13-5 lists the parameters for FlashClass1 which can't be modified by users.

Table 13-5 FlashClass1 parameters

Command List
FLASH_InitStruct->FLASH_Id = FLASH_ID_WINBOND; /*1.1 status bit define */ FLASH_InitStruct->FLASH_QuadEn_bit = BIT(9); FLASH_InitStruct->FLASH_Busy_bit = BIT(0); FLASH_InitStruct->FLASH_WLE_bit = BIT(1); FLASH_InitStruct->FLASH_Status2_exist = 1; /*1.2 dummy cycle */ FLASH_InitStruct->FLASH_rd_dummy_cyle[0] = 0; FLASH_InitStruct->FLASH_rd_dummy_cyle[1] = 0x04; FLASH_InitStruct->FLASH_rd_dummy_cyle[2] = 0x06; /*1.3 set 2 bits mode command */ FLASH_InitStruct->FLASH_rd_dual_io = 0xBB; FLASH_InitStruct->FLASH_rd_dual_o = 0x3B; /*1.4 set 4 bits mode command */ FLASH_InitStruct->FLASH_rd_quad_io = 0xEB; FLASH_InitStruct->FLASH_rd_quad_o = 0x6B; /*1.5 other flash command set */

```
FLASH_InitStruct->FLASH_cmd_wr_en = 0x06;
FLASH_InitStruct->FLASH_cmd_rd_id = 0x9F;
FLASH_InitStruct->FLASH_cmd_rd_status = 0x05;
FLASH_InitStruct->FLASH_cmd_rd_status2 = 0x35;    //FLASH_CMD_RDSR2;
FLASH_InitStruct->FLASH_cmd_wr_status = 0x01;
FLASH_InitStruct->FLASH_cmd_wr_status2 = 0;
FLASH_InitStruct->FLASH_cmd_chip_e = 0x60;
FLASH_InitStruct->FLASH_cmd_block_e = 0xD8;
FLASH_InitStruct->FLASH_cmd_sector_e = 0x20;
FLASH_InitStruct->FLASH_cmd_pwdn_release = 0xAB;
FLASH_InitStruct->FLASH_cmd_pwdn = 0xB9;
```

13.3.3 FlashClass2

Table 13-6 lists the parameters for FlashClass2 which can't be modified by users. FLASH_cmd_wr_status2 would be set when density is larger than 2MB in SDK.

Table 13-6 FlashClass2 parameters

Command List

```
FLASH_InitStruct->FLASH_Id = FLASH_ID_GD;

/*1.1 status bit define */
FLASH_InitStruct->FLASH_QuadEn_bit = BIT(9);
FLASH_InitStruct->FLASH_Busy_bit = BIT(0);    //WIP
FLASH_InitStruct->FLASH_WLE_bit = BIT(1);    //WLE
FLASH_InitStruct->FLASH_Status2_exist = 1;

/*1.2 dummy cycle */
FLASH_InitStruct->FLASH_rd_dummy_cyle[0] = 0;
FLASH_InitStruct->FLASH_rd_dummy_cyle[1] = 0x04;    //(M7-4)2 + 2 dummy
FLASH_InitStruct->FLASH_rd_dummy_cyle[2] = 0x06;    //(M7-0)2 + 4 dummy;

/*1.3 set 2 bits mode command */
FLASH_InitStruct->FLASH_rd_dual_io = 0xBB;
FLASH_InitStruct->FLASH_rd_dual_o = 0x3B;

/*1.4 set 4 bits mode command */
FLASH_InitStruct->FLASH_rd_quad_io = 0xEB;
FLASH_InitStruct->FLASH_rd_quad_o = 0x6B;

/*1.5 other flash command set */
FLASH_InitStruct->FLASH_cmd_wr_en = 0x06;
FLASH_InitStruct->FLASH_cmd_rd_id = 0x9F;
FLASH_InitStruct->FLASH_cmd_rd_status = 0x05;
FLASH_InitStruct->FLASH_cmd_rd_status2 = 0x35;
FLASH_InitStruct->FLASH_cmd_wr_status = 0x01;
FLASH_InitStruct->FLASH_cmd_wr_status2 = 0;
FLASH_InitStruct->FLASH_cmd_chip_e = 0x60;
FLASH_InitStruct->FLASH_cmd_block_e = 0xD8;
FLASH_InitStruct->FLASH_cmd_sector_e = 0x20;
FLASH_InitStruct->FLASH_cmd_pwdn_release = 0xAB;
FLASH_InitStruct->FLASH_cmd_pwdn = 0xB9;
```

13.3.4 FlashClass3

Table 13-7 lists the parameters for FlashClass3 which can't be modified by users.

Table 13-7 FlashClass3 parameters

Command List

FLASH_InitStruct->FLASH_Id = FLASH_ID_MXIC;
 	
/*1.1 status bit define */
FLASH_InitStruct->FLASH_QuadEn_bit = BIT(6);
FLASH_InitStruct->FLASH_Busy_bit = BIT(0); //WIP
FLASH_InitStruct->FLASH_WLE_bit = BIT(1); //WLE
FLASH_InitStruct->FLASH_Status2_exist = 0;
 	
/*1.2 dummy cycle */
FLASH_InitStruct->FLASH_rd_dummy_cyle[0] = 0;
FLASH_InitStruct->FLASH_rd_dummy_cyle[1] = 0x04;
FLASH_InitStruct->FLASH_rd_dummy_cyle[2] = 0x06;
 	
/*1.3 set 2 bits mode command */
FLASH_InitStruct->FLASH_rd_dual_io = 0xBB;
FLASH_InitStruct->FLASH_rd_dual_o = 0x3B;
 	
/*1.4 set 4 bits mode command */
FLASH_InitStruct->FLASH_rd_quad_io = 0xEB;
FLASH_InitStruct->FLASH_rd_quad_o = 0x6B;
 	
/*1.5 other flash command set */
FLASH_InitStruct->FLASH_cmd_wr_en = 0x06;
FLASH_InitStruct->FLASH_cmd_rd_id = 0x9F;
FLASH_InitStruct->FLASH_cmd_rd_status = 0x05;
FLASH_InitStruct->FLASH_cmd_rd_status2 = 0;
FLASH_InitStruct->FLASH_cmd_wr_status = 0x01;
FLASH_InitStruct->FLASH_cmd_wr_status2 = 0;
FLASH_InitStruct->FLASH_cmd_chip_e = 0x60;
FLASH_InitStruct->FLASH_cmd_block_e = 0xD8;
FLASH_InitStruct->FLASH_cmd_sector_e = 0x20;
FLASH_InitStruct->FLASH_cmd_pwdn_release = 0xAB;
FLASH_InitStruct->FLASH_cmd_pwdn = 0xB9;

13.3.5 FlashClass4

Table 13-8 lists the parameters for FlashClass4 which can't be modified by users.

Table 13-8 FlashClass4 parameters

Command List

FLASH_InitStruct->FLASH_Id = FLASH_ID_MXIC;
 	
/*1.1 status bit define */
FLASH_InitStruct->FLASH_QuadEn_bit = 0;
FLASH_InitStruct->FLASH_Busy_bit = BIT(0); //WIP
FLASH_InitStruct->FLASH_WLE_bit = BIT(1); //WLE
FLASH_InitStruct->FLASH_Status2_exist = 0;
 	
/*1.2 dummy cycle */
FLASH_InitStruct->FLASH_rd_dummy_cyle[0] = 0;
FLASH_InitStruct->FLASH_rd_dummy_cyle[1] = 0x04;
FLASH_InitStruct->FLASH_rd_dummy_cyle[2] = 0x06;
 	
/*1.3 set 2 bits mode command */
FLASH_InitStruct->FLASH_rd_dual_io = 0xBB;

```
FLASH_InitStruct->FLASH_rd_dual_o = 0x3B;

/*1.4 set 4 bits mode command */
FLASH_InitStruct->FLASH_rd_quad_io = 0xEB;
FLASH_InitStruct->FLASH_rd_quad_o = 0x6B;

/*1.5 other flash command set */
FLASH_InitStruct->FLASH_cmd_wr_en = 0x06;
FLASH_InitStruct->FLASH_cmd_rd_id = 0x9F;
FLASH_InitStruct->FLASH_cmd_rd_status = 0x05;
FLASH_InitStruct->FLASH_cmd_rd_status2 = 0;
FLASH_InitStruct->FLASH_cmd_wr_status = 0x01;
FLASH_InitStruct->FLASH_cmd_wr_status2 = 0;
FLASH_InitStruct->FLASH_cmd_chip_e = 0x60;
FLASH_InitStruct->FLASH_cmd_block_e = 0xD8;
FLASH_InitStruct->FLASH_cmd_sector_e = 0x20;
FLASH_InitStruct->FLASH_cmd_pwdn_release = 0xAB;
FLASH_InitStruct->FLASH_cmd_pwdn = 0xB9;
```

13.3.6 FlashClass5

Table 13-9 lists the parameters for FlashClass5 which can't be modified by users.

Table 13-9 FlashClass5 parameters

Command List

```
FLASH_InitStruct->FLASH_id = FLASH_ID_MICRON;

/*1.1 status bit define */
FLASH_InitStruct->FLASH_QuadEn_bit = 0;
FLASH_InitStruct->FLASH_Busy_bit = BIT(0);
FLASH_InitStruct->FLASH_WLE_bit = BIT(1);
FLASH_InitStruct->FLASH_Status2_exist = 0;

/*1.2 dummy cycle */
FLASH_InitStruct->FLASH_rd_dummy_cyle[0] = 0;
/* set Micron rd_dummy_cyle based on baud rate, default 100MHz */
FLASH_InitStruct->FLASH_rd_dummy_cyle[1] = 0x05;
FLASH_InitStruct->FLASH_rd_dummy_cyle[2] = 0x09;

/*1.3 set 2 bits mode command */
FLASH_InitStruct->FLASH_rd_dual_io = 0xBB;
FLASH_InitStruct->FLASH_rd_dual_o = 0x3B;

/*1.4 set 4 bits mode command */
FLASH_InitStruct->FLASH_rd_quad_io = 0xEB;
FLASH_InitStruct->FLASH_rd_quad_o = 0x6B;

/*1.5 other flash command set */
FLASH_InitStruct->FLASH_cmd_wr_en = 0x06;
FLASH_InitStruct->FLASH_cmd_rd_id = 0x9F;
FLASH_InitStruct->FLASH_cmd_rd_status = 0x05;
FLASH_InitStruct->FLASH_cmd_rd_status2 = 0;
FLASH_InitStruct->FLASH_cmd_wr_status = 0x01;
FLASH_InitStruct->FLASH_cmd_wr_status2 = 0;
FLASH_InitStruct->FLASH_cmd_chip_e = 0xC7;
FLASH_InitStruct->FLASH_cmd_block_e = 0xD8;
FLASH_InitStruct->FLASH_cmd_sector_e = 0x20;
```

```
FLASH_InitStruct->FLASH_cmd_pwdn_release = 0xAB;
FLASH_InitStruct->FLASH_cmd_pwdn = 0xB9;
```

13.3.7 FlashClass6

Table 13-10 lists the parameters for FlashClass6 which can't be modified by users.

Table 13-10 FlashClass6 parameters

Command List

```
FLASH_InitStruct->FLASH_Id = FLASH_ID_MXIC;

/*1.1 status bit define */
FLASH_InitStruct->FLASH_QuadEn_bit = BIT(6);
FLASH_InitStruct->FLASH_Busy_bit = BIT(0);    //WIP
FLASH_InitStruct->FLASH_WLE_bit = BIT(1);    //WLE
FLASH_InitStruct->FLASH_Status2_exist = 0;

/*1.2 dummy cycle */
FLASH_InitStruct->FLASH_rd_dummy_cyle[0] = 0;
FLASH_InitStruct->FLASH_rd_dummy_cyle[1] = 0x04;
FLASH_InitStruct->FLASH_rd_dummy_cyle[2] = 0x06;

/*1.3 set 2 bits mode command */
FLASH_InitStruct->FLASH_rd_dual_io = 0xBB;
FLASH_InitStruct->FLASH_rd_dual_o = 0x3B;

/*1.4 set 4 bits mode command */
FLASH_InitStruct->FLASH_rd_quad_io = 0xEB;
FLASH_InitStruct->FLASH_rd_quad_o = 0x6B;

/*1.5 other flash command set */
FLASH_InitStruct->FLASH_cmd_wr_en = 0x06;
FLASH_InitStruct->FLASH_cmd_rd_id = 0x9F;
FLASH_InitStruct->FLASH_cmd_rd_status = 0x05;
FLASH_InitStruct->FLASH_cmd_rd_status2 = 0;
FLASH_InitStruct->FLASH_cmd_wr_status = 0x01;
FLASH_InitStruct->FLASH_cmd_wr_status2 = 0;
FLASH_InitStruct->FLASH_cmd_chip_e = 0x60;
FLASH_InitStruct->FLASH_cmd_block_e = 0xD8;
FLASH_InitStruct->FLASH_cmd_sector_e = 0x20;
FLASH_InitStruct->FLASH_cmd_pwdn_release = 0xAB;
FLASH_InitStruct->FLASH_cmd_pwdn = 0xB9;
```

13.4 pinmapcfg

It needs doing I/O pull control when enter deep sleep and sleep mode. Otherwise it results power leakage. For example, UART voltage level is high. If we pull down UART pin or not pull, then power leakage happens. So we need make sure each pin has proper pull control.
In SDK you should set GPIO function pull control and sleep pull control in pmap_func based on your PCB board, and SDK will set pull control based on your setting between suspend and resume.

```
const PMAP_TypeDef pmap_func[]=
{
//      Pin Name        Func PU/PD          Slp PU/PD           DSlp PU/PD          LowPowerPin
    {_PA_0,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_1,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_2,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_3,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_4,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_5,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_6,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_7,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //UART_LOG_TXD
    {_PA_8,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //UART_LOG_RXD
    {_PA_9,     GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_10,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_11,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_12,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_13,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_14,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_15,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE},//keyscan
    {_PA_16,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_17,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_18,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_19,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_20,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_21,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_22,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_23,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_24,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_25,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_26,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     TRUE}, //keyscan
    {_PA_27,        GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SWD_DATA or normal_m
    {_PA_28,        GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_29,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_30,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PA_31,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_0,     GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_1,     GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_2,     GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_3,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SWD_CLK
    {_PB_4,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch0
    {_PB_5,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch1
    {_PB_6,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch2
    //{_PB_6,       GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE},
    {_PB_7,     GPIO_PuPd_DOWN,     GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //Touch3
    {_PB_8,     GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_9,     GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_10,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_11,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_12,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA3
    {_PB_13,        GPIO_PuPd_UP,       GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     FALSE}, //SPI_CLK
    {_PB_14,        GPIO_PuPd_UP,       GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA0
    {_PB_15,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA2
    {_PB_16,        GPIO_PuPd_UP,       GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //SPI_CS
    {_PB_17,        GPIO_PuPd_UP,       GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     FALSE}, //SPI_DATA1
    {_PB_18,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_19,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_20,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_21,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_22,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //IR pin should no-pull
    //{_PB_22,      GPIO_PuPd_NOPULL,   GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE},
    {_PB_23,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_24,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_25,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_26,        GPIO_PuPd_SHUTDOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, // this is a pcb bo
    {_PB_27,        GPIO_PuPd_UP,   GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_28,        GPIO_PuPd_UP,   GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_29,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_30,        GPIO_PuPd_UP,   GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PB_31,        GPIO_PuPd_DOWN, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //
    {_PNC,      GPIO_PuPd_KEEP, GPIO_PuPd_KEEP,     GPIO_PuPd_KEEP,     FALSE}, //table end
};
```

13.5 sleepcfg

Keep default settings, you can only change these settings on Realtek RD's help.

13.6 bootcfg

Boot loader can be configured through this file:

```
/*
* @brief     MMU Configuration.
*     There are 8 MMU entries totally. Entry 0 & Entry 1 are already used by OTA, Entry 2~7 can be used by Users.
*/
BOOT_RAM_DATA_SECTION
MMU_ConfDef Flash_MMU_Config[] = {
    /*VAddrStart,   VAddrEnd,           PAddrStart,         PAddrEnd*/
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},    //Entry 2
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},    //Entry 3
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},    //Entry 4
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},    //Entry 5
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},    //Entry 6
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},    //Entry 7
    {0xFFFFFFFF,    0xFFFFFFFF,         0xFFFFFFFF,     0xFFFFFFFF},
};

/*
* @brief     OTA start address. Because KM0 & KM4 IMG2 are combined, users only need to set the start address
*     of KM0 IMG2.
*/
BOOT_RAM_DATA_SECTION
u32 OTA_Region[2] = {
    0x08006000,         /* OTA1 region start address */
    0x08106000,         /* OTA2 region start address */
};

/*
* @brief     RSIP Mask Configuration.
*     There are 4 RSIP mask entries totally. Entry 0 is already used by System Data, Entry 3 is reserved by Realtek.
*     Only Entry 1 & Entry 2 can be used by Users.
*     MaskAddr: start address for RSIP Mask, should be 4KB aligned
*     MaskSize: size of the mask area, unit is 4KB
*/
BOOT_RAM_DATA_SECTION
RSIP_MaskDef RSIP_Mask_Config[] = {
    /*MaskAddr,         MaskSize*/
    {0x08001000,    3},         //Entry 0: 4K system data & 4K backup & OPK

    /* customer can set here */
    {0xFFFFFFFF,    0xFFFF},    //Entry 1: can be used by users
    {0xFFFFFFFF,    0xFFFF},    //Entry 2: can be used by users
    {0xFFFFFFFF,    0xFFFF},    //Entry 3: Reserved by Realtek. If RDP is not used, this entry can be used by users.

    /* End */
    {0xFFFFFFFF,    0xFFFF},
};
/**
* @brief  GPIO force OTA1 as image2. 0xFF means force OTA1 trigger is disabled.
*     BIT[7]: active level, 0 is low level active, 1 is high level active,
*     BIT[6:5]: port, 0 is PORT_A, 1 is PORT_B
*     BIT[4:0]: pin num 0~31
*/
BOOT_RAM_DATA_SECTION
u8 Force_OTA1_GPIO = 0xFF;

/**
* @brief  boot log enable or disable.
*     FALSE: disable
*     TRUE: enable
*/
BOOT_RAM_DATA_SECTION
u8 Boot_Log_En = FALSE;


/**
* @brief  Firmware verify callback handler.
*     If users need to verify checksum/ hash for image2, implement the function and assign the address
*     to this function pointer.
* @param  None
* @retval 1: Firmware verify successfully, 0: verify failed
*/
BOOT_RAM_DATA_SECTION
FuncPtr FwCheckCallback = NULL;

/**
* @brief  Firmware select hook.
*     If users need to implement own OTA select method, implement the function and assign the address
*     to this function pointer.
* @param  None
* @retval 0: both firmwares are invalid, select none, 1: boot from OTA1, 2: boot from OTA2
*/
BOOT_RAM_DATA_SECTION
FuncPtr OTASelectHook = NULL;
```

Table 13-11 User bootcfg

Items	Configuration Items
Flash_MMU_Config	Used for MMU configure OTA2 address mapping should be settled here
RSIP_Mask_Config	RSIP mask configure
Force_OTA1_GPIO	Force OTA1 GPIO configure
Boot_Log_En	Used to disable Reaktek boot log
FwCheckCallback	Firmware verify callback handler
OTASelectHook	Firmware select hook
OTA_Region	OTA start address

13.7 ipccfg

You can add IPC entry in the following table, so that KM4 & KM0 can send message to each other by the method you defined.
For USER_MSG_TYPE:

- IPC_USER_POINT: Message in IRQFUNC is a pointer (address).
- IPC_USER_DATA: Message in IRQFUNC is just a value.

```
#if defined (ARM_CORE_CM4)
const IPC_INIT_TABLE ipc_init_config[] =
{
    // USER_MSG_TYPE        IRQFUNC                         IRQDATA
    {IPC_USER_DATA,         shell_switch_ipc_int,       (VOID*) NULL},//channel 0: IPC_INT_CHAN_SHELL_SWITCH
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 1: IPC_INT_CHAN_WIFI_FW
    {IPC_USER_DATA,         FLASH_Write_IPC_Int,        (VOID*) NULL},//channel 2: IPC_INT_CHAN_FLASHPG_REQ
    {IPC_USER_POINT, NULL,                              (VOID*) NULL},//channel 3: IPC_INT_KM4_TICKLESS_INDICATION
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 4: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 5: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 6: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 7: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 8: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 9: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 10: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 11: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 12: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 13: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 14: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 15: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 16: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 17: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 18: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 19: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 20: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 21: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 22: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 23: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 24: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 25: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 26: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 27: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 28: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 29: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 30: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 31: Reserved for Customer use

    {0xFFFFFFFF,            OFF,                            OFF},     /* Table end */
};
#else
#if CONFIG_WIFI_FW_EN
extern void driver_fw_flow_ipc_int(VOID *Data, u32 IrqStatus, u32 ChanNum);
#define fw_flow_ipc_int driver_fw_flow_ipc_int
#else
#define fw_flow_ipc_int NULL
#endif
const IPC_INIT_TABLE ipc_init_config[] =
{
    // USER_MSG_TYPE        IRQFUNC                         IRQDATA
    {IPC_USER_DATA,         shell_switch_ipc_int,       (VOID*) NULL},//channel 0: IPC_INT_CHAN_SHELL_SWITCH
    {IPC_USER_DATA,         fw_flow_ipc_int,                (VOID*) IPCM4_DEV},//channel 1: IPC_INT_CHAN_WIFI_FW
    {IPC_USER_DATA,         FLASH_Write_IPC_Int,        (VOID*) NULL},//channel 2: IPC_INT_CHAN_FLASHPG_REQ
    {IPC_USER_POINT, km4_tickless_ipc_int,          (VOID*) NULL},//channel 3: IPC_INT_KM4_TICKLESS_INDICATION
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 4: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 5: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 6: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 7: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 8: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 9: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 10: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 11: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 12: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 13: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 14: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 15: Reserved for Realtek use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 16: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 17: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 18: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 19: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 20: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 21: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 22: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 23: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 24: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 25: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 26: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 27: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 28: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 29: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 30: Reserved for Customer use
    {IPC_USER_DATA,         NULL,                           (VOID*) NULL},//channel 31: Reserved for Customer use

    {0xFFFFFFFF,            OFF,                            OFF},     /* Table end */
};
#endif
```

Note: Please use IPC channel 16~31 and IPC semaphore index 8~15, Channel 0~15 and semaphore index 0~7 are reserved for Realtek use.

13.8 lpintcfg

UART low power RX enable/disable can be configured through this file:

```
UARTCFG_TypeDef uart_config[2]=
{
    /* UART0 */
    {
        .LOW_POWER_RX_ENABLE = DISABLE, /* Enable low power RX*/
    },
    /* UART1 */
    {
        .LOW_POWER_RX_ENABLE = DISABLE,
    },
};
```

13.9 hpintcfg

PSRAM parameters, SDIO host parameters and FTL parameters can be configured through this file:

```
PSRAMCFG_TypeDef psram_dev_config = {
    .psram_dev_enable = FALSE,                  //enable psram
    .psram_dev_cal_enable = FALSE,              //enable psram calibration function
    .psram_dev_retention = FALSE,               //enable psram retention
    .psram_heap_start_address = 0x02000400,     //config psram heap start address, should be 8 bytes aligned
    .psram_heap_size = 0x200000,                //config psram heap size, should be 8 bytes aligned
};

SDIOHCFG_TypeDef sdioh_config = {
    .sdioh_bus_speed = SD_SPEED_HS,             //SD_SPEED_DS or SD_SPEED_HS
    .sdioh_bus_width = SDIOH_BUS_WIDTH_4BIT,    //SDIOH_BUS_WIDTH_1BIT or SDIOH_BUS_WIDTH_4BIT
    .sdioh_cd_pin = _PB_25,                     //_PB_25/_PA_6/_PNC
    .sdioh_wp_pin = _PNC,                       //_PB_24/_PA_5/_PNC
};

#if defined(CONFIG_FTL_ENABLED)
#define FTL_MEM_CUSTEM          0
#if FTL_MEM_CUSTEM == 0
#error "You should allocate flash sectors to for FTL physical map as following, then set FTL_MEM_CUSTEM to 1. For more information, Please refer to Application Note, FTL chapter."
#else
const u8 ftl_phy_page_num = 3;                  /* The number of physical map pages, default is 3*/
const u32 ftl_phy_page_addr[] = {0x000FD000, 0x000FE000, 0x000FF000}; /* The offset of flash sectors which is allocated to FTL physical map.
                                                                         Users should modify it according to their own memory layout. */
#endif
#endif
```

14 Hardware Crypto Engine

14.1 Introduction

Hardware Crypto Engine is used to authenticate, encrypt and decrypt packets. It can support the following Hash and Cipher functions:

- Hash (include HMAC): MD5/SHA1/SHA2 (224, 256), Poly1305
- Cipher: AES (ECB/CBC/CFB/OFB/CTR/GMAC/GHASH/GCM), 3DES (ECB/CBC/CFB/OFB/CTR), DES(ECB/CBC/CFB/OFB/CTR), ChaCha20, Chacha20_poly1305

Hardware Crypto Engine also support sequential hash for long plaintext length.

The following section illustrates the Hardware Crypto Engine APIs and how to use these APIs.

14.2 Hardware Crypto Engine APIs

14.2.1 Crypto Engine Initialize API

14.2.1.1 rtl_cryptoEngine_init

Items	Description
Introduction	Hardware crypto engine initialization
Parameters	N/A
Return	Status value: ● SUCCESS: initialization ok

14.2.2 Hash APIs

14.2.2.1 rtl_crypto_md5

Items	Description
Introduction	MD5 calculation digest
Parameters	● message: Plaintext ● msglen: Plaintext length ● pDigest: Result of MD5 function
Return	Status value: ● SUCCESS: MD5 ok ● Others: fail, refer to ERRNO

14.2.2.2 rtl_crypto_md5_init

Items	Description
Introduction	MD5 initialization (used for sequential hash)
Parameters	N/A
Return	Status value: ● SUCCESS: MD5 initialization ok ● Others: fail, refer to ERRNO

14.2.2.3 rtl_crypto_md5_update

Items	Description
Introduction	MD5 block calculation (used for sequential hash)
Parameters	● message: Plaintext ● msglen: Plaintext length
Return	Status value: ● SUCCESS: MD5 update ok ● Others: fail, refer to ERRNO

14.2.2.4 rtl_crypto_md5_ final

Items	Description
Introduction	MD5 last block calculation (used for sequential hash)
Parameters	pDigest: Result of MD5 function
Return	Status value: ● SUCCESS: MD5 final ok ● Others: fail, refer to ERRNO

14.2.2.5 rtl_crypto_sha1

Items	Description
Introduction	SHA1 calculation digest
Parameters	● message: Plaintext ● msglen: Plaintext length ● pDigest: Result of SHA1 function
Return	Status value: ● SUCCESS: SHA1 ok ● Others: fail, refer to ERRNO

14.2.2.6 rtl_crypto_sha1_init

Items	Description
Introduction	SHA1 initialization (used for sequential hash)
Parameters	N/A
Return	Status value: ● SUCCESS: SHA1 initialization ok ● Others: fail, refer to ERRNO

14.2.2.7 rtl_crypto_sha1_update

Items	Description
Introduction	SHA1 block calculation (used for sequential hash)
Parameters	● message: Plaintext ● msglen: Plaintext length
Return	Status value: ● SUCCESS: SHA1 update ok ● Others: fail, refer to ERRNO

14.2.2.8 rtl_crypto_sha1_ final

Items	Description

Introduction	SHA1 last block calculation (used for sequential hash)
Parameters	pDigest: Result of SHA1 function
Return	Status value: ● SUCCESS: SHA1 final ok ● Others: fail, refer to ERRNO

14.2.2.9 rtl_crypto_sha2

Items	Description
Introduction	SHA2 calculation digest
Parameters	● sha2type: SHA2 type, SHA2_224 or SHA2_256 ● message: Plaintext ● msglen: Plaintext length ● pDigest: Result of SHA2 function
Return	Status value: ● SUCCESS: SHA2 ok ● Others: fail, refer to ERRNO

14.2.2.10 rtl_crypto_sha2_init

Items	Description
Introduction	SHA2 initialization (used for sequential hash)
Parameters	sha2type: SHA2 type, SHA2_224 or SHA2_256
Return	Status value: ● SUCCESS: SHA2 initialize ok ● Others: fail, refer to ERRNO

14.2.2.11 rtl_crypto_sha2_update

Items	Description
Introduction	SHA2 block calculation (used for sequential hash)
Parameters	● message: Plaintext ● msglen: Plaintext length
Return	Status value: ● SUCCESS: SHA2 update ok ● Others: fail, refer to ERRNO

14.2.2.12 rtl_crypto_sha2_ final

Items	Description
Introduction	SHA2 last block calculation (used for sequential hash)
Parameters	pDigest: Result of SHA2 function
Return	Status value: ● SUCCESS: SHA2 final ok ● Others: fail, refer to ERRNO

14.2.2.13 rtl_crypto_hmac_md5

Items	Description
Introduction	HMAC-MD5 calculation digest
Parameters	● key: HMAC-MD5 key, need to be 4-byte alignment ● keylen: key length ● message: Plaintext

	● msglen: Plaintext length ● pDigest: Result of HMAC-MD5 function
Return	Status value: ● SUCCESS: HMAC-MD5 ok ● Others: fail, refer to ERRNO

14.2.2.14 rtl_crypto_hamc_md5_init

Items	Description
Introduction	HMAC-MD5 initialization (used for sequential hash)
Parameters	● key: HMAC-MD5 key, need to be 4-byte alignment ● keylen: key length
Return	Status value: ● SUCCESS: HMAC-MD5 initialization ok ● Others: fail, refer to ERRNO

14.2.2.15 rtl_crypto_hmac_md5_update

Items	Description
Introduction	HMAC-MD5 block calculation (used for sequential hash)
Parameters	● message: Plaintext ● msglen: Plaintext length
Return	retval status value: ● SUCCESS: HMAC-MD5 update ok ● Others: fail, refer to ERRNO

14.2.2.16 rtl_crypto_hmac_md5_ final

Items	Description
Introduction	HMAC-MD5 last block calculation (used for sequential hash)
Parameters	pDigest: Result of HMAC-MD5 function
Return	Status value: ● SUCCESS: HMAC-MD5 final ok ● Others: fail, refer to ERRNO

14.2.2.17 rtl_crypto_hmac_sha1

Items	Description
Introduction	HMAC-SHA1 calculation digest
Parameters	● key: HMAC-SHA1 key, need to be 4-byte alignment ● keylen: key length ● message: Plaintext ● msglen: Plaintext length ● pDigest: Result of HMAC-SHA1 function
Return	Status value: ● SUCCESS: HMAC-SHA1 ok ● Others: fail, refer to ERRNO

14.2.2.18 rtl_crypto_hmac_sha1_init

Items	Description
Introduction	HMAC-SHA1 initialization (used for sequential hash)
Parameters	● key: HMAC-SHA1 key, need to be 4-byte alignment

	● keylen: key length
Return	Status value: ● SUCCESS: HMAC-SHA1 initialization ok ● Others: fail, refer to ERRNO

14.2.2.19 rtl_crypto_hmac_sha1_update

Items	Description
Introduction	HMAC-SHA1 block calculation (used for sequential hash)
Parameters	● message: Plaintext ● msglen: Plaintext length
Return	Status value: ● SUCCESS: HMAC-SHA1 update ok ● Others: fail, refer to ERRNO

14.2.2.20 rtl_crypto_hmac_sha1_ final

Items	Description
Introduction	HMAC-SHA1 last block calculation (used for sequential hash)
Parameters	pDigest: Result of HMAC-SHA1 function
Return	Status value: ● SUCCESS: HMAC-SHA1 final ok ● Others: fail, refer to ERRNO

14.2.2.21 rtl_crypto_hmac_sha2

Items	Description
Introduction	HMAC-SHA2 calculation digest
Parameters	● sha2type: SHA2 type, SHA2_224 or SHA2_256 ● key: HMAC-SHA2 key, need to be 4-byte alignment ● keylen: Key length ● message: Plaintext ● msglen: Plaintext length ● pDigest: Result of HMAC-SHA2 function
Return	Status value: ● SUCCESS: HMAC-SHA2 ok ● Others: fail, refer to ERRNO

14.2.2.22 rtl_crypto_hmac_sha2_init

Items	Description
Introduction	HMAC-SHA2 initialization (used for sequential hash)
Parameters	● sha2type: SHA2 type, SHA2_224 or SHA2_256 ● key: HMAC-SHA2 key, need to be 4-byte alignment ● keylen: Key length
Return	Status value: ● SUCCESS: HMAC-SHA2 initialization ok ● Others: fail, refer to ERRNO

14.2.2.23 rtl_crypto_hmac_sha2_update

Items	Description
Introduction	HMAC-SHA2 block calculation (used for sequential hash)

Parameters	● message: Plaintext ● msglen: Plaintext length
Return	Status value: ● SUCCESS: HMAC-SHA2 update ok ● Others: fail, refer to ERRNO

14.2.2.24 rtl_crypto_hmac_sha2_ final

Items	Description
Introduction	HMAC-SHA2 last block calculation (used for sequential hash)
Parameters	pDigest: Result of HMAC-SHA2 function
Return	Status value: ● SUCCESS: HMAC-SHA2 final ok ● Others: fail, refer to ERRNO

14.2.2.25 rtl_crypto_poly1305

Items	Description
Introduction	poly1305 calculation digest
Parameters	● key: poly1305 key, need to be 4-byte alignment ● keylen: Key length ● message: Plaintext ● msglen: Plaintext length ● pDigest: Result of poly1305 function
Return	Status value: ● SUCCESS: poly1305 ok ● Others: fail, refer to ERRNO

14.2.3 Cipher APIs

14.2.3.1 rtl_crypto_xxx_xxx _init

Items	Description
Introduction	xxx initialization xxx can be aes_cbc/aes_ecb/aes_ctr/aes_cfb/aes_ofb/aes_gcm/3des_cbc/3des_ecb/3des_ctr/3des_cfb/3des_ofb/ des_cbc/des_ecb/des_ctr/des_cfb/des_ofb
Parameters	● key: Cipher key. ● keylen: Key length
Return	status value: ● SUCCESS: initialization OK ● Others: fail, refer to ERRNO

14.2.3.2 rtl_crypto_xxx_xxx_encrypt

Items	Description
Introduction	xxx encryption xxx can be aes_cbc/aes_ecb/aes_ctr/aes_cfb/aes_ofb/aes_gcm/3des_cbc/3des_ecb/3des_ctr/3des_cfb/3des_ofb/ des_cbc/des_ecb/des_ctr/des_cfb/des_ofb
Parameters	● message: Point to source message ● msglen: Message length ● iv: Initial vector ● ivlen: iv length ● pResult: Point to cipher result

Return	Status value: ● SUCCESS: encryption OK ● Others: fail, refer to ERRNO

14.2.3.3 rtl_crypto_xxx_xxx_decrypt

Items	Description
Introduction	xxx decryption xxx can be aes_cbc/aes_ecb/aes_ctr/aes_cfb/aes_ofb/aes_gcm/3des_cbc/3des_ecb/3des_ctr/3des_cfb/3des_ofb/ des_cbc/ des_ecb/des_ctr/des_cfb/des_ofb
Parameters	● message: Point to source message ● msglen: Message length ● iv: Initial vector ● ivlen: IV (initialization vector) length ● pResult: Point to cipher result
Return	Status value: ● SUCCESS: decryption OK ● Others: fail, refer to ERRNO

14.2.3.4 rtl_crypto_aes_gcm _init

Items	Description
Introduction	AES-GCM initialization
Parameters	● key: Cipher key ● keylen: Key length
Return	Status value: ● SUCCESS: initialization OK ● Others: fail, refer to ERRNO

14.2.3.5 rtl_crypto_aes_gcm_encrypt

Items	Description
Introduction	AES-GCM encryption
Parameters	● message: Point to source message ● msglen: Message length ● iv: Initial vector ● aad: Point to AAD (additional authentication data) ● aadlen: AAD length ● pResult: Point to cipher result ● pTag: Point to MAC (Message Authentication Code)
Return	Status value: ● SUCCESS: encryption OK ● Others: fail, refer to ERRNO

14.2.3.6 rtl_crypto_aes_gcm_decrypt

Items	Description
Introduction	AES-GCM decryption
Parameters	● message: Point to source message ● msglen: Message length ● iv: Initial vector ● aad: Point to AAD (additional authentication data) ● aadlen: AAD length ● pResult: Point to cipher result

	● pTag: Point to MAC (Message Authentication Code) .
Return	Status value: ● SUCCESS: decryption OK ● Others: fail, refer to ERRNO

14.2.3.7 rtl_crypto_chacha_init

Items	Description
Introduction	ChaCha20 initialization
Parameters	key: Cipher key.
Return	Status value: ● SUCCESS: initialization OK ● Others: fail, refer to ERRNO

14.2.3.8 rtl_crypto_chacha_encrypt

Items	Description
Introduction	ChaCha20 encryption
Parameters	● message: Point to source message. ● msglen: Message length ● iv: Point to IV (initialization vector) ● count: Counter value. ● pResult: Point to cipher result
Return	Status value: ● SUCCESS: encryption OK ● Others: fail, refer to ERRNO

14.2.3.9 rtl_crypto_chacha_decrypt

Items	Description
Introduction	ChaCha20 decryption
Parameters	● message: Point to source message ● msglen: Message length ● iv: Point to IV (initialization vector) ● count: Counter value ● pResult: Point to cipher result
Return	Status value: ● SUCCESS: decryption OK ● Others: fail, refer to ERRNO

14.2.3.10 rtl_crypto_chacha_poly1305_init

Items	Description
Introduction	ChaCha Poly1305 initialization
Parameters	key: Cipher key
Return	Status value: ● SUCCESS: initialization OK ● Others: fail, refer to ERRNO

14.2.3.11 rtl_crypto_chacha_poly1305_encrypt

Items	Description
Introduction	ChaCha Poly1305 encryption

Parameters	● message: Point to source message ● msglen: Message length ● nonce: Random value ● aad: Point to AAD (additional authentication data) ● aadlen: AAD length ● pResult: Point to cipher result ● pTag: Point to MAC (Message Authentication Code)
Return	Status value: ● SUCCESS: encryption OK ● Others: fail, refer to ERRNO

14.2.3.12 rtl_crypto_chacha_poly1305_decrypt

Items	Description
Introduction	ChaCha Poly1305 decryption
Parameters	● message: Point to source message ● msglen: Message length ● nonce: Random value ● aad: Point to AAD (additional authentication data) ● aadlen: AAD length ● pResult: Point to cipher result ● pTag: Point to MAC (Message Authentication Code)
Return	Status value: ● SUCCESS: decryption OK ● Others: fail, refer to ERRNO

14.3 Hardware Crypto Engine APIs Usage

14.3.1 Start Hardware Crypto Engine

Before using hardware crypto engine, you need initialize it first. You can use Crypto Engine Initialize API to do this.

14.3.2 Start Crypto Engine Calculation

Choose a hash or cipher algorithm, and call the following APIs to calculate hash digest or ciphertext.

14.3.2.1 Hash Algorithm

- Hash

If a hash algorithm is selected, then Hash APIs *(rtl_crypto_xxx)* can be used to calculate digest.

- Sequential Hash

Sequential hash is needed when source message length beyond hardware support. Sequential hash breaks a whole long message into several piece of message payload, then calculate the payload in sequence, until the last message payload.

When use sequential hash, you can follow the steps below:

(1) Initialize sequential hash
Use Hash APIs (*rtl_crypto_xxx_init*) to initialize.

(2) Handle each piece of message payload
Call Hash APIs (*rtl_crypto_xxx_update*) once when one piece message payload.

(3) Handle the last piece of message payload
Call Hash APIs (*rtl_crypto_xxx_final*) once when one piece message payload.

14.3.2.2 Cipher Algorithm

Steps to encrypt or decrypt message are as following:
(1) Initialize
Use Cipher APIs (*rtl_crypto_xxx_xxx_init*) to initialize.
(2) Encrypt or decrypt message
Call Cipher APIs (*rtl_crypto_xxx_xxx_encrypt*) to encrypt source message.
Call Cipher APIs (*rtl_crypto_xxx_xxx_decrypt*) to decrypt source message.

14.4 Demo Code Path

Hardware Crypto Engine demo code locates in folder: **project/realtek_amebaD_cm4_gcc_verification/example_sources/CRYPTO**

15 eFuse

eFuse belongs to One Time Programmable (OTP) technology, its default value is '1', and can only be changed from '1' to '0'. eFuse can be used to hold the individual and stable data such as key, calibration data, MAC address, specific setting.

15.1 Power Requirement

The power requirement of eFuse operation is listed in Table 15-1.

Table 15-1 Operation under different voltage

Supply Voltage (V)	Operation
1.8	Only read operation is allowed.
3.3	Read & Write

Note: If you want to program eFuse, you must switch the power supply to 3.3V.

