



AN1000

RTL8721Dx Application Note

This document describes the application note of RTL8721Dx.

Rev. 2.0

Mar., 2024



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

COPYRIGHT

©2023 Realtek Semiconductor Corp. All rights reserved. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form or by any means without the written permission of Realtek Semiconductor Corp.

DISCLAIMER

Please Read Carefully:

Realtek Semiconductor Corp., (Realtek) reserves the right to make corrections, enhancements, improvements and other changes to its products and services. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

Reproduction of significant portions in Realtek data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Realtek is not responsible or liable for such reproduced documentation. Information of third parties may be subject to additional restrictions.

Buyers and others who are developing systems that incorporate Realtek products (collectively, "Customers") understand and agree that Customers remain responsible for using their independent analysis, evaluation and judgment in designing their applications and that Customers have full and exclusive responsibility to assure the safety of Customers' applications and compliance of their applications (and of all Realtek products used in or for Customers' applications) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to their applications, Customer has all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. Customer agrees that prior to using or distributing any applications that include Realtek products, Customer will thoroughly test such applications and the functionality of such Realtek products as used in such applications.

Realtek's provision of technical, application or other design advice, quality characterization, reliability data or other services or information, including, but not limited to, reference designs and materials relating to evaluation kits, (collectively, "Resources") are intended to assist designers who are developing applications that incorporate Realtek products; by downloading, accessing or using Realtek's Resources in any way, Customer (individually or, if Customer is acting on behalf of a company, Customer's company) agrees to use any particular Realtek Resources solely for this purpose and subject to the terms of this Notice.

Realtek's provision of Realtek Resources does not expand or otherwise alter Realtek's applicable published warranties or warranty disclaimers for Realtek's products, and no additional obligations or liabilities arise from Realtek providing such Realtek Resources. Realtek reserves the right to make corrections, enhancements, improvements and other changes to its Realtek Resources. Realtek has not conducted any testing other than that specifically described in the published documentation for a particular Realtek Resource.

Customer is authorized to use, copy and modify any individual Realtek Resource only in connection with the development of applications that include the Realtek product(s) identified in such Realtek Resource. No other license, express or implied, by estoppel or otherwise to any other Realtek intellectual property right, and no license to any technology or intellectual property right of Realtek or any third party is granted herein, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Realtek products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of Realtek Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from Realtek under the patents or other Realtek's intellectual property.

Realtek's Resources are provided "as is" and with all faults. Realtek disclaims all other warranties or representations, express or implied, regarding resources or use thereof, including but not limited to accuracy or completeness, title, any epidemic failure warranty and any implied warranties of merchantability, fitness for a particular purpose, and non-infringement of any third party intellectual property rights.

Realtek shall not be liable for and shall not defend or indemnify Customer against any claim, including but not limited to any infringement claim that related to or is based on any combination of products even if described in Realtek Resources or otherwise. In no event shall Realtek be liable for any actual, direct, special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of Realtek's Resources or use thereof, and regardless of whether Realtek has been advised of the possibility of such damages. Realtek is not responsible for any failure to meet such industry standard requirements.

Where Realtek specifically promotes products as facilitating functional safety or as compliant with industry functional safety standards, such products are intended to help enable customers to design and create their own applications that meet applicable functional safety standards and requirements. Using products in an application does not by itself establish any safety features in the application. Customers must ensure compliance with safety-related requirements and standards applicable to their applications. Designer may not use any Realtek products in life-critical medical equipment unless authorized officers of the parties have executed a special contract specifically governing such use. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death. Such equipment includes, without limitation, all medical devices identified by the U.S.FDA as Class III devices and equivalent classifications outside the U.S.

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

This document is intended for the software engineer's reference and provides detailed programming information.

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

Contents	4
Conventions	11
1 Build Environment	12
1.1 Introduction	12
1.2 Preparing GCC Environment	12
1.2.1 Windows	12
1.2.2 Linux	13
1.2.3 Troubleshooting.....	15
1.3 Installing Toolchain	15
1.3.1 Windows	15
1.3.2 Linux	15
1.4 Configuring SDK	16
1.5 Building Code.....	16
1.5.1 Build One by One.....	17
1.5.2 Build Together	17
1.6 Setting Debugger	18
1.6.1 Windows.....	19
1.6.2 Linux	20
1.7 Downloading Image to Flash	22
1.8 Entering Debug Mode.....	23
1.8.1 GDB Server	23
1.8.2 J-Link.....	24
1.9 Command Lists	24
1.10 GDB Debugger Basic Usage	24
2 SDK Architecture.....	26
2.1 project	26
2.2 component	26
2.3 tools.....	27
2.4 Critical Header Files	27
3 GCC Makefile	28
3.1 Makefile Architecture	28
3.2 How to Build Library	29
3.3 How to Add Library.....	29
4 SDK Examples	30
4.1 Application Example	30
4.2 Peripheral Example.....	31
5 Device Profile Editor	32
5.1 Introduction.....	32
5.2 Environment Setup	32
5.3 Modify a Device Profile.....	32
6 Image Tool	34
6.1 Introduction.....	34
6.2 Environment Setup	34
6.2.1 Hardware Setup.....	34
6.2.2 Software Setup	35
6.3 Image Download.....	35