15.2 eFuse Mapping

The eFuse layout of Ameba-D is shown in Fig 15-1. The physical eFuse size is 512 bytes, and the first 288 bytes can be mapping to logical eFuse with size of 1024 bytes.

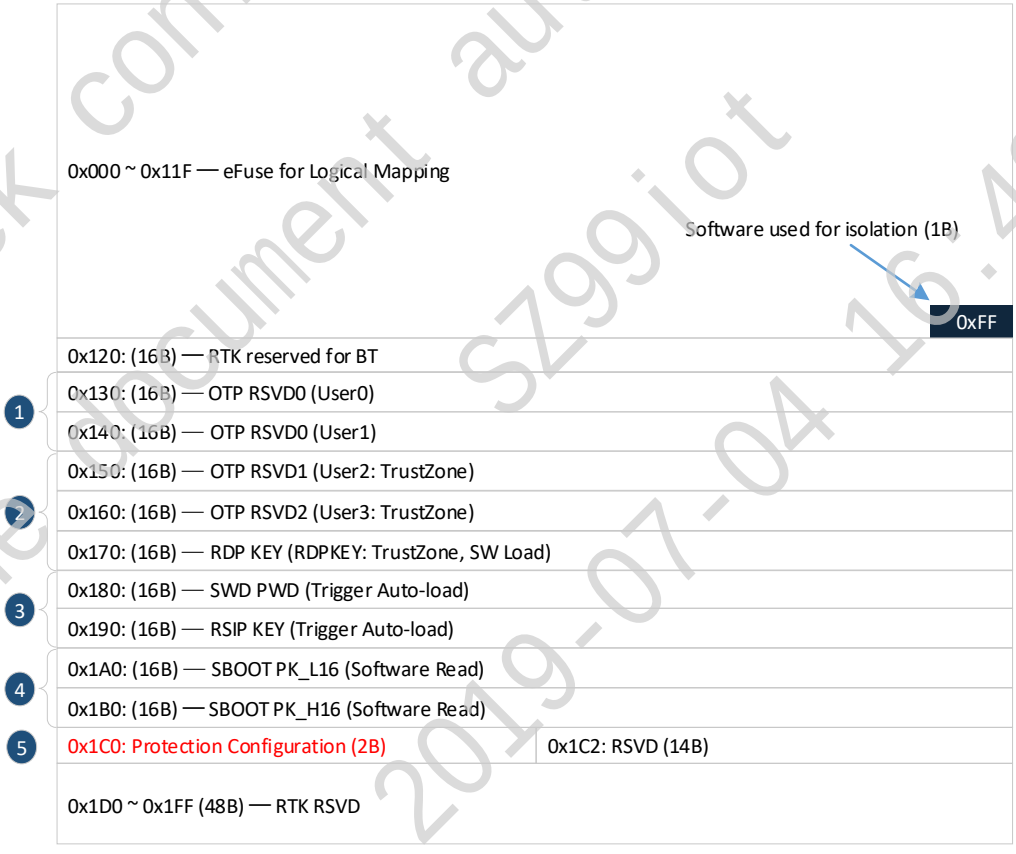


Fig 15-1 eFuse layout diagram

The way to program the eFuse for logical mapping and other physical eFuse is not the same. The detailed APIs are described in section 15.6.

Note: Logical eFuse program will program header and package, and the package is in word, so it takes at least three bytes a time. Software also reserve one byte for isolation, so when the space of eFuse for logical mapping is less than four bytes, it cannot be programmed in any mode.

15.3 eFuse Auto-load

eFuse can auto-load part of setting to control the circuit. eFuse auto-load is changed in word (two bytes). Under default condition, value of all the System Data bytes is 0xFF, and the System Configure Registers have its default values.

The way to change the value of System Configure Registers is as follows:

- Make sure that the first two bytes of System Data area are written to 0x21, 0x87 correctly.
- Program the corresponding bytes in word. If just programming in byte, another byte will load the eFuse default value 0xFF, which may make mistakes.
- Reboot the chip, and the System Configure Registers will load the new values.

15.4 Physical eFuse

15.4.1 Physical eFuse Layout

Table 15-2 Physical eFuse layout

Area	Name	Address	Size	Access	Usage
❶	User0	0x130	16B	Software R/W	Defined by user
	User1	0x140	16B		Defined by user
❷	User2	0x150	16B	• Software R/W • TrustZone Protection	Defined by user
	User3	0x160	16B		Defined by user
	RDP KEY	0x170	16B		Fixed by hardware
❸	SWD KEY	0x180	16B	• Software Trigger auto-load	Fixed by hardware
	RSIP KEY	0x190	16B	• Software Read forbidden • Software Write forbidden	Fixed by hardware
❹	Security Boot KEY	0x1A0	32B	• Software Read • Software Write forbidden	Fixed by ROM
❺	Protection Configuration	0x1C0	2B	Auto-load to SYSON Register	Fixed by hardware to configure ❶~❹

15.4.2 Protection Configuration (R/W)

The protection configuration of eFuse is shown in Table 15-3.

Table 15-3 eFuse protection configuration bits

Bit	Protection	Size Bit	Description
0	SWD Protection	SWD_PWD_RD_Forbidden_EN	0: Read Forbidden
3		SWD_PWD_WR_Forbidden_EN	0: Write Forbidden
8		SWD_Protection_Enable	0: Enable SWD Protection
9	SWD Disable	SWD Secure Noninvasive Debug Disable	0: SWD Secure Noninvasive Debug Disable
10		SWD Secure Invasive Debug Disable	0: SWD Secure Invasive Debug Disable
11		SWD Non-Secure Noninvasive Debug Disable	0: SWD Non-Secure Noninvasive Debug Disable
12		SWD Non-Secure Invasive Debug Disable	0: SWD Non-Secure Invasive Debug Disable
2	RSIP Protection	RSIP_KEY_RD_Forbidden_EN	0: Read Forbidden
5		RSIP_KEY_WR_Forbidden_EN	0: Write Forbidden
6	SBOOT Protection	SBOOT_PK_ WR_Forbidden_EN	0: Write Forbidden
13		SECURE BOOT EN	Software used 0: Enable Security Boot (ECC check)
14	DEBUG Protection	CPU_PC_DBGEN	• 1: Enable to get KM4/KM0 PC value through debug port • 0: Disable

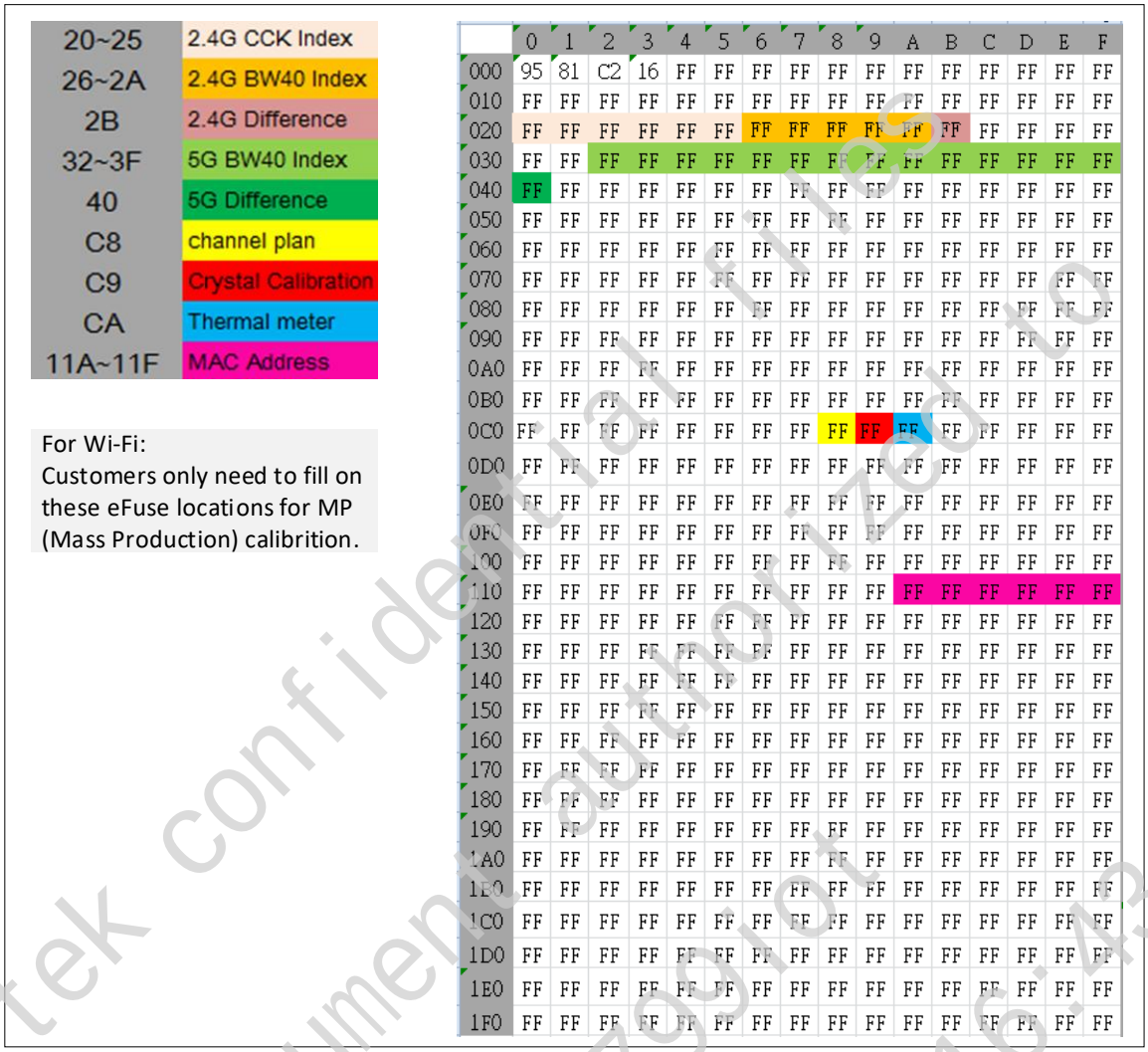
7		RSVD	Reserved
1		RSVD	Reserved
4		RSVD	Reserved
15		RSVD	Reserved

15.5 Logical eFuse

There are 1024 bytes logical eFuse in Ameba-D, as Table 15-4 shows.

Table 15-4 Logical eFuse layout

Start Address	End Address	Description
0x000	0x01F	System Data to be auto-loaded
0x020	0x15F	Wi-Fi calibration data
0x160	0x17F	User MTP
0x180	0x183	Cap-Touch
0x184	0x18F	Reserved
0x190	0x1A4	BT parameters
0x1A5	0x1BF	Reserved
0x1C0	0x1CF	HCI USB
0x1D0	0x1DF	ADC calibration
0x1E0	0x1FF	Reserved
0x200	0x22F	PMC patch
0x230	0x2FF	Reserved



20~25	2.4G CCK Index
26~2A	2.4G BW40 Index
2B	2.4G Difference
32~3F	5G BW40 Index
40	5G Difference
C8	channel plan
C9	Crystal Calibration
CA	Thermal meter
11A~11F	MAC Address

For Wi-Fi:
Customers only need to fill on these eFuse locations for MP (Mass Production) calibrition.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	95	81	C2	16	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
060	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
100	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
110	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
120	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
130	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
140	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
150	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
160	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
170	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
180	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
190	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Fig 15-2 Wi-Fi eFuse data

15.5.1 Wi-Fi 2.4G Power Index

The address and specification of 2.4G power index is shown in Fig 15-3 and Fig 15-4.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Fig 15-3 2.4G Wi-Fi power index address

offset	name	comment
20~25	2.4G CCK Index	Path A CCK Power Index for Ch 1,2, Range 0~127,0.25dbm/step.
		Path A CCK Power Index for Ch 3, 4, 5, Range 0~127,0.25dbm/step.
		Path A CCK Power Index for Ch 6, 7 ,8, Range 0~127,0.25dbm/step.
		Path A CCK Power Index for Ch 9, 10, 11, Range 0~127,0.25dbm/step.
		Path A CCK Power Index for Ch 12, 13, Range 0~127,0.25dbm/step.
		Path A CCK Power Index for Ch 14, Range 0~127,0.25dbm/step.
26~2A	2.4G BW40 Index	Path A 2G BW40-1S Power Index for Ch 1, 2, Range 0~127,0.25dbm/step.
		Path A 2G BW40-1S Power Index for Ch 3, 4, 5, Range 0~127,0.25dbm/step.
		Path A 2G BW40-1S Power Index for Ch 6, 7 ,8, Range 0~127,0.25dbm/step.
		Path A 2G BW40-1S Power Index for Ch 9, 10, 11, Range 0~127,0.25dbm/step.
		Path A 2G BW40-1S Power Index for Ch 12, 13, 14 Range 0~127,0.25dbm/step.
2B	2.4G Difference	Power Index Difference between BW20-1S and BW40-1S Bit[7:4]: Path A 2G Offset, Range –8~7,0.5dbm/step.
		Power Index Difference between OFDM-1Tx and BW40-1S. Bit[3:0]: Path A 2G Offset, Range –8~7,0.5dbm/step.

Fig 15-4 2.4G Wi-Fi power index specification

15.5.2 Wi-Fi 5G Power Index

The address and specification of 5G power index is shown in Fig 15-5 and Fig 15-6.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
040	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Fig 15-5 5G Wi-Fi power index address

32~3F	5G BW40 Index	Path A 5G BW40-1S Power Index for Ch 36,38,40, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 44,46,48, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 52,54,56, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 60,62,64, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 100,102,104, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 108,110,112, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 116,118,120, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 124,126,128, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 132,134,136, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 140,142,144, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 149,151,153, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 157,159,161, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 165,167,169, Range 0~127,0.25dbm/step.
		Path A 5G BW40-1S Power Index for Ch 173,175,177, Range 0~127,0.25dbm/step.
40	5G Difference	Power Index Difference between BW20-1S and BW40-1S. Bit[7:4]: Path A 5G Offset, Range –8~7,0.5dbm/step.
		Power Index Difference between OFDM-1Tx and BW40-1S. Bit[3:0]: Path A 5G Offset, Range –8~7,0.5dbm/step.

Fig 15-6 5G Wi-Fi power index specification

15.5.3 Wi-Fi Channel Plan

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Fig 15-7 Channel plan

eFuse	Name	Bit	Default	Comment
C8	Channel Plan	8	7Fh	Bit[7]: Software configure mode ● 0h: Enable software configure (refer to Channel Plane Domain Code) ● 1h: Disable software configure (can't change Channel Plan Setting) Bit[6:0]: Channel Plan, default means: 2G_WORLD, 5G_WORLD

You can program eFuse channel plan based on Table 15-5.

Table 15-5 eFuse channel plan in driver

Channel Plan in Driver

```
//===== new channel plan mapping, (2GDOMAIN_5GDOMAIN) =====//
RT_CHANNEL_DOMAIN_WORLD_NULL = 0x20,
RT_CHANNEL_DOMAIN_ETSI1_NULL = 0x21,
RT_CHANNEL_DOMAIN_FCC1_NULL = 0x22,
RT_CHANNEL_DOMAIN_MKK1_NULL = 0x23,
RT_CHANNEL_DOMAIN_ETSI2_NULL = 0x24,
RT_CHANNEL_DOMAIN_FCC1_FCC1 = 0x25,
RT_CHANNEL_DOMAIN_WORLD_ETSI1 = 0x26,
RT_CHANNEL_DOMAIN_MKK1_MKK1 = 0x27,
RT_CHANNEL_DOMAIN_WORLD_KCC1 = 0x28,
RT_CHANNEL_DOMAIN_WORLD_FCC2 = 0x29,
RT_CHANNEL_DOMAIN_FCC2_NULL = 0x2A,
RT_CHANNEL_DOMAIN_IC1_IC2 = 0x2B,
RT_CHANNEL_DOMAIN_MKK2_NULL = 0x2C,
RT_CHANNEL_DOMAIN_WORLD_CHILE1 = 0x2D,
RT_CHANNEL_DOMAIN_WORLD1_WORLD1 = 0x2E,
RT_CHANNEL_DOMAIN_WORLD_CHILE2 = 0x2F,
RT_CHANNEL_DOMAIN_WORLD_FCC3 = 0x30,
RT_CHANNEL_DOMAIN_WORLD_FCC4 = 0x31,
RT_CHANNEL_DOMAIN_WORLD_FCC5 = 0x32,
RT_CHANNEL_DOMAIN_WORLD_FCC6 = 0x33,
RT_CHANNEL_DOMAIN_FCC1_FCC7 = 0x34,
RT_CHANNEL_DOMAIN_WORLD_ETSI2 = 0x35,
RT_CHANNEL_DOMAIN_WORLD_ETSI3 = 0x36,
RT_CHANNEL_DOMAIN_MKK1_MKK2 = 0x37,
RT_CHANNEL_DOMAIN_MKK1_MKK3 = 0x38,
RT_CHANNEL_DOMAIN_FCC1_NCC1 = 0x39,
RT_CHANNEL_DOMAIN_FCC1_NCC2 = 0x40,
RT_CHANNEL_DOMAIN_GLOBAL_NULL = 0x41,
RT_CHANNEL_DOMAIN_ETSI1_ETSI4 = 0x42,
RT_CHANNEL_DOMAIN_FCC1_FCC2 = 0x43,
RT_CHANNEL_DOMAIN_FCC1_NCC3 = 0x44,
RT_CHANNEL_DOMAIN_WORLD_ACMA1 = 0x45,
RT_CHANNEL_DOMAIN_FCC1_FCC8 = 0x46,
RT_CHANNEL_DOMAIN_WORLD_ETSI6 = 0x47,
RT_CHANNEL_DOMAIN_WORLD_ETSI7 = 0x48,
RT_CHANNEL_DOMAIN_WORLD_ETSI8 = 0x49,
RT_CHANNEL_DOMAIN_WORLD_ETSI9 = 0x50,
```

```
RT_CHANNEL_DOMAIN_WORLD_ETSI10 = 0x51,
RT_CHANNEL_DOMAIN_WORLD_ETSI11 = 0x52,
RT_CHANNEL_DOMAIN_FCC1_NCC4 = 0x53,
RT_CHANNEL_DOMAIN_WORLD_ETSI12 = 0x54,
RT_CHANNEL_DOMAIN_FCC1_FCC9 = 0x55,
RT_CHANNEL_DOMAIN_WORLD_ETSI13 = 0x56,
RT_CHANNEL_DOMAIN_FCC1_FCC10 = 0x57,
RT_CHANNEL_DOMAIN_MKK2_MKK4 = 0x58,
RT_CHANNEL_DOMAIN_WORLD_ETSI14 = 0x59,
RT_CHANNEL_DOMAIN_FCC1_FCC5 = 0x60,
RT_CHANNEL_DOMAIN_FCC2_FCC7 = 0x61,
RT_CHANNEL_DOMAIN_FCC2_FCC1 = 0x62,
RT_CHANNEL_DOMAIN_WORLD_ETSI15 = 0x63,
RT_CHANNEL_DOMAIN_MKK2_MKK5 = 0x64,
RT_CHANNEL_DOMAIN_ETSI1_ETSI16 = 0x65,
RT_CHANNEL_DOMAIN_FCC1_FCC14 = 0x66,
RT_CHANNEL_DOMAIN_FCC1_FCC12 = 0x67,
RT_CHANNEL_DOMAIN_FCC2_FCC14 = 0x68,
RT_CHANNEL_DOMAIN_FCC2_FCC12 = 0x69,
RT_CHANNEL_DOMAIN_ETSI1_ETSI17 = 0x6A,
RT_CHANNEL_DOMAIN_WORLD_FCC16 = 0x6B,
RT_CHANNEL_DOMAIN_WORLD_FCC13 = 0x6C,
RT_CHANNEL_DOMAIN_FCC2_FCC15 = 0x6D,
RT_CHANNEL_DOMAIN_WORLD_FCC12 = 0x6E,
RT_CHANNEL_DOMAIN_NULL_ETSI8 = 0x6F,
RT_CHANNEL_DOMAIN_NULL_ETSI18 = 0x70,
RT_CHANNEL_DOMAIN_NULL_ETSI17 = 0x71,
RT_CHANNEL_DOMAIN_NULL_ETSI19 = 0x72,
RT_CHANNEL_DOMAIN_WORLD_FCC7 = 0x73,
RT_CHANNEL_DOMAIN_FCC2_FCC17 = 0x74,
RT_CHANNEL_DOMAIN_WORLD_ETSI20 = 0x75,
RT_CHANNEL_DOMAIN_FCC2_FCC11 = 0x76,
RT_CHANNEL_DOMAIN_WORLD_ETSI21 = 0x77,
RT_CHANNEL_DOMAIN_FCC1_FCC18 = 0x78,
RT_CHANNEL_DOMAIN_MKK2_MKK1 = 0x79,

//===== Add new channel plan above this line===============//
RT_CHANNEL_DOMAIN_MAX,
RT_CHANNEL_DOMAIN_REALTEK_DEFINE = 0x7F
```

15.5.4 Wi-Fi Crystal Calibration

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Fig 15-8 Crystal Calibration

eFuse	Name	Bit	Default	Comment
C9	Crystal Calibration	8	40h	XTAL_K Value ● Bit[6:0]: Xi=Xo, range 0~7Fh ● Bit[7]: reserved ● FFh = 00h

15.5.5 Wi-Fi Thermal Meter

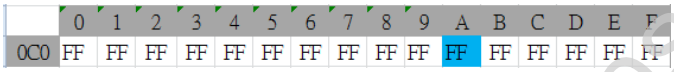


Fig 15-9 Thermal Meter

eFuse	Name	Bit	Default	Comment
CA	Thermal Meter	8	16h	Thermal Meter Default Value System maker will calibrate a value and save it in EEPROM. Bit[7:0]: Thermal Meter Value 0xFF: Disable Tx power tracking function

15.5.6 Wi-Fi MAC Address

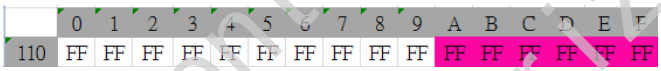


Fig 15-10 MAC Address

Address Offset	Bit	Default	Comment
Byte 11A	[7:0]	FF	MAC Address: [7:0]
Byte 11B	[7:0]	FF	MAC Address: [15:8]
Byte 11C	[7:0]	FF	MAC Address: [23:16]
Byte 11D	[7:0]	FF	MAC Address: [31:24]
Byte 11E	[7:0]	FF	MAC Address: [39:32]
Byte 11F	[7:0]	FF	MAC Address: [47:40]

15.5.7 Cap-Touch

Address Offset	Bit	Default	Description
180h	[7:0]	FF	Touch key channel 0 threshold[7:0]
181h	[6:0]	FF	Touch key channel 1 threshold[6:0]
	[7]	FF	Touch key channel 0 threshold[8]
182h	[6:0]	FF	Touch key channel 2 threshold diff[6:0]
	[7]	FF	Touch key channel 0 threshold[9]
183h	[6:0]	FF	Touch key channel 3 threshold diff[6:0]
	[7]	FF	Touch key channel 0 threshold[10]

15.5.8 Bluetooth

Address Offset	Bit	Default	Description
190~195h	[47:0]		BD address (module must)
196h	[7:0]		ENABLE_PHY_INIT[0] iqm_txgaink_module_valid iqm_txgain_flatk_module_valid lbt_mode[4:3] lbt_enable[5] rsvd1
197h	[7:0]		iqm_txgaink_module
198~199h	[15:0]		iqm_txgain_flatk_module
19Ah	[7:0]		iqm_txgain_10dBm_raw_index

19Bh	[7:0]		lbt_ant_gain
19Ch	[7:0]		iqm_max_txgain_LE1M
19Dh	[7:0]		iqm_max_txgain_LE2M
19Eh	[7:0]		iqm_max_txgain_LE1M_adv
19Fh	[7:0]		iqm_max_txgain_LE2M_adv
1A0h	[7:0]		otp.phy_init_cfg.tmeterx4_txgaink_module
1A1h	[7:0]		iqm_max_txgain_LE1M_2402
1A2h	[7:0]		iqm_max_txgain_LE1M_2480
1A3h	[7:0]		iqm_max_txgain_LE2M_2402
1A4h	[7:0]		iqm_max_txgain_LE2M_2480

15.5.9 HCI USB

Address Offset	Name	Bit
Byte 1C0	VID[7:0]	[7:0]
Byte 1C1	VID[15:8]	[7:0]
Byte 1C2	PID[7:0]	[7:0]
Byte 1C3	PID[15:8]	[7:0]
Byte 1C4	USB device type	[7:0]
Byte 1C5~1CF	RSVD	

15.5.10 ADC Calibration

Address Offset	Bit	Description
1D0 ~ 1D1h	[16:0]	Normal channel (CH0~CH6) single-ended input, OFFSET parameter
1D2 ~ 1D3h	[16:0]	Normal channel (CH0~CH6) single-ended input, GAIN parameter
1D4 ~ 1D5h	[16:0]	VBAT channel OFFSET
1D6 ~ 1D7h	[16:0]	VBAT channel GAIN
1D8 ~ 1D9h	[16:0]	Normal channel (CH0~CH6) differential input, OFFSET parameter
1DA ~ 1DBh	[16:0]	Normal channel (CH0~CH6) differential input, GAIN parameter
1DC ~ 1DFh	[31:0]	RSVD for ADC calibration

15.6 eFuse PG APIs

Items	API	Comment
Low Level API	EFUSE_PMAP_WRITE8	Physical map writes one byte
	EFUSE_PMAP_READ8	Physical map reads one byte
	EFUSE_LMAP_WRITE	Write eFuse logical address by length
	EFUSE_LMAP_READ	Read total eFuse logical map
mbed API	efuse_otp_write	Write content to OTP eFuse by length
	efuse_otp_read	Read eFuse OTP content by length
	efuse_mtp_write	Write user's content to USER MTP Space (0x160~0x17F)
	efuse_mtp_read	Read eFuse content from address (0x160 ~ 0x17F)

15.6.1 Low Level APIs

15.6.1.1 EFUSE_PMAP_WRITE8

Items	Description
Introduction	Physical map writes one byte.

Parameters	● CtrlSetting: eFuse control setting read from REG_LP_EFUSE_CTRL1 (suggest 0) ● Addr: eFuse physical address ● Data: 1 byte data to write ● L25OutVoltage: L25EOUTVOLTAGE.
Return	Status value: ● _TRUE: Write is ok. ● _FALSE: Write is failed.

15.6.1.2 EFUSE_PMAP_READ8

Items	Description
Introduction	Physical map reads one byte.
Parameters	● CtrlSetting: eFuse control setting read from REG_LP_EFUSE_CTRL1 (suggest 0) ● Addr: eFuse physical address ● Data: 1 byte data buffer for eFuse data read ● L25OutVoltage: L25EOUTVOLTAGE.
Return	Status value: ● _TRUE: Read is ok. ● _FALSE: Read is failed.

15.6.1.3 EFUSE_LMAP_WRITE

Items	Description
Introduction	Write eFuse logical address by length
Parameters	● addr: logical address, should be word-aligned ● cnts: byte number, should be even ● data: data buffer to be write
Return	Status value: ● _SUCCESS: Write is ok. ● _FAIL: Write is failed.

15.6.1.4 EFUSE_LMAP_READ

Items	Description
Introduction	Read total eFuse logical map
Parameters	pbuf: 1024 bytes length buffer used for eFuse Logical map
Return	Status value: ● _SUCCESS: Read is ok. ● _FALSE: Read is failed.

15.6.2 Mbed APIs

15.6.2.1 efuse_otp_write

Items	Description
Introduction	Write content to OTP eFuse by length (mbed API)
Parameters	● address: Specifies the offset of the programmed OTP. ● len: Specifies the data length of programmed data. ● buf: Specified the data to be programmed.
Return	Status value: ● 0: Success ● -1: Failure

15.6.2.2 efuse_otp_read

Items	Description
Introduction	Read eFuse OTP content by length (mbed API)
Parameters	● address: Specifies the offset of the OTP. ● len: Specifies the length of readback data. ● buf: Specified the address to save the readback data.
Return	Status value: ● 0: Success ● -1: Failure

15.6.2.3 efuse_mtp_write

Items	Description
Introduction	Write user's content to USER MTP Space (0x160~0x17F)
Parameters	● data: Specified the data to write ● len: Specified the data length to write
Return	Status value: ● 0 ~ 32: Success ● 0 or -1: Failure

15.6.2.4 efuse_mtp_read

Items	Description
Introduction	Read eFuse content from address (0x160 ~ 0x17F)
Parameters	Data: Specified the address to save the read back data
Return	N/A

15.7 eFuse PG Command

Items	Offset	Command
Tx Power Index	0x20~0x2B	iwpriv config_set wmap, 020, 20202020202020202020202002
Channel plan, XTAL & Thermal & RTK reserved	0xC8~0xCF	iwpriv config_set wmap, 0C8, 20201A05000000FF[1]
MAC Address	0x11A~0x11F	iwpriv config_set wmap, 11a, 00e04c870102
Realtek RSVD	0x130~0x139	iwpriv config_set wmap, 130, FF01001000FF00FF1000
Get all eFuse map		iwpriv config_get realmap

1. Channel plan does not affect the results of RF verification, so you can choose whether or not to write the correct channel plan here.

15.8 eFuse Check Sequence (TBD)

eFuse should be checked to make sure if it's programed right after program.

- eFuse write
- HAL_WRITE32 (0x40000000, 0x00EC, 0x00000000)
- eFuse read
- HAL_WRITE32 (0x40000000, 0x00EC, 0x04000000)
- Check if eFuse write is ok.

16 Flash Operation

16.1 Functional Description

Ameba-D uses SPI Controller (SPIC) to communicate with SPI NOR flash.

There are two operation modes for SPIC: auto mode and user mode.

- User mode

User mode is a typical software flow to implement all serial transfer. User can transmit or receive data from SPI flash through setting up SPIC registers.

- Auto mode

Compared to user mode, auto mode is a hardware control flow to execute. After initialize setting, user don't need to configure the related control register for each transfer operation. Auto mode is a convenient way to access SPI flash just as access memory (SRAM or DRAM).

The SPIC is initialized automatically in boot sequence, and user must not initialize SPIC manually. The supported mode for different operations can be found in Table 16-1.

Table 16-1 SPIC operation mode

Flash Operation	SPIC Operation Mode
Read data	Auto mode/User mode
Program data	User mode
Erase	User mode
Receive/transmit command	User mode

Note: Auto mode and user mode can't be used at the same time.

16.2 Protection Method

Considering Ameba-D supports flash executed in place (XIP), which enables code execute directly in flash memory without copying to RAM. Both KM0 and KM4 CPU cores can issue request to SPIC in auto mode to read instructions from flash to execute.

To prevent being interrupted by auto mode reading, the user mode operation should be protected until it is finished. Fig 16-1 shows the flow of KM0 manipulating flash safely in user mode, which has been protected. The method of KM4 manipulating flash is in the same way.

The protection methods are implemented in **FLASH_Write_Lock()** and **FLASH_Write_Unlock()** functions. The APIs provided by Realtek in 16.3 and 16.4 are already protected and can be called by users directly.

Note: If users need to implement user mode function by themselves, the code of manipulating SPIC should locate in RAM instead of flash. They should also call **FLASH_Write_Lock()** before operation and call **FLASH_Write_Unlock()** to release from protection after operation.

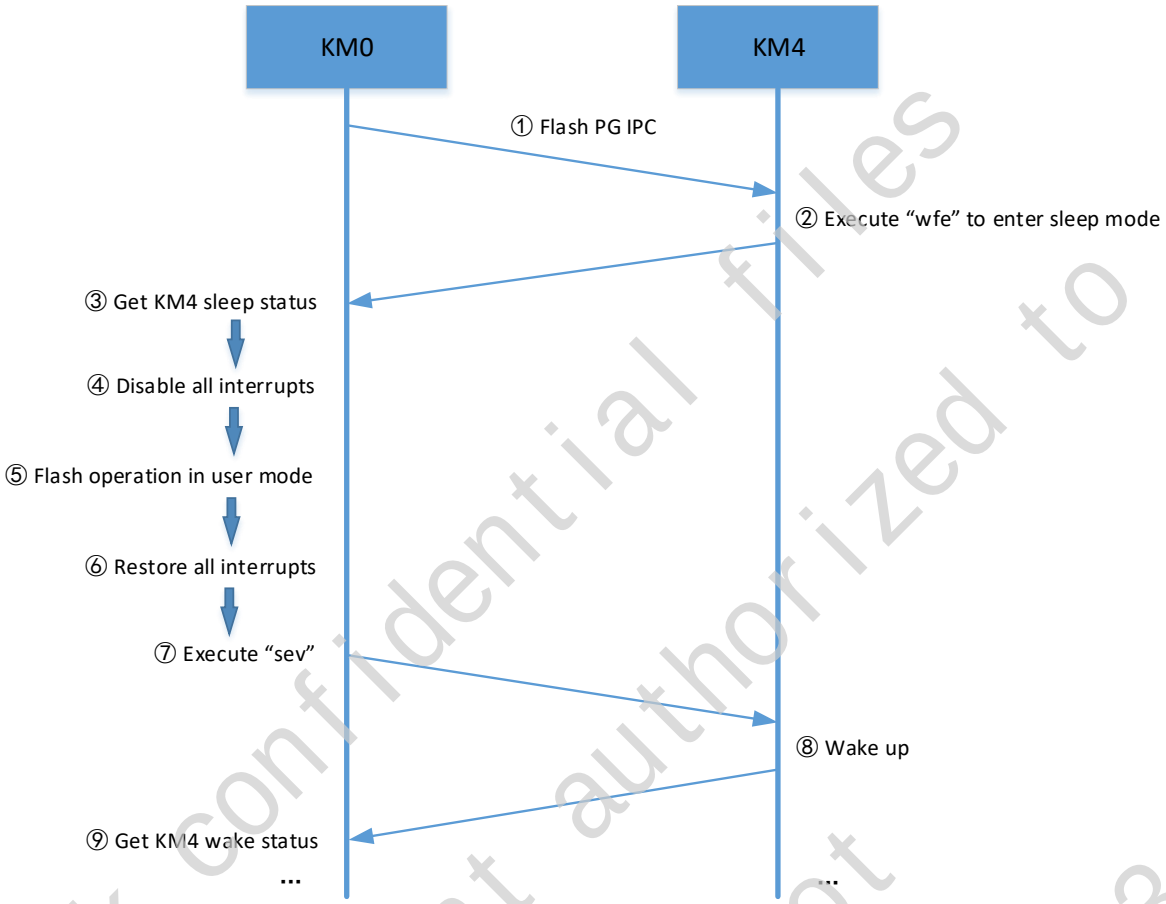


Fig 16-1 User mode operation flow

16.3 Flash Raw APIs

16.3.1 Read

16.3.1.1 HAL_READ8/ HAL_READ16/ HAL_READ32

Items	Description
Introduction	Read 1-byte/1-word/1-dword data from flash in **auto mode**, just as access SRAM.
Parameters	● base: the base address of flash memory. It should be SPI_FLASH_BASE which is 0x08000000. ● addr: the offset address in flash memory. It should be byte-aligned for HAL_READ8, word-aligned for HAL_READ16 and dword-aligned for HAL_READ32.
Return	The read data

16.3.1.2 FLASH_ReadStream

Items	Description
Introduction	Read a stream of data from specified offset address of flash in **auto mode**.
Parameters	● address: specify the starting offset address to read from. It has no alignment restrictions. ● len: specify the length of the data to read. ● data: specify the buffer to save the read-back data.
Return	Status:

	● 1: success ● Others: fail

16.3.2 Write

16.3.2.1 FLASH_TxData12BXIP

Items	Description
Introduction	Write data to flash in **user mode**. This function is protected and can be called by user directly.
Parameters	● StartAddr: start offset address in flash from which SPIC writes. ● DataPhaseLen: the number of bytes that SPIC sends in Data Phase. It should not be more than 12 bytes. ● pData: pointer to a byte array that is to be sent.
Return	N/A

16.3.2.2 FLASH_WriteStream

Items	Description
Introduction	Write a stream of data to specified address in **user mode**. The function is already protected and can be called by user directly.
Parameters	● address: specifies the starting offset to write to. ● len: specifies the length of the data to write. It has no restrictions of less than 12 bytes. ● data: pointer to a byte array that is to be written.
Return	Status: ● 1: success ● Others: fail

16.3.3 Erase

16.3.3.1 FLASH_EraseXIP

Items	Description
Introduction	Erase sector/block/chip for flash in **user mode**. The function is already protected and can be called by users directly.
Parameters	● EraseType: can be one of the following parameters: ■ EraseChip: erase the whole chip. ■ EraseBlock: erase the specified block (64KB). ■ EraseSector: erase the specified sector (4KB). ● Address: address should be 4-byte aligned. The block/sector which the address in will be erased.
Return	N/A

16.3.4 Receive/Transmit Command

16.3.4.1 FLASH_RxCmdXIP

Items	Description
Introduction	Send Rx command to flash to get status register or flash ID in **user mode**. The function is already protected and can be called by user directly.
Parameters	● cmd: command that need to be sent. ● read_len: the number of bytes that will be read by SPIC after sending command. ● read_data: pointer to a byte array which is used to save data received.
Return	N/A

16.3.4.2 FLASH_SetStatusXIP

Items	Description
Introduction	Set flash status register in **user mode**. The function is already protected and can be called by users directly.
Parameters	● cmd: command to be sent. ● len: the number of bytes to be sent after sending command. ● status: pointer to byte array to be sent.
Return	N/A

16.4 Flash Mbed APIs

The Flash mbed API descriptions can be found in “AN403 Peripheral Driver Mbed API.chm”. All those functions have already been protected and can be called directly.

16.5 User Configuration

Ameba-D provides **rtl8721dlp_flashcfg.c** to configure flash speed and read mode.

16.5.1 Flash Speed

```
/**
* @brif Indicate the flash baudrate. It can be one of the following value:
*       0xFFFF: 80MHz
*       0x7FFF: 100MHz
*       0x3FFF: 67MHz
*       0x1FFF: 57MHz
*       0x0FFF: 50MHz
*/
const u16 Flash_Speed = 0xFFFF;
```

16.5.2 Flash Read Mode

```
/**
* @brif Indicate the flash read I/O mode. It can be one of the following value:
*       0xFFFF: Read quad IO, Address & Data 4 bits mode
*       0x7FFF: Read quad O, Just data 4 bits mode
*       0x3FFF: Read dual IO, Address & Data 2 bits mode
*       0x1FFF: Read dual O, Just data 2 bits mode
*       0x0FFF: 1 bit mode
* @note If the configured read mode is not supported, other modes would be searched until find the appropriate mode.
*/
const u16 Flash_ReadMode = 0xFFFF;
```

16.5.3 New Flash Support

For some Flash chip which is not available in Flash AVL, the principle to determine whether it can be supported by SDK, and the method to add a new flash can be found in User Configuration.

17 Battery Measurement

17.1 Functional Description

LP-ADC has a total of 11 channels, of which 8 are external channels and 3 are internal channels. 8 external channels include 7 normal channels and 1 battery measurement channel called VBAT channel, as Table 17-1 shows. The VBAT channel measurable range is 0~5V.

Table 17-1 Channel description

Function	ADC Channel	Pin Name	Voltage Range
Normal channel	CH0 ~ CH6	PB_4 ~ PB_7 (CH0 ~ 3)	0 ~ 3.3V (when power is 3.3V)
		PB_1 ~ PB_3 (CH4 ~ 6)	0 ~ 1.8V (when power is 1.8V)
VBAT	CH7	VBAT_MEAS	0 ~ 5V

When one channel is added to channel switch list, LP-ADC would convert the voltage of corresponding pin to digital data. The resolution of digital sample data is 12-bit.

To obtain sample data, auto mode, timer-trigger mode and software-trigger mode can be adopted according to different usages. LP-ADC can also be configured to send wakeup and interrupt signal to system when the sample data matches criteria.

17.2 Calibration

The relationship between input voltage and sample data is almost linear. Fig 17-1 shows the conversion of sample data and input voltage of VBAT_MEAS.

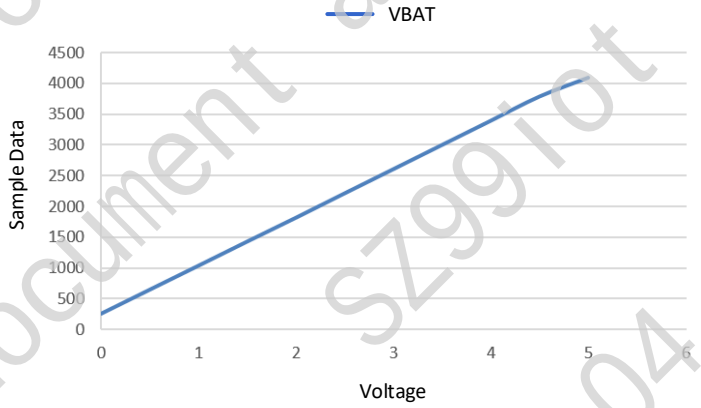


Fig 17-1 Relationship between sample data and input voltage

To obtain voltage from sample data, the following formula can be used:

Voltage = (10 * Data - OFFSET)/GAIN (V)

where

- Data: 12-bit sample data
- OFFSET: 10 times of sample data of 0.000v
- GAIN: 10 times of sample data increment when voltage rises 1V

To enhance conversion accuracy, ADC calibration is necessary to obtain precise OFFSET and GAIN parameters for channels.

The calibration operations can be done in FT test. The FT test would sample for different voltages and use Linear Fitting Method to calculate OFFSET and GAIN.

Pay attention that the OFFSET and GAIN values gotten from FT test are 10 times of the actual values. When calculating voltage using the above two values, they must be divided by 10 first; that is, voltage = (data – OFFSET/10)/(GAIN/10) = (10 * data – offset)/GAIN.

As normal external channels use the same voltage divider circuit, they can use one set of calibration parameters. But VBAT channel need to be calibrated independently.

These parameters would be programmed into eFuse after calibration, as Table 17-2 shows.

Table 17-2 Calibration parameters location

Channel	Input Mode	Parameter	Address
Normal (CH0 ~ CH6)	Single-end	OFFSET	Logical map 0x1A0 ~ 0x1A1
		GAIN	Logical map 0x1A2 ~ 0x1A3
	Differential	OFFSET	Logical map 0x1A8 ~ 0x1A9
		GAIN	Logical map 0x1AA ~ 0x1AB
VBAT (CH7)	Single-end	OFFSET	Logical map 0x1A4 ~ 0x1A5
		GAIN	Logical map 0x1A6 ~ 0x1A7

To calculate voltage from sample data, user only need to obtain calibration parameters from eFuse and call the voltage computational formula.

18 Wi-Fi

18.1 Wi-Fi Data Structures

Data Structures	Introduction
<_cus_ie>	The structure is used to set Wi-Fi custom IE list, and type match CUSTOM_IE_TYPE. The IE will be transmitted according to the type.
<rtw_ssid>	The structure is used to describe the SSID.
<rtw_mac>	The structure is used to describe the unique 6-byte MAC address.
<rtw_ap_info>	The structure is used to describe the setting about SSID, security type, password and default channel, used to start AP mode.
<rtw_network_info>	The structure is used to describe the station mode setting about SSID, security type and password, used when connecting to an AP.
<rtw_scan_result>	The structure is used to describe the scan result of the AP.
<rtw_scan_handler_result>	The structure is used to describe the data needed by scan result handler function.
<rtw_wifi_setting>	The structure is used to store the Wi-Fi setting gotten from Wi-Fi driver.
<rtw_wifi_config>	The structure is used to describe the setting when configure the network.
<rtw_maclist_t>	The structure is used to describe the maclist.
<rtw_bss_info_t>	The structure is used to describe the bss info of the network. It includes the version, BSSID, beacon_period, capability, SSID, channel, atm_window, dtim_period, RSSI etc.

18.2 Wi-Fi APIs

18.2.1 System APIs

API	Introduction
<wifi_on>	Enable Wi-Fi.
<wifi_off>	Disable Wi-Fi.
<wifi_off_fastly>	Turn off the Wi-Fi device.
<wifi_is_up>	Check if the specified interface is up.
<wifi_is_ready_to_transceive>	Determines if a particular interface is ready to transceiver Ethernet packets.
<wifi_rf_on>	Enable Wi-Fi RF.
<wifi_rf_off>	Disable Wi-Fi RF.

18.2.1.1 wifi_on

Enable Wi-Fi.

- Bring the wireless interface “Up”.
- Initialize the driver thread which arbitrates access to the SDIO/SPI bus.

Parameter	Type	Introduction
<mode>	rtw_mode_t	Decide to enable Wi-Fi in which mode. The optional modes are enumerated in rtw_mode_t.

18.2.1.2 wifi_off

Disable Wi-Fi.

Parameter	Type	Introduction
None	-	-

18.2.1.3 wifi_off_fastly

Turn off the Wi-Fi device.

- Bring the wireless interface “Down”.
- De-Initializes the driver thread which arbitrates access to the SDIO/SPI bus.

Parameter	Type	Introduction
None	-	-

18.2.1.4 wifi_is_up

Check if the specified interface is up.

Parameter	Type	Introduction
<interface>	rtw_interface_t	The interface can be set as RTW_STA_INTERFACE or RTW_AP_INTERFACE. (rtw_interface_t)

18.2.1.5 wifi_is_ready_to_transceive

Determines if a particular interface is ready to transceive Ethernet packets.

Parameter	Type	Introduction
<interface>	rtw_interface_t	interface to check, options are RTW_STA_INTERFACE, RTW_AP_INTERFACE.

18.2.1.6 wifi_rf_on

Enable Wi-Fi RF.

Parameter	Type	Introduction
None	-	-

Note: The difference between wifi_rf_on() and wifi_on() is that wifi_rf_on() simply enable RF HAL, it does not enable the driver or allocate memory.

18.2.1.7 wifi_rf_off

Disable Wi-Fi RF.

Parameter	Type	Introduction
None	-	-

Note: The difference between wifi_rf_off() and wifi_off() is that wifi_rf_off() simply disable RF HAL, the driver and used heap memory will not be released.

18.2.2 Scan APIs

API	Introduction
<wifi_scan>	Initiate a scan to search for 802.11 networks, a higher level API based on wifi_scan to simplify the scan operation.
<wifi_scan_networks_with_ssid>	Initiate a scan to search for 802.11 networks with specified SSID.

18.2.2.1 wifi_scan

Initiate a scan to search for 802.11 networks, a higher level API based on wifi_scan to simplify the scan operation.

Parameter	Type	Introduction
<scan_type>	rtw_scan_type_t	Specifies whether the scan should be active, passive or scan prohibited channels.
<bss_type>	rtw_bss_type_t	Specifies whether the scan should search for infrastructure networks (those using an AP), Ad-hoc networks, or both types.
<result_ptr>	void *	Scan specific ssid. The first 4 bytes is ssid length, and ssid name append after it. If no specific ssid need to scan, please clean result_ptr before pass it into parameter.
< result_ptr >	void *	[out] a pointer to a pointer to a result storage structure.

Note:

- The scan progressively accumulates results over time, and may take between 1 and 3 seconds to complete. The results of the scan will be individually provided to the callback function.
- The callback function will be executed in the context of the RTW thread.
- When scanning specific channels, devices with a strong signal strength on nearby channels may be detected.

18.2.2.2 wifi_scan_networks_with_ssid

Initiate a scan to search for 802.11 networks with specified SSID.

Parameter	Type	Introduction
<results_handler>	int	The callback function which will receive and process the result data.
<user_data>	void *	User specified data that will be passed directly to the callback function.
<scan_buflen>	int	The length of the result storage structure.
< ssid>	char *	The SSID of target network.
<ssid_len>	int	The length of the target network SSID.

Note: Callback must not use blocking functions, since it is called from the context of the RTW thread. The callback, user_data variables will be referenced after the function returns. Those variables must remain valid until the scan is completed.

18.2.3 Connection APIs

API	Introduction
<wifi_connect>	Join a Wi-Fi network. Scan for, associate and authenticate with a Wi-Fi network. On successful return, the system is ready to send data packets.
<wifi_connect_bssid>	Join a Wi-Fi network with specified BSSID. Scan for, associate and authenticate with a Wi-Fi network. On successful return, the system is ready to send data packets.
<wifi_disconnect>	Disassociates from current Wi-Fi network.
<wifi_is_connected_to_ap>	Check if Wi-Fi has connected to AP before dhcp.
<wifi_config_autoreconnect>	Set reconnection mode with configuration.
<wifi_set_autoreconnect>	Set reconnection mode with 3 retry limit and 5 seconds timeout as default.
<wifi_get_autoreconnect>	Get the result of setting reconnection mode.
<wifi_get_last_error>	Present the device disconnect reason while connecting.

18.2.3.1 wifi_connect

Join a Wi-Fi network. Scan for, associate and authenticate with a Wi-Fi network. On successful return, the system is ready to send data packets.

Parameter	Type	Introduction
<ssid>	char *	A null terminated string containing the SSID name of the network to join.
<security_type>	rtw_security_t	Authentication type:

		● RTW_SECURITY_OPEN: open security ● RTW_SECURITY_WEP_PSK: WEP Security with open authentication ● RTW_SECURITY_WEP_SHARED: WEP Security with shared authentication ● RTW_SECURITY_WPA_TKIP_PSK: WPA Security ● RTW_SECURITY_WPA2_AES_PSK: WPA2 Security using AES cipher ● RTW_SECURITY_WPA2_TKIP_PSK: WPA2 Security using TKIP cipher ● RTW_SECURITY_WPA2_MIXED_PSK: WPA2 Security using AES and/or TKIP ciphers
<password>	char *	A byte array containing either the cleartext security key for WPA/WPA2 secured networks, or a pointer to an array of rtw_wep_key_t structures for WEP secured networks.
<ssid_len>	int	The length of the SSID in bytes.
<password_len>	int	The length of the security_key in bytes.
<key_id>	int	The index of the wep key (0, 1, 2, or 3). If not using it, leave it with value -1.
<semaphore>	void *	A user provided semaphore is flagged when the join is complete. If not using, leave it with NULL value.

Note: Make sure the Wi-Fi is enabled before invoking this function. (wifi_on())

18.2.3.2 wifi_connect_bssid

Join a Wi-Fi network with specified BSSID. Scan for, associate and authenticate with a Wi-Fi network. On successful return, the system is ready to send data packets.

Parameter	Type	Introduction
<bssid>	unsigned char	The specified BSSID to connect.
<ssid>	char *	A null terminated string containing the SSID name of the network to join.
<security_type>	rtw_security_t	Authentication type: ● RTW_SECURITY_OPEN: open security ● RTW_SECURITY_WEP_PSK: WEP Security with open authentication ● RTW_SECURITY_WEP_SHARED: WEP Security with shared authentication ● RTW_SECURITY_WPA_TKIP_PSK: WPA Security ● RTW_SECURITY_WPA2_AES_PSK: WPA2 Security using AES cipher ● RTW_SECURITY_WPA2_TKIP_PSK: WPA2 Security using TKIP cipher ● RTW_SECURITY_WPA2_MIXED_PSK: WPA2 Security using AES and/or TKIP ciphers
<password>	char *	A byte array containing either the cleartext security key for WPA/WPA2 secured networks, or a pointer to an array of rtw_wep_key_t structures for WEP secured networks.
<ssid_len>	int	The length of the SSID in bytes.
<password_len>	int	The length of the security_key in bytes.
<key_id>	int	The index of the wep key (0, 1, 2, or 3). If not using it, leave it with value -1.
<semaphore>	void *	A user provided semaphore is flagged when the join is complete. If not using, leave it with NULL value.
<semaphore>	void *	A user provided semaphore is flagged when the join is complete. If not using, leave it with NULL value.

Note:

- Make sure the Wi-Fi is enabled before invoking this function. (wifi_on())
- The difference between wifi_connect_bssid() and wifi_connect() is that BSSID has higher priority as the basis of connection in wifi_connect_bssid().

18.2.3.3 wifi_disconnect

Disassociates from current Wi-Fi network.

Parameter	Type	Introduction
None		

18.2.3.4 wifi_is_connected_to_ap

Check if Wi-Fi has connected to AP before dhcp.

Parameter	Type	Introduction
None	-	-

18.2.3.5 wifi_config_autoreconnect

Set reconnection mode with configuration.

Parameter	Type	Introduction
<mode>	_u8	Set 1/0 to enable/disable the reconnection mode.
<callback>	_u8	The number of retry limit.
<len_used>	_u16	The timeout value (in seconds).

Note:

- Defining CONFIG_AUTO_RECONNECT in **autoconf.h** needs to be done before compiling, or this API won't be effective.
- The difference between **wifi_config_autoreconnect()** and **wifi_set_autoreconnect()** is that user can specify the retry times and timeout value in **wifi_config_autoreconnect()**. But in **wifi_set_autoreconnect()** these values are set with 3 retry limit and 5 seconds timeout as default.

18.2.3.6 wifi_set_autoreconnect

Set reconnection mode with 3 retry limit and 5 seconds timeout as default.

Parameter	Type	Introduction
<mode>	_u8	Set 1/0 to enalbe/disable the reconnection mode.

Note:

- Defining CONFIG_AUTO_RECONNECT in **autoconf.h** needs to be done before compiling, or this API won't be effective.
- The difference between **wifi_config_autoreconnect()** and **wifi_set_autoreconnect()** is that user can specify the retry times and timeout value in **wifi_config_autoreconnect()**. But in **wifi_set_autoreconnect()** these values are set with 3 retry limit and 5 seconds timeout as default.

18.2.3.7 wifi_get_autoreconnect

Get the result of setting reconnection mode.

Parameter	Type	Introduction
<mode>	_u8	Pointer to the result of setting reconnection mode.

Note: Defining CONFIG_AUTO_RECONNECT in **autoconf.h** needs to be done before compiling, or this API won't be effective.

18.2.3.8 wifi_get_last_error

Present the device disconnect reason while connecting.

Parameter	Type	Introduction
None	-	-

18.2.4 Channel APIs

API	Introduction
<wifi_set_channel>	Set the listening channel on STA interface.
<wifi_get_channel>	Get the current channel on STA interface.
<wifi_set_channel_plan>	Set channel plan into flash/eFuse, must reboot after setting channel plan.
<wifi_get_channel_plan>	Get channel plan from calibration section.

18.2.4.1 wifi_set_channel

Set the listening channel on STA interface.

Parameter	Type	Introduction
<channel>	int	The desired channel.

Note: do not need to call this function for STA mode Wi-Fi driver, since it will determine the channel from received beacon.

18.2.4.2 wifi_get_channel

Get the current channel on STA interface.

Parameter	Type	Introduction
<channel>	int *	A pointer to the variable where the channel value will be written.

18.2.4.3 wifi_set_channel_plan

Set channel plan into flash/eFuse, must reboot after setting channel plan.

Parameter	Type	Introduction
<channel_plan>	uint8_t	The value of channel plan, define in **wifi_constants.h**

18.2.4.4 wifi_get_channel_plan

Get channel plan from calibration section.

Parameter	Type	Introduction
<channel_plan>	uint8_t	Point to the value of channel plan, define in **wifi_constants.h**

18.2.5 Power APIs

API	Introduction
<wifi_enable_powersave>	Enable Wi-Fi power save mode. When power save mode is enabled, RF will wake up only when there is data to be sent or beacon to be received. Otherwise, RF will be awake all the time.
<wifi_disable_powersave>	Disable Wi-Fi power save mode.
<wifi_get_txpower>	Get the Tx power in index units.
<wifi_set_txpower>	Set the Tx power in index units.
<wifi_set_power_mode>	Set IPS/LPS mode.
<wifi_set_lps_dtim>	Set LPS DTIM.
<wifi_get_lps_dtim>	Get LPS DTIM.

18.2.5.1 wifi_enable_powersave

Enable Wi-Fi power save mode. When power save mode is enabled, RF will wake up only when there is data to be sent or beacon to be received. Otherwise, RF will be awake all the time.

Parameter	Type	Introduction
None	-	-

18.2.5.2 wifi_disable_powersave

Disable Wi-Fi power save mode.

Parameter	Type	Introduction
None	-	-

18.2.5.3 wifi_get_txpower

Get the Tx power in index units.

Parameter	Type	Introduction
<poweridx>	int *	The variable to receive the Tx power in index.

18.2.5.4 wifi_set_txpower

Set the Tx power in index units.

Parameter	Type	Introduction
<poweridx>	int	The desired Tx power in index.

18.2.5.5 wifi_set_power_mode

Set IPS/LPS mode.

Parameter	Type	Introduction
<ips_mode>	unsigned char	The desired IPS mode. It becomes effective when WLAN enters IPS. ips_mode is inactive power save mode. Wi-Fi automatically turns RF off if it is not associated to AP. Set 1 to enable inactive power save mode.
<lps_mode>	unsigned char	The desired LPS mode. It becomes effective when WLAN enters LPS. lps_mode is leisure power save mode. Wi-Fi automatically turns RF off during the association to AP is traffic is not busy while it also automatically turns RF on to listen to beacon. Set 1 to enable leisure power save mode.

18.2.5.6 wifi_set_lps_dtim

Set LPS DTIM.

Parameter	Type	Introduction
<dtim>	unsigned char	In LPS, the package can be buffered at AP side. STA leave LPS until DTIM count of packages buffered at AP side.

Note: DTIM is the duration to listen beacon. The default DTIM is 1. If DTIM is set bigger than 1, the advantage is that power can be saved and the disadvantage is that STA has the risk of losing packets.

18.2.5.7 wifi_get_lps_dtim

Get LPS DTIM.

Parameter	Type	Introduction
<dtim>	unsigned char *	In LPS, the package can be buffered at AP side. STA leave LPS until DTIM count of packages buffered at AP side.

18.2.6 AP Mode APIs

API	Introduction
<wifi_start_ap>	Trigger Wi-Fi driver to start an infrastructure Wi-Fi network.
<wifi_start_ap_with_hidden_ssid>	Start an infrastructure Wi-Fi network with hidden SSID.
<wifi_restart_ap>	Trigger Wi-Fi driver to restart an infrastructure Wi-Fi network.
<wifi_get_associated_client_list>	Get the associated clients with SoftAP.
<wifi_enable_forwarding>	Enable packets forwarding in AP mode.
<wifi_disable_forwarding>	Disable packets forwarding in AP mode.

18.2.6.1 wifi_start_ap

Trigger Wi-Fi driver to start an infrastructure Wi-Fi network.

Parameter	Type	Introduction
<ssid>	char *	A null terminated string containing the SSID name of the network.
<security_type>	security_type	● RTW_SECURITY_OPEN: Open Security ● RTW_SECURITY_WPA_TKIP_PSK: WPA Security ● RTW_SECURITY_WPA2_AES_PSK: WPA2 Security using AES cipher ● RTW_SECURITY_WPA2_MIXED_PSK: WPA2 Security using AES and/or TKIP ciphers WEP security is NOT IMPLEMENTED. It is NOT SECURE!
<password>	char *	A byte array containing the cleartext security key for the network.
<ssid_len>	int	The length of the SSID in bytes.
<password_len>	int	The length of the security_key in bytes.
<channel>	int	802.11 channel number.

Note:

- Make sure the Wi-Fi is enabled before invoking this function. (wifi_on())
- if a STA interface is active when this function is called, the softAP will start on the same channel as the STA. It will not use the channel provided.

18.2.6.2 wifi_start_ap_with_hidden_ssid

Start an infrastructure Wi-Fi network with hidden SSID.

Parameter	Type	Introduction
<ssid>	char *	A null terminated string containing the SSID name of the network.
<security_type>	security_type	● RTW_SECURITY_OPEN: Open Security ● RTW_SECURITY_WPA_TKIP_PSK: WPA Security ● RTW_SECURITY_WPA2_AES_PSK: WPA2 Security using AES cipher ● RTW_SECURITY_WPA2_MIXED_PSK: WPA2 Security using AES and/or TKIP ciphers WEP security is NOT IMPLEMENTED. It is NOT SECURE!
<password>	char *	A byte array containing the cleartext security key for the network.
<ssid_len>	int	The length of the SSID in bytes.
<password_len>	int	The length of the security_key in bytes.

<channel>	int	802.11 channel number.

Note: if a STA interface is active when this function is called, the softAP will start on the same channel as the STA. It will not use the channel provided.

18.2.6.3 wifi_restart_ap

Trigger Wi-Fi driver to restart an infrastructure Wi-Fi network.

Parameter	Type	Introduction
<ssid>	unsigned char *	A null terminated string containing the SSID name of the network.
<security_type>	rtw_security_t	● RTW_SECURITY_OPEN: Open Security ● RTW_SECURITY_WPA_TKIP_PSK: WPA Security ● RTW_SECURITY_WPA2_AES_PSK: WPA2 Security using AES cipher ● RTW_SECURITY_WPA2_MIXED_PSK: WPA2 Security using AES and/or TKIP ciphers WEP security is NOT IMPLEMENTED. It is NOT SECURE!
<password>	unsigned char *	A byte array containing the cleartext security key for the network.
< ssid_len>	int	The length of the SSID in bytes.
<password_len>	int	The length of the security_key in bytes.
<channel>	int	802.11 channel number.

Note:
- Make sure the Wi-Fi is enabled before invoking this function. (wifi_on())
- if a STA interface is active when this function is called, the softAP will start on the same channel as the STA. It will not use the channel provided.

18.2.6.4 wifi_get_associated_client_list

Get the associated clients with SoftAP.

Parameter	Type	Introduction
<client_list_buffer>	int *	The location where the client list will be stored.
<buffer_length>	unsigned short	The buffer length.

18.2.6.5 wifi_enable_forwarding

Enable packets forwarding in AP mode.

Parameter	Type	Introduction
None	-	-

18.2.6.6 wifi_disable_forwarding

Disable packets forwarding in AP mode.

Parameter	Type	Introduction
None	-	-

18.2.7 Custom IE APIs

API	Introduction
<wifi_add_custom_ie>	Setup custom IE list.
<wifi_update_custom_ie>	Update the item in Wi-Fi custom IE list.

<wifi_del_custom_ie>	Delete Wi-Fi custom IE list.

Note: These three APIs are only effective on beacon, probe request and probe response frames.

18.2.7.1 wifi_add_custom_ie

Setup custom IE list.

Parameter	Type	Introduction
<cus_ie>	void *	Pointer to Wi-Fi CUSTOM IE list.
<ie_num>	int	The number of Wi-Fi CUSTOM IE list.

Note:
- This API can't be executed twice before deleting the previous custom IE list.
- Defining CONFIG_CUSTOM_IE in **autoconf.h** needs to be done before compiling, or this API won't be effective.

18.2.7.2 wifi_update_custom_ie

Update the item in Wi-Fi custom IE list.

Parameter	Type	Introduction
<cus_ie>	void *	Pointer to Wi-Fi CUSTOM IE list.
<ie_index>	int	The number of Wi-Fi CUSTOM IE list.

Note: Defining CONFIG_CUSTOM_IE in **autoconf.h** needs to be done before compiling, or this API won't be effective.

18.2.7.3 wifi_del_custom_ie

Delete Wi-Fi custom IE list.

Parameter	Type	Introduction
None	-	-

Note: Defining CONFIG_CUSTOM_IE in **autoconf.h** needs to be done before compiling, or this API won't be effective.

18.2.8 Wi-Fi Setting APIs

API	Introduction
<wifi_get_mac_address>	Retrieves the current Media Access Control (MAC) address (or Ethernet hardware address) of the 802.11 device.
<wifi_get_ap_bssid>	Get connected AP's BSSID.
<wifi_get_ap_info>	Get the SoftAP information.
<wifi_set_country>	Set the country code to driver to determine the channel set.
<wifi_get_sta_max_data_rate>	Retrieved STA mode max. data rate.
<wifi_get_rssi>	Retrieved the latest RSSI value.
<wifi_register_multicast_address>	Register interest in a multicast address. Once a multicast address has been registered, all packets detected on the medium destined for that address are forwarded to the host. Otherwise they are ignored.
< wifi_unregister_multicast_address >	Unregister interest in a multicast address. Once a multicast address has been unregistered, all packets detected on the medium destined for that address are ignored.
<wifi_set_mib>	Setup the adaptive mode. You can replace this weak function by the same name function to setup adaptive mode you can.

<wifi_set_country_code>	Setup country code. You can replace this weak function by the same name function to setup country code you want.
<wifi_set_tdma_param>	Set TDMA parameters.
<wifi_get_setting>	Get current Wi-Fi setting from driver.
<wifi_show_setting>	Show the network information stored in a rtw_wifi_setting_t structure.
<wifi_set_network_mode>	Set the network mode according to the data rate it supported. Driver works in BGN mode in default after driver initialization. This function is used to change wireless network mode for station mode before connecting to AP.
<wifi_get_network_mode>	Get the network mode. Driver works in BGN mode in default after driver initialization. This function is used to get the current wireless network mode for station mode.
<wifi_get_antenna_info>	Get antenna information.
<wifi_set_ch_deauth>	Set flag for concurrent mode wlan1 issue_deauth when channel switched by wlan0. Usage: wifi_set_ch_deayth(0) -> wlan0 wifi_connect -> wifi_set_ch_deauth(1)

18.2.8.1 wifi_get_mac_address

Retrieve the current Media Access Control (MAC) address (or Ethernet hardware address) of the 802.11 device.

Parameter	Type	Introduction
<mac>	char *	Point to the result of the mac address will be get.

18.2.8.2 wifi_get_ap_bssid

Get connected AP's BSSID.

Parameter	Type	Introduction
<bssid>	unsigned char *	The location where the AP BSSID will be stored.

18.2.8.3 wifi_get_ap_info

Get the SoftAP information.

Parameter	Type	Introduction
<ap_info>	rtw_bss_info_t *	The location where the AP info will be stored.
<security>	rtw_security_t *	The security type.

18.2.8.4 wifi_set_country

Set the country code to driver to determine the channel set.

Parameter	Type	Introduction
<country_code>	rtw_country_code_t	Specify the country code.

18.2.8.5 wifi_get_sta_max_data_rate

Retrieved STA mode max. data rate.

Parameter	Type	Introduction
<inidata_rate>	_u8 *	Max data rate.

18.2.8.6 wifi_get_rssi

Retrieved the latest RSSI value.

Parameter	Type	Introduction
<pRSSI>	int *	Points to the integer to store the RSSI value gotten from driver.

18.2.8.7 wifi_register_multicast_address

Register interest in a multicast address.

Once a multicast address has been registered, all packets detected on the medium destined for that address are forwarded to the host, otherwise they are ignored.

Parameter	Type	Introduction
<mac>	rtw_mac_t *	Ethernet MAC address.

18.2.8.8 wifi_unregister_multicast_address

Unregister interest in a multicast address.

Once a multicast address has been unregistered, all packets detected on the medium destined for that address are ignored.

Parameter	Type	Introduction
<mac>	rtw_mac_t *	Ethernet MAC address.

18.2.8.9 wifi_set_mib

Setup the adaptive mode. You can replace this weak function by the same name function to setup adaptive mode you can.

Parameter	Type	Introduction
None	-	-

18.2.8.10 wifi_set_country_code

Setup country code. You can replace this weak function by the same name function to setup country code you want.

Parameter	Type	Introduction
None	-	-

18.2.8.11 wifi_set_tdma_param

Set TDMA parameters.

Parameter	Type	Introduction
<slot_period>	unsigned char	We separate TBTT into 2 or 3 slots. If we separate TBTT into 2 slots, then slot_period should be larger or equal to 50ms. It means 2 slot period is slot_period, 100-slot_period If we separate TBTT into 3 slots, then slot_period should be less or equal to 33ms. It means 3 slot period is 100 - 2 * slot_period, slot_period, slot_period.
<rfon_period_len_1>	unsigned char	RF on period of slot 1.
<rfon_period_len_2>	unsigned char	RF on period of slot 2.
<rfon_period_len_3>	unsigned char	RF on period of slot 3.

18.2.8.12 wifi_get_setting

Get current Wi-Fi setting from driver.

Parameter	Type	Introduction
<ifname>	const char *	The WLAN interface name, can be WLAN0_NAME or WLAN1_NAME.
<pSetting>	rtw_wifi_setting_t *	Points to the rtw_wifi_setting_t structure to store the Wi-Fi setting gotten from driver.

18.2.8.13 wifi_show_setting

Show the network information stored in a rtw_wifi_setting_t structure.

Parameter	Type	Introduction
<ifname>	const char *	The WLAN interface name, can be WLAN0_NAME or WLAN1_NAME.
<pSetting>	rtw_wifi_setting_t *	Points to the rtw_wifi_setting_t structure which information is gotten by wifi_get_setting().

18.2.8.14 wifi_set_network_mode

Set the network mode according to the data rate it supported. Driver works in BGN mode in default after driver initialization. This function is used to change wireless network mode for station mode before connecting to AP.

Parameter	Type	Introduction
<mode>	rtw_network_mode_t	Network mode to set. The value can be RTW_NETWORK_B/RTW_NETWORK_BG/RTW_NETWORK_BGN

18.2.8.15 wifi_get_network_mode

Get the network mode. Driver works in BGN mode in default after driver initialization. This function is used to get the current wireless network mode for station mode.

Parameter	Type	Introduction
<pmode>	rtw_network_mode_t *	Network mode to get.

18.2.8.16 wifi_get_antenna_info

Get antenna information.

Parameter	Type	Introduction
<antenna>	unsigned char *	Points to store the antenna value gotten from driver. ● 0: main ● 1: aux

18.2.8.17 wifi_set_ch_deauth

Set flag for concurrent mode wlan1 issue_deauth when channel switched by wlan0.

Usage: wifi_set_ch_deayth(0) -> wlan0 wifi_connect -> wifi_set_ch_deauth(1)

Parameter	Type	Introduction
<enable>	_u8	0 for disable and 1 for enable.

18.2.9 Wi-Fi Indication APIs

API	Introduction
<wifi_manager_init>	Initialize Realtek Wi-Fi API System.
<wifi_indication>	WLAN driver indicate event to upper layer through wifi_indication.
<init_event_callback_list>	Initialize the event callback list.
<wifi_reg_event_handler>	Register the event listener.
<wifi_unreg_event_handler>	Un-register the event listener.

18.2.9.1 wifi_manager_init

Initialize Realtek Wi-Fi API System.

Parameter	Type	Introduction
None	-	-

18.2.9.2 wifi_indication

WLAN driver indicates event to upper layer through wifi_indication().

Parameter	Type	Introduction
<event>	rtw_event_indicate_t	An event reported from driver to upper layer application. Refer to rtw_event_indicate_t enum.
<buf>	char *	If it is not NUL, buf is a pointer to the buffer for message string.
<buf_len>	int	The length of the buffer.
<flags>	int	Indicate some extra information, sometimes it is 0.

Note: If upper layer application triggers additional operations on receiving of wext_wlan_indicate, please strictly check current stack size usage (by using uxTaskGetStackHighWaterMark()), and tries not to share the same stack with WLAN driver if remaining stack space is not available for the following operations.
Ex.: Using semaphore to notice another thread instead of handing event directly in **wifi_indication()**.

18.2.9.3 init_event_callback_list

Initialize the event callback list.

Parameter	Type	Introduction
None	-	-

Note: Make sure this function has been invoked before using the event handler related mechanism.

18.2.9.4 wifi_reg_event_handler

Register the event listener.

Parameter	Type	Introduction
<event_cmds>	unsigned int	The event command number indicated.
<handler_func>	rtw_event_handler_t	The callback function which will receive and process the event.
<handler_user_data>	void *	User specific data that will be passed directly to the callback function.

Note: set the same even_cmds with empty handler_func will unregister the event_cmds.

18.2.9.5 wifi_unreg_event_handler

Un-register the event listener.

Parameter	Type	Introduction
<event_cmds>	unsigned int	The event command number indicated.
<handler_func>	rtw_event_handler_t	The callback function which will receive and process the event.

18.2.10 eFuse writing APIs

API	Introduction
<wifi_set_mac_address>	This function sets the current Media Access Control (MAC) address of the 802.11 device.

Note: eFuse writing related API can only be called when the voltage is 3.3V. eFuse writing with the voltage 1.8V may fail. What's more, eFuse writing can only be operated one time, so be careful to do eFuse writing.

18.2.10.1 wifi_set_mac_address

Sets the current Media Access Control (MAC) address of the 802.11 device.

Parameter	Type	Introduction
<mac>	char *	Wi-Fi MAC address.

18.3 Fast Connect

This section illustrates principle of fast connect and how to implement user's own fast connect code.

Fast connect is used to reconnect with AP automatically after Wi-Fi initialized, the principle is to store the AP information in flash and reconnect to AP after Wi-Fi initialized.

18.3.1 Implement

18.3.1.1 Store AP Information

User should implement a function to write AP information to flash, just like demo function **wlan_wrtie_reconnect_data_to_flash()** in example code. In this function, you should reserve some space for AP information, and write the AP information to the reserved space in a pre-defined data format. The address of the function must be assigned to the global variable **p_write_reconnect_ptr**. After Wi-Fi connection success, if **p_write_reconnect_ptr** points to a valid address, **wlan_wrtie_reconnect_data_to_flash()** will be called.

Note: The path of example source code is **SDK/component/common/example/example_wlan_fast_connect/example_wlan_fast_connect.c.**

18.3.1.2 Reconnect

User should implement his own function to read AP information from flash and connect to AP, just like demo function **wlan_init_done_callback()** in example code. The address of the function must be assigned to the global variable **p_wlan_init_done_callback**. This global variable should be defined before Wi-Fi initializing. After Wi-Fi initialized, if **p_wlan_init_done_callback** points to a valid address, this function will be called.

18.3.1.3 Erase fast connect data

User should implement his own function to erase fast connect data, just like demo function **Erase_Fastconnect_data()** in example code.

18.3.2 APIs

API	Introduction
<wlan_wrtie_reconnect_data_to_flash>	Wi-Fi connection indication trigger this function to save current Wi-Fi profile in flash.
<wlan_init_done_callback>	After Wi-Fi initialization done, WLAN driver calls this function to check whether auto-connect is required. This function reads previous saved WLAN profile in flash and execute connection.

18.3.2.1 wlan_wrtie_reconnect_data_to_flash

Wi-Fi connection indications trigger this function to save current Wi-Fi profile in flash. Write AP information to flash.

Parameter	Type	Introduction
<data>	u8 *	Data will be written to flash
<len>	uint32_t	Length of data
return	int	Status value: ● 0: write is ok ● -1: write is failed

Note: This is only a demo API, user should define his own API.

18.4 WPS APIs

API	Introduction
<wps_start>	Start WPS enrollee process.
<wps_stop>	Stop WPS enrollee process.

18.4.1 wps_start

Start WPS enrollee process.

Parameter	Type	Introduction
<wps_config>	u16	WPS configure method: ● WPS_CONFIG_DISPLAY ● WPS_CONFIG_KEYPAD ● WPS_CONFIG_PUSHBUTTON
<pin>	char *	PIN number. Can be set to NULL if using WPS_CONFIG_PUSHBUTTON.
<channel>	u8	Channel. Currently un-used, can be set to 0.
<ssid>	char *	Target network SSID. Can be set to NULL if no target network specified.

Note:
- Before invoking this function, the Wi-Fi should be enabled by using **wifi_on()**.
- Make sure CONFIG_ENABLE_WPS is enabled in **platform_opts.h**. After calling wps_start(), the longest time of WPS is 120s. You can call wps_stop() to quit WPS.

18.4.2 wps_stop

Stop WPS enrollee process.

Parameter	Type	Introduction
None	-	-

Note: Make sure CONFIG_ENABLE_WPS is enabled in **platform_opts.h**.

19 Inter Processor Communication (IPC)

Inter Processor Communication (IPC) hardware is designed for the purpose of making two CPUs communicate with each other. The system architecture is shown in Fig 19-1.

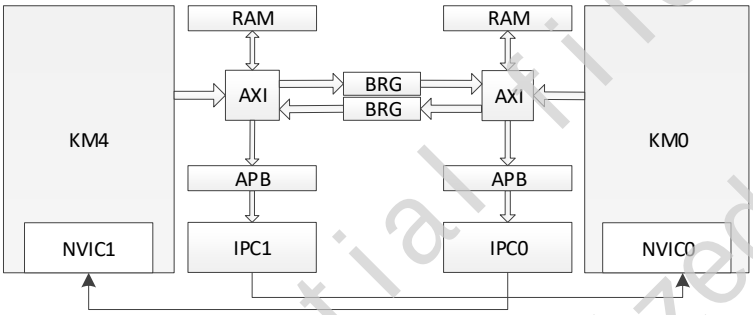


Fig 19-1 IPC system architecture

If you want KM0 to execute its IPC handler, IPC interrupt of KM4 should be enable and the status should be set, vice versa.

19.1 How to Use IPC?

It is complex because both KM0 and KM4 should be set to realize IPC. SDK has provided API and configuration file to simplify the procedure. Take KM4 requests an interrupt into direction KM0 for example.
Add new item in KM0 core section of **ipc_init_config[]** which is in **rtl8721d_ipccfg.c** for selected channel. IRQ handler and data should be set and defined. If message exchange is needed, you should specify the message type: data or point.

```
const IPC_INIT_TABLE ipc_init_config[] =
{
#if defined (ARM_CORE_CM4)
    //USER_MSG_TYPE     IRQFUNC                         IRQDATA
    {IPC_USER_DATA,     (VOID*) shell_switch_ipc_int,   (VOID*) NULL},//channel 0: IPC_INT_CHAN_SHELL_SWITCH
    {IPC_USER_DATA,     (VOID*) NULL,                   (VOID*) NULL},//channel 1: IPC_INT_CHAN_WIFI_FW
    {IPC_USER_DATA,     (VOID*) FLASH_Write_IPC_Int,    (VOID*) NULL},//channel 2: IPC_INT_CHAN_FLASHPG_REQ
    {IPC_USER_POINT,    (VOID*) NULL,                   (VOID*) NULL},//channel 3: IPC_INT_KM4_TICKLESS_INDICATION
#else
    //USER_MSG_TYPE     IRQFUNC                         IRQDATA
    {IPC_USER_DATA,     (VOID*) shell_switch_ipc_int,   (VOID*) NULL},//channel 0: IPC_INT_CHAN_SHELL_SWITCH
    {IPC_USER_DATA,     (VOID*) driver_fw_flow_ipc_int, (VOID*) IPCM4_DEV},//channel 1: IPC_INT_CHAN_WIFI_FW
    {IPC_USER_DATA,     (VOID*) FLASH_Write_IPC_Int,    (VOID*) NULL},//channel 2: IPC_INT_CHAN_FLASHPG_REQ
    {IPC_USER_POINT,    (VOID*) km4_tickless_ipc_int,   (VOID*) NULL},//channel 3: IPC_INT_KM4_TICKLESS_INDICATION
#endif
    {0xFFFFFFFF,        OFF,                            OFF},    /* Table end */
};
```

SDK will enable the IPC interrupt of KM4 for the channel according to **ipc_init_config[]**, and register the corresponding KM0 IRQ handler and data for the channel.
When KM4 wants to request an interrupt into direction KM0, it should call **ipc_send_message()** and specify the channel number and message. Corresponding KM0 IRQ handler will be executed, call **ipc_get_message()** to get message if need.

Note: Use IPC channel 16~31 and IPC semaphore index 8~15, Channel 0~15 and semaphore index 0~7 are reserved for Realtek use.

19.2 IPC Program APIs

19.2.1 ipc_table_init

Items	Description
Introduction	Initialize IPC channels according to **ipc_init_config[]**
Parameters	N/A

Return	N/A

19.2.2 ipc_send_message

Items	Description
Introduction	Request interrupt to target CPU by specified channel and exchange messages between KM0 and KM4
Parameters	● IPC_ChNum: the IPC channel number ● Message: the message to be exchanged, and should not be stored in stack
Return	N/A

Note:
- The message supports two types: data or point. The point is used to point to complex message structure.
- The message is shared between two CPUs, so the point can't be stored in stack.

19.2.3 ipc_get_message

Items	Description
Introduction	Get IPC message from specified channel.
Parameters	IPC_ChNum: the IPC channel number
Return	Message: the message to be exchanged

Note: Cache in the system is write-through type, so the corresponding data cache must be invalidated before data fetch.

19.3 IPC Driver Code

19.3.1 IPC_INTConfig

Items	Description
Introduction	Enable or disable the specified IPC channel interrupts.
Parameters	● IPC*x*: ■ IPCM0_DEV for CM0 ■ IPCM4_DEV for CM4 ● IPC_ChNum: the channel that want to be set. This parameter must be set to a value of 0 ~ 31. ● NewState: ■ ENABLE ■ DISABLE
Return	N/A

19.3.2 IPC_IERSet

Items	Description
Introduction	Set IER of specified IPC channel interrupts
Parameters	● IPC*x*: ■ IPCM0_DEV for CM0 ■ IPCM4_DEV for CM4 ● IPC_Ch: the channel that want to be enable
Return	N/A

19.3.3 IPC_IERGet

Items	Description
Introduction	Get IER of specified IPC*x*
Parameters	IPC*x*: ● IPCM0_DEV for CM0 ● IPCM4_DEV for CM4
Return	The IER of specified IPC*x*

19.3.4 IPC_INTRequest

Items	Description
Introduction	Request a core-to-core interrupt
Parameters	● IPC*x*: ■ IPCM0_DEV for CM0 ■ IPCM4_DEV for CM4 ● IPC_ChNum: the channel that want to request interrupt. This parameter must be set to a value of 0 ~ 31.
Return	N/A

19.3.5 IPC_INTClear

Items	Description
Introduction	Clear a core-to-core interrupt
Parameters	● IPC*x*: ■ IPCM0_DEV for CM0 ■ IPCM4_DEV for CM4 ● IPC_ChNum: the channel that want to clear interrupt. This parameter must be set to a value of 0 ~ 31.
Return	N/A

19.3.6 IPC_INTGet

Items	Description
Introduction	Get a core-to-core interrupt status
Parameters	IPC*x*: ● IPCM0_DEV for CM0 ● IPCM4_DEV for CM4
Return	The ISR of specified IPC*x*

19.3.7 IPC_CPUID

Items	Description
Introduction	Get the current CPU ID
Parameters	N/A
Return	CPU ID: ● 0: KM0 ● 1: KM4

19.3.8 IPC_SEMGet

Items	Description

Introduction	Get a core-to-core hardware semaphore
Parameters	SEM_Idx: the semaphore index that want to get. This parameter must be set to a value of 0 ~ 15. ● 0 ~ 7: Reserved for Realtek use ● 8 ~ 15: Reserved for Customer use
Return	Result: ● TRUE ● FALSE

19.3.9 IPC_SEMFree

Items	Description
Introduction	Free a core-to-core hardware semaphore
Parameters	SEM_Idx: the semaphore index that want to free. This parameter must be set to a value of 0 ~ 15. ● 0 ~ 7: Reserved for Realtek use ● 8 ~ 15: Reserved for Customer use
Return	Result: ● TRUE ● FALSE

19.3.10 IPC_INTHandler

Items	Description
Introduction	The common IPC interrupt handler
Parameters	Data: the data passed to the IRQ Handler
Return	0

Note: Before entering the specified IRQ handler, the common IPC interrupt handler clears the pending interrupt first. There is no need to clear the pending interrupt in user defined IRQ handler.