6.4	Flash Erase	36
6.5	Flash Register Access	36
6.5.1	NOR Flash Register Access	37
6.6	Flash Block Protection Process	38
7	Trace Tool	39
7.1	Introduction	39
7.2	Environment Setup	39
7.2.1	Hardware Connection	39
7.2.2	Software Setup	40
7.3	Usage	40
7.3.1	Log Print	40
7.3.2	Command Send	41
7.3.3	Command Prefix	42
7.3.4	Debug	42
7.3.5	AUTO Script	44
8	Layout Tool	46
8.1	Introduction	46
8.2	Environment Setup	46
8.2.1	Software Setup	46
8.3	Open	46
8.3.1	Select SDK Path	46
8.3.2	Configure Memory	46
8.4	Edit	47
8.4.1	Configure Link	47
8.4.2	Resize Memory Section	47
8.4.3	Delete Section	48
8.5	Preview	48
8.6	Save and Save As	49
8.6.1	Save	49
8.6.2	Save As	49
9	Flash Layout	50
9.1	Introduction	50
9.2	Memory Management Unit (MMU)	51
9.3	How to Modify Flash Layout	51
9.4	Flash Protect Enable	51
10	Memory Layout	53
10.1	RAM Layout	53
10.1.1	SRAM0 (First 64KB) Layout	53
10.1.2	RAM & PSRAM Layout	54
10.2	How to Modify Memory Layout	54
11	MPU	55
11.1	Functional Description	55
11.2	MPU APIs	55
11.2.1	mpu_init	55
11.2.2	mpu_set_mem_attr	55
11.2.3	mpu_region_cfg	56
11.2.4	mpu_entry_free	56
11.2.5	mpu_entry_alloc	56
11.3	Usage	56
12	Cache	57
12.1	Functional Description	57

12.2	Cache APIs.....	57
12.2.1	ICache_Enable	57
12.2.2	ICache_Disable	57
12.2.3	ICache_Invalidate	57
12.2.4	DCache_IsEnabled	57
12.2.5	DCache_Enable.....	58
12.2.6	DCache_Disable.....	58
12.2.7	DCache_Invalidate	58
12.2.8	DCache_Clean.....	58
12.2.9	DCache_CleanInvalidate	58
12.3	How to Define a Non-cacheable Data Buffer	58
12.4	Cache Consistency When Using DMA	58
13	MP Image.....	60
13.1	How to Build MP Image.....	60
13.2	How to Combine Images	60
13.3	Boot Flow	61
13.4	How to Switch Image	62
14	Virtual File System.....	63
14.1	Introduction	63
14.2	VFS Initialization	63
14.3	Usage of VFS.....	64
14.3.1	VFS on Flash.....	64
14.3.2	Common File Operation	64
14.3.3	Key-Value Operation	65
14.3.4	Code Conversion.....	65
14.3.5	VFS Bin File Generation	65
15	OTA Firmware Update	67
15.1	Introduction	67
15.1.1	Image Slot.....	67
15.1.2	Version Number.....	67
15.1.3	Anti-rollback	68
15.2	Bootloader	68
15.2.1	OTA Image.....	68
15.2.2	OTA Select Flow	69
15.3	Application	69
15.3.1	OTA Image.....	69
15.3.2	OTA Select Flow	70
15.4	Build OTA Image	70
15.4.1	Modify Configurations.....	70
15.4.2	Generate OTA Image Automatically.....	71
15.5	Update from Local Server	72
15.5.1	Firmware Format	72
15.5.2	OTA Flow	73
15.6	Run OTA Demo	73
15.7	OTA Firmware Swap	74
15.8	User Configuration	74
16	Boot Process.....	76
16.1	Features	76
16.2	Boot Address.....	76
16.3	Pin Description	76
16.4	Boot Flow	76
16.5	Boot API	77