19.3.11 IPC_INTUserHandler

Items	Description
Introduction	Rregister a user interrupt handler for a specified IPC channel
Parameters	● IPC_ChNum: the channel that want to register IRQ handler. This parameter must be set to a value of 0 ~ 31. ● IrqHandler: the IRQ handler to be assigned to the specified IPC channel ● IrqData: the pointer will be passed to the IRQ handler
Return	N/A

20 PSRAM

Pseudo Static Random Access Memory (PSRAM) is used for high speed transmission of data stream, and is suitable for audio codec. RTL8721D uses PSRAM controller to communicate with PSRAM. PSRAM is in KM4 platform, so only KM4 can access it.

The features of PSRAM are:

- Density: 32Mbit
- Address Mapping: 0x0200_0000 ~ 0x0240_0000
- Clock rata: 50MHz
- Double Data Rate (DDR)
- 16/32/64/128 bytes burst access
- Half sleep mode and deep power-down mode

20.1 Throughput

PSRAM supports direct access and DMA access.

Table 20-1 PSRAM throughput

Access Mode		**Write**		**Read**	
		Theory	Test	Theory	Test
Direct Access	Cache off	200Mbps	< 160Mbps	177.78Mbps	< (32)/(180ns+360ns) = 59.26 Mbps
	Cache on	200Mbps	160Mbps (CS is high for 40ns between two transmits)	556.52Mbps	< (32*8)/(460ns+360ns) = 312Mbps
DMA		731.43Mbps	711.11Mbps	721.13Mbps	589.18Mbps

Table 20-2 PSRAM throughput theoretical calculation

Time	**Write 32 bits**	**Read 32 bits**
Header + delay	[3 + (3-1)] * 20ns = 100ns	[3 + (3-1)] * 20ns = 100ns
Data transmit period	2 * 20ns = 40ns	16 * 20ns = 320ns
Hardware hold	1 * 20ns = 20ns	2 * 20ns = 40ns
Total without consider instruction execution time	100ns + 40ns + 20ns = 160ns	100ns + 320ns + 40ns = 460ns
Throughput theoretical value	32/160ns = 200Mbps	(32*8)/460ns = 556.52Mbps

Note:

- When Cache off:
  - Read or write operation only accesses 32 bits once with header and delay.
  - Instruction execution time also needs to take into consideration.
  - The 360ns in test is caused by hardware characteristic. Command from CPU to PSRAM controller needs 152ns to sync, PSRAM controller controlling PHY circuit to work needs 82us, PHY giving read data to CPU needs 126ns.
- When Cache on:
  - Read operation reads 32 bytes (cache line) once.
  - Write operation writes 32 bits once, but the interval between two write operations is greatly reduced.
  - The 360ns in test is caused by hardware characteristic. Command from CPU to PSRAM controller needs 152ns to sync, PSRAM controller controlling PHY circuit to work needs 82us, PHY giving read data to CPU needs 126ns.
- DMA sets burst length 128 bytes and disables cache.
  - For DMA write, DAM moving data to PSRAM FIFO and PSRAM controller writing FIFO data to PSRAM slave can work simultaneously, so the interval between two burst write operations is small.
  - For DMA read, similar operation as DMA write is not supported, so the interval between two burst read operations is large.
  - The test data above takes variable initial latency and 3 clocks initial latency for example.

20.2 Power Management

When entering KM4 powergate mode, PSRAM will be reset and its memory will be lost. If you want to retain the PSRAM, KM4 powergate mode is not supported to use. If you want to keep the PSRAM and save power, KM4 clockgate mode is supported.

Table 20-3 PSRAM power management

Retention	Options	Power Consumption (uA)	Comment
NO	● KM4 PG ● PSRAM_LDO OFF	30 ~ 50	PSRAM cannot keep. So memory retention applications are not supported.
Yes	● KM4 CG ● PSRAM_LDO ON ● PSRAM Half Sleep mode	400+	

20.3 How to Use PSRAM?

20.3.1 Initialize PSRAM

Before accessing PSRAM, PSRAM power should be enabled, PSRAM controller and PSRAM slave should be initialized to synchronize related parameters.

In SDK, you should set **psram_dev_enable** in **rtl8721dhp_intfcfg.c**. If the chip works in the environment with large fluctuations in temperature, you should also set **psram_dev_cal_enable** to enable calibration function. If you want to keep the PSRAM and save power when KM4 enters sleep mode, you should set **psram_dev_retention** to enable PSRAM retention.

```
PSRAMCFG_TypeDef psram_dev_config = {
    .psram_dev_enable = TRUE,                   //enable psram
    .psram_dev_cal_enable = TRUE,               //enable psram calibration function
    .psram_dev_retention = TRUE,                //enable psram retention
    .psram_heap_start_address = 0x02000400,     //config psram heap start address, should be 8 bytes aligned
    .psram_heap_size = 0x400000-0x400,          //config psram heap size, should be 8 bytes aligned
};
```

Fig 20-1 PSRAM initial configuration

20.3.2 Add Section into PSRAM Region

(1) To add section into PSRAM region, the following steps are necessary.
(2) To put a data buffer in PSRAM, add **PSRAM_BSS_SECTION** before the buffer definition.

```
PSRAM_BSS_SECTION u32 PSRAM_Testbuf[1024];
```

(3) Rebuild KM4 project.

20.4 How to Allocate Heap from PSRAM?

Before allocating heap, Initialize PSRAM must be completed, then follow the steps below.
(1) Confirm the heap size and the start address you want in **rtl8721dhp_intfcfg.c**, **psram_heap_start_address** and **psram_heap_size** must be 8 bytes aligned.

```
PSRAMCFG_TypeDef psram_dev_config = {
    .psram_dev_enable = FALSE,                  //enable psram
    .psram_dev_cal_enable = FALSE,              //enable psram calibration function
    .psram_dev_retention = FALSE,             //enable psram retention
    .psram_heap_start_address = 0x02000400,     //config psram heap start address, should be 8 bytes aligned
    .psram_heap_size = 0x400000- 0x400,             //config psram heap size, should be 8 bytes aligned
};
```

(2) Use **void *Psram_reserve_malloc(int size)** to allocate a heap or use **void *Psram_reserve_calloc(int num, int size)** to allocate several consecutive spaces. Both functions return a pointer to the start address of the allocation. Use **void Psram_reserve_free(void *mem)** to free the allocation.

Note：

- The **psram_heap_start_address** must be larger than the __psram_bss_end__. The **psram_heap_start_address** must be increased when "psram_heap_start_address should larger than …" is shown, and the **psram_heap_size** should be decreased depending on the **psram_heap_start_address** configured.
- The PSRAM_BSS_SECTION will be cleared only in the function **void app_init_psram(void)** of **main.c** (in KM4).

21 MPU and Cache

21.1 Functional description

21.1.1 MPU

Memory Protection Unit (MPU) is used to provide Hardware Protection by Software definition. Our code provides **mpu_region_config** structure to include the region memory attribute. Default attribute of all KM0 & KM4 SRAM are write-through.

Table 21-1 shows member variables of the **mpu_region_config** structure.

Table 21-1 mpu_region_config structure

Member Variable Name	Type	Description
region_base	uint32_t	MPU region base, 32 bytes aligned
region_size	uint32_t	MPU region size, 32 bytes aligned
xn	uint8_t	Execute Never attribute ● MPU_EXEC_ALLOW: Allows program execution in this region ● MPU_EXEC_NEVER: Does not allow program execution in this region
ap	uint8_t	Access permissions ● MPU_PRIV_RW: Read/write by privileged code only ● MPU_UN_PRIV_RW: Read/write by any privilege level ● MPU_PRIV_R: Read only by privileged code only ● MPU_PRIV_W: Read only by any privilege level
sh	uint8_t	Share ability for Normal memory ● MPU_NON_SHAREABLE: Non-shareable ● MPU_OUT_SHAREABLE: Outer shareable ● MPU_INR_SHAREABLE: Inner shareable
attr_idx	uint8_t	Memory attribute indirect index This parameter can be a value of 0 ~ 7, the detailed attribute is defined in **mpu_init()** and is customized. The typical definition is as following: ● 0: MPU_MEM_ATTR_IDX_NC, defines memory attribute of Normal memory with non-cacheable ● 1: MPU_MEM_ATTR_IDX_WT_T_RA, defines memory attribute of Normal memory with write-through transient, read allocation ● 2: MPU_MEM_ATTR_IDX_WB_T_RWA, defines memory attribute of Normal memory with write-back transient, read and write allocation ● 3 ~ 7: MPU_MEM_ATTR_IDX_DEVICE, defines memory attribute of Device memory with non-gathering, non-recording, non-early Write Acknowledge

Table 21-2 shows how to set a MPU region.

Table 21-2 How to set a MPU region

Steps	Description
New variable and structure	● Variable to store MPU entry index ● Structure **mpu_region_config** to store the region attribution
Allocate a free MPU entry	Call **mpu_entry_alloc()**
Set region attribution	Set region attribution structure
Configure MPU region memory attribute	Call **mpu_region_cfg()**

21.1.2 Cache

Cache is used to improve the CPU performance of data access. Ameba-D Cache supports Enable/Disable, Flush and Clean Operation, as Table 21-3 lists.

Table 21-3 Enable/disable, flush and clean operation supported by Cache

Operation	Description	I-Cache	D-Cache
Enable/Disable	Enable or Disable Cache function	✓	✓
Flush (Invalidate)	● Flush Cache ● D-Cache can be flushed by address ● Can be used after DMA Rx, and CPU reads DMA data from DMA buffer for D-Cache	✓	✓
Clean	● Clean D-Cache ● D-Cache will be write back to memory ● D-Cache can be cleaned by address ● Can be used before DMA Tx, after CPU writes data to DMA buffer for D-Cache	×	✓

21.2 MPU APIs

21.2.1 mpu_init

Items	Description
Introduction	Initialize MPU memory attribute to typical value
Parameters	N/A
Return	N/A

21.2.2 mpu_set_mem_attr

Items	Description
Introduction	Change MPU memory attribute
Parameters	● attr_idx: memory attribute index, where *x* can be 0 ~ 7. ● mem_attr: memory attributes.
Return	N/A

21.2.3 mpu_region_cfg

Items	Description
Introduction	Configure MPU region memory attribute.
Parameters	● region_num: where x can be 0 ~ 3 (KM0 & KM4 both). ● pmpu_cfg: pointer to an **mpu_region_config** structure which has been configured
Return	N/A

21.2.4 mpu_entry_free

Items	Description
Introduction	Free MPU entry
Parameters	entry_index: ● KM0: 0 ~ 3 ● KM4_NS: 0 ~ 7 ● KM4_S: 0 ~ 3
Return	N/A

21.2.5 mpu_entry_alloc

Items	Description
Introduction	Allocate a free MPU entry

Parameters	N/A
Return	MPU entry index: ● KM0: 0 ~ 3 ● KM4_NS: 0 ~ 7 ● KM4_S: 0 ~ 3 ● Fail: -1

21.3 Cache APIs

21.3.1 ICache_Enable

Items	Description
Introduction	Enable I-Cache
Parameters	N/A
Return	N/A

21.3.2 ICache_Disable

Items	Description
Introduction	Disable I-Cache
Parameters	N/A
Return	N/A

21.3.3 ICache_Invalidate

Items	Description
Introduction	Invalidate I-Cache
Parameters	N/A
Return	N/A

21.3.4 DCache_IsEnabled

Items	Description
Introduction	Check D-Cache Enabled or not
Parameters	N/A
Return	D-Cache enable status: ● 1: Enable ● 0: Disable

21.3.5 DCache_Enable

Items	Description
Introduction	Enable D-Cache
Parameters	N/A
Return	N/A

21.3.6 DCache_Disable

Items	Description

Introduction	Disable D-Cache
Parameters	N/A
Return	N/A

21.3.7 DCache_Invalidate

Items	Description
Introduction	D-Cache invalidate by address
Parameters	● Address: invalidate address (aligned to 32-byte boundary) ● Bytes: size of memory block (in number of bytes)
Return	N/A

21.3.8 DCache_Clean

Items	Description
Introduction	D-Cache clean by address
Parameters	● Address: clean address (aligned to 32-byte boundary). Address set 0xFFFFFFFF is used for all D-Cache be cleaned. ● Bytes: size of memory block (in number of bytes).
Return	N/A

21.3.9 DCache_CleanInvalidate

Items	Description
Introduction	D-Cache clean and invalidate by address
Parameters	● Address: clean and invalidate address (aligned to 32-byte boundary) Address set 0xFFFFFFFF is used for all D-Cache be cleaned and flushed ● Bytes: size of memory block (in number of bytes)
Return	N/A

21.4 How to Define a Non-cacheable Data Buffer?

To define a data buffer with non-cacheable attribute, you should add **SRAM_NOCACHE_DATA_SECTION** before the buffer definition. The non-cacheable buffer initialization is shown as below.

```
SRAM_NOCACHE_DATA_SECTION u8 noncache_buffer[DATA_BUFFER_SIZE];
```

22 Liquid Crystal Display Controller (LCDC)

Ameba-D LCDC supports Thin Film Transistor (TFT) color display. It provides I8080 MCU interface (8-/16-bit bus width) and RGB interface (6-/16-bit bus width), and supports RGB565 data format

This chapter introduces how to use LCDC I/F to control LCD module.

22.1 Interface

The available LCDC interfaces for different IC packages are shown in Table 22-1.

Table 22-1 LCDC I/F supported for different packages

IC Package	MCU 8-bit	RGB 6-bit	MCU 16-bit (TBD)	RGB 16-bit (TBD)
RTL8722D/RTL8722CS	Yes	Yes	No	No
RTL8721D/RTL8721CS/RTL8720D/RTL8720CS	No	No	No	No

The data format supported by each interface is shown in Table 22-2. In order to display normally, the data format of LCM and LCDC interface should match.

Table 22-2 LCDC I/F data format

Interface	Data Format	Comment
MCU 8-bit	Count 0 1 2 3 4 … RS 0 1 1 1 1 … D7 C7 P1R4 P1G2 P2R4 P2G2 … D6 C6 P1R3 P1G1 P2R3 P2G1 … D5 C5 P1R2 P1G0 P2R2 P2G0 … D4 C4 P1R1 P1B4 P2R1 P2B4 … D3 C3 P1R0 P1B3 P2R0 P2B3 … D2 C2 P1G5 P1B2 P2G5 P2B2 … D1 C1 P1G4 P1B1 P2G4 P2B1 … D0 C0 P1G3 P1B0 P2G3 P2B0 … P1R4 : **P**ixel**1**/**R**ed_bit**4**	2 transfers/pixel, RGB565
MCU 16-bit	Count 0 1 2 … RS 0 1 1 … D15 P1R4 P2R4 … D14 P1R3 P2R3 … D13 P1R2 P2R2 … D12 P1R1 P2R1 … D11 P1R0 P2R0 … D10 P1G5 P2G5 … D9 P1G4 P2G4 … D8 P1G3 P2G3 … D7 C7 P1G2 P2G2 … D6 C6 P1G1 P2G1 … D5 C5 P1G0 P2G0 … D4 C4 P1B4 P2B4 … D3 C3 P1B3 P2B3 … D2 C2 P1B2 P2B2 … D1 C1 P1B1 P2B1 … D0 C0 P1B0 P2B0 … P1R4 : **P**ixel**1**/**R**ed_bit**4**	1 transfer/pixel, RGB565

RGB 6-bit

Count	0	1	2	3	4	…
D5	P1R4	P1G5	P1B4	P2R4	P2G5	…
D4	P1R3	P1G4	P1B3	P2R3	P2G4	…
D3	P1R2	P1G3	P1B2	P2R2	P2G3	…
D2	P1R1	P1G2	P1B1	P2R1	P2G2	…
D1	P1R0	P1G1	P1B0	P2R0	P2G1	…
D0		P1G0			P2G0	…

P1R4 : **P**ixel**1**/**R**ed_bit**4**

3 transfers/pixel, RGB565

RGB 16-bit

Count	0	1	…
D15	P1R4	P2R4	…
D14	P1R3	P2R3	…
D13	P1R2	P2R2	…
D12	P1R1	P2R1	…
D11	P1R0	P2R0	…
D10	P1G5	P2G5	…
D9	P1G4	P2G4	…
D8	P1G3	P2G3	…
D7	P1G2	P2G2	…
D6	P1G1	P2G1	…
D5	P1G0	P2G0	…
D4	P1B4	P2B4	…
D3	P1B3	P2B3	…
D2	P1B2	P2B2	…
D1	P1B1	P2B1	…
D0	P1B0	P2B0	…

P1R4 : **P**ixel**1**/**R**ed_bit**4**

1 transfer/pixel, RGB565

22.2 Resolution

The maximum resolution supported by each interface is listed in Table 22-3.

Table 22-3 Maximum supported resolution

Interface	Display Type	Max. Support Resolution
MCU 8-bit	Static Display	1024*1024
	Animate Display	645*645 (30fps)
MCU 16-bit	Static Display	1024*1024
	Animate Display	912*912 (30fps)
RGB 6-bit	Animate Display	600*400 (60fps)
RGB 16-bit	Animate Display	1024*760 (60fps)

For RGB LCD, the typical frame rate is 60fps. As a result, the parameters in Table 22-1 are given in the case of refresh frequency 60Hz.

In order to support RGB-LCD with higher resolution than parameters in Table 22-1, users have to set the frame rate lower. The maximum frame rate for a RGB-LCD is calculated as follows:

- Max. dot clock: MAX_DOT_CLK = system_clock/2 = 100M/2 = 50MHz
- Width, height, HBP, HFP, VBP, VFP are specified by LCD datasheet.

For 16-bit I/F mode:

image_size = MAX_DOT_CLK / F – (width + HBP + HFP) * (VBP+VFP) – height * (HBP + HFP);

➔ F = 50M / (width * height + (width + HBP + HFP) * (VBP+VFP) + height * (HBP + HFP));

For 6-bit I/F mode:

image_size = MAX_DOT_CLK / (3*F) – (width + HBP + HFP) * (VBP+VFP) - height * (HBP + HFP);
➔ F = 50M / (width * height + (width + HBP + HFP) * (VBP+VFP) + height * (HBP + HFP)) / 3;

When frame rate is lower than 30fps, the screen flickering may happen. Users should evaluate the visual artifacts when setting the frame rate lower.

22.3 Pinmux

The pin assignments of LCDC are listed in Table 22-4.

Table 22-4 LCDC pin assignments

Port Name	MCU 8-bit	MCU 16-bit	RGB 6-bit	RGB 16-bit	LED I/F
PB[19]	-	D[15]	-	D[15]	-
PB[18]	-	D[14]	-	D[14]	-
PB[11]	-	D[13]	-	D[13]	-
PB[10]	-	D[12]	-	D[12]	-
PB[9]	-	D[11]	-	D[11]	-
PB[8]	-	D[10]	-	D[10]	-
PA[25]	-	D[9]	-	D[9]	-
PA[26]	-	D[8]	-	D[8]	-
PA[28]	D[7]	D[7]	-	D[7]	-
PA[30]	D[6]	D[6]	-	D[6]	-
PB[0]	D[5]	D[5]	D[5]	D[5]	D[5]
PA[31]	D[4]	D[4]	D[4]	D[4]	D[4]
PA[24]	D[3]	D[3]	D[3]	D[3]	D[3]
PA[23]	D[2]	D[2]	D[2]	D[2]	D[2]
PA[20]	D[1]	D[1]	D[1]	D[1]	D[1]
PA[19]	D[0]	D[0]	D[0]	D[0]	D[0]
PB[20]	TE/VSYNC		VSYNC		-
PB[21]	RS		-		
PB[22]	RD		HSYNC		LAT
PB[23]	WR		DCLK		DCLK
PB[28]	CS		ENABLE		OE

22.4 JPG Decompressor

The picture of bitmap format can be displayed on LCD directly after transmitting pixel data into LCM. This format contains color bits of each pixel, causing the file size too large and taking too much memory.

JPEG is a commonly used method of lossy compression for digital images. The storage size decreases with little perceptible loss in image quality after compression.

Ameba-D provides SW JPG decoder for .jpg file decompression. Users can refer to JPG decode demo to know how to use. The following table shows the performance of decoding test .jpg whose resolution is 250*250.

Resolution	Memory Size	Time Consumption
250*250	Heap: 3K	70ms

22.5 LCDC APIs

22.5.1 MCU Function

22.5.1.1 LCDC_MCUStructInit

Items	Description
Introduction	Initializes the parameters in the LCDC_MCUInitStruct with default values.
Parameters	LCDC_MCUInitStruct: pointer to an LCDC_MCUInitTypeDef structure which is initialized.
Return	N/A

Note: LCDC_MCUInitStruct contains the MCU configurable parameters for LCDC, which determine that the MCU I/F is 8-bit or 16-bit, IO mode or DMA mode, the plane size and some timing control. The parameters in this structure should be set according to LCD module datasheet.

22.5.1.2 LCDC_MCUInit

Items	Description
Introduction	Initializes the LCDC peripheral according to the specified parameters in LCDC_MCUInitStruct.
Parameters	● LCDCx: where LCDCx can be LCDC. ● LCDC_MCUInitStruct: pointer to a LCDC_MCUInitTypeDef structure that contains the configuration information.
Return	N/A

22.5.1.3 LCDC_MCUIOWriteCmd

Items	Description
Introduction	Writes command to LCD module via MCU I/F.
Parameters	● LCDCx: where LCDCx can be LCDC. ● Cmd: the command to transmit.
Return	N/A

22.5.1.4 LCDC_MCUIOWriteData

Items	Description
Introduction	Writes data to LCD module via MCU I/F.
Parameters	● LCDCx: where LCDCx can be LCDC. ● Data: the data to transmit.
Return	N/A

22.5.1.5 LCDC_MCUIOReadData

Items	Description
Introduction	Reads data from LCD module via MCU I/F.
Parameters	LCDCx: where LCDCx can be LCDC.
Return	The read value

22.5.1.6 LCDC_MCUDMATrigger

Items	Description
Introduction	Triggers to transfer data of one frame from DMA buffer to LCDC Transmit FIFO.
Parameters	LCDCx: where LCDCx can be LCDC.

Return	N/A

22.5.2 RGB Function

22.5.2.1 LCDC_RGBStructInit

Items	Description
Introduction	Initializes the parameters in the LCDC_RGBInitStruct with default values.
Parameters	LCDC_RGBInitStruct: pointer to an LCDC_RGBInitTypeDef structure which is initialized.
Return	N/A

Note: LCDC_RGBInitStruct contains the RGB configurable parameters for LCDC, which determine that the RGB I/F is 6-bit or 16-bit, DE or HV mode, the plane size, the refresh frequency and VSYNC & HSYNC control. The parameters in this structure should be set according to LCD module datasheet.

22.5.2.2 LCDC_RGBInit

Items	Description
Introduction	Initializes the LCDC peripheral according to the specified parameters in the LCDC_RGBInitStruct.
Parameters	● LCDCx: where LCDCx can be LCDC. ● LCDC_RGBInitStruct: pointer to a LCDC_RGBInitTypeDef structure that contains the configuration information.
Return	N/A

22.5.3 LED Function

22.5.3.1 LCDC_LEDStructInit

Items	Description
Introduction	Initializes the parameters in the LCDC_LEDInitStruct with default values.
Parameters	LCDC_LEDInitStruct: pointer to an LCDC_LEDInitTypeDef structure which will be initialized.
Return	N/A

22.5.3.2 LCDC_LEDInit

Items	Description
Introduction	Initializes the LCDC peripheral according to the specified parameters in the LCDC_LEDInitStruct.
Parameters	● LCDCx: where LCDCx can be LCDC. ● LCDC_LEDInitStruct: pointer to an LCDC_LEDInitTypeDef structure which will be initialized.
Return	N/A

22.5.4 Common Function

22.5.4.1 LCDC_DMAModeConfig

Items	Description
Introduction	Configures LCDC DMA burst size.
Parameters	● LCDCx: where LCDCx can be LCDC. ● BurstSize: DMA burst size; unit is 64 bytes.
Return	N/A

Note:

- If BurstSize = 1, the actual burstsize = 1x64 bytes; if BurstSize = 2, the actual burstsize = 2x64 = 128 bytes; ...
- The parameter BurstSize is not more than 8. The recommended value is 2.

22.5.4.2 LCDC_DMAImageBaseAddrConfig

Items	Description
Introduction	Configures image base address.
Parameters	● LCDCx: where LCDCx can be LCDC. ● ImgBaseAddr: the buffer address.
Return	N/A

22.5.4.3 LCDC_INTConfig

Items	Description
Introduction	Enables or disables the specified LCDC interrupts.
Parameters	● LCDCx: where LCDCx can be LCDC. ● LCDC_IT: specifies the LCDC interrupt sources to be enabled or disabled. This parameter can be any combination of the following values: ■ LCDC_IT_DMAUNDFW: DMA FIFO underflow interrupt ■ LCDC_IT_FRDN: LCD refresh done interrupt ■ LCDC_IT_LINE: line interrupt ■ LCDC_IT_IO_TIMEOUT: I/O write/read timeout interrupt ■ LCDC_IT_FRM_START: Frame Start interrupt ● NewState: new state of the specified LCDC interrupts. This parameter can be ENABLE or DISABLE.
Return	N/A

22.5.4.4 LCDC_GetINTStatus

Items	Description
Introduction	Gets LCDC interrupt status.
Parameters	LCDCx: where LCDCx can be LCDC.
Return	Interrupt status

22.5.4.5 LCDC_ClearINT

Items	Description
Introduction	Clears the LCDC's interrupt pending bits.
Parameters	● LCDC_IT: specifies the interrupt to be cleared. This parameter can be any combination of the following values: ■ LCDC_IT_LINE: line interrupt ■ LCDC_IT_FRDN: refresh frame done interrupt ■ LCDC_IT_DMAUNDFW: DMA FIFO under flow interrupt ■ LCDC_IT_IO_TIMEOUT: IO write/read timeout interrupt ■ LCDC_IT_FRM_START: Frame Start interrupt
Return	N/A

22.5.4.6 LCDC_Cmd

Items	Description
Introduction	Enables or disables the LCDC.
Parameters	● LCDCx: where LCDCx can be LCDC. ● NewState: new state of the LCDC. This parameter can be: ENABLE or DISABLE.
Return	N/A

Note: When NewState is DISABLE, during the period of valid line (VTIMING =valid data), the disabled action is performed after the last valid line has transferred. If you want to disable the LCDC instantly, use the API LCDC_InsDisable(). During the other periods, the disabled action is performed instantly.

22.5.4.7 LCDC_InsDisable

Items	Description
Introduction	Disables the LCDC instantly.
Parameters	LCDCx: where LCDCx can be LCDC.
Return	N/A

22.5.4.8 LCDC_DeInit

Items	Description
Introduction	De-initializes the LCDC.
Parameters	LCDCx: where LCDCx can be LCDC.
Return	N/A

Note: when disabling LCDC instantly, all interrupts are cleared and disabled.

22.6 How to Use LCDC?

Table 22-5 lists the typical application scenarios of LCDC.

Table 22-5 Typical application scenario of LCDC

Interface	Data Mode	LCD GRAM	Ameba-D Frame Buffer	Application
MCU	I/O mode	Yes	No	Static Display
	DMA Trigger-mode	Yes	Yes	Static Display
	DMA Auto-mode (VSYNC/TE mode)	Yes	Yes	Animation Display
RGB	DMA Auto-mode (DE/HV mode)	No	Yes	Animation Display

22.6.1 MCU Interface

22.6.1.1 I/O Mode

To use the LCDC MCU interface I/O mode, the following steps are mandatory.

(1) Configure the LCDC pinmux according to Table 22-4.

For example, in order to use PA[19] as LCDC pin, call the following function. It is the same for other LCDC pins.

```
Pinmux_Config (_PA_19, PINMUX_FUNCTION_LCD);
```

(2) Initialize the LCDC_MCUInitStruct variable.

a) Use the following function to initialize LCDC_MCUInitStruct variable with default parameters.

```
LCDC_MCUStructInit (LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
```

b) Change other parameters according to LCM datasheet, such as 8-bit or 16-bit interface mode, Data/WR/RD/CS/RS pulse polarity, WR/RD pulse width.

(3) Initialize the LCDC using the initialized structure in step (2).

```
LCDC_MCUInit (LCDC_TypeDef * LCDCx, LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
```

(4) Enable the LCDC using the function LCDC_Cmd().

(5) Send commands and parameters to LCM using the function LCDC_MCUIOWriteCmd()/LCDC_MCUIOWriteData() to initialize the LCD module.

(6) After LCM is initialized, call LCDC_MCUIOWriteCmd()/LCDC_MCUIOWriteData()/LCDC_MCUIOReadData() to send commands/write data/read data from LCM to drive LCD displaying.

22.6.1.2 Trigger DMA Mode

To use the LCDC MCU interface trigger DMA mode, the following steps are mandatory.
(1) Configure the LCDC pinmux according to Table 22-4.
(2) Configure LCDC to work in MCU I/O mode, then send commands and parameters to LCM to initialize LCD. After that, users need to send a command to inform LCM to start receiving data before transferring frame data.
(3) Initialize the LCDC_MCUInitStruct variable.
    a) Configure the LCDC_MCUInitStruct parameter to Trigger DMA mode.
    LCDC_MCUStructInit (LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
    LCDC_MCUInitStruct->LCDC_MCUMode = LCDC_MCU_DMA_MODE;
    LCDC_MCUInitStruct->LCDC_MCUDMAMode = LCDC_TRIGGER_DMA_MODE;
    b) Change other parameters according to LCM datasheet, such as LCD width/height, 8-bit or 16-bit I/F mode, Data/WR/RD/CS/RS pulse polarity, WR/RD pulse width.
(4) Initialize the LCDC using the initialized structure in step (3).
    LCDC_MCUInit (LCDC_TypeDef * LCDCx, LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
(5) Configure the LCDC DMA parameters.
    a) Configure the LCDC DMA burst size using LCDC_DMAModeConfig().
    b) Allocate DMA buffer and assign the address to LCDC using LCDC_DMAImageBaseAddrConfig().
(6) Enable LCDC interrupt using the function LCDC_INTConfig(), if needed.
(7) Enable the LCDC using the function LCDC_Cmd().
(8) Trigger one frame transfer using the function LCDC_MCUDMATrigger(), and update the frame buffer to change the display effect.

22.6.1.3 VSYNC Mode

To use the LCDC MCU I/F VSYNC mode, the following steps are mandatory.
(1) Configure the LCDC pinmux according to Table 22-4.
(2) Configure LCDC to work in MCU I/O mode, then send commands and parameters to LCM to make LCD work in VSYNC mode.
(3) Initialize the LCDC_MCUInitStruct variable.
    a) Configure the LCDC_MCUInitStruct parameter corresponding to VSYNC mode.
    LCDC_MCUStructInit (LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
    LCDC_MCUInitStruct->LCDC_MCUMode = LCDC_MCU_DMA_MODE;
    LCDC_MCUInitStruct->LCDC_MCUDMAMode = LCDC_AUTO_DMA_MODE;
    LCDC_MCUInitStruct->LCDC_MCUSyncMode = LCDC_MCU_SYNC_WITH_VSYNC;
    b) Change other parameters according to LCM datasheet, such as VSYNC pulse polarity/pulse width/idle period, LCD width/height.
(4) Initialize the LCDC using the initialized structure in step (3).
    LCDC_MCUInit (LCDC_TypeDef * LCDCx, LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
(5) Configure the LCDC DMA parameters.
    a) Configure the LCDC DMA burst size using the function LCDC_DMAModeConfig().
    b) Allocate DMA buffer and assign the address to LCDC using the function LCDC_DMAImageBaseAddrConfig().
(6) Enable the specified LCDC interrupt using the function LCDC_INTConfig(), if needed.
(7) Enable the LCDC using the function LCDC_Cmd().
(8) The LCDC can transfer frame data to LCM automatically synchronized with the VSYNC signal to LCM, and you can update the frame buffer to change the display.

22.6.1.4 TE Mode

(1) To use the LCDC MCU IinterfaceF TE mode, the following steps are mandatory.
(2) Configure the LCDC pinmux according to Table 22-4.
(3) Configure LCDC to work in MCU I/O mode, then send commands and parameters to LCM to let LCD work in TE mode.
(4) Initialize the LCDC_MCUInitStruct variable.
    a) Configure the LCDC_MCUInitStruct parameter corresponding to VSYNC mode.
    LCDC_MCUStructInit (LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);
    LCDC_MCUInitStruct->LCDC_MCUMode = LCDC_MCU_DMA_MODE;
    LCDC_MCUInitStruct->LCDC_MCUDMAMode = LCDC_AUTO_DMA_MODE;
    LCDC_MCUInitStruct->LCDC_MCUSyncMode = LCDC_MCU_SYNC_WITH_TE;
    b) Change other parameters if needed, such as TE pulse polarity, TE Delay, LCD width/height.
(5) Initialize the LCDC using the initialized structure in step (3).

LCDC_MCUInit (LCDC_TypeDef * LCDCx, LCDC_MCUInitTypeDef * LCDC_MCUInitStruct);

(6) Configure the LCDC DMA parameters.
   a) Configure the LCDC DMA burst size using the function LCDC_DMAModeConfig().
   b) Allocate DMA buffer and assign the address to LCDC using the function LCDC_DMAImageBaseAddrConfig().
(7) Enable the specified LCDC interrupt using the function LCDC_INTConfig(), if needed.
(8) Enable the LCDC using the function LCDC_Cmd().
(9) The LCDC can transfer frame data to LCM automatically synchronized with the TE signal from LCM, and you can update the frame buffer to change the display.

22.6.2 RGB Interface

22.6.2.1 DE Mode

To use the LCDC RGB interface DE mode, the following steps are mandatory.

(1) Configure the LCDC pinmux according to Table 22-4.
(2) Configure LCM parameters through SPI or other interfaces if needed.
(3) Initialize the LCDC_RGBInitStruct variable.
   a) Configure the LCDC_RGBInitStruct parameter corresponding to DE mode.
   LCDC_RGBStructInit (LCDC_RGBInitTypeDef * LCDC_RGBInitStruct);
   LCDC_RGBInitStruct->LCDC_RGBSyncMode = LCDC_RGB_DE_MODE;
   b) Change other parameters if needed, such as Date/ENABLE/VSYNC/HSYNC pulse polarity, DCLK active edge, VFP, VBP, VSW, HBP, HFP, HSW, refresh frequency, LCD width/height.
(4) Initialize the LCDC using the initialized structure in step (3).
   LCDC_RGBInit (LCDC_TypeDef* LCDCx, LCDC_RGBInitTypeDef* LCDC_RGBInitStruct);
(5) Configure the LCDC DMA parameters.
   a) Set burst size using the function LCDC_DMAModeConfig().
   b) Set DMA FIFO under flow mode and error data using the functions LCDC_DMAUnderFlowModeConfig() and LCDC_DMAUnderFlowModeConfig().
   c) Allocate DMA buffer and assign the address to LCDC using the functionL CDC_DMAImageBaseAddrConfig().
(6) Enable the specified LCDC interrupt using the function LCDC_INTConfig(), if needed.
(7) Enable the LCDC using the function LCDC_Cmd().
(8) The LCDC can transfer frame data to LCM automatically according to the refresh frequency, and you can update the frame buffer to change the display.

22.6.2.2 HV Mode

To use the LCDC RGB I/F HV mode, the following steps are mandatory.

(1) Configure the LCDC pinmux according to Table 22-4.
(2) Configure LCM parameters through SPI or other interfaces if needed.
(3) Initialize the LCDC_RGBInitStruct variable.
   a) Configure the LCDC_RGBInitStruct parameter corresponding to HV mode.
   LCDC_RGBStructInit (LCDC_RGBInitTypeDef * LCDC_RGBInitStruct);
   b) Change other parameters if needed, Date/ENABLE/VSYNC/HSYNC pulse polarity, DCLK active edge, VFP, VBP, VSW, HBP, HFP, HSW, refresh frequency, LCD width/height, 6-bit or 16-bit parallel I/F mode, refresh frequency.
(4) Initialize the LCDC using the initialized structure in step (3).
   LCDC_RGBInit (LCDC_TypeDef* LCDCx, LCDC_RGBInitTypeDef* LCDC_RGBInitStruct)
(5) Configure the LCDC DMA parameters
   a) Set burst size using the function LCDC_DMAModeConfig().
   b) Set DMA FIFO under flow mode and error data using the functions LCDC_DMAUnderFlowModeConfig() and LCDC_DMAUnderFlowModeConfig().
   c) Allocate DMA buffer and assign the address to LCDC using the function LCDC_DMAImageBaseAddrConfig().
(6) Enable the specified LCDC interrupt using the function LCDC_INTConfig(), if needed.
(7) Enable the LCDC using the function LCDC_Cmd().
(8) The LCDC can transfer frame data to LCM automatically according to the refresh frequency, and you can update the frame buffer to change the display.

22.6.3 LED Interface

To use the LCDC LED interface mode, the following steps are mandatory.

(1) Configure the LCDC pinmux according to Table 22-4.
(2) Initialize the LCDC_RGBInitStruct variable.
    a) Configure the LCDC_LEDInitStruct parameter corresponding to LED interface mode
    LCDC_LEDStructInit (LCDC_LEDInitTypeDef * LCDC_LEDInitStruct)
    b) Change other parameters if needed, such as color channel, color numbers, timing (latch start time, latch pulse width, OE active width), refresh frequency, LED width/height.
(3) Initialize the LCDC using the initialized structure in step (2).
    LCDC_LEDInit (LCDC_TypeDef* LCDCx, LCDC_LEDInitTypeDef * LCDC_LEDInitStruct)
(4) Configure the LCDC DMA parameters
    a) Set burst size using the function LCDC_DMAModeConfig().
    b) Allocate DMA buffer and assign the address to LCDC using the function LCDC_DMAImageBaseAddrConfig().
(5) Enable the specified LCDC interrupt using the function LCDC_INTConfig(), if needed.
(6) Enable the LCDC using the function LCDC_Cmd().
(7) The LCDC can transfer frame data to LED array board automatically according to the refresh frequency, and you can update the frame buffer to change the display.

22.7 GUI

emWin is the embedded GUI solution. It can be adapted to any size, either physical or virtual display, not depending on the display controller, which makes it a professional GUI for the embedded market, usable for multiple different scenarios.

Realtek has reached an emWin Pro Buyout Agreement with SEGGER. The emWin LICENSED SOFTWARE, which is located in "component\common\ui\emwin", is available in object code form for Realtek customers who have been authorized.

22.7.1 Authorization

In order to be authorized to use emWin SOFTWARE, you need to follow these steps:

(1) Read the "emWin_Software_License_Agreement.pdf" carefully under the path "component\common\ui". This document is a binding, legal agreement between Realtek and you (either an individual or a legal entity). It explains the terms and conditions that you should accept and agree.
(2) If you do not accept and agree to this Agreement, do not unzip "emwin.zip" and do not use any of the LICENSED SOFTWARE. If you do accept and agree this Agreement, you can unzip "emwin.zip" with the password, which can be found in "emWin_Software_License_Agreement.pdf".

22.7.2 emWin Software

After extracting the "emwin.zip", you can see the following directories of emWin software.

- Config: Contains the configuration files which are used to adapter for different LCD module and different scenarios.
  The default configuration files are for TCX043DTLN-04 TFT-LCD module (480*272, RGB 6-bit I/F). You need to modify these to adapter for different LCD module. Details can be found in *Chapter 38 Configuration, UM03001_emWin5.pdf*.
- Doc
  The *UM03001_emWin5.pdf* is the User Guide and Reference Manual of emWin.
- GUI_X
  GUI_X_FreeRTOS.c is the configuration of the timing routines, the debugging routines and the kernel interface routines.
- Include
  The header files of emWin.
- Lib
  The object code of emWin.
- Tool
  The available tools that are useful during UI development, such as Font Converter, GUIBuilder.
- Truetype
  TTF (TrueType Font) support for emWin, which should be used if fonts need to be scalable at run-time.

This is the adapted version of FreeType font library from David Turner, Robert Wilhelm and Werner Lemberg.