17	SoC Clock Modification	78
17.1	Introduction	78
17.2	SoC Clock Switch	78
17.2.1	Flow	78
17.2.2	Example	79
17.3	Flash Clock Switch	79
18	Hardware Crypto Engine	80
18.1	Introduction	80
18.1.1	OTP Keys for Crypto	80
18.1.2	Crypto Key Order	81
18.1.3	Crypto Key Flash Procedure	82
18.2	Usage	82
18.2.1	Start Hardware Crypto Engine	82
18.2.2	Use Auto-loaded OTP Key from OTP	82
18.2.3	Start Crypto Engine Calculation	82
18.3	Demo Code	83
19	One-time Programmable Controller (OTPC)	84
19.1	Introduction	84
19.2	OTP Programming APIs	84
19.2.1	OTP_Read8	84
19.2.2	OTP_Write8	85
19.2.3	OTP_LogicalMap_Read	85
19.2.4	OTP_LogicalMap_Write	85
19.2.5	OTP_RemainLength	85
19.2.6	OTPGetCRC	85
19.3	OTP Programming Commands	85
19.3.1	Logical Zone	85
19.3.2	Physical Zone	86
19.4	Usage	87
19.4.1	Logical Zone	87
19.4.2	Security Zone	88
19.4.3	Hidden Physical Zone	92
20	Power Saving	93
20.1	Power-Saving Mode	93
20.1.1	Summary	93
20.1.2	Sleep Mode	94
20.1.3	Deep-Sleep Mode	95
20.2	Wakeup Source	96
20.3	Entering Sleep Mode	96
20.3.1	UART	97
20.3.2	LOGUART	97
20.4	Entering Deep-Sleep Mode	97
20.5	Power-Saving Configuration	97
20.5.1	Wakeup Mask Setup	97
20.5.2	AON Wakepin Configuration	98
20.5.3	Clock and Voltage Configuration	98
20.5.4	Sleep Type Configuration	99
20.6	Power-Saving Related APIs	99
20.6.1	Wakelock APIs	99
20.6.2	pmu_set_sysactive_time	100
20.6.3	Sleep/Wake Callback APIs	101
20.6.4	pmu_set_max_sleep_time	101
20.6.5	Wakeup Reason APIs	102

20.7	Wi-Fi Power Saving	102
21	CHIP Enable	104
21.1	Introduction	104
21.2	Three Working Modes	104
21.2.1	Level Reset Mode	104
21.2.2	Interrupt Reset Mode	104
21.2.3	Pulse Reset Mode	104
21.3	How to Choose Work Mode	105
21.4	CHIP_EN Driver APIs	105
21.4.1	CHIPEN_WorkMode	105
21.4.2	CHIPEN_DebounceSet	105
21.4.3	CHIPEN_ThresHoldSet	105
21.4.4	CHIPEN_AckTimeSet	106
21.4.5	CHIPEN_ClearINT	106
21.4.6	CHIPEN_GetINT	106
22	Inter-Processor Communication (IPC)	107
22.1	Introduction	107
22.2	General Principle	107
22.3	How to Use IPC	107
22.3.1	IPC Usage Procedure	107
22.3.2	Suggested Usage: ipc_get_message()	108
23	Direct Memory Access Controller (DMAC)	111
23.1	Introduction	111
23.2	GDMA Performance	111
23.3	Usage	111
23.3.1	GDMA Configuration	112
23.3.2	Demo for Single Block	117
23.3.3	Demo for Multi-block	117
24	Pseudo-Static Random Access Memory (PSRAM)	119
24.1	Introduction	119
24.2	Throughput	119
24.3	Boot from PSRAM	120
24.4	PSRAM Cache “Write Back” Policy	120
24.4.1	Row Hammer	120
24.4.2	Notice	120
24.5	TCEM Setting	121
24.5.1	Winbond	121
24.5.2	APM	121
25	General Purpose Input/Output (GPIO)	122
25.1	Output	122
25.2	Input	122
25.3	Interrupt	122
26	Pin Controller (Pinctrl)	123
27	Pin Multiplexing Instructions	124
27.1	Introduction	124
27.2	Trap Pins	124
27.3	Wake Pins	124
27.4	SWD Pins	124
27.5	Function Multiplexing	124
27.5.1	Function ID 0-18	124