22.7.3 How to Use emWin in SDK?

To build and run emWin in SDK, you need to follow these steps:

(1) Enable emWin compile
KM4 make menuconfig > 'MENUCONFIG FOR CHIP CONFIG' > 'GUI Config' > 'Enable GUI', and select 'emwin'.

(2) Enable PSRAM if needed
By default, the VRAM buffer, which is necessary for RGB I/F LCM, is located is PSRAM, so you should enable PSRAM by setting psram_dev_config, psram_dev_enable to TRUE in rtl8721dhp_intfcfg.c.
If you want to put the VRAM buffer in RAM instead of PSRAM, you should modify LCDConf.h as follows:

```
#undef PSRAM_BUF_USED
```

(3) Add your own code or run emWin demo code.
The emWin demo code is located in "project\realtek_amebaD_va0_example\example_sources\LCDC\GUI_demo\emWin". The ReadMe.txt in each demo folder demonstrates how to use the demo.

23 Audio Codec Controller Guide

23.1 Audio Codec

Ameba-D audio codec (AC) is often used to play and record audio data. It is a stereo audio codec with stereo headphone amplifiers, as well as 2-way inputs and 1-way stereo/mono output that are programmable to single-ended or differential.

Ameba-D audio codec integrates anti-pop circuit for audible pop noise cancellation, MIC bias circuit for MIC power supply and programmable MIC boost ability.

Ameba-D audio codec transmits record data to or receives playback data from platform through SPORT interface. By means of specific serial interface (SI), Ameba-D platform configures and drives audio codec.

Ameba-D also provides external I²S interface for audio application extension, which supports the highest 384kHz sampling frequency.

23.1.1 Diagram

The diagram of AC is shown in Fig 23-1.

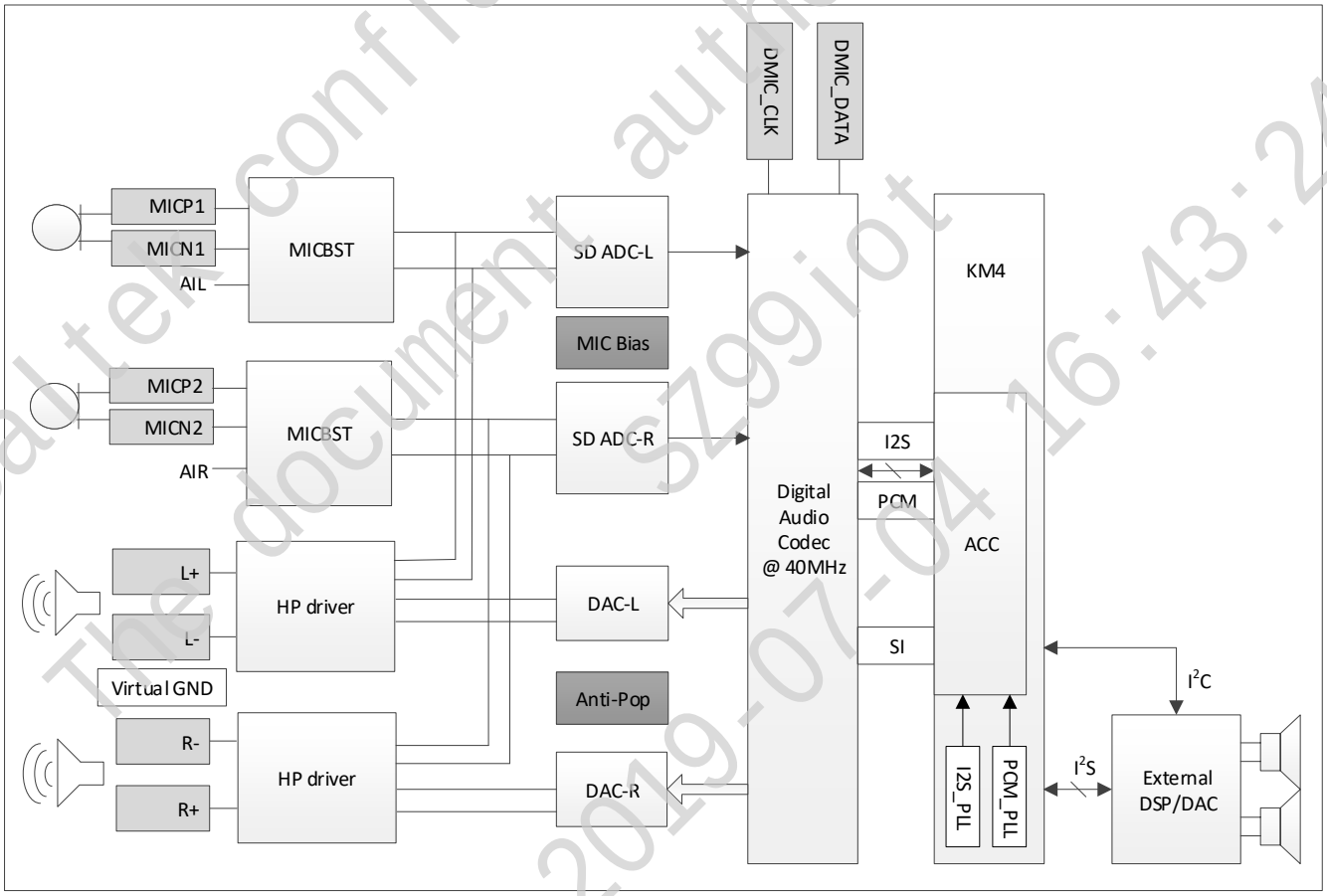


Fig 23-1 Audio codec diagram

23.1.2 Features

- Mono and stereo channel
- 8-bit, 16-bit and 24-bit sample bits
- 8k, 16k, 32k, 48k, 96k, 44.1k, 48k and 88.2k sample rate
- I2S, left justify, PCM mode A, PCM mode B, PCM mode A-N and PCM mode B-N data format
- Anti-pop function to reduce audible pop
- Programmable MIC boost gain
- Programmable gain in ADC and DAC path
- Three line-out output modes: cap-less, differential and single-end
- Three input ways: line-in, AMIC-in and DMIC-in

23.1.3 Application Mode

Audio codec supports three input ways: line-in, AMIC-in and DMIC-in, but only supports one output way: line-out.

23.1.3.1 Line-out

Line-out has no amplifier to amplify output voice. Line-out supports three output modes: cap-less, differential and single-ended. User can select the wanted mode by setting the related registers.

- Line-out cap-less mode

In this mode, the N-end of L/R channel outputs common-level voltage, while P-end drives the available analog audio signal. When earphone inserts into jack, the ground must short with N-end output for audio signal de-couple. That is why the ground is called virtual ground.

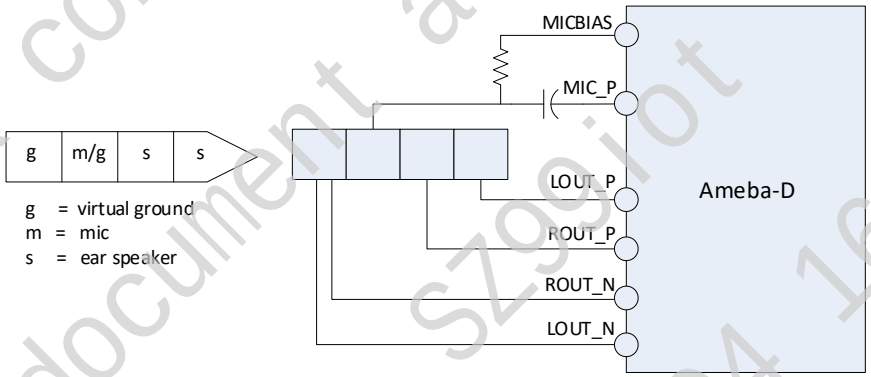


Fig 23-2 Cap-less mode connection with headphone jack

- Line-out differential mode

In this mode, both N-end and P-end drive the available analog audio signal. User should select the differential jack and earphone accordingly.

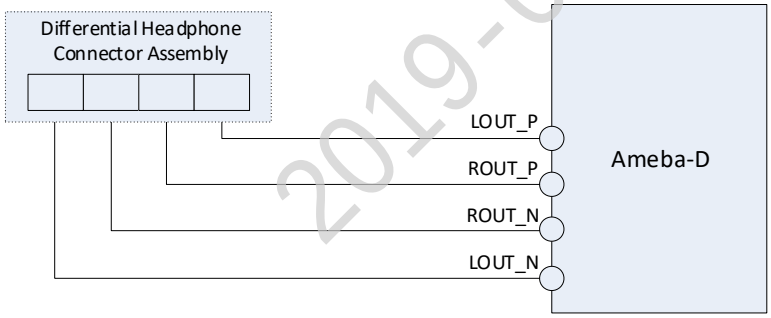


Fig 23-3 Differential mode connection with headphone jack

- Line-out single-ended mode

In this mode, board circuit designer needs to place a capacitor to the P-end output path for analog audio signal pick-up. No N-end output is required.

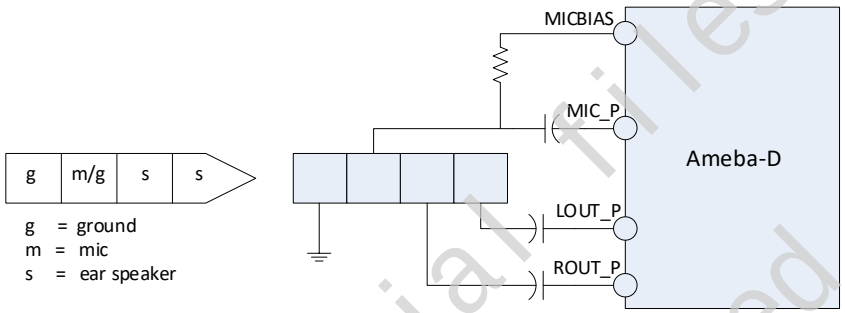


Fig 23-4 Single-ended mode connection with headphone jack

23.1.3.2 Line-in

Line-in has no preamplifier, its input signal often has a large output power. It often connects to the audio output of equipment such as electric guitar, electronic organ and synthesizer.

Connect the left channel of line-in signal to AUX_L, and the right channel to AUX_R accordingly.

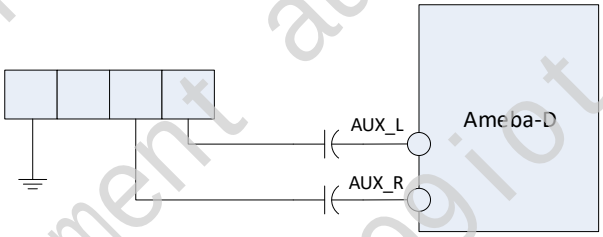


Fig 23-5 Line-in mode connection

23.1.3.3 AMIC-in

Analog microphone (AMIC) records audio data, it has preamplifier, its input signal often has a low output power. AMIC-in supports differential mode and single-ended mode.

- AMIC-in single-ended mode

Connect MIC_P with single-ended analog microphone, while MICBIAS provides the microphone bias voltage.

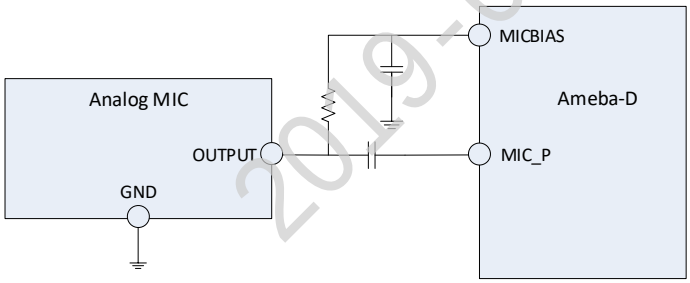


Fig 23-6 AMIC-in single-ended mode connection

- AMIC-in differential mode

Connect MIC_P/MIC_N with differential analog microphone, while MICBIAS provides the microphone bias voltage.

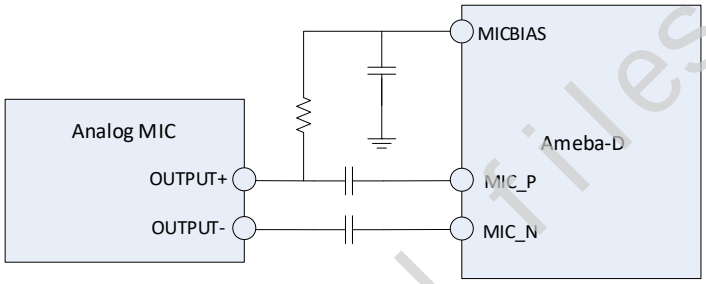


Fig 23-7 AMIC-in differential mode connection

23.1.3.4 DMIC-in

Digital microphone (DMIC) records audio data, it is integrated with ADC internal, and can directly output digital signal. DMIC-in supports mono mode and stereo mode.

- DMIC-in mono mode

Tie the L/R of digital microphone to ground or VDD if only one digital microphone is placed.

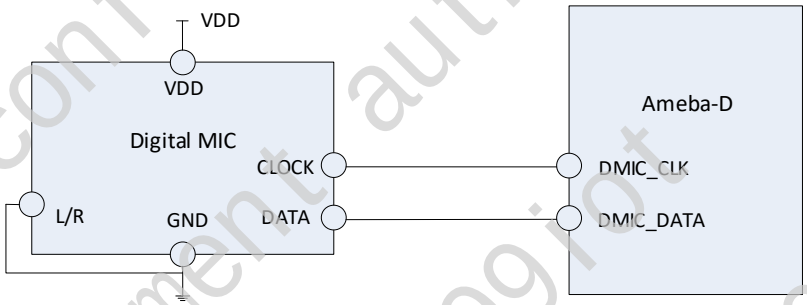


Fig 23-8 DMIC-in mono mode connection

- DMIC-in stereo mode

Tie the L/R of two digital microphones to ground and VDD respectively if stereo microphone is needed. The two microphones share the DMIC_DATA according to the rising/falling edge.

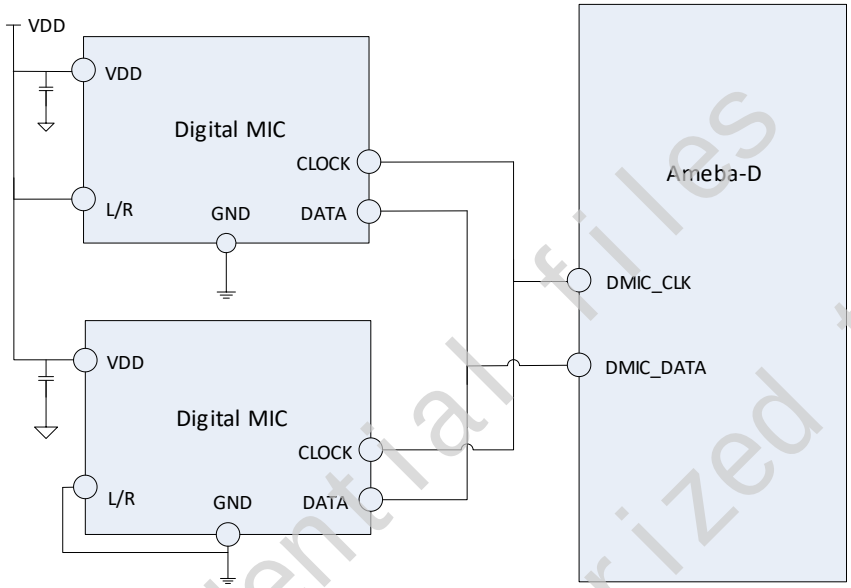


Fig 23-9 DMIC-in stereo mode connection

23.2 Audio Codec Controller

Ameba-D audio codec controller (ACC) is the bridge between host audio buffers and audio codec module. It is used for audio codec input/output control.

Ameba-D audio codec controller uses GDMA to move data, and transfers audio data to or from audio codec module via SPORT, and configure audio codec module via SI.

23.2.1 Diagram

The diagram of ACC is shown in Fig 23-10.

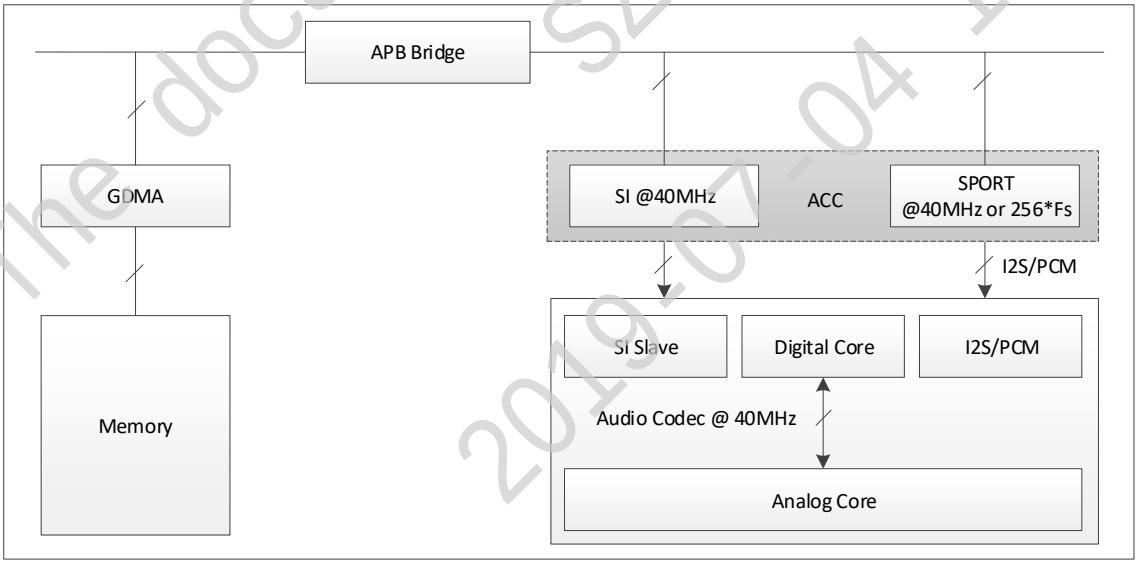


Fig 23-10 Audio codec controller diagram

23.2.2 Features

- Mono and stereo channel
- 8-bit, 16-bit and 24-bit sample bits
- I2S, left justify, PCM mode A, PCM mode B, PCM mode A-N and PCM mode B-N data format
- Mandatory or optional sample rate which audio codec module declares for support
- GDMA for data moving
- Data loopback between SDI and SDO

23.2.3 Control Interface

There are two control interfaces: SPORT interface and SI interface.

- SPORT interface

Audio codec transfers audio data via SPORT interface sequentially according to user's setting. It supports multiple data format, such as I2S, left justify, PCM mode A, PCM mode B, PCM mode A-N and PCM mode B-N.

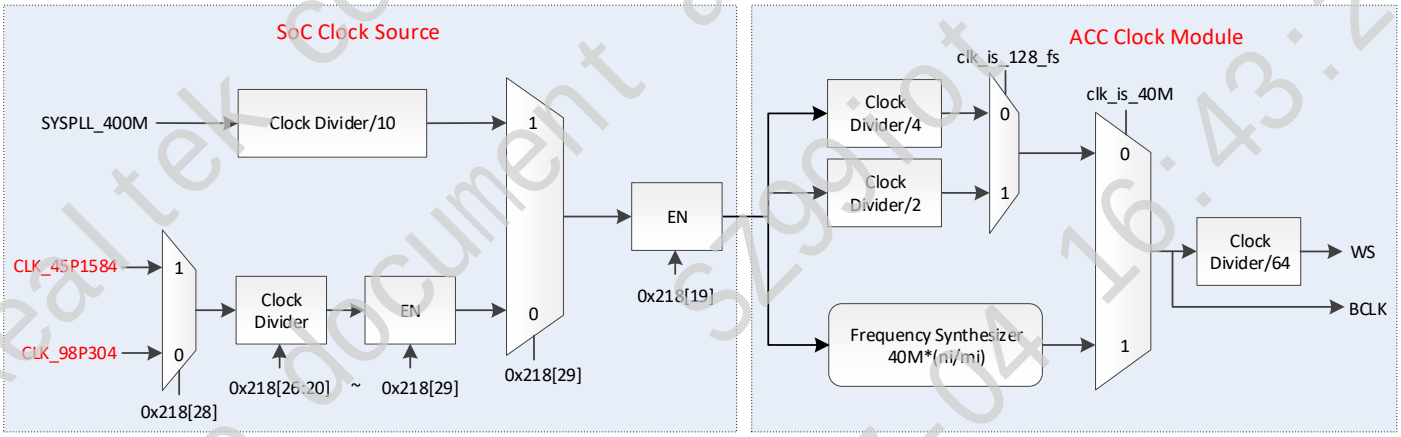
- SI interface

Audio codec needs to configure the related analog and digital parameters before transferring audio data. SI interface is used to configure codec parameters. It can read or write codec register.

For more details about AC and AAC, refer to UM0400 Ameba-D User Manual.

23.3 Audio PLL

The ACC clock architecture is shown in Fig 23-11.



Note: CLK_45P1584 is clock derived from PCM PLL internal, while CLK_98P304 from I^{2}S PLL.

Fig 23-11 ACC clock architecture

23.3.1 Diagram

The ACC clock diagram is illustrated in Fig 23-12.

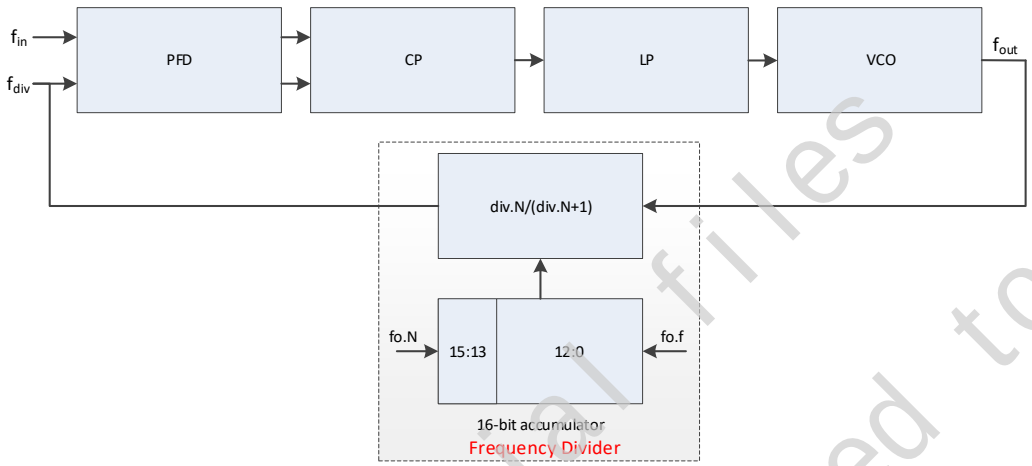


Fig 23-12 ACC clock diagram

$f_{out}/f_{in} = (div.N*(2^{16} - ((fo.N*2^{13}) + fo.f)) + (div.N + 1)*((fo.N*2^{13}) + fo.f))/2^{16} = div.N + fo.N/2^{3} + fo.f/2^{16}$
where f_{in} is derived from the clock of crystal, while div.N, fo.N and fo.f from the register settings. Therefore, the resolution of f_{out} equals to $f_{in}/2^{16}$.

The relationship between crystal clock and f_{in} is listed in Table 23-1.

Table 23-1 Relationship between crystal clock and f_{in}

Crystal Clock (MHz)	f_{in} (MHz)	Divider
40	10	4
25	12.5	2
13	13	1
19.2	9.6	2
20	10	2
26	13	2
38.4	9.6	4
17.664	8.832	2
16	8	2
14.318	14.318	1
12	12	1
52	13	4
48	12	4
27	13.5	2
24	12	2

23.3.2 Operation Mode

23.3.2.1 Auto Mode

In auto mode, PLL circuit automatically sets the parameters (div.N, fo.N, fo.f, etc.) to output clock with the exact frequency, 196.608MHz (=98.304MHz x 2) is offered by I²S PLL while 180.6336MHz (=45.1584MHz x 4) by PCM PLL.

23.3.2.2 Manual Mode

If fine tuning PLL clock is required, users could switch the I²S or PCM PLL to manual mode. In this mode, users could control the output frequency of I²S PLL max ± 100ppm around 196.608MHz or PCM PLL max ± 100ppm around 180.6336MHz by calling the corresponding APIs. As the adjusted step of f_{out} equals to $f_{in}/2^{16}$, f_{in} decides the exact ppm per step ($(f_{in}/2^{16})/196.608$ or $(f_{in}/2^{16})/180.6336$).

The relationship between f_{in} and ppm per step is listed in Table 23-2.

Table 23-2 Relationship between f_{in} and ppm per step

f_{in} (MHz)	I²S ppm per step	PCM ppm per step
10	0.78	0.84
12.5	0.97	1.06
13	1.01	1.10
9.6	0.75	0.81
10	0.78	0.84
13	1.01	1.10
9.6	0.75	0.81
8.832	0.69	0.75
8	0.62	0.68
14.318	1.11	1.21
12	0.93	1.01
13	1.01	1.10
12	0.93	1.01
13.5	1.05	1.14
12	0.93	1.01

23.4 Audio Codec APIs

23.4.1 PLL APIs

API	Introduction
<PLL_Div>	Divider to generate 256*fs or 128*fs clock
<PLL_Sel>	Selects I²S PLL or PCM PLL
<PLL_I2S_Set>	Enables or disables I²S PLL
<PLL_PCM_Set>	Enables or disables PCM PLL
<PLL_I2S_ClkTune>	Tunes I²S PLL output faster or slower, or resets it to auto mode
<PLL_PCM_ClkTune>	Tunes PCM PLL output faster or slower, or resets it to auto mode

23.4.1.1 PLL_Div

Parameter	Type	Introduction
	u32	Divider to generate 256*fs or 128*fs clock.

23.4.1.2 PLL_Sel

Parameter	Type	Introduction
<sel>	u32	● Selects I²S PLL if fs=8/16/32/48/96kHz ● Selects PCM PLL if fs=44.1/88.2kHz

23.4.1.3 PLL_I2S_Set

Parameter	Type	Introduction
<new_state>	u32	Enables or disables I²S PLL

23.4.1.4 PLL_PCM_Set

Parameter	Type	Introduction
< new_state >	u32	Enables or disables PCM PLL

23.4.1.5 PLL_I2S_ClkTune

Parameter	Type	Introduction
<ppm>	u32	Required fine tuning ppm value
<action>	U32	Faster or slower, or reset to auto mode

23.4.1.6 PLL_PCM_ClkTune

Parameter	Type	Introduction
<ppm>	u32	Required fine tuning ppm value
<action>	U32	Faster or slower, or reset to auto mode

23.4.2 SPORT APIs

API	Introduction
<AUDIO_SP_StructInit>	Fills each SP_StructInit member with its default value
<AUDIO_SP_Init>	Initializes the audio SPORT registers according to the specified parameters in SP_InitStruct
<AUDIO_SP_TxStart>	Starts or stops SPORT Tx path
<AUDIO_SP_RxStart>	Starts or stops SPORT Rx path
<AUDIO_SP_TdmaCmd>	Enables or disables SPORT Tx DMA request
<AUDIO_SP_RdmaCmd>	Enables or disables SPORT Rx DMA request
<AUDIO_SP_SetWordLen>	Sets the AUDIO SPORT word length
<AUDIO_SP_GetWordLen>	Gets the AUDIO SPORT word length
<AUDIO_SP_SetMonoStereo>	Sets the AUDIO SPORT channel number
<AUDIO_SP_TXGDMA_Init>	Initializes GDMA peripheral for Tx data
<AUDIO_SP_RXGDMA_Init>	Initializes GDMA peripheral for Rx data

23.4.2.1 AUDIO_SP_StructInit

Fills each SP_StructInit member with its default value.

Parameter	Type	Introduction
<SP_InitStruct >	SP_InitTypeDef*	SP_InitTypeDef structure that contains the configuration information for the specified audio SPORT peripheral

23.4.2.2 AUDIO_SP_Init

Initializes the audio SPORT registers according to the specified parameters in SP_InitStruct.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
<SP_InitStruct >	SP_InitTypeDef*	SP_InitTypeDef structure that contains the configuration information for the specified AUDIO SPORT peripheral

23.4.2.3 AUDIO_SP_TxStart

Starts or stops SPORT Tx path.

If playing with audio codec, it starts SPORT Tx path.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
<NewState>	u32	State (enable or disable) of the SPORT Tx

23.4.2.4 AUDIO_SP_RxStart

Starts or stops SPORT Rx path.

If recording with audio codec, it starts SPORT Rx path.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
<NewState>	u32	State (enable or disable) of the SPORT Rx

23.4.2.5 AUDIO_SP_TdmaCmd

Enables or disables SPORT Tx DMA request.

If Tx DMA request is not enabled, you should start Tx when GDMA completes every time.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
<NewState>	u32	State (enable or disable) of the SPORT Tx DMA request

23.4.2.6 AUDIO_SP_RdmaCmd

Enables or disables SPORT Rx DMA request.

If Rx DMA request is not enabled, you should start Rx when GDMA completes every time.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
<NewState>	u32	State (enable or disable) of the SPORT Rx DMA request

23.4.2.7 AUDIO_SP_SetWordLen

Sets the audio SPORT word length.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
<SP_WordLen>	u32	The value of word length

23.4.2.8 AUDIO_SP_GetWordLen

Gets the audio SPORT word length.

Parameter	Type	Introduction

<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral

23.4.2.9 AUDIO_SP_SetMonoStereo

Sets the audio SPORT channel number.

SPORT only supports stereo channel and mono channel.

Parameter	Type	Introduction
<SPORTx>	AUDIO_SPORT_TypeDef*	The base address of audio SPORT peripheral
< SP_MonoStereo>	u32	Mono or stereo channel

23.4.2.10 AUDIO_SP_TXGDMA_Init

Initializes GDMA peripheral for sending data.

Parameter	Type	Introduction
<Index>	u32	GDMA index
<GDMA_InitStruct>	GDMA_InitTypeDef *	GDMA_InitTypeDef structure that contains the configuration information for the GDMA peripheral
<CallbackData>	void *	GDMA callback data
<CallbackFunc>	IRQ_FUN	GDMA callback function
<pTxData>	u8 *	Tx buffer that stores Tx data
<Length>	u32	Tx data length

23.4.2.11 AUDIO_SP_RXGDMA_Init

Initializes GDMA peripheral for receiving data.

Parameter	Type	Introduction
<Index>	u32	GDMA index
<GDMA_InitStruct>	GDMA_InitTypeDef *	GDMA_InitTypeDef structure that contains the configuration information for the GDMA peripheral
<CallbackData>	void *	GDMA callback data
<CallbackFunc>	IRQ_FUN	GDMA callback function
<pRxData>	u8 *	Rx buffer that stores the received data
<Length>	u32	Rx data length

23.4.3 SI APIs

SI APIs are used to read and write codec register.

API	Introduction
<AUDIO_SI_Cmd>	Enables or disables the specified audio SI peripheral
<AUDIO_SI_WriteReg>	SI writes codec register
<AUDIO_SI_ReadReg>	SI reads codec register
<AUDIO_SI_ClkCmd>	Turns on or turns off the clock of register bank of audio codec

23.4.3.1 AUDIO_SI_Cmd

Enables or disables the specified audio SI peripheral.

Parameter	Type	Introduction
<new_state>	u8	New state of the SI peripheral

23.4.3.2 AUDIO_SI_WriteReg

Uses SI interface to write codec register.

Parameter	Type	Introduction
<address>	u32	Codec register address
<data>	u32	Data value written to the register

23.4.3.3 AUDIO_SI_ReadReg

Uses SI interface to read codec register.

Parameter	Type	Introduction
<address>	u32	Codec register address

23.4.3.4 AUDIO_SI_ClkCmd

Turns on or turns off the clock of register bank of audio codec.

Parameter	Type	Introduction
<new_state>	u8	New state of the clock of register bank of audio codec

23.4.4 Codec APIs

API	Introduction
<CODEC_Init>	Initializes codec peripheral according to the application mode
<CODEC_SetVolume>	Sets codec volume by controlling mon DAC channel DVOL gain
<CODEC_GetVolume>	Gets codec mon DAC channel gain control
<CODEC_SetSr>	Sets codec ADC and DAC sample rate
<CODEC_SetAdcGain>	Sets codec ADC gain
<CODEC_SetAmicBst>	Set codec AMIC boost
<CODEC_SetDmicBst>	Set codec DMIC boost
<CODEC_SetMicBias>	Sets MIC_BIAS output voltage
<CODEC_MuteRecord>	Mutes or unmutes per AD channel
<CODEC_MutePlay>	Mutes or unmutes per DA channel.
<CODEC_DeInit>	De-initializes codec peripheral

23.4.4.1 CODEC_Init

Initializes codec peripheral according to the application mode.

Parameter	Type	Introduction
<sample_rate>	u32	Codec ADC and DAC sample rate
<word_len>	u32	Codec data sample bit
<mono_stereo>	u32	Codec mono channel or stereo channel
<application>	u32	Codec application mode, such as APP_AMIC_IN, APP_LINE_OUT

23.4.4.2 CODEC_SetVolume

Sets codec volume by controlling mon DAC channel DVOL gain.

Parameter	Type	Introduction
<vol_lch>	u8	Codec mon DAC left channel DVOL gain control (0.375dB/step) ● 8'hAF: 0dB ● 8'h00: -65.625dB
<vol_rch>	u8	Codec mon DAC right channel DVOL gain control (0.375dB/step) ● 8'hAF: 0dB ● 8'h00: -65.625dB

23.4.4.3 CODEC_GetVolume

Gets codec mon DAC channel gain control.

Parameter	Type	Introduction
<vol>	u16 *	Mon DAC channel DVOL gain (high 8 bits is RCH gain, low 8 bits are LCH gain)

23.4.4.4 CODEC_SetSr

Sets codec ADC and DAC sample rate.

Parameter	Type	Introduction
<sample_rate>	u32	Codec ADC and DAC sample rate

23.4.4.5 CODEC_SetAdcGain

Sets codec ADC gain.

Parameter	Type	Introduction
<ad_gain_left>	u32	ADC left channel digital volume gain
<ad_gain_right>	u32	ADC right channel digital volume gain

23.4.4.6 CODEC_SetAmicBst

Sets codec AMIC boost.

Parameter	Type	Introduction
<amic_bst_left>	u32	AMIC left channel boost gain
<amic_bst_right>	u32	AMIC right channel boost gain

23.4.4.7 CODEC_SetDmicBst

Sets codec DMIC boost.

Parameter	Type	Introduction
<dmic_bst_left>	u32	DMIC left channel boost gain
<dmic_bst_right>	u32	DMIC right channel boost gain

23.4.4.8 CODEC_SetMicBias

Sets MIC_BIAS output voltage.

Parameter	Type	Introduction
<mic_bias>	u8	Microphone bias voltage setting

23.4.4.9 CODEC_MuteRecord

Mutes or unmutes per AD channel.

Parameter	Type	Introduction
< mute_lch>	u32	Mutes for left AD channel
< mute_rch>	u32	Mutes for right AD channel

23.4.4.10 CODEC_MutePlay

Mutes or unmutes per DA channel.

Parameter	Type	Introduction
< mute_lch>	u32	Mutes for left DA channel
< mute_rch>	u32	Mutes for right DA channel

23.4.4.11 CODEC_DeInit

De-initializes codec peripheral.

Parameter	Type	Introduction
<application>	u32	Codec application mode

23.5 How to Use AC APIs?

23.5.1 Audio Play Steps

To use audio codec playing the audio data, follow the steps below:

(1) Open audio codec clock and function
    PLLx_Set (0, ENABLE); (x is 0 or 1)
    RCC_PeriphClockCmd (APBPeriph_AUDIOC, APBPeriph_AUDIOC_CLOCK, ENABLE);
    RCC_PeriphClockCmd (APBPeriph_SPORT, APBPeriph_SPORT_CLOCK, ENABLE);

(2) Enable pin for audio codec function
    PAD_CMD (PinName, DISABLE);

(3) Initialize codec with desired parameters
    CODEC_Init (SampleRate, WordLen, MonoStereo, Application); (Application is APP_LINE_OUT)

- If you need to change codec volume, use
    CODEC_SetVolume (vol_lch, vol_rch);
- If you want to adjust microphone bias output voltage, use
    CODEC_SetMicBias (mic_bias);

(4) Fill the SPORT desired parameters
    AUDIO_SP_InitStruct (&SP_InitStruct);

(5) Configure audio SPORT with the corresponding configuration
    AUDIO_SP_Init (AUDIO_SP_DEV, &SP_InitStruct);

(6) Start Tx path
    AUDIO_SP_TdmaCmd (AUDIO_SPORT_DEV, ENABLE);

AUDIO_SP_TxStart (AUDIO_SPORT_DEV, ENABLE);

(7) Activate GDMA to Tx data

AUDIO_SP_TXGDMA_Init (Index, &GDMA_InitStruct, *CallbackData, CallbackFunc, pTxData, Length);

23.5.2 Audio Record Steps

To use audio codec recording the audio data, follow the steps below:

(1) Open audio codec clock and function

PLLx_Set (0, ENABLE); (x is 0 or 1)

RCC_PeriphClockCmd (APBPeriph_AUDIOC, APBPeriph_AUDIOC_CLOCK, ENABLE);

RCC_PeriphClockCmd (APBPeriph_SPORT, APBPeriph_SPORT_CLOCK, ENABLE);

(2) Enable pin for audio codec function

PAD_CMD (PinName, DISABLE);

(3) Initialize codec with desired parameters

CODEC_Init (SampleRate, WordLen, MonoStereo, Application);

Application can select APP_AMIC_IN for analog microphone, select APP_DMIC_IN for digital microphone, or select APP_LINE_IN.

- If codec needs to change volume, use

  CODEC_SetVolume (vol_lch, vol_rch);

- If codec needs to change ADC gain, use

  CODEC_SetAdcGain (ad_gain_left, ad_gain_right);

(4) Fill the desired parameters

AUDIO_SP_InitStruct (&SP_InitStruct);

(5) Configure audio SPORT with the corresponding configuration

AUDIO_SP_Init (AUDIO_SP_DEV, &SP_InitStruct);

(6) Start Rx path

AUDIO_SP_RdmaCmd (AUDIO_SPORT_DEV, ENABLE);

AUDIO_SP_RxStart (AUDIO_SPORT_DEV, ENABLE);

(7) Activate GDMA to Rx data

AUDIO_SP_RXGDMA_Init (Index, &GDMA_InitStruct, *CallbackData, CallbackFunc, pRxData, Length);

23.5.3 Example List

Example		Location
Playback		/project/realtek_amebaD_cm4_gcc_verification/example_sources/Audio/dac
Record and playback	AMIC	/project/realtek_amebaD_cm4_gcc_verification/example_sources/Audio/adc
	DMIC	/project/realtek_amebaD_cm4_gcc_verification/example_sources/Audio/dmic
Record and store	AMIC	/component/common/example/audio_sport/audio_recorder
Record and upload	DMIC	/component/common/example/audio_sport/audio_pcm_upload
Decode algorithm	AC3	/component/common/example/audio_sport/audio_ac3
	AMR	/component/common/example/audio_sport/audio_amr
	FLAC	/component/common/example/audio_sport/audio_flac
	Helix AAC	/component/common/example/audio_sport/audio_helix_aac
	Helix MP3	/component/common/example/audio_sport/audio_helix_mp3
	HLS	/component/common/example/audio_sport/audio_hls
	M4A	/component/common/example/audio_sport/audio_m4a
	M4A self-parse	/component/common/example/audio_sport/audio_m4a_selfparse
	MP3	/component/common/example/audio_sport/audio_mp3

Note: In the example code, we don't disable the GDMA even if no valid data are available. If you want to disable the GDMA, it should be done in a GDMA related interrupt routine. Disabling the GDMA outside of an interrupt routine may cause an exception if the GDMA is still transferring data.

23.6 Hardware Design Guide

23.6.1 Line-out

The line-out connection of audio codec is illustrated in Fig 23-13. The capacitors between 3.5mm jack and IC should be 47uF tantalum capacitors rather than ceramic capacitors. The reason is that capacitance value of ceramic capacitors may decrease when bias voltage is applied to them, which causes audio performance bad.

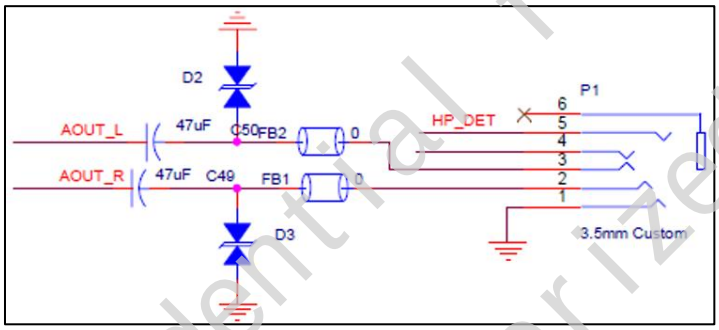


Fig 23-13 Line-out connection

23.6.2 AMIC-in

The AMIC-in connection of audio codec is illustrated in Fig 23-14. The capacitor between analog microphone and IC should be 1uF. Larger capacitance value makes longer period needed for capacitor charging.

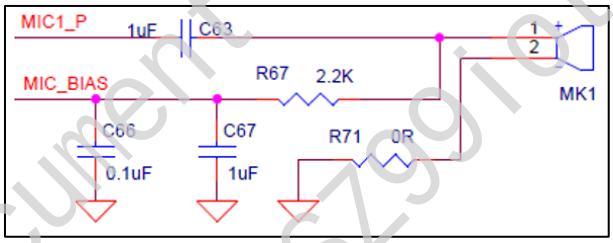


Fig 23-14 AMIC-in connection

MIC_BIAS connects to the positive side of microphone through a 2.2kohm resistor to offer bias voltage

- Short the negative side of microphone to ground if working at single-ended mode, or connect to ground through a 2.2kohm resistor at differential mode.
- Connect the negative side of microphone to MIC_N through a 1uF capacitor at differential mode.

23.6.3 Power

The power connection of audio codec is illustrated in Fig 23-15.

- The capacitor between AUDIO_VREF and ground should be 4.7nF. Larger capacitance value makes longer period needed for AVCC's stabilization.
- The capacitor between AVCC or AVCC_DRIV and ground should be 1uF or a little larger to keep voltage stable.

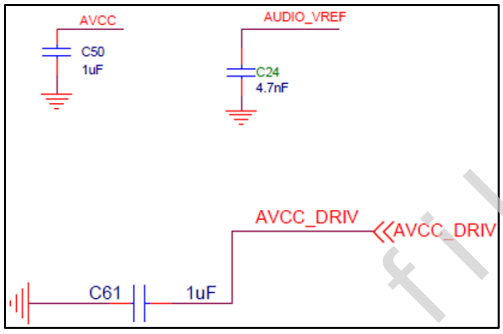


Fig 23-15 Power connection

23.7 Performance of Encoding & Decoding

23.7.1 AC3 Format

The performance of decoding AC3 format audio data is listed in Table 23-3.

Table 23-3 Performance of decoding AC3 format audio data

AC Channel	Bit Rate (kHz)	Sample Rate (kHz)	Output Channel	Average MIPS Measured[1]	Average MIPS Simulated[2]	Max. MIPS Simulated[2]
5.1	640	48	1	50.5	53.7	54.5
5.1	448	48	1	72.2	70.8	76.8
5.1	448	48	1	69.9	68.2	90.8
5.1	448	48	1	68.7	65.9	92.3
2.1	192	48	1	54.8	50.6	52.2
2.0	192	48	1	TBD	50.7	49.6

1. The values are evaluated on the AmebaD_QFN88_EVB_V1.
2. The values are evaluated with the Keil simulator.

23.7.2 OPUS Format

The simulated CPU load of encoding and decoding of OPUS audio data is listed in Table 23-4. Keil simulator is used during the whole process.

Table 23-4 Simulated CPU load of encoding and decoding of OPUS audio data

Rate (kHz)	Channel	Bit Rate (kbps)	Complexity	MIPS (decoding)	MIPS (encoding)	Measured Data
48	2	256	10	63	153.5	TBD
48	2	256	3	58	122.5	
48	2	256	0	53	102	
48	1	256	3	38	74.5	
16	1	48	3	8.5	46.5	
16	1	20	3	7	44.5	

If users decide to use libopus library to deal with their data, the code size requirement is listed in Table 23-5. The version used here is libopus 1.1.4.

Table 23-5 Code size requirement using libopus

Type	Program Size	Data Size
SILK encoder	77.1k	8.5k
Original Source Code (Encode + Decode)	133k	23.6k

In Table 23-6, CPU load on using Silk encoder is listed.

Table 23-6 CPU load on using Silk encoder

Test Condition	Average CPU Load (MHz) Complexity 3	Average CPU Load (MHz) Complexity 10	Remarks
SILK encoder, 16k, 1ch, 20kbps	42	182	SILK VBR

23.7.3 FLAC Format

The simulated (with Keil simulator) and measured CPU load for decoding FLAC audio data is listed in Table 23-7.

Table 23-7 CPU load of decoding FLAC audio data

Rate (kHz)	Channel	Word Length	MIPS Simulated	MIPS Measured
44.1	2	16	19	9.66

The tested file lasts for 279.64s, and the decoding time is 13.51s, which means 2.89s decoding time for 1-minute audio data.

23.7.4 AAC Format

The measured CPU load for decoding AAC audio data is listed in Table 23-8.

Table 23-8 CPU load of decoding AAC audio data

Rate (kHz)	Channel	Bit Rate (kbps)	MIPS
48	2	320	28.93

The tested file lasts for 24.576s, and the decoding time is 3.55s, which means 8.68s decoding time for 1-minute audio data.

23.7.5 MP3 Format

The measured CPU load for decoding MP3 audio data is listed in Table 23-9.

Table 23-9 CPU load of decoding MP3 audio data

Rate (kHz)	Channel	Bit Rate (kbps)	MIPS
32	2	320	30.43

The tested file lasts for 5.58s, and the decoding time is 849ms, which means 9.13s decoding time for 1-minute audio data.

23.8 Q & A

Q1 (EN): How to connect the output of DAC to a power amplifier?
Q1 (CN)：怎样将 DAC 的输出接至功放一侧？

A1 (EN): In our SDK, the output of DAC is configured as single-ended by default. The N-end should be left alone. The P-end of L/R should be connected to the amplifier respectively. Choose either the P-end of L or R if only one channel is needed.

- If the power amplifier is an AB type amplifier, refer to the design shown in Fig 23-16 (power amplifier is LM4991).
- If the power amplifier is a D type amplifier, refer to the design shown in Fig 23-17 (power amplifier is ALC1003).

A1 (CN)：在我们的 SDK 中，默认将 DAC 的输出配置为单端输出，所以不需要接 N 端。L/R 的 P 端应该分别接至功放对应的端口。若只需要一路输出，可以选择 L 或 R 任意一路。

- 当您使用 AB 类功放时，请参考 Fig 23-16 设计（功率放大器的型号是 LM4991）。
- 当您使用 D 类功放时，请参考 Fig 23-17 设计（功率放大器的型号是 ALC1003）。

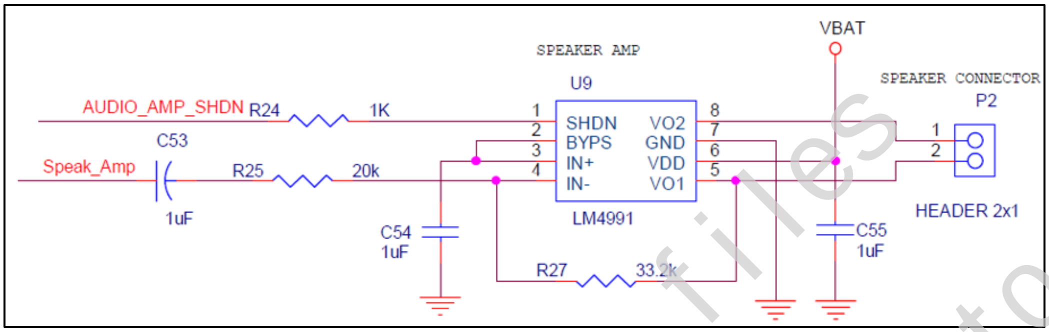


Fig 23-16 Reference design of using AB type power amplifier

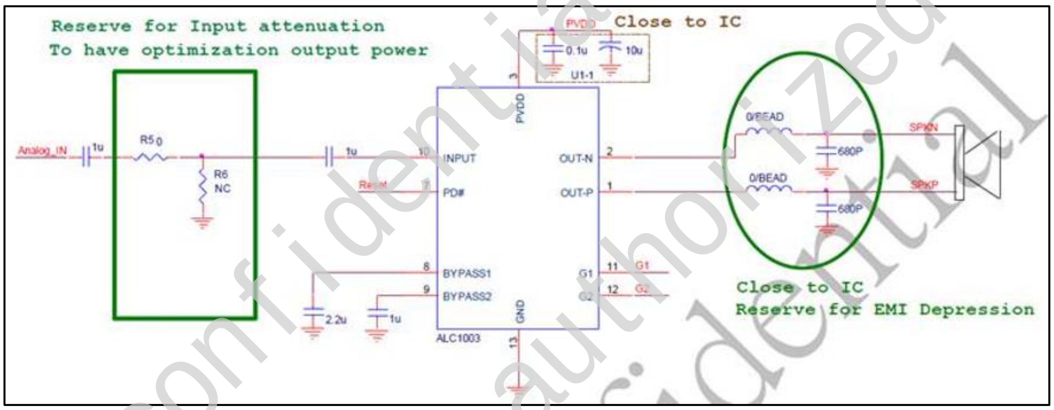


Fig 23-17 Reference design of using D type power amplifier

Q2 (EN): What's the difference between a single-ended mic input and a differential mic input?
Q2 (CN)：单端 mic 输入和差分 mic 输入有什么区别？

A2 (EN): The SNR of differential mic input is better than single-ended input, but one more pin is needed. Ameba-D supports 2 mics. One supports both single-ended input and differential inputs, but another only supports single-ended input.
A2 (CN)：差分输入的 SNR 比单端输入要好，但需要多用一个引脚。Ameba-D 有两个 mic，其中一个支持单端和差分输入，另一个只支持单端输入。

Q3 (EN): How to play local audio files?
Q3 (CN)：如何播放本地音频文件？

A3 (EN): If audio files are stored in SD card, you can use APIs related to SD card and file system to read and play audio files. If audio files are stored in flash, you need to copy the audio files to SRAM or PSRAM first, then call AUDIO_SP_TXGDMA_Init() to play them. Passing address directly to AUDIO_SP_TXGDMA_Init() doesn't work. The reason is that flash doesn't support GDMA transfer, but SRAM or PSRAM supports.
A3 (CN)：如果音频文件存放在 SD 卡中，则调用 SD card 和 file system 相关的 API 进行读取播放。如果音频文件存放在 flash 中，则需要将音频文件拷贝到 SRAM 或 PSRAM 中，再调用 AUDIO_SP_TXGDMA_Init()函数进行播放，而不能直接将文件地址传递给 AUDIO_SP_TXGDMA_Init()。这是因为 Flash 不支持 GDMA 传输。

24 DuerOS

24.1 DuerOS Platform

DuerOS is an important application of Baidu artificial intelligence technology. It has huge amounts of data, and can control or communication with the hardware through natural language. DuerOS is widely used in intelligent toys, bluetooth speakers, intelligent household appliances and other equipment.

DuerOS development platform aims to create intelligent terminal era of Artificial Intelligence (AI). Users can use it to build their own exclusive artificial intelligence products.

This section uses story machine as an example.

24.1.1 Apply product

DuerOS development platform home page url: http://open.duer.baidu.com/didp/main/index.

To apply an own product, please follow the steps:
(1) Enter DuerOS home page and register as a developer.

Fig 24-1 Register guide

(2) Configure new device.
a) After successful registered, click **Configure New Device** and enter the next step.

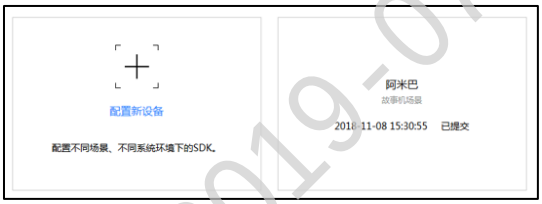


Fig 24-2 Device control platform

b) Configure the device as your need, story machine is chosen as an example.

Fig 24-3 Configure new product

c) Choose FreeRTOS and enter the next step.

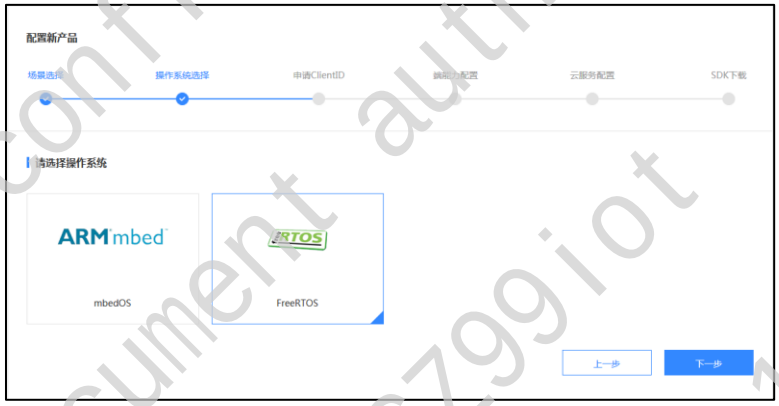


Fig 24-4 Choose FreeRTOS

d) Give the product name and push the **Apply ClientID** button.

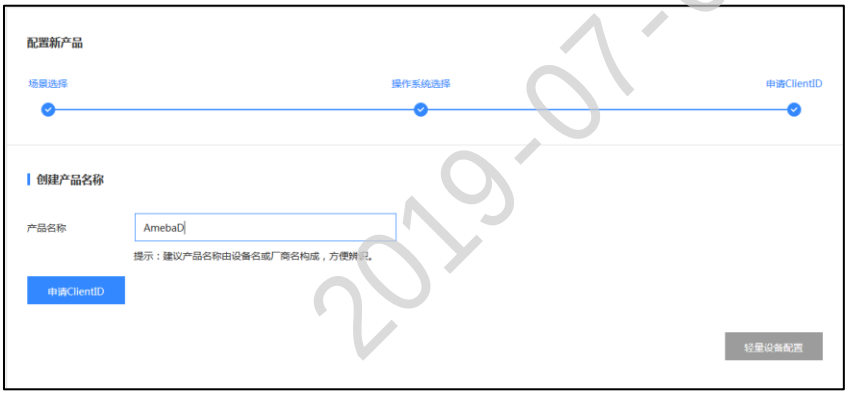


Fig 24-5 Apply ClientID

e) Push the **Light Device Configuration** button.

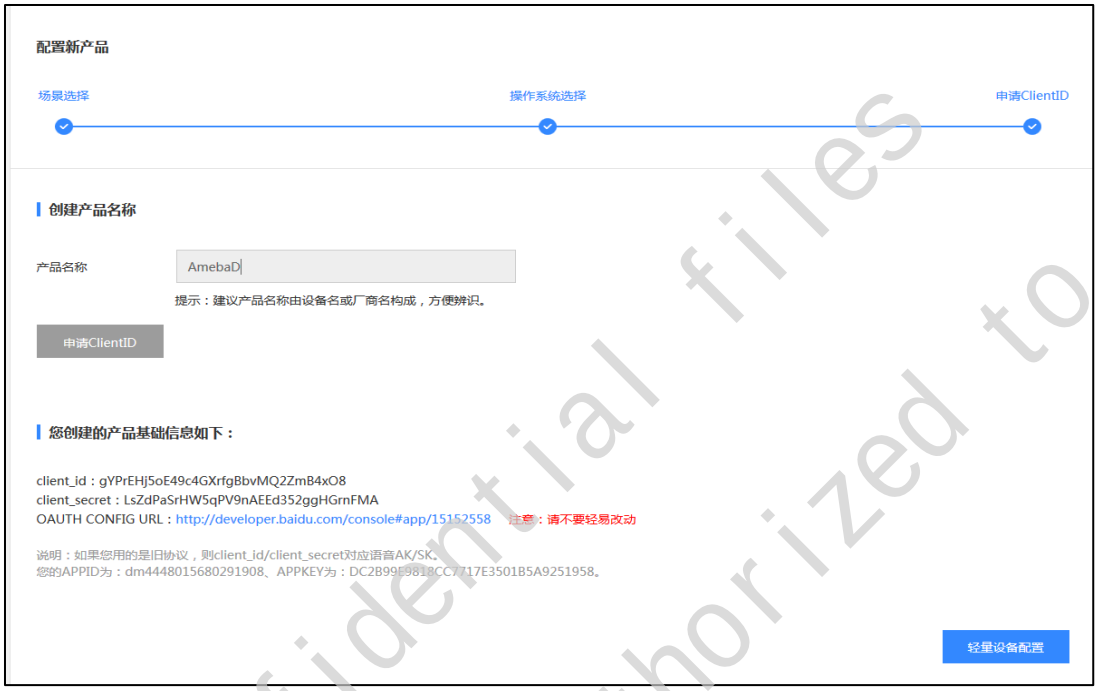


Fig 24-6 Configure device

After the above steps, story machine product has been created. You can get story machine information through **Product Information** menu, and can customize DuerOS service.

Fig 24-7 Product center

24.1.2 Apply Profile

After product has been applied, get a set of profiles through **Device Development** menu.

Fig 24-8 Get profile

Duer profile contains UUID, Token, server address, port, and root certificate. It is used to establish a TLS connection with the cloud and register to the cloud.

Note:

- Every UUID can only be used by one device.
- The profile can't be used directly, because we only support mp3 format now. PM should connect to Baidu to filter the other format for these profiles.

24.2 Hardware

- Ameba-D development board

Use AmebaD_QFN88_EVB_V1 and connect as Fig 24-9.

- MicroUSB

MicroUSB is used to provide power for Ameba-D board, and can view serial port log.

- Play equipment

Headset or speaker can be used to play audio data.

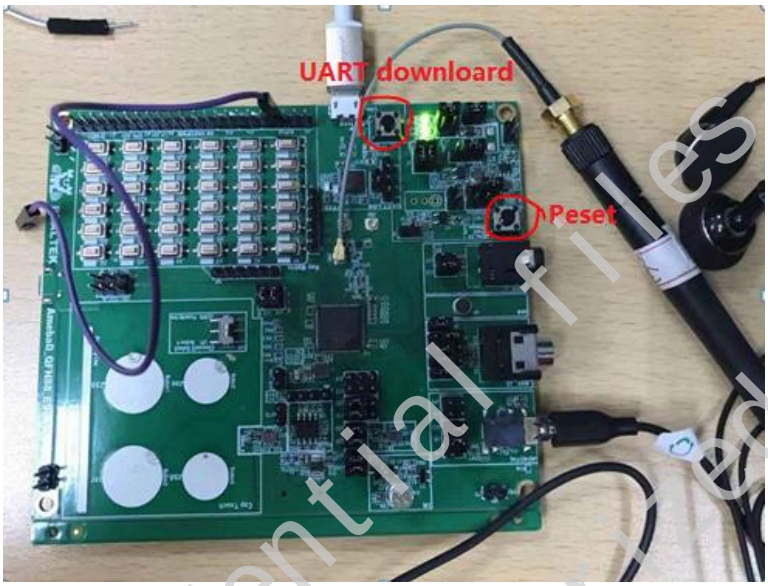


Fig 24-9 AmebaD_QFN88_EVB_V1

24.3 Software Component

DuerOS software component is shown in Fig 1-1.

Fig 24-10 Softwarecomponent

24.3.1 duerapp

Items	Description
include	duerapp header files
src	● public: Ameba-D initializes peripheral: Audio codec, Wi-Fi, OTA … ● private: parse play file, record from file, load profile.

24.3.2 libduer-device

Items	Description
external	DuerOS external library, include mbedtls, speex, zlibilte …
framework	Utilities and core.
modules	Modules: OTA, HTTP, dcs, coap …
platform	Connect and communicate with cloud

DuerOS libduer-device consists of three parts: Lightduer, Baidu CA, Speex.

- Lightduer is the interface between DuerOS and external, it communicates with FreeRtos, Lwip, Mbedtls.
- Baidu CA is used to connect and communicate with Baidu Cloud.
- Speex is responsible for voice compression.

24.4 How to Use DuerOS?

24.4.1 Build and Download

(1) Modify profile.

As every device should have a unique profile, so we should change the profile for each device. Zip the profile and choose one file to open. Copy the profile to the **component\common\application\baidu\duerapp\src\public\duerapp.c** and modify to the correct format.

```
#define DUER_PROFILE_UUID   1  //Range: 5,6,7,8,9
#if (PROFILE_FROM_SDCARD == 0)
#if (DUER_PROFILE_UUID == 1)
SDRAM_DATA_SECTION static const char duer_profile[] =
    "{\"configures\":\"{}\",\"bindToken\":\"9071f6a0954cfe179b9699c4b83c2437\",\"coapPort\":443,\
\"token\":\"hpUun7nsMgXgk3Vviq8jc7Gg89CBjeAq\",\"serverAddr\":\"device.iot.baidu.com\",\"lwm2mPort\":443,\
\"uuid\":\"19c800000000001\",\"rsaCaCrt\":\"-----BEGIN CERTIFICATE-----\\n\
MIIDUDCCAjgCCQCmVPUErMYmCjANBgkqhkiG9w0BAQUFADBqMQswCQYDVQQGEwJD\\n\
TjETMBEGA1UECAwKU29tZS1TdGF0ZTEOMAwGA1UECgwFYmFpZHUxGDAWBgNVBAMM\\n\
DyouaW90LmJhaWR1LmNvbTEcMBoGCSqGSIb3DQEJARYNaW90QGJhaWR1LmNvbTAe\\n\
Fw0xNjAzMTEwMzMwND1aFw0yNjAzMDkwMzMwND1aMGoxCzAJBgNVBAYTAkNOMRMw\\n\
EQYDVQQIDApTb211LVN0YXR1MQ4wDAYDVQQKDAViYW1kdTEYMBYGA1UEAwwPKi5p\\n\
b3QuYmFpZHUuY29tMRwwGgYJKoZIhvcNAQkBFg1pb3RAYmFpZHUuY29tMIIBIjAN\\n\
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtbhIeiN7pznzuMwsLKQj2xB02+51\\n\
OvCJ5d116ZFLjecp9qtllqOfN7bm+AJa5N2aAHJtsetcTHMitY4dtGmOpw4d1Gqx\\n\
luoz50kWJWQjVR+z6DLPnGE4uELOS8vbKHUoPPQTf80eNVn19S9h/17DcjEAJYC\\n\
IYJbf6+K9x+Ti9VRChvWcvgZQHMRym9j1g/7CKGMCiwkC+6ihkGD/XG40r7KRCyH\\n\
bD53KnBjBO9FH4IL3rGlZWKWzMw3zC6RTS2ekfEsgAtYDvROKd4rNs+uDU9xaBLO\\n\
dXTl5uxgudH2VnVzWtj09OUbBtXcQFD2IhmOi20BrckYul+HEIMR0oDibwIDAQAB\\n\
MA0GCSqGSIb3DQEBBQUAA4IBAQCzTTH91jNh/uYBEFekSVNg1h1kPSuj1wEDDf/\\n\
pjqPJPqrZvW0w0cmYsYibNDy985JB87MJMfJVESG/v0Y/YbvcnRoi5gAenWXQNL4\\n\
h2hf08A5wEQfLO/EaD1GTH3OIierkYZ6GItGrz4uFKHV5fTMiflABCdu37ALGjrA\\n\
rIjwjxQG6WwLr9468hkKrWNG3dMBHKvmqO8x42sZOFRJMkqBbKzaBd1uW4xY5XwM\\n\
S1QX56tVrgO0A3S+4dEg5uiLVN4YVP/Vqh4SMtYkL7ZZiZAxD9GtNnhRyFsW1C2r\\n\
OVSdXs1ttZxEaEBGUl7tgs8te556bIvufZX+BXGyycVJdBu3\\n\
-----END CERTIFICATE-----\\n\",\"macId\":\"\",\"version\":12237}";

#elif (DUER_PROFILE_UUID == 5)
```

Fig 24-11 Profile in duerapp.c

(2) Select AMIC or DMIC to record.

Modify macro CONFIG_DMIC_SEL in **project\realtek_amebaD_cm4_gcc_verification\inc\platform_opts.h**.

Items	Description
#define CONFIG_DMIC_SEL 0	Use AMIC to record
#define CONFIG_DMIC_SEL 1	Use DMIC to record

(3) Build project to generate DuerOS image.

(4) Use ImageTool to download DuerOS image into Ameba-D platform.

24.4.2 Configure the Network

After downloading the image successful, reset the Ameba-D board. If it is the first up, it enters the simple configuration mode. Then you can use the simple configuration tools to configure the network. If it is not the first reset, the SSID and password are saved on the flash, it connects to the network automatically.

```
18:09:01.397  Firmware Enable
18:09:01.400  ===>Retention Ram Init
18:09:01.413  Firmware Disable
18:09:01.417  [rltk_wlan_deinit] Wait for RxStop
18:09:01.446  WIFI deinitialized
18:09:01.449  Firmware Enable
18:09:01.451  ===>Retention Ram Init
18:09:01.464  Initializing WIFI ...
18:09:01.479  WIFI initialized
18:09:02.245
18:09:02.249  Switch to channel(2)
18:09:02.432
18:09:02.432  Switch to channel(3)
18:09:02.550
18:09:02.555  Switch to channel(4)
18:09:02.719
18:09:02.719  Switch to channel(5)
18:09:02.854
18:09:02.856  Switch to channel(6)
18:09:03.014
18:09:03.020  Switch to channel(7)
18:09:03.157
18:09:03.160  Switch to channel(8)
18:09:03.317
18:09:03.319  Switch to channel(9)
18:09:03.487
18:09:03.487  Switch to channel(10)
18:09:03.642
18:09:03.642  Switch to channel(11)
18:09:03.749  Input frame da: 01 00 5E 00 00 FB, data length: 39
18:09:03.781
```

Fig 24-12 Simple configuration log

24.4.3 DuerOS Connection Success

After connecting to the network successfully, the device connects to the Baidu Clouds automatically. When printing the log like Fig 24-13, it means DuerOS connection successful.

```
18:38:02.748  duerapp: duer_events_call_internal, handler not initialied...
18:38:02.749  duerapp: duer_engine_start, g_handler:0x10053e00, length:1469, profile:0x10042690
18:38:02.750  duerapp:     duer_conf_get_string: uuid = 19c80000000001
18:38:02.751  duerapp:     duer_conf_get_string: serverAddr = device.iot.baidu.com
18:38:02.807  duerapp: DNS lookup succeeded. IP=180.97.33.165
18:38:08.612  duerapp: soc:0x100435a8, destroy:1, fd:-1, ref_count:0, timer:0x10043580
18:38:09.136  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:09.142  duerapp: duer_engine_start, g_handler:0x10053e00, length:0, profile:0x0
18:38:09.520  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:09.921  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:10.048  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:10.143  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:10.150  duerapp: duer_engine_start, g_handler:0x10053e00, length:0, profile:0x0
18:38:10.367  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:12.130  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:12.164  duerapp: duer_engine_start, g_handler:0x10053e00, length:0, profile:0x0
18:38:16.158  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:16.178  duerapp: duer_engine_start, g_handler:0x10053e00, length:0, profile:0x0
18:38:23.929  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:23.968  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:26.670  duerapp: will start latter(DUER_ERR_TRANS_WOULD_BLOCK)
18:38:26.676  duerapp: connect started!
18:38:26.680  duerapp: Mutex initializing
18:38:26.682  duerapp: event: 0
18:38:26.685  duerapp: current vol 10, mute 0
18:38:26.687
18:38:26.689  duerapp: add resource successfully!!
18:38:26.692  duerapp: add resource successfully!!
```

Fig 24-13 DuerOS success log

24.4.4 Triger Record and Speak

(1) GPIOA_22 pull high to trigger audio codec record.
In driver GPIO_IRQ_PIN, set IRQ_RISE to trigger audio codec record, and GPIO_IRQ_PIN is PA_22, so use GPIOA_22 to trigger.

(2) Make a voice command.
After GPIOA_22 pull high, DuerOS would response with 'hello' tone. So after it says 'hello', you can make a voice command such as 'weather', 'sing a song', 'play a joke', 'what's your name' and so on to chat with DuerOS.

```
void init_voice_trigger_irq(void (*callback) (uint32_t id, gpio_irq_event event))
{
    gpio_irq_t voice_irq;

    //init voice trigger pin
    gpio_irq_init(&voice_irq, GPIO_IRQ_PIN, callback, NULL);
    gpio_irq_set(&voice_irq, IRQ_RISE, 1);
    gpio_irq_enable(&voice_irq);
}
```

Fig 24-14 Trigger code

24.5 OTA Upgrade

24.5.1 Generate OTA Image

(1) Change FIRMWARE_VERSION in the files: **component\common\application\baidu\duerapp\include\duerapp_ota.h**

```
#define FIRMWARE_VERSION "1.0.1.5"  //new version,update it when make new version
#define CHIP_VERSION     "RTL8721d"
#define SDK_VERSION      "1.0"
```

Fig 24-15 Firmware version

(2) Change OTA2 address mapping.
As the size of DuerOS image is more than 1M, there should be at least 4M flash if the OTA is needed. If the flash size is 4M, modify the **component\soc\realtek\amebad\fwlib\usrcfg\rtl8721d_bootcfg.c** as follows.

```
00038: /*
00039: * @brief    OTA start address. Because KM0 & KM4 IMG2 are combined, users only need to set the start address
00040: *     of KM0 IMG2.
00041: */
00042: BOOT_RAM_DATA_SECTION
00043: u32 OTA_Region[2] = {
00044:       0x08006000,         /* OTA1 region start address */
00045:       0x08206000,         /* OTA2 region start address */
00046: };
```

Fig 24-16 OTA Region

(3) Generate OTA image.
For Ameba-D, **km0_km4_image2.bin** in folder **project\realtek_amebaD_cm4_gcc_verification\asdk\image** is the OTA image.

24.5.2 OTA with DuerOS Platform

(1) Add OTA firmware.
(2) Click **Add** button, then fill in the OTA firmware information, such as version, file, file name, and file type.

Fig 24-17 Add OTA firmware

(3) Enter OTA verify page.

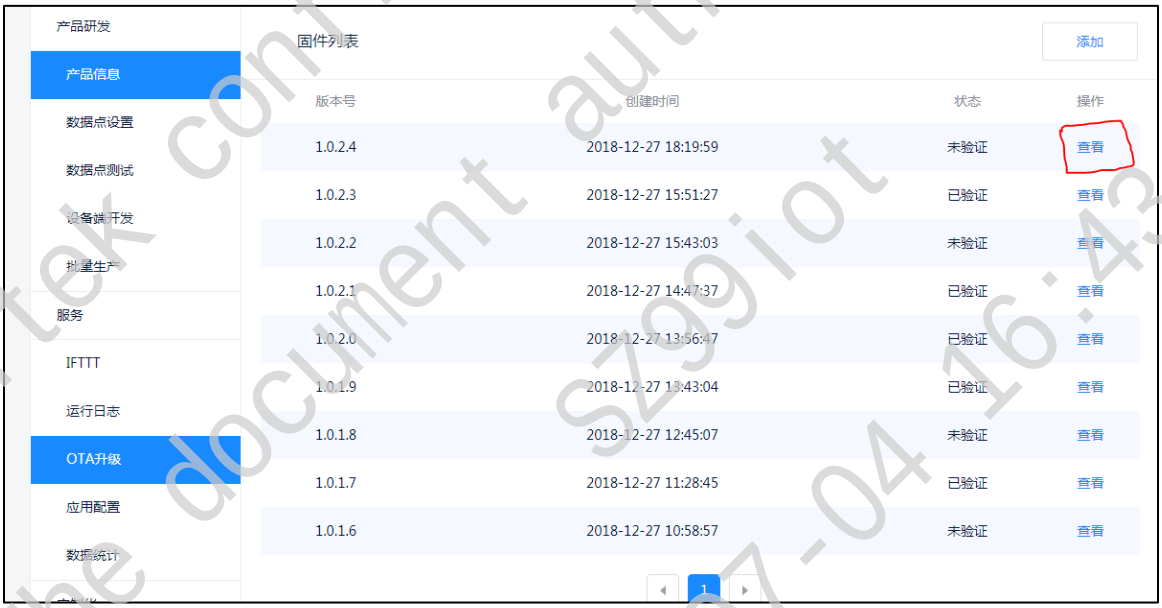


Fig 24-18 Push check button

(4) Verify the OTA firmware.

Fig 24-19 Push verify button

(5) Fill in the UUID to show OTA status.

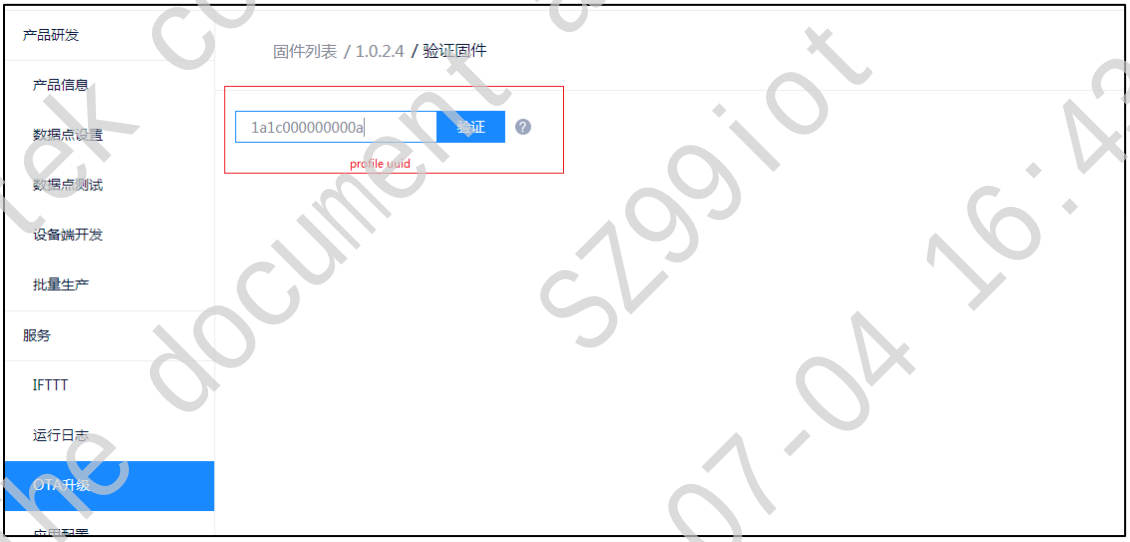


Fig 24-20 Input device profile number and verify

(6) Check whether OTA update succeed or not.

If OTA update succeed, then state upgrade is successful as Fig 24-21. Otherwise, OTA upgrade is failed.

Fig 24-21 OTA update result

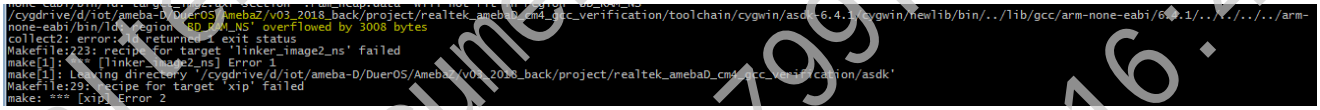
24.5.3 Attention

- OTA upgrade needs 4M byte flash.
- OTA firmware file name must consistent with the name in duer_ota_register_installer function in duerapp_ota.c.
- OTA firmware version should be later than the old image.

24.6 How to Debug?

24.6.1 Build Error

If find the following error, you must reduce the heap size in the file **project\realtek_amebaD_cm4_gcc_verification\inc\ FreeRTOSConfig.h.**



24.6.2 Wi-Fi Signal

Use ATWS command to check the Wi-Fi signal. If the signal is bad, remove the electric resistance shown in Fig 24-22 to check.

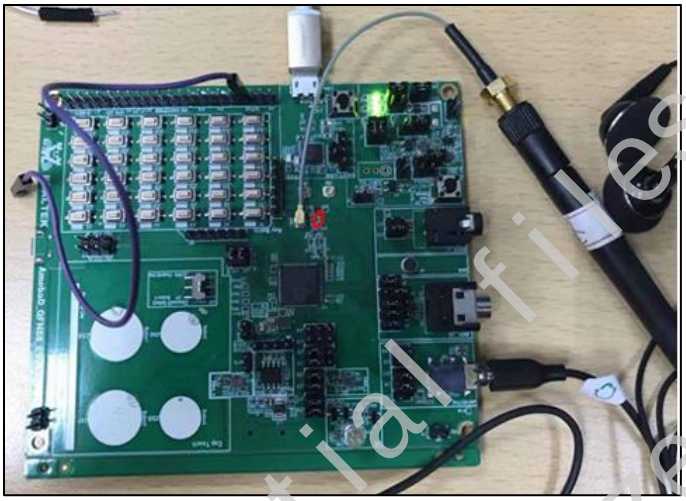


Fig 24-22 Electric resistance

24.6.3 The Recorder Cannot Be Recognized

(1) Check if the DMIC or AMIC is set correctly.
Check if the jumpers are connected correctly, and the DMIC or AMIC is set correctly.
Open **project\realtek_amebaD_cm4_gcc_verification\inc\platform_opts.h and check the macro CONFIG_DMIC_SEL.**

(2) Check if the network is ok.

```
18:57:25.976  #duerapp:     duer_conf_get_string: serverAddr = device.iot.baidu.com
18:57:30.875  duerapp: DNS failed!!!
18:57:30.879  duerapp: got ip failed!
18:57:30.882  duerapp: duer_coap_connect: transport connect failed by -52
18:57:30.885  duerapp: cached tracecode: 0x12010302
18:57:30.888  duerapp: start fail, status:0, rs:-52
18:57:30.891  duerapp: =====> event: 1[DUER_EVT_START], status: -1
18:57:30.893  duerapp: start fail! status:-1
18:57:30.896  duerapp: Action failed: event: 1, status: -1
18:57:30.900  duerapp: stop timer:0x0
18:57:30.904  duerapp: start last timer:0x0, soc:0x10043580
18:57:30.908  duerapp: no action
18:57:30.912  duerapp: =====> event: 5[DUER_EVT_STOP], status: -1
18:57:30.916  duerapp: connect stopped, status:-1!!
18:57:30.917  duerapp: event: 1
18:57:30.919  duerapp: duer_events_call_internal, handler not initialied...
18:57:30.920  duerapp: duer_events_call_internal, handler not initialied...
18:57:30.921  duerapp: duer_engine_start, g_handler:0x10053e00, length:1469, profile:0x10042690
18:57:30.923  duerapp:     duer_conf_get_string: uuid = 19c80000000001
```

Fig 24-23 Network fail log

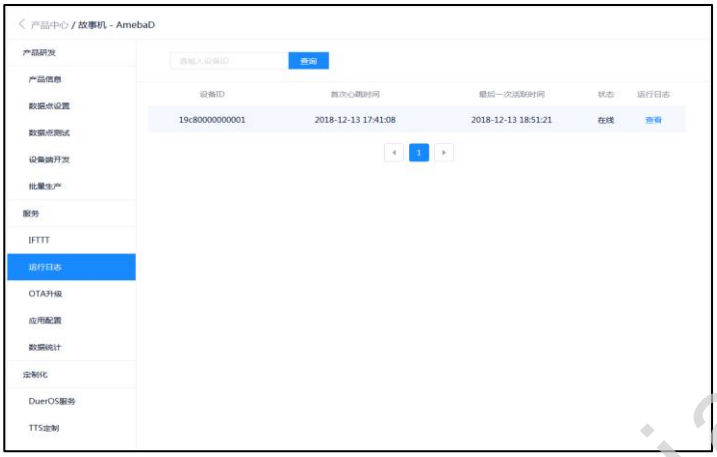
24.6.4 Check the memory

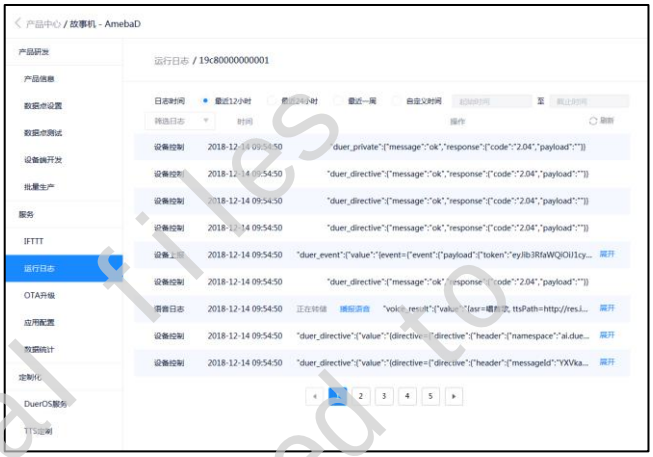
Use any command to check the heap size.

```
18:57:36.170
18:57:36.170  [MEM] After do cmd, available heap 55232
18:57:36.170
18:57:36.170
```

24.6.5 Check the Device status

You can see the running log on Baidu cloud.





25 Infrared Radiation (IR)

Ameba-D Infrared Radiation (IR) provides hardware modulation for Infrared Radiation sending and hardware auto capture for receiving.

This chapter introduces how to use IR.

25.1 Pinmux

The pin assignments of IR are listed in Table 25-1.

Table 25-1 IR pin assignments

Port Name	Pin Name	QFN48	QFN68	QFN88
PA[25]	IR_TX	Y	Y	Y
PA[26]	IR_RX	Y	Y	Y
PB[23]	IR_TX	N	Y	Y
PB[22]	IR_RX	N	Y	Y
PB[29]	IR_TX	N	Y	Y
PB[31]	IR_RX	N	Y	Y

25.1 Data Format

This section introduces the data format of Tx FIFO and Rx FIFO.

25.1.1 IR Tx

To send IR data, you should write the data into IR Tx FIFO register, so it's important to understand the data format to be sent to Tx FIFO register. Before sending data, you should convert the data into the appropriate format that Tx FIFO register can recognized. The size of Tx FIFO register is 32 bits, where:

- Bit[31] indicates data type.
  - 0: inactive carrier
  - 1: active carrier
- Bit[30] is data end flag.
  - 0: normal packet
  - 1: last packet
- Bit[27:0], represented by cycle, stores the data to be sent and the data format is the number of carrier cycles. Let $f_{carrier}$ represents the carrier frequency (the unit of $f_{carrier}$ is KHz), $T_{duration}$ represents the duration of carrier or no carrier symbol (the unit of $T_{duration}$ is us). Then,

$$\text{bit}[27:0] = f_{carrier} * T_{duration} / 1000$$

Take NEC protocol for example, the carrier frequency is 38KHz. The format of NEC is shown in Fig 25-1 and the modulation of NEC is shown in Fig 25-2. The NEC format consists of 2 start symbols, 64 data symbols and 1 stop symbol.

Fig 25-1 NEC format

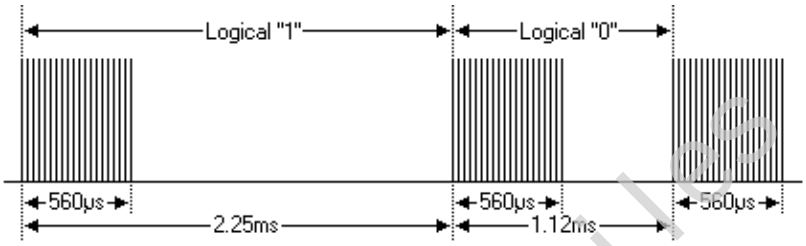


Fig 25-2 NEC modulation

- To send logical “1”, you should write two data into Tx FIFO register.
  - For the first data, bit[31] = 1, bit[30] = 0, bit[27:0] = 38*560/1000 = 21.
  - For the second data, bit[31] = 0, bit[30] = 0, bit[27:0] = 38*(2250-560)/1000 = 64.
- To send logical “0”, you should write two data into Tx FIFO register.
  - For the first data, bit[31] = 1, bit[30] = 0, bit[27:0] = 38*560/1000 = 21.
  - For the second data, bit[31] = 0, bit[30] = 0, bit[27:0] = 38*(1120-560)/1000 =21.

At last, you need to send the stop symbol. In this symbol, set bit[30] = 1 because the stop symbol indicates the data end of the current transmission.

25.1.2 IR Rx

To receive IR data, you should read the data in Rx FIFO register, so it’s important to understand the data format to be received in Rx FIFO register. The size of Rx FIFO register is 32 bits, where:

- Bit[31] indicates data level (0: high level, 1: low level)
- Bit[30:0] stores the data to be received and the data format is cycle duration. Let $f_{sampling}$ represents the sampling frequency (the unit of $f_{sampling}$ is KHz), T_{level} represents the duration of each high/low level (the unit of T_{level} is us). Then,

$$T_{level} = \frac{bit[30:0]}{f_{sampling}} * 1000$$

With the use of T_{level}, you can do further processing to get the information you need.

25.2 APIs

25.2.1 IR Setting APIs

25.2.1.1 IR_StructInit

Items	Description
Introduction	Fills each IR_InitStruct member with its default value.
Parameters	IR_InitStruct: pointer to an IR_InitTypeDef structure which will be initialized.
Return	N/A

25.2.1.2 IR_Init

Items	Description
Introduction	Initializes the IR peripheral according to the specified parameters in the IR_InitStruct.
Parameters	● IRx: selected IR peripheral. ● IR_InitStruct: pointer to a IR_InitTypeDef structure that contains the configuration information for the specified IR peripheral.
Return	N/A

25.2.1.3 IR_Cmd

Items	Description
Introduction	Enables or disables the selected IR mode.
Parameters	● IRx: selected IR peripheral. ● mode: selected IR operation mode. This parameter can be the following values: ■ IR_MODE_TX: Transmission mode. ■ IR_MODE_RX: Receiving mode. ● NewState: new state of the operation mode. This parameter can be: ENABLE or DISABLE.
Return	N/A

25.2.1.4 IR_INTConfig

Items	Description
Introduction	Enables or disables the specified IR interrupts.
Parameters	● IRx: selected IR peripheral. ● IR_INT: specifies the IR interrupt sources to be enabled or disabled. This parameter can be one or combinations of the following values: ■ IR_TX_FIFO_EMPTY_INT_EN: Tx FIFO empty interrupt. ■ IR_TX_FIFO_LEVEL_INT_EN: Tx FIFO threshold interrupt. ■ IR_TX_FIFO_OVER_INT_EN: Tx FIFO overflow interrupt. ■ IR_RX_FIFO_FULL_INT_EN: Rx FIFO full interrupt. ■ IR_RX_FIFO_LEVEL_INT_EN: Rx FIFO threshold interrupt. ■ IR_RX_CNT_OF_INT_EN: Rx counter overflow interrupt. ■ IR_RX_FIFO_OF_INT_EN: Rx FIFO overflow interrupt. ■ IR_RX_CNT_THR_INT_EN: Rx counter threshold interrupt. ■ IR_RX_FIFO_ERROR_INT_EN: Rx FIFO error read interrupt. Trigger when Rx FIFO empty and read Rx FIFO. ● NewState: new state of the specified IR interrupts. This parameter can be: ENABLE or DISABLE.
Return	N/A

25.2.1.5 IR_MaskINTConfig

Items	Description
Introduction	Mask or unmask the specified IR interrupts.
Parameters	● IRx: selected IR peripheral. ● IR_INT: specifies the IR interrupt sources to be mask or unmask. This parameter can be one or combinations of the following values: ■ IR_TX_FIFO_EMPTY_INT_MASK: Tx FIFO empty interrupt mask. ■ IR_TX_FIFO_LEVEL_INT_MASK: Tx FIFO threshold interrupt mask. ■ IR_TX_FIFO_OVER_INT_MASK: Tx FIFO overflow interrupt mask. ■ IR_RX_FIFO_FULL_INT_Msk: Rx FIFO full interrupt mask. ■ IR_RX_FIFO_LEVEL_INT_Msk: Rx FIFO threshold interrupt mask. ■ IR_RX_CNT_OF_INT_Msk: Rx counter overflow interrupt mask. ■ IR_RX_FIFO_OF_INT_Msk: Rx FIFO overflow interrupt mask. ■ IR_RX_CNT_THR_INT_Msk: Rx counter threshold interrupt mask. ■ IR_RX_FIFO_ERROR_INT_Msk: Rx FIFO error read interrupt mask. Trigger when Rx FIFO empty and read Rx FIFO. ● NewState: new state of the specified IR interrupts. This parameter can be: ENABLE or DISABLE.
Return	N/A

25.2.1.6 IR_GetINTStatus

Items	Description
Introduction	Get the specified IR interrupt status.

Parameters	IRx: selected IR peripheral.
Return	The new state of IR_INT (SET or RESET).

25.2.1.7 IR_GetIMR

Items	Description
Introduction	Get the specified IR interrupt mask status.
Parameters	IRx: selected IR peripheral.
Return	The new mask state of IR_INT (SET or RESET).

25.2.1.8 IR_FSMRunning

Items	Description
Introduction	Get the specified IR FSM status.
Parameters	IRx: selected IR peripheral.
Return	The new state of FSM: ● TRUE: RUN ● FALSE: IDLE

25.2.1.9 IR_ClearINTPendingBit

Items	Description
Introduction	Clears the IR interrupt pending bits.
Parameters	● IRx: selected IR peripheral. ● IR_CLEAR_INT: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values: ■ IR_TX_FIFO_EMPTY_INT_CLR: Clear Tx FIFO empty interrupt. ■ IR_TX_FIFO_LEVEL_INT_CLR: Clear Tx FIFO threshold interrupt. ■ IR_TX_FIFO_OVER_INT_CLR: Clear Tx FIFO overflow interrupt. ■ IR_RX_FIFO_FULL_INT_CLR: Clear Rx FIFO full interrupt. ■ IR_RX_FIFO_LEVEL_INT_CLR: Clear Rx FIFO threshold interrupt. ■ IR_RX_CNT_OF_INT_CLR: Clear Rx counter overflow interrupt. ■ IR_RX_FIFO_OF_INT_CLR: Clear Rx FIFO overflow interrupt. ■ IR_RX_CNT_THR_INT_CLR: Clear Rx counter threshold interrupt. ■ IR_RX_FIFO_ERROR_INT_CLR: Clear Rx FIFO error read interrupt. Trigger when Rx FIFO empty and read Rx FIFO.
Return	N/A

25.2.2 IR Tx APIs

25.2.2.1 IR_SendBuf

Items	Description
Introduction	Send data buffer.
Parameters	● IRx: selected IR peripheral. ● pBuf: data buffer to send. ● len: buffer length. ● IsLastPacket: this parameter can be the following values: ■ ENABLE: The last data in IR packet and there is no continuous data. In other words, an infrared data transmission is completed. ■ DISABLE: There is data to be transmitted continuously.
Return	N/A

Note: the difference between IR_SendBuf() and IR_SendData() is that IR_SendBuf() send a data buffer once and the number of sending data is the length of data buffer, while IR_SendData() only send one data once.

25.2.2.2 IR_SendData

Items	Description
Introduction	Send one data.
Parameters	● IRx: selected IR peripheral. ● data: data to send.
Return	N/A

25.2.2.3 IR_SetTxThreshold

Items	Description
Introduction	Set Tx threshold. When Tx FIFO depth <= threshold value, trigger interrupt.
Parameters	● IRx: selected IR peripheral. ● thd: Tx threshold.
Return	N/A

25.2.2.4 IR_GetTxFIFOFreeLen

Items	Description
Introduction	Get free size of Tx FIFO.
Parameters	IRx: selected IR peripheral.
Return	The free size of Tx FIFO.

25.2.2.5 IR_ClearTxFIFO

Items	Description
Introduction	Clear IR Tx FIFO.
Parameters	IRx: selected IR peripheral.
Return	N/A

25.2.3 IR Rx APIs

25.2.3.1 IR_ReceiveBuf

Items	Description
Introduction	Read data from Rx FIFO.
Parameters	● IRx: selected IR peripheral. ● pBuf: buffer address to receive data. ● len: read data length.
Return	N/A

Note: the difference between IR_ReceiveBuf() and IR_ReceiveData() is that IR_ReceiveBuf() read a data buffer once and the number of receiving data can be defined by users but shouldn't be larger than the data size in Rx FIFO, while IR_ ReceiveData() only read one data once. The data size in Rx FIFO can be required by calling IR_GetRxDataLen().

25.2.3.2 IR_ReceiveData

Items	Description

Introduction	Read one data.
Parameters	IRx: selected IR peripheral.
Return	Data which read from Rx FIFO.

25.2.3.3 IR_SetRxThreshold

Items	Description
Introduction	Set Rx threshold. When Rx FIFO depth > threshold value, trigger interrupt
Parameters	● IRx: selected IR peripheral. ● thd: Rx threshold.
Return	N/A

25.2.3.4 IR_GetRxDataLen

Items	Description
Introduction	Get data size in Rx FIFO.
Parameters	IRx: selected IR peripheral.
Return	Current data size in Rx FIFO.

25.2.3.5 IR_ClearRxFIFO

Items	Description
Introduction	Clear IR Rx FIFO.
Parameters	IRx: selected IR peripheral.
Return	N/A

25.2.3.6 IR_SetRxCounterThreshold

Items	Description
Introduction	Configure counter threshold value in receiving mode. You can use it to stop receiving IR data.
Parameters	● IRx: selected IR peripheral. ● IR_RxCntThrType: This parameter can be the following values: ■ IR_RX_Count_Low_Level: Low level counter value >= IR_RxCntThr, trigger IR_INT_RX_CNT_THR interrupt. ■ IR_RX_Count_High_Level: High level counter value >= IR_RxCntThr, trigger IR_INT_RX_CNT_THR interrupt. ● IR_RxCntThr: Configure IR Rx counter threshold value which can be 0 to 0x7fffffff.
Return	N/A

25.2.3.7 IR_StartManualRxTrigger

Items	Description
Introduction	Start trigger only in manual receive mode.
Parameters	IRx: selected IR peripheral.
Return	N/A

Note: manual Rx mode is seldom used.

25.3 IR Usage

25.3.1 Sending

25.3.1.1 Tx Polling Mode

To use IR sending function, the following steps are mandatory.
(1) Configure the IR pinmux according to Table 22-4.
For example, in order to use PB[23] as IR Tx pin, call the following function. And it is the same for other IR pins.
    Pinmux_Config(_PB_23, PINMUX_FUNCTION_IR);
(2) Call IR_Cmd() to disable IR.
(3) Set parameters, change some parameter if needed.
    IR_StructInit(IR_InitTypeDef *IR_InitStruct);
(4) Initialize hardware using the parameters in step (3).
    IR_Init(IR_InitTypeDef *IR_InitStruct);
(5) Write Tx data to FIFO using IR_SendBuf() or IR_SendData().
(6) Call IR_Cmd() to enable IR to start transmission.
(7) Write more data to FIFO if needed.

Note:
- In step (2) and step (6), It is suggested that disabling IR at first, and then enabling IR after writing data to FIFO.
- In step (5), pay attention to convert the data into the appropriate format that Tx FIFO register can recognized before writing data to FIFO. You can refer to section 25.1.1.

25.3.1.2 Special Notes

25.3.1.2.1 Tx FIFO Offset Issue

If you want to judge whether Tx data in FIFO has been sent completely or not, you'd better check Tx FIFO empty flag rather than TX_FIFO_OFFSET.

25.3.1.2.2 Tx Last Packet Cannot Let FSM Enter Idle Issue

if the last packet written to Tx FIFO cannot let Tx state machine enter idle, it is suggested that before enabling IR Tx, writing some data packets to Tx FIFO. You can refer to step (2) and step (6) in section 25.3.1.1.

25.3.2 Receiving

25.3.2.1 Rx Interrupt Mode

To use IR receiving function, the following steps are mandatory.
(1) Configure the IR pinmux according to Table 22-4.
For example, in order to use PB[22] as IR Tx pin, call the following function. And it is the same for other IR pins.
    Pinmux_Config(_PB_22, PINMUX_FUNCTION_IR);
(2) Set parameters, such as sampling frequency, Rx FIFO threshold level, Rx count threshold type, Rx count threshold level, Rx trigger mode if needed.
    IR_StructInit(IR_InitTypeDef *IR_InitStruct);
(3) Initialize hardware using the parameters in step (2).
    IR_Init(IR_InitTypeDef *IR_InitStruct);
(4) Configure interrupt if needed and register interrupt callback function.
    IR_INTConfig(IR_DEV, IR_RX_INT_ALL_EN, ENABLE);
    InterruptRegister((IRQ_FUN) IR_irq_handler, IR_IRQ, (u32)NULL, 10);
    InterruptEn(IR_IRQ, 10);
(5) Call IR_Cmd() to enable IR.

(6) Clear Rx FIFO by calling IR_ClearRxFIFO().
(7) When Rx FIFO threshold interrupt triggers, read data from Rx FIFO with the use of IR_ReceiveBuf() and IR_ReceiveData(), and make further processing in interrupt handle function.

Note:
- In step (7), to decode the receiving data correctly, you should understand the data format in Rx FIFO register. You can refer to section 25.1.2.
- Waveform inverse issue: in Rx ending, if the waveform is inverse, you should #define INVERSE_DATA in Ir_nec_protocol.h and set IR_InitStruct.IR_RxCntThrType = IR_RX_COUNT_HIGH_LEVEL.

25.3.2.2 Rx Learning

The process of Rx learning is similar to common Rx introduced in section 25.3.1.2.1. As shown in Fig 25-3, the difference is that in interrupt handle function, Rx learning should store each pulse of the Rx waveform, while common Rx only needs to store the carrier or un-carrier duration.

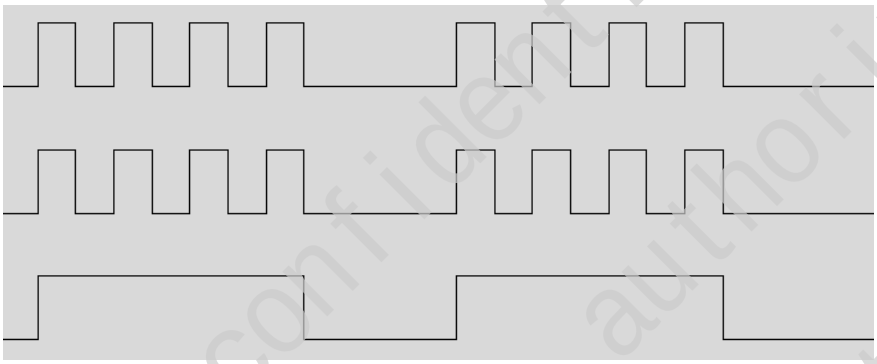


Fig 25-3 Difference of waveform between Rx learning and common Rx

Note:
- It is advised that putting the interrupt handle function code in RAM, and close other peripheral interrupt to avoid interfere.
- If the carrier frequency of learning waveform is larger than 400KHz, hardware may cannot respond to interrupt in time, which will result in decoding carrier frequency failed.

25.4 Compensation Mechanism

Tx waveforms are composed of some carrier symbols and no carrier symbols. Software calculates the duration of certain symbol by application specification (such as NEC). But no carrier symbols cannot be divisible by carrier frequency accurately. If there is no effective compensation mechanism, the error will increase. So Tx compensation is proposed.

Compensation mechanism is used in IR Tx to decrease the error. In most scenarios, it is not necessary to adopt compensation mechanism. Therefore, if you're not interested in compensation mechanism, just skip this section. Next we would introduce the application of compensation mechanism.

25.4.1 Tx Compensation Mechanism Application

Take NEC application for example, the Tx NEC waveform shown in Fig 25-4 is with $f_{carrier} = 38Khz$, and duty = 1/3. The system clock is 100MHz.

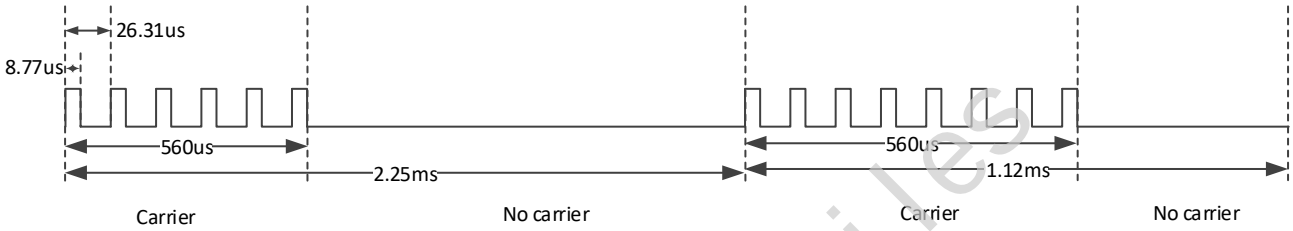


Fig 25-4 Tx NEC waveform

For 38KHz carrier frequency, $T_{carrier} = \frac{1}{f_{carrier}} = \frac{1}{38K} = 26.31us$, then $T_{carrier\_duty} = T_{carrier} * duty = 8.77us$

Compensation frequency f_{comp} is a dependent clock, you can set $f_{comp}$to any value you want. In this example, we set f_{comp} =1MHz, so $T_{comp} = \frac{1}{f_{comp}} = 1us$.

We divided this waveform into four symbols: 560us carrier symbol, 2250us ~ 560us no carrier symbol, 560us carrier symbol, and 1120us ~ 560us no carrier symbol.

- If no compensation mechanism is adopted, the real wave can be calculated as follows.
  For IR_TX_COMPENSATION = 0 (IR_TX_COMPENSATION is bit[28:29] of IR_TX_FIFO register)
  (1) The first carrier symbol: Referring to section 25.1.1, cycle $= 21$, 560us $\approx$ 21*26.31+8.77=561.28us.
  (2) The second no carrier symbol: cycle $= (2250 - 560) * 38/1000$=64, (2250-560)us = 1690us $\approx$ 64*26.31+(26.31-8.77)=1701.38us.
  (3) The third carrier symbol is the same as the first one.
  (4) The forth no carrier symbol: cycle $= 21$, 560us $\approx$ 21*26.31+(26.31-8.77)=570.05us.
- If compensation mechanism is adopted, the real wave can be calculated as follows.
  For IR_TX_COMPENSATION = 3
  (1) The first carrier symbol: Referring to section25.1.1, cycle $= 21$, 560us $\approx$21*26.31+8.77=561.28us.
  (2) The second no carrier symbol: cycle $= (2250 - 560) * 1000/1000$=1690, (2250-560)us = 1690us = 1690 * 1us.
  (3) The third carrier symbol is the same as the first one.
  (4) The forth no carrier symbol: cycle $= 560$, 560us = 560 * 1us.

Compare the above two calculating methods, we can find that by using compensation mechanism, the accuracy can be improved.

Note:

- Compensation mechanism can only be used for no carrier symbol.
- Compensation (IR_TX_COMPENSATION = 1) and method 2 (IR_TX_COMPENSATION = 2) are not recommended. If you want to use compensation mechanism, refer to method 3 (IR_TX_COMPENSATION = 3).

25.5 IR Schematic Design Guideline

25.5.1 Leakage

To avoid the leakage problem, we suggest using the IR circuit which is shown in Fig 25-5.

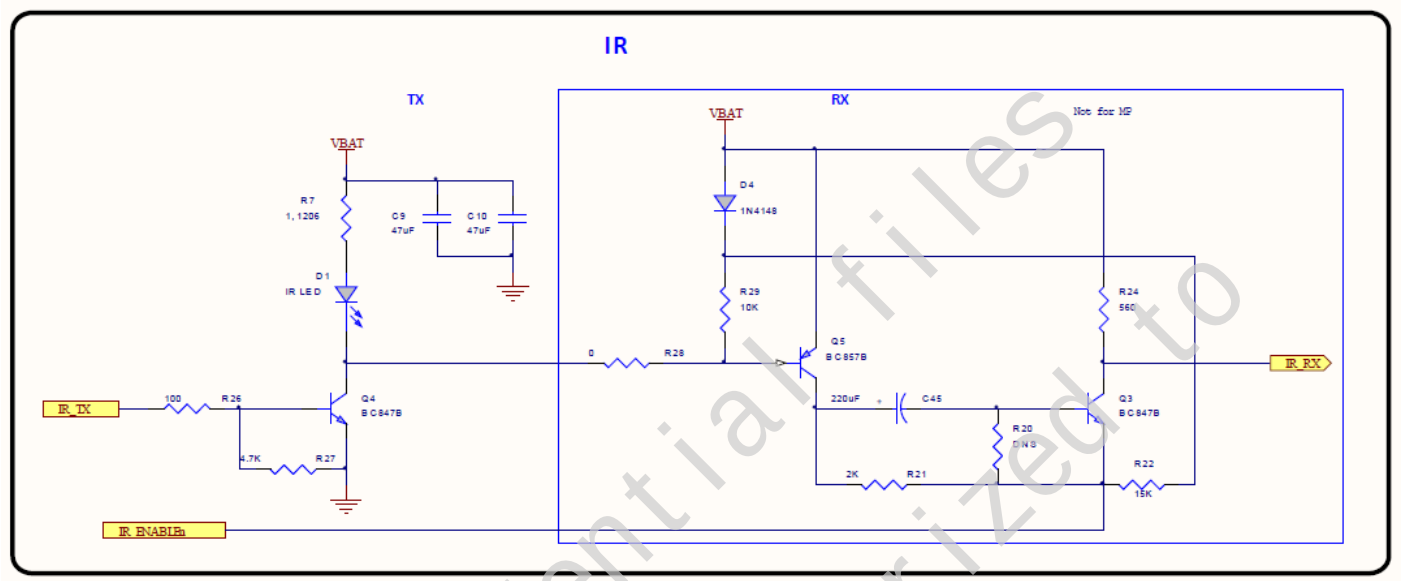


Fig 25-5 Circuit of IR

25.5.2 Carrier Problem in Rx Learning

Due to the characteristics of transistor, the response speed will slow down when the transistor works in deep saturation area. Therefore, when carrier frequency is set too large, the hardware has the risk of receiving carrier waveform failed in receiving end. In Rx learning, the maximum carrier frequency that can achieve is related to Rx circuit and the choice of transistor. The circuit we suggested in section 25.5.1 can support the maximum carrier frequency 70KHz, but you can choose the better transistor to acquire higher supported carrier frequency.

26 BOD

BOD is mainly used to notify a user that the voltage level is low for the applications which use batteries. Ameba-D provides many thresholds to choose, the alternative levels and corresponding voltage value is shown in Table 26-1 and Table 26-2.

Table 26-1 The high threshold of interrupt mode and reset mode (3.3V/1.8V)

Voltage	Value	Symbol	Reset	Interrupt
3.3V	001	BOR_TH_HIGH1	1.887	2.297
	010	BOR_TH_HIGH2	1.970	2.397
	011	BOR_TH_HIGH3	2.061	2.507
	100	BOR_TH_HIGH4	2.139	2.602
	101	BOR_TH_HIGH5	2.224	2.704
	110	BOR_TH_HIGH6	2.315	2.815
	111	BOR_TH_HIGH7	2.388	2.904
1.8V	001	BOR_TH_HIGH1	1.422	1.498
	010	BOR_TH_HIGH2	1.480	1.560
	011	BOR_TH_HIGH3	1.543	1.627
	100	BOR_TH_HIGH4	1.598	1.684
	101	BOR_TH_HIGH5	1.656	1.746
	110	BOR_TH_HIGH6	1.719	1.812
	111	BOR_TH_HIGH7	1.770	1.865

Table 26-2 The low threshold of interrupt mode and reset mode (3.3V/1.8V)

Voltage	Value	Symbol	Reset	Interrupt
3.3V	001	BOR_TH_LOW1	1.784	2.194
	010	BOR_TH_LOW2	1.863	2.290
	011	BOR_TH_LOW3	1.949	2.395
	100	BOR_TH_LOW4	2.023	2.486
	101	BOR_TH_LOW5	2.103	2.584
	110	BOR_TH_LOW6	2.190	2.690
	111	BOR_TH_LOW7	2.260	2.775
1.8V	001	BOR_TH_LOW1	1.345	1.422
	010	BOR_TH_LOW2	1.400	1.480
	011	BOR_TH_LOW3	1.460	1.543
	100	BOR_TH_LOW4	1.512	1.598
	101	BOR_TH_LOW5	1.567	1.656
	110	BOR_TH_LOW6	1.627	1.719
	111	BOR_TH_LOW7	1.674	1.770

Note: The voltage value may have ±5% error.

26.1 Recommended Threshold Parameter

The recommended threshold parameters are shown in Table 26-3.

Table 26-3 Recommended Threshold Parameter

Work Voltage	Reset		Interrupt	
	High Threshold	Low Threshold	High Threshold	Low Threshold
3.3V	BOR_TH_HIGH7	BOR_TH_LOW6	BOR_TH_HIGH7	BOR_TH_LOW6
1.8V	BOR_TH_HIGH5	BOR_TH_LOW3	BOR_TH_HIGH5	BOR_TH_LOW3

Note: To avoid voltage fluctuation during work trigger BOD interrupt or reset by mistake, the difference between high and low threshold can be appropriately increased.

26.2 BOD APIs

26.2.1 BOR_ModeSet

Items	Description
Introduction	Choose BOD mode and enable BOD function
Parameters	● Option: ■ BOR_RESET: BOD reset mode ■ BOR_INTR: BOD interrupt mode ● NewStatus: ■ ENABLE ■ DISABLE
Return	Null

26.2.2 BOR_ThresholdSet

Items	Description
Introduction	Set BOD high and low thresholds
Parameters	● Thres_Low: BOD low threshold ■ BOR_TH_LOW1 ■ BOR_TH_LOW2 ■ BOR_TH_LOW3 ■ BOR_TH_LOW4 ■ BOR_TH_LOW5 ■ BOR_TH_LOW6 ■ BOR_TH_LOW7 ● Thres_High: BOD high threshold ■ BOR_TH_HIGH1 ■ BOR_TH_HIGH2 ■ BOR_TH_HIGH3 ■ BOR_TH_HIGH4 ■ BOR_TH_HIGH5 ■ BOR_TH_HIGH6 ■ BOR_TH_HIGH7
Return	● Null

26.2.3 BOR_ClearINT

Items	Description
Introduction	Clear BOD interrupt.
Parameters	Null
Return	Null

26.2.4 BOR_DbncSet

Items	Description
Introduction	Set BOD interrupt mode debounce cycle
Parameters	● Option: ■ BOR_INTR: BOD interrupt mode

	● Dbnc_Value: debounce cycle, in unit of ANA4M clock cycles.
Return	Null

Note: Only BOD interrupt mode can set debounce cycle.

27 Flash Translation Layer (FTL)

27.1 Overview

NOR-Flash is comprised of blocks, which contains pages, and they contain individual cells of data. Flash read/write operations take place at page level. But erase operations take place at the block level. The flash needs to be erased before write. The memory portion for erasing differs in size from that for reading or writing, resulting in the major performance degradation of the overall flash memory system.

Therefore, a type of system software termed FTL (Flash Translation Layer) has been introduced, which is provided to make the flash a friendly medium to store data.

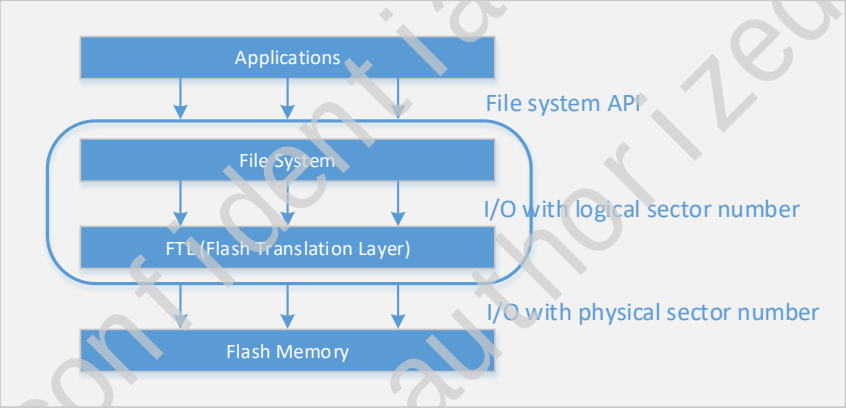


Fig 27-1 Architecture of flash memory system

The FTL algorithm provides the following functionalities:

- **Logical-to-physical address mapping:** Convert logical addresses from the file system to physical addresses in flash memory.
- **Power-off recovery**: Even when a sudden power-off event occurs during FTL operations, FTL data structures should be preserved and data consistency should be guaranteed.
- **Wear-leveling**: Wear down memory blocks as evenly as possible.

To write data to logical map, FTL would generate a data packet in specified format and store it in flash. To modify data in logical map, a new packet would be generated and appended to the end of physical map. When the physical pages are nearly full, the garbage collection is triggered to recycle the old pages which would be erased.

To read data from logical map, FLT would search the physical map to find the newest packet, which contains the data of specified address.

When using FTL, users don't need to care about physical map, which is maintained automatically by FTL.

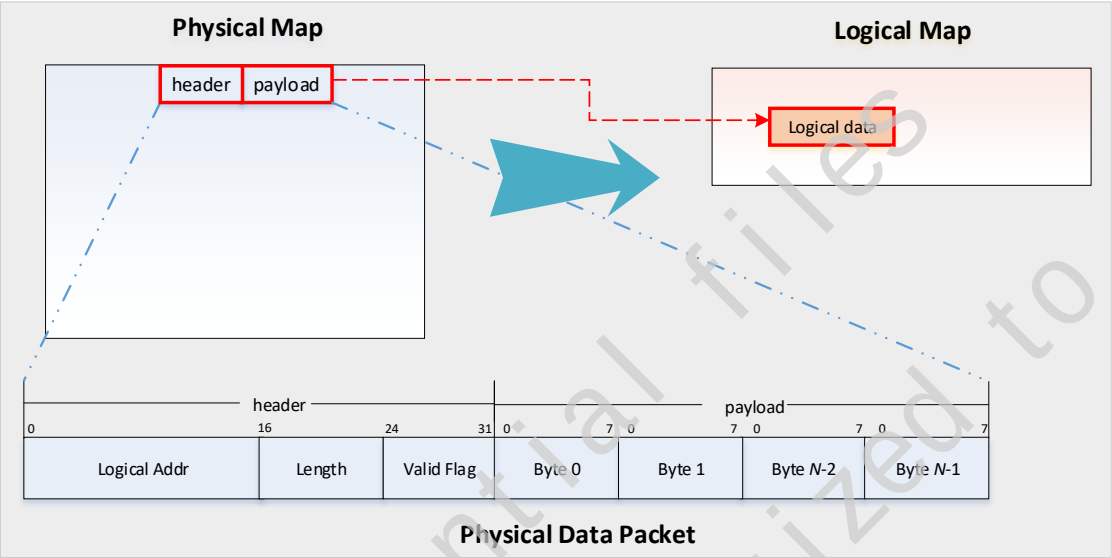


Fig 27-2 FTL overview

27.2 Features

- Physical Map
  - Physical Page size: 4096 bytes
  - Configurable physical page number
  - Physical map size = 4096 * physical page number
- Logical Map
  - The maximum logical map size is determined by physical map size: ((511 * (physical page number -1))-1) * 4
- Auto Garbage Collection
- Abnormal Power-off Protection

27.3 FTL APIs

API	Introduction
<ftl_init>	Initializes FTL
<ftl_load_from_storage>	Gets specified length of data from logical map.
<ftl_save_to_storage>	Writes specified length of data to logical map.

27.3.1 ftl_init

Items	Description
Introduction	Initializes FTL
Parameters	● u32PageStartAddr: The start address of physical map ● pagenum: The page number of physical map
Return	N/A

27.3.2 ftl_load_from_storage

Items	Description
Introduction	Gets specified length of data from logical map.
Parameters	● pdata_tmp: Pointer to a buffer to save logical data

	● offset: From which address to read the logical map ● size: The number of bytes to read
Return	● 0: Get logical map values successfully ● Others: Fail to get logical map value

27.3.3 ftl_save_to_storage

Items	Description
Introduction	Writes specified length of data to logical map.
Parameters	● pdata_tmp: Pointer to byte array of new logical data ● offset: From which address to update the logical map ● size: The number of bytes to update
Return	● 0: Write logical map values successfully ● Others: Fail to write logical map value

27.4 How to Use FTL

To use FTL in Ameba-D, the following steps are necessary.

(1) Define the following macro in **autoconf.h**, and the FTL initialization will be contained in main.

```
#define CONFIG_FTL_ENABLED 1
```

If Bluetooth is enabled in your system, **CONFIG_FTL_ENABLED** is defined automatically after define **CONFIG_BT_EN**.

(2) The default configurations for logical & physical map is as following:

a) 3 pages allocated for physical map: 0x0810_2000~0x0810_4FFF

b) logical map size: 4084 bytes

Note: If the memory layout is modified and overwrites FTL area 0x0810_2000~0x0810_4FFF, users need to modify the physical page address and define **FTL_MEM_CUSTEM** to **1** in rtl8721dhp_intfcfg.c as following:

```
#if defined(CONFIG_FTL_ENABLED)
#define FTL_MEM_CUSTEM      1
#if FTL_MEM_CUSTEM == 0
#error "You should allocate flash sectors to for FTL physical map as following, then set FTL_MEM_CUSTEM to 1. For more information, Please refer to Application Note, FTL chapter. "
#else
const u8 ftl_phy_page_num = 3;                    /* The number of physical map pages, default is 3*/
const u32 ftl_phy_page_start_addr = 0x00102000;   /* The start offset of flash pages which is allocated to FTL physical map.
                                                     Users should modify it according to their own memory layout! */
#endif
#endif
```

Otherwise, an error would be thrown out to remind users to modify FTL start address as following. If no other region overlay with FTL area, only need to define FTL_MEM_CUSTEM to 1.

```
  -o rtl8721dhp_intfcfg.o
/cygdrive/e/AmebaD/AmebaD_svn/project/realtek_amebaD_cm4_gcc_verification/asdk/../../../component/soc/realtek/amebad/fwlib/usrcfg/rtl8721dhp_intfcfg.c:64:2: error: #error "You shou
ld allocate flash sectors to for FTL physical map as following, then set FTL_MEM_CUSTEM to 1. For more information, Please refer to Application Note, FTL chapter. "
 #error "You should allocate flash sectors to for FTL physical map as following, then set FTL_MEM_CUSTEM to 1. For more information, Please refer to Application Note, FTL chapter.
"
  ^~~~~
make[4]: *** [/cygdrive/e/AmebaD/AmebaD_svn/project/realtek_amebaD_cm4_gcc_verification/asdk/Makefile.include.gen:449: rtl8721dhp_intfcfg.o] Error 1
make[4]: Leaving directory '/cygdrive/e/AmebaD/AmebaD_svn/project/realtek_amebaD_cm4_gcc_verification/asdk/make/target/fwlib'
make[3]: *** [Makefile:19: all] Error 2
```

(3) Call *ftl_load_from_storage/ftl_save_to_storage* functions to read from/write to logical map.

28 Audio Signal Generation and Analysis

28.1 Introduction

When developing digital mic (DMIC) related applications, you may need to generate a signal of a certain frequency to test whether your DMIC works well or not. In this application note, we first introduce how to generate a signal and how to analyze the frequency of a signal collecting by the DMIC. Then, digital analog – analog digital (DAAD) loopback is explained. DAAD loopback is helpful in determining which part (codec or DMIC) doesn't work well when problems occur.

28.1.1 Compilation

You need to add the **example_audio_signal_generate.c** to your project before you can use all the commands mentioned in this AN. Related File used are placed under the path: **/component/common/example/audio_sport/audio_signal_generate**.

Follow two steps to use this function:
(1) Set the macro named "AUDIO_SIGNAL_GENERATE" to 1 in **platform_opts.h** under the path: **/project/realtek_amebaD_va0_example/inc/inc_hp**
(2) In Cygwin terminal, change to the directory **/project/ realtek_amebaD_va0_example / GCC-RELEASE / project_hp,** type **make menuconfig,** and enable audio related configurations (MENUCONFIG FOR CHIP CONFIG > Audio Config > Enable Audio).

28.1.2 Generating a Signal of a Certain Frequency

Frequencies of signals supported by the program range from 20~20000Hz, which is the frequency range that people can hear.

Type command "Audio_generate tone spkr (f)" to generate a signal whose frequency is equal to f ($20 < f < 20000$).

28.1.3 Analyzing Signals Collected by DMIC

One way of testing a DMIC is to analyze the signals collected by the DMIC in frequency domain. If the detected frequency is the same as the frequency of the signal, the DMIC works well.

Type command "Audio_generate dmic fft 1" to analyze the signal collected by DMIC in frequency domain. The main frequency and its relative strength are printed out. Due to frequency leaks, the frequency detected may not be the same as the one you play, but the number should be closed. You can use this command to analyze signals whose frequencies range from 20~20000Hz.

28.1.4 DAAD

DMIC related problems can be caused by internal codec or DMIC itself. In this part, we will explain how to use DAAD loopback to determine whether the internal codec works in a normal way or not.

Fig 28-1 depicts how DAAD loopback works. The data transmitted from DA Digital IP is received directly by AD Digital IP without the interference of DMIC.

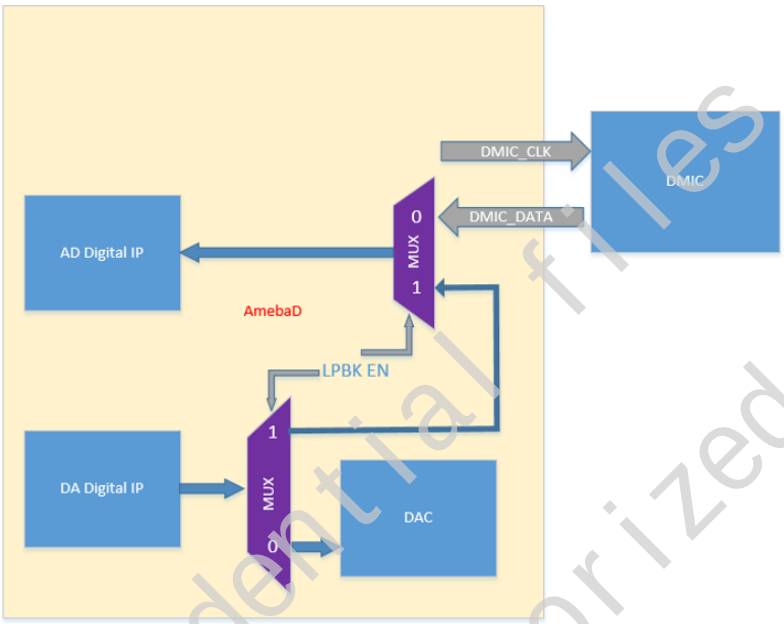


Fig 28-1 Illustration of DAAD loopback

Type command “Audio_generate daad fft (*f*)” to generate a signal whose frequency is *f*. The main frequency detected and relative strength calculated will be printed out. If the frequency detected is closed to the one you generate, it means that the internal codec works well. The problem may be caused by DMIC.

28.1.5 Command

All the commands that can be used are listed in Table 28-1. Each command is composed of several parameters, and the parameter which are surrounded by () should be set by users.

Table 28-1 Available commands in audio signal generation and analysis

Command Format	Command Function	Example
Audio_generate tone spkr (*f*)	Generates a tone whose frequency is *f*	Audio_generate tone spkr 1000
Audio_generate dmic fft 1	Analyzes the signal collected by DMIC	Audio_generate dmic fft 1
Audio_generate daad fft (*f*)	DAAD loopback analysis	Audio_generate daad fft 1000
Audio_generate stop	Stops audio test	Audio_generate stop

29 Key-Scan

Ameba-D Key-Scan provides up to 6*6 (36) keypad array with 12 GPIOs, which can be configured for different applications, for example: 5*7 or 3*4 keypad arrays. Also multi-key detection and low power mode are supported.

This chapter introduces the APIs of Key-Scan, and how to use Key-Scan.

29.1 Pinmux

The pin assignments of Key-Scan are listed in Table 22-4.

Table 29-1 Key-Scan pin assignment

Port Name	Pin Name	QFN48	QFN68	QFN88
PA[12]	KEY_ROW0	Y	Y	Y
PA[13]	KEY_ROW1	Y	Y	Y
PA[14]	KEY_ROW2	Y	Y	Y
PA[15]	KEY_ROW3/KEY_COL6	Y	Y	Y
PA[16]	KEY_ROW4/KEY_COL5	N	Y	Y
PA[17]	KEY_ROW6/KEY_COL3	N	Y	Y
PA[18]	KEY_ROW5/KEY_COL4	N	Y	Y
PA[19]	KEY_COL2	N	Y	Y
PA[20]	KEY_COL7	N	N	Y
PA[21]	KEY_ROW7	N	N	Y
PA[25]	KEY_COL1	Y	Y	Y
PA[26]	KEY_COL0	Y	Y	Y

29.2 APIs

29.2.1 keyscan_array_pinmux

Items	Description
Introduction	Initialization of pinmux settings and pad settings
Parameters	● col: selected column number depending on KeyColumn (for example: col0 & col2, col is 5) ● row: selected column number depending on KeyRow
Return	N/A

29.2.2 keyscan_getdatanum

Items	Description
Introduction	Gets data number of Key-Scan FIFO
Parameters	obj: Key-Scan object defined in application software
Return	Data number of Key-Scan FIFO

29.2.3 keyscan_read

Items	Description
Introduction	Reads data from Key-Scan FIFO
Parameters	● obj: Key-Scan object defined in application software. ● pBuf: buffer to save data read from Key-Scan FIFO

	● num: number of data to be read
Return	N/A

29.2.4 keyscan_init

Items	Description
Introduction	Initializes the Key-Scan device.
Parameters	obj: Key-Scan object defined in application software
Return	N/A

29.2.5 keyscan_set_irq

Items	Description
Introduction	Enables or disables the specified Key-Scan interrupts mask.
Parameters	● obj: Key-Scan object defined in application software. ● keyscan_IT: specifies the Key-Scan interrupt sources to be enabled or masked. ● newstate: new state of the specified Key-Scan interrupts mask.
Return	N/A

29.2.6 keyscan_clear_irq

Items	Description
Introduction	Clears the specified Key-Scan interrupt pending bit.
Parameters	● obj: Key-Scan object defined in application software. ● keyscan_IT: specifies the Key-Scan interrupt to be cleared.
Return	N/A

29.2.7 keyscan_get_irq_status

Items	Description
Introduction	Gets Key-Scan interrupt status.
Parameters	obj: Key-Scan object defined in application software.
Return	Interrupt status

29.2.8 keyscan_set_irq_handler

Items	Description
Introduction	Sets Key-Scan interrupt handler.
Parameters	● obj: Key-Scan object defined in application software. ● func: the interrupt handler function.
Return	N/A

29.2.9 keyscan_enable

Items	Description
Introduction	Enables the specified Key-Scan peripheral
Parameters	obj: Key-Scan object defined in application software.
Return	N/A

29.2.10 keyscan_disable

Items	Description
Introduction	Disables the specified Key-Scan peripheral
Parameters	obj: Key-Scan object defined in application software.
Return	N/A

29.2.11 keyscan_clearfifodata

Items	Description
Introduction	Clears the FIFO data
Parameters	obj: Key-Scan object defined in application software.
Return	N/A

29.3 Key-Scan Usage

There are two ways to use the Key-Scan: using default settings or using APIs listed in 29.2.

29.3.1 Using Default Configuration

When using default settings, only the keypad can be configured.

The Key-Scan initialization is implemented by the function **app_keyscan_init**() in **main.c** . User can follow these steps to use it.
(1) Set the km0_enable_key_touch in ps_config (**Rtl8721dlp_sleepcfg.c**) to BIT_KEY_ENABLE to enable Key-Scan when KM0 boots.

```
PSCFG_TypeDef ps_config = {
    .km0_config_wifi_enable = TRUE,
    .km0_enable_key_touch = FALSE, //BIT_KEY_ENABLE | BIT_CAPTOUCH_ENABLE,
    .km0_tickles_debug = TRUE,          /* if open WIFI FW, should close it, or beacon will lost in WOWLAN */
    .km0_osc2m_close = TRUE,
    .km0_pg_enable = FALSE,
    .km0_rtc_calibration = FALSE,
    .km0_audio_pad_enable = TRUE,
};
```

(2) Configure the GPIOs in the following structure (**touch_key.c**), and set valid to 1 for the pins which will be used in keypad.

```
static KeyMatrix_TypeDef KeyRow[8] =
{
    /* pinmux,      valid */
    {_PA_12, 1}, /* row 0 */
    {_PA_13, 1}, /* row 1 */
    {_PA_14, 1}, /* row 2 */
    {_PA_15, 1}, /* row 3 */
    {_PA_16, 1}, /* row 4 */
    {_PA_18, 0}, /* row 5 */
    {_PA_17, 0}, /* row 6 */
    {_PA_21, 0}, /* row 7 */
};

static KeyMatrix_TypeDef KeyColumn[8] =
{
    /* pinmux,      valid */
    {_PA_26, 1}, /* Col 0 */
    {_PA_25, 1}, /* Col 1 */
    {_PA_19, 1}, /* Col 2 */
    {_PA_17, 1}, /* Col 3 */
    {_PA_18, 1}, /* Col 4 */
    {_PA_16, 0}, /* Col 5 */
    {_PA_15, 0}, /* Col 6 */
    {_PA_20, 1}, /* Col 7 */
};
```

(3) Rebuild SDK and re-burn images, the Key-Scan would work after boot up.

29.3.2 Using APIs

When using APIs, more parameters can be configured.

(1) A Key-Scan object must be defined in application software first, then the necessary parameters must be configured.
For the parameter row, if row 0, row 2 and row 5 will be used, row must be set to 0x25 (100101), the same as parameter col. For the parameter clk, scan clock=bus clock/(clk+1).

```
struct keyscan_s {
    u32 row;
    u32 col;
    u32 clk;
    u32 workmode;     //0 for regular scan mode, 1 for event trigger mode
    u32 keylimit;
    u32 overctrl;   //0 for discard new, 1 for discard oldest
};
```

(2) Use **keyscan_init()** in **keyscan_api.c** to initialize Key-Scan.
(3) Use **keyscan_set_irq()** in **keyscan_api.c** to enable the interrupt wanted.
(4) Use **keyscan_set_irq_handler()** in **keyscan_api.c** to handle the triggered interrupt.
(5) Use **keyscan_enable()** in **keyscan_api.c** to enable the Key-Scan. Then the Key-Scan would work.

30 BT Coexistence

30.1 Introduction

Bluetooth (BT) is a wireless technology standard. It enables short-distance data exchange between fixed devices, mobile devices, and builds personal area networks.

Bluetooth (BT) and WLAN both occupy the 2.4GHz to 2.4835GHz unlicensed band. They will most likely work concurrently in the same environment, so the mutual interference between them should be considered. Here we provide the so-called PTA (Packet Traffic Arbitration) mode to relieve the interference.

30.2 PTA Mode

The PTA is actually a block of hardware circuit, which is used only when both Bluetooth and WLAN send transfer request to RF. As depicted in Fig 30-1, the main idea is that the PTA circuit gathers the transfer status information (Tx/Rx, including the corresponding priority) from both sides to decide which side to use RF. The judgement strategy is based on the coexistence table (1# or 2#) and break table, which can be configured by software. The input signals ANT_SEL and OOB mean antenna selection and out of band selection respectively, and they are used to select Coex_Table_1 or Coex_Table_2 to apply. The outcome of PTA is GNT_BT signal, which is further transferred to RF circuit, which is adopted by RFC to make switches between WLAN and BT Modem.

Note: The output signal GNT_WL is mainly used in a chip which has dual-antenna, while the chip refered here is a single-antenna chip.

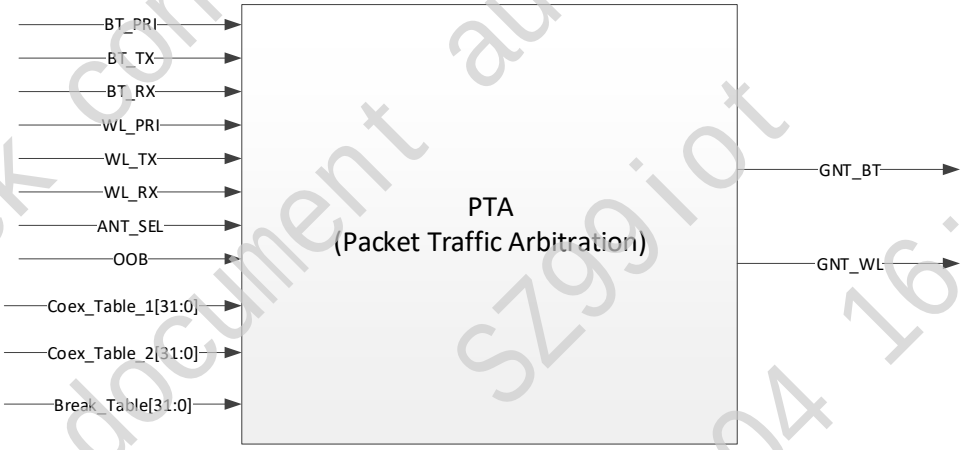


Fig 30-1 PTA module port diagram

Table 30-1 PTA module pin definition

Pin	Description
BT_PRI	Bluetooth High priority traffic indication ● 0: Low Priority ● 1: High Priority
BT_TX	Bluetooth Request Tx or is on transmitting
BT_RX	Bluetooth Request Rx or is on receiving
WL_PRI	WLAN High priority traffic indication ● 0: Low Priority ● 1: High Priority
WL_TX	WLAN Request Tx or is on transmitting
WL_RX	WLAN Request Rx or is on receiving
ANT_SEL	Single or dual antenna selection ● 0: Single

	● 1: Dual
OOB	Out of band indication ● 0: In band ● 1: Out of band This bit should come from software.
Coex_Table_1 [31:0]	Coexistence table, used at dual antenna & out of band
Coex_Table_2 [31:0]	Coexistence table, used at dual antenna & in band or single antenna
Break_Table_1[15:0]	Break table, used to break BT activity, when new WLAN request is coming and BT is on transmitting or receiving, BT should break their activity if GNT_WL asserts.
Break_Table_1[31:15]	Break table, used to break WLAN activity, when new BT request is coming and WLAN is on transmitting or receiving.
GNT_BT	Grant Bluetooth Request. It is used by WLAN BB, which can break the WLAN Tx/Rx on air immediately or not. And to WLAN MAC, it is like the CCA coming from AIR.
GNT_WL	Grant WLAN Request. BT should break their activity if GNT_WL asserts.

30.3 Related Parameters

30.3.1 PTA Parameters

From the perspective of software design, the parameters of PTA are mainly refered to Coex_Table and Break_Table values. The common cases are given in Table 30-2.

Table 30-2 Common cases of Coex_Table and Break_Table settings

Index	Priority Cases	Coex_Table_1 and Coex_Table_2	Break Table
1	BT > WLAN	0x55555555	0x0000FFFF
2	WLAN > BT	0xAAAAAAAA	0xFFFF0000
3	WLAN high-Priority > BT > WLAN low-Priority	0xAAAA5555	0xFF0000FF
4	BT high-Priority > WLAN > BT low-Priority	0x5A5A5A5A	0x3333CCCC
5	WLAN high-Priority > BT high-Priority > WLAN low-Priority> BT low-Priority	0xAAAA5A5A	0xFF3300CC

Among the five cases, we use the fourth case (index = 4) in our project, which distinguishes the BT high and low priority traffic. That is to say, if WLAN is transferring data while BT high-priority Tx/Rx is requesting, the WLAN traffic will be preempted. If BT low-priority transfer is ongoing, the WLAN Tx/RxX request can preempt RF's usage from BT.

30.3.2 BT Peripheral Parameters

If the chip acts as a BT peripheral, all the activities related to BLE are set to high in the BLE Controller. Because the BT peripheral acts as the slave role after the connection is established. The advertising is configurable, while the other two parameters, pairing interval and data transfer interval, depend on the counterpart.

Table 30-3 BT peripheral parameters

BT Peripheral Item	Value (ms)	Priority	Configurable
Advertising interval	200	high	√
Pairing interval	50 (it depends)	high	×
Data transfer interval	7.5 (typical Android mobile)	high	×

30.3.3 BT Central Parameters

If the chip acts as a BT central, all the parameters can be configured in the BLE Controller. The default values are listed in Table 30-4.

Table 30-4 BT central parameters

BT Central Item	Value (ms)	Priority	Configurable
Scan window	10	high once, followed by low 4 times	√

Scan interval	25	/	√
Connection interval	100 (default)	high	√

30.4 How to Configure BT Coexistence?

30.4.1 PTA

The only thing you can deal with PTA is changing coexistence table. The default setting allows WLAN high-priority traffic to preempt BT traffic and BT traffic to preempt WLAN low-priority traffic, which corresponds to the index 3 in Table 30-2.

If you want to change the settings of PTA, you need to have knowledge of the traffic priority in use. The traffic priorities of WLAN and BT are decided by WLAN MAC and BT Controller, respectively. Therefore, we recommend you NOT to change the PTA coexistence table. If you do it by yourself, it may cause a crash in coexistence decision.

Table 30-5 Coexistence table configuration in PTA

Related Register/Variable	Function Name	File Name
HAL_WRITE32 (WIFI_REG_BASE, 0x6C0, 0x5A5A5A5A); HAL_WRITE32 (WIFI_REG_BASE, 0x6C4, 0x5A5A5A5A);	u32 bt_coex_pta_only(void);	Halbtc8721d_phy.c
File location: \component\common\drivers\wlan\realtek\src\hal\btc		

30.4.2 BT Peripheral

The only thing you can deal with BT peripheral is changing the peripheral's advertising interval. The advertising interval relates to a tradeoff between energy-efficiency and responsiveness. If the advertising interval is set to a bigger value, the better energy-efficiency and worse responsive latency are expected.

Typically, the advertising interval can be configured to a value between 20ms and 10.24s according to BLE specification. The default setting is 200ms in the project.

Table 30-6 Advertising interval configuration in BT peripheral

Related Register/Variable	Function Name	File Name
#define DEFAULT_ADVERTISING_INTERVAL_MIN 320 #define DEFAULT_ADVERTISING_INTERVAL_MAX 320	void app_le_gap_init(void);	Ble_app_main.c
File location: \component\common\bluetooth\realtek\sdk\example\ble_peripheral		

30.4.3 BT Central

In the case of BT central, the configurable timing parameters are scan window, scan interval and connection interval.

The parameters, scan interval and scan widow, can change the duty-ratio of scanning operation. That is to say, the quotient of the scan window and scan interval can reflect the intensity of BLE scanning, which is commonly between 0 and 1. Furthermore, the scan interval and scan window have an effect on the latency performance of BLE during the scan process. The more time is spent on the scan, the lower latency we can expect. The typical range of scan interval or scan window is between 20ms and 10.24s according to BLE specification. In the default setting, the scan interval and scan window are set to 25ms and 10ms, respectively. The tests for quality control show that the default setting can get a good performance of scanning latency. To conclude, we do NOT recommend you to modify scan interval and scan widow only if a strict latency performance is required in your project.

The parameter, connection interval, represents the data transfer frequency between BLE master and slave after connection has been established. The direct result of modifying connection interval is the BLE throughput. If we set connection interval to a smaller value, the BLE throughput may become higher. The typical value of the connection interval is set to an integer multiple of 1.25ms. The default setting is 100ms, which corresponds to an estimated BLE throughput of 1k bytes/s. A smaller connection interval is usually linked with frequent RF switches between BLE and WLAN, which can have an impact on the WLAN performance. Therefore, the connection interval less than 50ms is NOT recommended to apply.

Table 30-7 Scan and connection related parameter configuration in BT central

Related Register/Variable	Function Name	File Name

conn_req_param.scan_interval = 0x10;
conn_req_param.scan_window = 0x10;
conn_req_param.conn_interval_min = 80;
conn_req_param.conn_interval_max = 80;
	int ble_central_at_cmd_connect(int argc, char **argv);	Ble_central_at_cmd.c

File location: \component\common\bluetooth\realtek\sdk\example\ble_central		
Remark: If the scan interval and scan window are set to the same value, the BLE controller enlarges the scan interval itself.		

31 BT Examples

Ameba-D BT examples provide a set of Bluetooth functionality, such as BT config, BT peripheral, BT central, BT Beacon.

This chapter illustrates the Bluetooth Low Energy (BLE) profiles and how to test BT examples in our SDK.

31.1 BLE Profiles

In Bluetooth Core Specification, profile defines features and functions that are available in protocol, and implementation of interaction details between devices, so as to accommodate Bluetooth protocol stack to application development in various scenarios.

The relationship between profile and protocol in Bluetooth specification is shown in Fig 15-1.

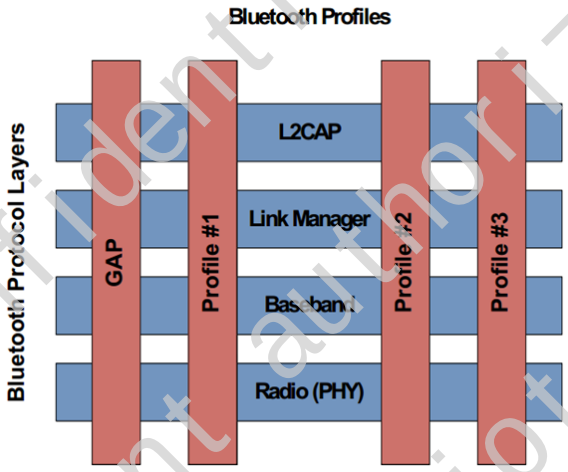


Fig 31-1 Bluetooth profiles

As shown in Fig 15-1, profiles are illustrated in red rectangular, containing GAP, Profile #1, Profile #2, and Profile #3. They can be classified into two types:

- Generic Access Profile (GAP)
- Generic Attribute Profile (GATT) Based Profile (Profile #1, Profile #2 and Profile #3).

31.1.1 GAP

GAP is the basic profile which must be implemented by all Bluetooth devices, and is used to describe actions and methods including device discovery, connection, security requirement, and authentication.

GAP makes your devices visible by other devices, and determines whether your device can connect to other devices or how to interact with other devices.

GAP has four application roles: Broadcaster, Observer, Peripheral and Central.

Roles	Description
Broadcaster	● Broadcaster is applicable to applications sending data only via broadcast. ● Send advertising events ● Cannot create connections
Observer	● Observer is applicable to applications receiving data via broadcast. ● Scan for advertising events ● Cannot initiate connections

Peripheral	● Peripheral is applicable to applications setting link connection. ● Send advertising events ● Can accept the establishment of LE link and become a slave role in the link
Central	● Central is applicable to applications setting a single or multiple link connections ● Scan for advertising events ● Can initiate connection and become a master role in the link

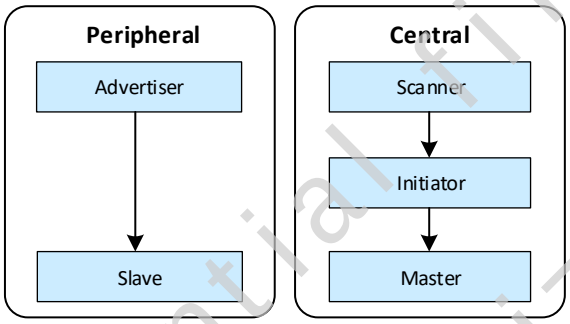


Fig 31-2 Peripheral and central roles

GAP has two network topology: Broadcast Network Topology and Connected Network Topology.

Network Topology	Description
Broadcast Network Topology	● A peripheral sends data to more than one device at a time. ● BLE peripheral sends data one-way to any devices in the listening range.
Connected Network Topology	● A peripheral can only be connected to one central device (such as a mobile phone) at a time. ● One central device can be connected to multiple peripherals. ● GATT services and characteristics is used to communicate in both directions.

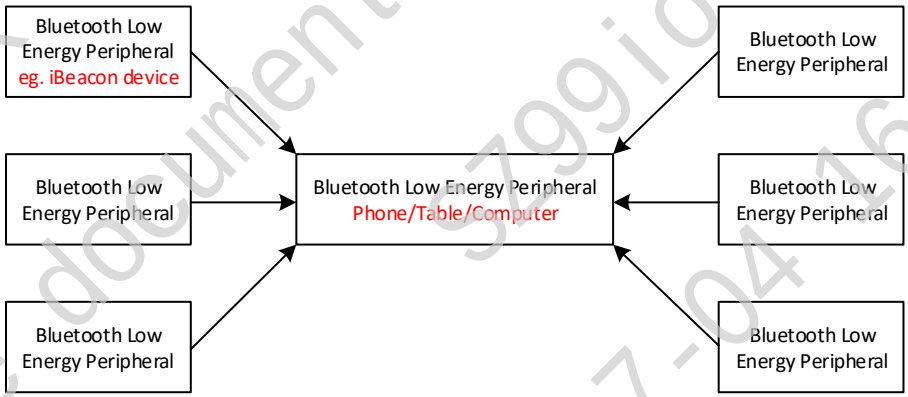


Fig 31-3 Broadcast network topology

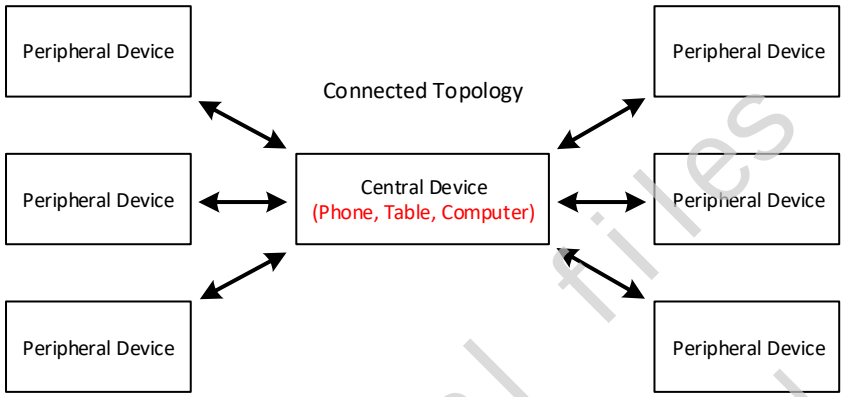


Fig 31-4 Connected network topology

31.1.2 GATT

GATT is a commonly used profile in Bluetooth specification. GATT profile is based on server-client interaction. It is used to meet various application cases and used for data interaction between devices as specified.

GATT is based on GAP connection, a BLE peripheral can only be connected to one central device (a mobile phone, etc.) at a time.

GATT has two roles: server and client.

Roles	Description
Server	● Accept incoming commands and requests from the client ● Send responses, indications and notifications to a client.
Client	● Discovery services at GATT Server ● Receive and handle indications and notifications from GATT Server ● Send read/write request to GATT Server.

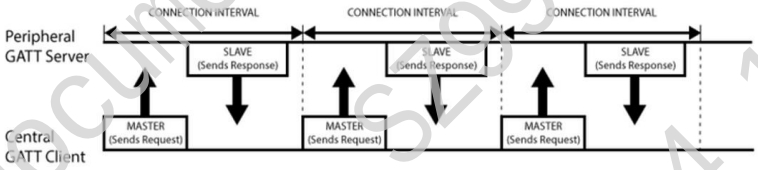


Fig 31-5 GATT server-client

GATT profile is made up in the form of Service and Characteristic. Profile may contain one or more GATT services, service is a group of characteristics in set, as shown in Fig 31-6.

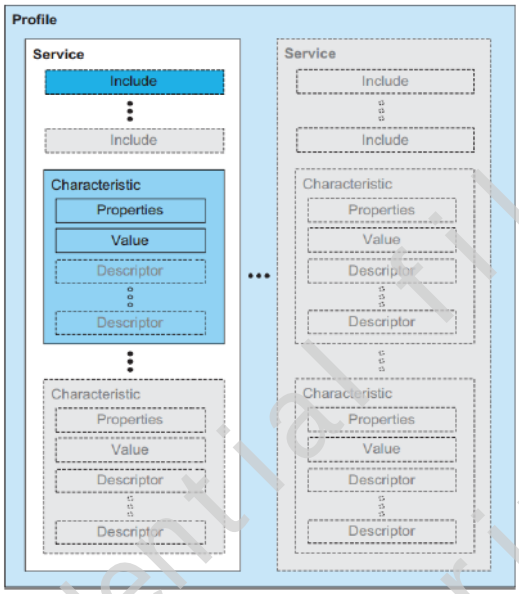


Fig 31-6 GATT based profile hierarchy

Services are used to break data up into logic entities, and contain specific chunks of data called characteristics. A service can have one or more characteristics, and each service distinguishes itself from other services by UUID.

The lowest level concept in GATT transactions is the characteristic, which encapsulates a single data point. Each characteristic consists of a type (a UUID), a value, and descriptors.

31.2 BT Config

BT config provides a simple way to use BT for a Wi-Fi device to associate to AP easily. User only has to enter passphrase (if needed) for the first time associating to a new AP. After that, whenever user wants Ameba to associate to that AP, user just needs to click a button on the BT Config app, and everything is done.

31.3 BT APIs

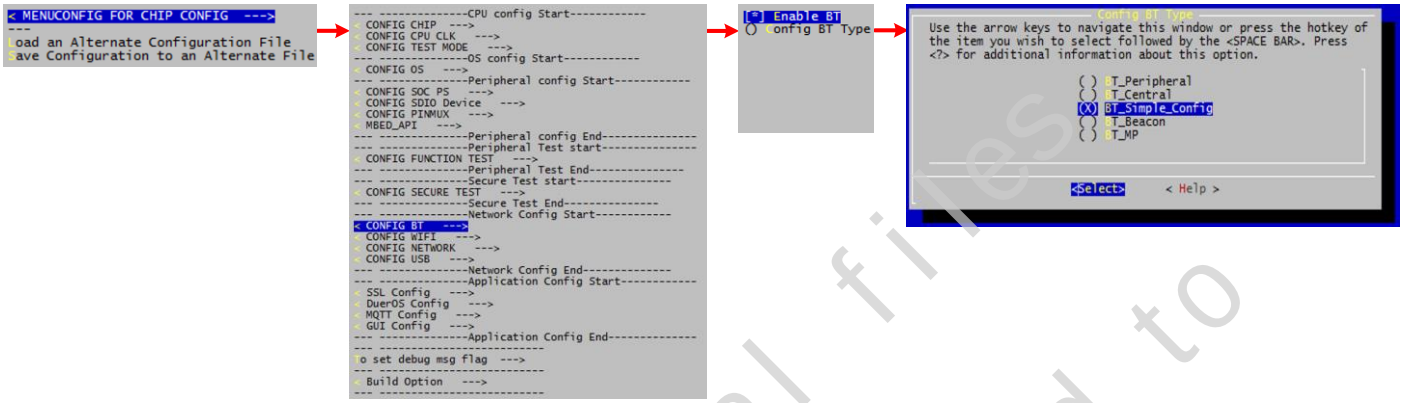
For details about APIs, please refer to document: UM0403 Ameba-D BLE Stack User Manual v1.0.0 EN.pdf.

31.4 How to Test BT Examples?

31.4.1 BT Config

31.4.1.1 Image Generation

(1) Enter SDK path: project/realtek_amebaD_cm4_gcc_verification, input command **make menuconfig** to enable BT Config.



(2) Allocate FTL physical map by setting FTL_MEM_CUSTEM to 1 in **rtl8721dhp_intfcfg.c** file.

```
#if defined(CONFIG_FTL_ENABLED)
#define FTL_MEM_CUSTEM          1
#if FTL_MEM_CUSTEM == 0
#error "You should allocate flash sectors to for FTL physical map as following, then set FTL_MEM_CUSTEM to 1. For mo
#else
const u8 ftl_phy_page_num = 3;                                  /* The number of physical map pages,
const u32 ftl_phy_page_start_addr = 0x00102000;                 /* The start offset of flash pages which
                                                                   Users should modify it accor
#endif
#endif
```

(3) Build image and use ImageTool to download image to your board.

31.4.1.2 APP Installation

The installation package is located at tools\bluetooth\BT Config in SDK. You can install Android or iOS as your phone OS.

31.4.1.3 Test Procedure

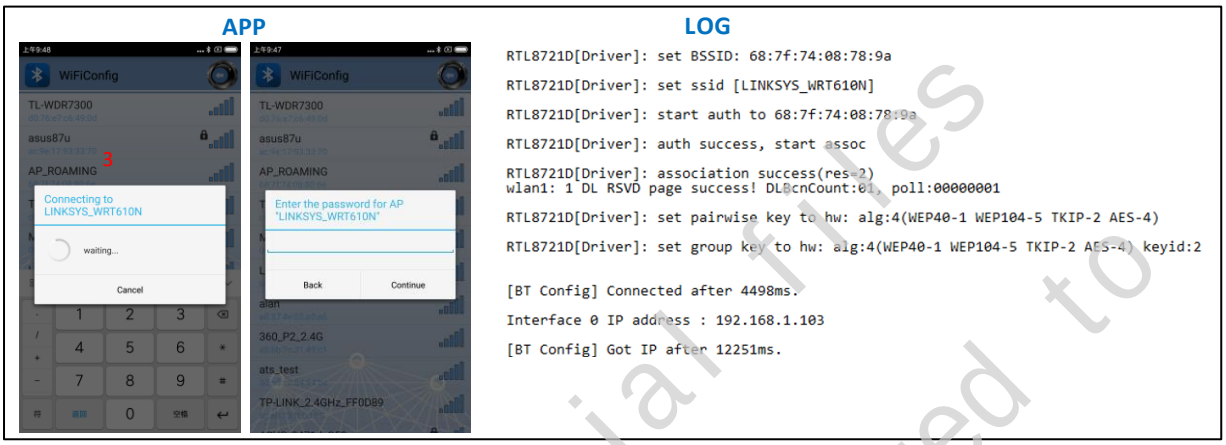
(1) Reset your Ameba-D board, and input command **ATBB=1** in your trace tool.

```
#ATBB=1
[ATBB]:_AT_BT_CONFIG_[ON]


BT BUILD [BT Config WDate: 2019/ifi] bt_conf05/06-15:20ig_app_task_:54
init

[MEM] After do cmd, availhcable heap 1i_rtk_parse_44952

config: BT A
#
DDRESS  11 28 36 12 51 89, use the defaut config
BT Reset...
bt_iqk_efuse_valid: no data
bt_iqk_logic_efuse_valid: no data
bt_check_iqk:  NO IQK LOK DATA need start LOK,
we need start iqk
hci_tp_rf_radio_ver
continue add  0
hci_tp_rf_radio_ver
continue add  1
hci_tp_rf_radio_ver
continue add  2
hci_tp_rf_radio_ver
bt_iqk_dump:    DUMP,
the IQK_xx  data is 0xfc,
the IQK_yy  data is 0x3f9,
the QDAC  data is 0x20,
the IDAC  data is 0x1f,

IQK OK

hci_tp_config:BT INIT success 7
Start upperStack

[BT Config Wifi] BT Config Wifi ready

[BT Config Wifi] ADV started
```

(2) Click the BT config icon to launch it. Scan and connect with Ameba-D BT using BT config app.

(3) Start to scan Aps of Ameba-D remotely.

(4) Select an AP to connect to.



(5) Confirm.

After confirming BT config result, Bluetooth connection is disconnected.

Also you can click "Try another AP" to go back to Wi-Fi scan list page and choose another AP to connect to.

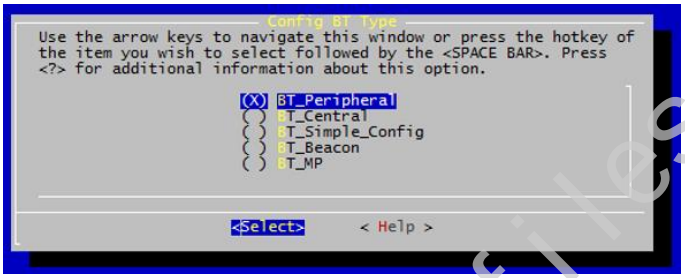
(6) You can use the command **ATBB=1** to restart BT config.

Command	Description	Usage
ATBB	Start or stop BT Config	• Start BT Config: ATBB=1 • Stop BT Config: ATBB=0

31.4.2 BT Peripheral

31.4.2.1 Image Generation

The image generation steps of BT Peripheral are mostly the same as BT Config, the only difference is to select BT_Peripheral in Config BT type.



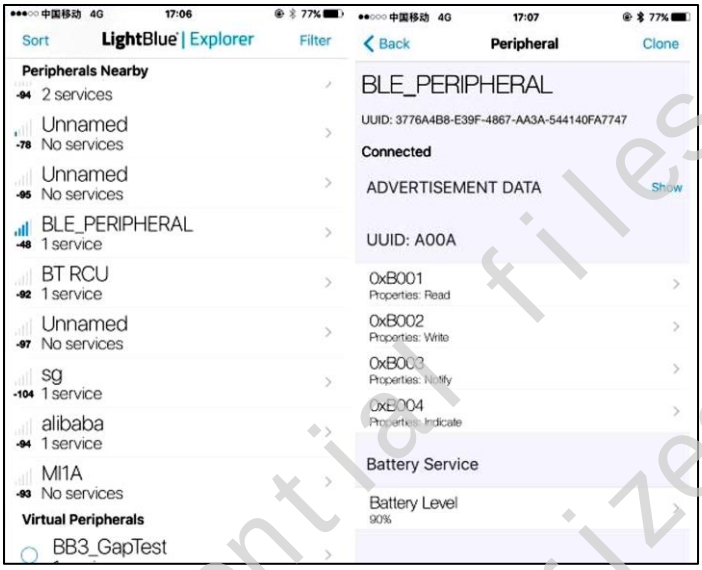
31.4.2.2 Test Procedure

- Test with mobile phone:

(1) After downloading image to your Ameba-D board, reset it. The board is a BLE peripheral device named BLE_PERIPHERAL.
(2) Enable Bluetooth in your mobile phone, scan peripheral devices through mobile phone Bluetooth.
(3) Search for BLE_PERIPHERAL device and connect to it.

- Test with iOS device:

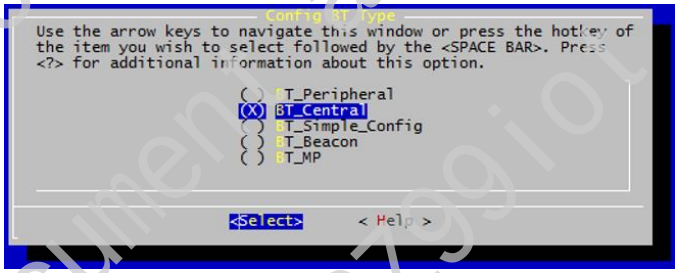
(1) After downloading image to your Ameba-D board, reset it. The board is a BLE peripheral device named BLE_PERIPHERAL.
(2) Download LightBlue in App Store, and run LightBlue on iOS device.
(3) Search for BLE_PERIPHERAL device and connect to it.



31.4.3 BT Central

31.4.3.1 Image Generation

The image generation steps of BT Central are mostly the same as BT Config, the only difference is to select BT_Central in Config BT type.



31.4.3.2 Test Procedure

(1) After downloading image to your Ameba-D board, reset it. The board is a BT central device.
(2) Input command **ATBS=1** in your trace tool to start scanning.
(3) Input command **ATBC=P/R, BLE_BD_ADDR** in your trace tool to start connecting.
(4) Use the following AT command to execute operation such as inquiry, read, write.

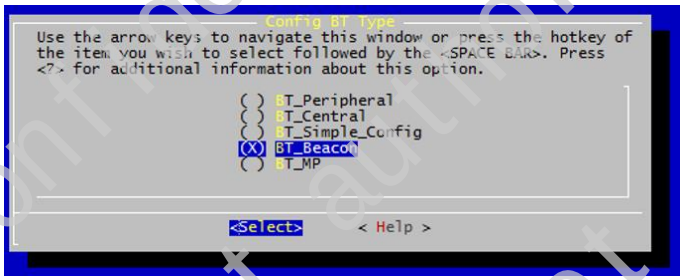
Command	Description	Usage
ATBC	Create a GATT connection	ATBC=P/R, BLE_BD_ADDR **Note**: P=public, R=random
ATBD	Disconnect a GATT connection	ATBD=connect_id
ATBG	Get the peripheral information	● Get all services: ATBG=ALL, connect_id ● Discover services by UUID: ATBG=SRV, connect_id, uuid_type,uuid ● Discover characteristic: ATBG=CHARDIS, connect_id, start_handle, end_handle ● Discover characteristic by UUID: ATBG=CHARUUID, connect_id, start_handle, end_handle, type, UUID ● Discover characteristic descriptor: ATBG=CHARDDIS, connect_id, start_handle, end_handle
ATBI	Get information of the connected device	ATBI

ATBK	Reply GAP passkey	● ATBK=SEND, conn_id ● ATBK=KEY, conn_id, passcode ● ATBK=MODE, auth_flags, io_cap, sec_enable, oob_enable
ATBS	Scan BT	ATBS=scan_enable, filter_policy, filter_duplicate
ATBY	Reply GAP user confirm	ATBY=[conn_id], [conf]
ATBR	GATT client read	● Read characteristic: ATBG=conn_id, handle ● Read characteristic value by UUID: ATBG=conn_id, start_handle, end_handle, uuid_type, uuid
ATBW	GATT client write	ATBW=conn_id, type, handle, length, value
ATBU	Update connection request	ATBU=conn_id, interval_min, interval_max, latency, supervision_timeout
ATBO	Get/clear bond information	● Clear bond information: ATBO=CLEAR ● Get bond information: ATBO=INFO

31.4.4 BT Beacon

31.4.4.1 Image Generation

The image generation steps of BT Beacon are mostly the same as BT Config, the only difference is to select BT_Beacon in Config BT type.



31.4.4.2 Test Procedure

Ameba-D provides two types of Beacon: Apple iBeacon and Radius Networks AltBeacon. Apple iBeacon is the default beacon type in SDK.

(1) After downloading image to your Ameba-D board, reset it. The board starts broadcasting beacons.

```
Initializing WIFI ...[FAST_CONNECT] Fast connect profile is empty, abort fast connection

WIFI initialized

init_thread(54), Available heap 0x284e0

[BT Beacon Example] Apple iBeacon

BT BUILD Date: 2019/05/14-17:05:02
hci_rtk_parse_config: BT ADDRESS  11 28 36 12 51 89, use the defaut config
BT Reset...
bt_iqk_efuse_valid: no data
bt_iqk_logic_efuse_valid: no data
bt_check_iqk:  NO IQK LOK DATA need start LOK,
we need start iqk
hci_tp_rf_radio_ver
continue add  0
hci_tp_rf_radio_ver
continue add  1
hci_tp_rf_radio_ver
continue add  2
hci_tp_rf_radio_ver
bt_iqk_dump:    DUMP,
the IQK_xx  data is 0xfc,
the IQK_yy  data is 0x3f9,
the QDAC  data is 0x20,
the IDAC  data is 0x1f,

IQK OK

hci_tp_config:BT INIT success 7
Start upperStack

GAP stack ready

GAP adv start
```

(2) Use “Locate Beacon” on iOS or “nRF Connect” on Android to scan beacons.

Revision History

Date	Version	Change
2019-06-03	v10	Add the following chapters: ● Key-Scan ● BT Coexistence ● BT Examples
2019-04-10	v09	Add the following chapters: ● GCC standard library ● IAR Build Environment Setup ● Brownout Detector (BOD) ● Flash Translation Layer (FTL) ● Audio Signal Generation and Analysis
2019-02-21	v08	Add Q&A list in GCC and Audio Codec
2019-01-10	v07	● Add IR application note ● Add more detailed description of flash classification in User Configuration ● Add Performance of Encoding and Decoding in Audio Codec ● Update Calibration in Cap-Touch
2018-12-29	v06	Add more information about different IP
2018-11-07	v05	Change format
2018-07-03	v04	Update system architecture
2018-06-28	v02	Change format
2018-06-15	v01	The draft