27.5.2	Function ID 19-81	125
28	RTC_IO	126
28.1	Introduction	126
28.2	Usage.....	126
29	Infrared Radiation (IR).....	127
29.1	Introduction	127
29.2	IR Sending	127
29.2.1	Tx Polling Mode	127
29.2.2	Special Notes	127
29.3	IR Receiving	127
29.3.1	Rx Interrupt Mode	127
29.3.2	Rx Learning	128
30	Light Emitting Diode Controller (LEDC)	129
30.1	Introduction	129
30.2	LEDC Control Application	129
30.3	Operation Flow.....	129
31	Universal Serial Bus On-The-Go (USB OTG).....	131
31.1	Introduction	131
31.1.1	Features.....	131
31.1.2	Architecture.....	131
31.1.3	Implementation.....	131
31.2	Configuration	131
31.3	USB HAL APIs.....	132
31.3.1	Overview	132
31.3.2	APIs for Core Driver	133
31.4	USB Device APIs	133
31.4.1	Overview	133
31.4.2	Core APIs	133
31.4.3	Class APIs	135
31.5	Design Suggestions.....	138
32	True Random Number Generator (TRNG)	140
32.1	Introduction	140
32.2	Features	140
32.3	Block Diagram	140
32.4	Usage.....	141
33	Analog-to-Digital Converter (ADC).....	142
33.1	Introduction	142
33.2	ADC Calibration	142
33.2.1	Calibration Parameters	142
33.2.2	Calculation.....	143
33.3	Internal R.....	143
33.4	Usage.....	144
33.4.1	Auto Mode.....	144
33.4.2	Software-trigger Mode	144
33.4.3	Timer-trigger Mode	145
33.4.4	Comparator-assist Mode	145
34	Cap-Touch Controller (CTC).....	146
34.1	Usage.....	146
34.1.1	Cap Test Tool	146
34.1.2	CTC Initialization	146

34.1.3	CTC Calibration	147
34.1.4	CTC Wakeup	147
35	Key-Scan	148
35.1	Introduction	148
35.1.1	Keypad	148
35.1.2	Work Mode	148
35.1.3	Stuck Key	149
35.2	Usage	149
35.2.1	Event Trigger Mode	149
35.2.2	Regular Scan Mode	149
35.2.3	Key Stuck	150
36	Audio	151
36.1	Introduction	151
36.1.1	Mixer Overview	151
36.1.2	Passthrough Overview	152
36.1.3	How to Choose Architecture	152
36.2	Terminology	152
36.3	Data Format	153
36.3.1	Framework Format	153
36.3.2	HAL Format	154
36.4	Architecture	155
36.4.1	Playback Architecture	155
36.4.2	Record Architecture	156
36.4.3	Control Architecture	157
36.5	Configurations	157
36.5.1	MenuConfig	157
36.5.2	Framework Configuration	158
36.5.3	HAL Configuration	158
36.6	Interfaces	158
36.6.1	Driver Interfaces	159
36.6.2	HAL Interfaces	161
36.6.3	Framework Interfaces	163
36.7	Audio Hardware Application	166
36.7.1	DMIC-in	166
36.7.2	I2S Data Pin	166
36.7.3	SPORT0 Rx with External I2S0	168
	Revision History	170

Conventions

The following abbreviations apply to indicate the MCUs/platforms of Realtek.

RTL8721Dx	32-bit MCU family including RTL8721DAx/RTL8721DCx/RTL8721DGx series
Real-M300 (KM4)	Arm® Cortex®-M55 instruction set compatible core based on Armv8.1-M mainline architecture, running at a frequency of up to 345MHz.
Real-M200 (KM0)	Arm® Cortex®-M23 instruction set compatible core based on Armv8-M baseline architecture, running at a frequency of up to 115MHz.
AP	Application Processor, designed for user application.
NP	Network Processor, designed for network protocol, provides network and power management services to AP.

1 Build Environment

1.1 Introduction

This chapter illustrates how to build Realtek's SDK under GCC environment. It focuses on both Windows platform and Linux distribution. The build and download procedures are quite similar between Windows and Linux operating systems.

- For Windows, Windows 10 64-bit is used as a platform.
- For Linux server, Ubuntu 16.04 64-bit is used as a platform.

1.2 Preparing GCC Environment

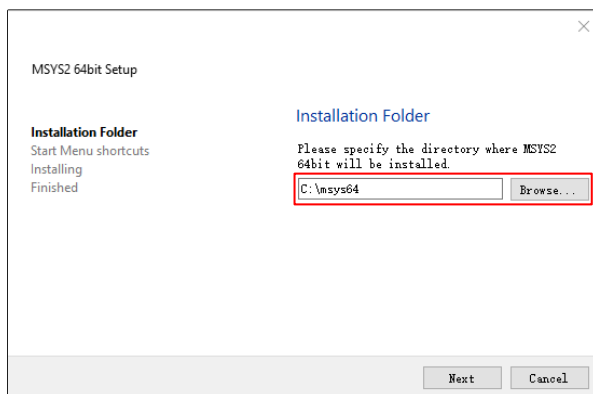
1.2.1 Windows

For Windows, use MSYS2 and MinGW as the GCC environment.

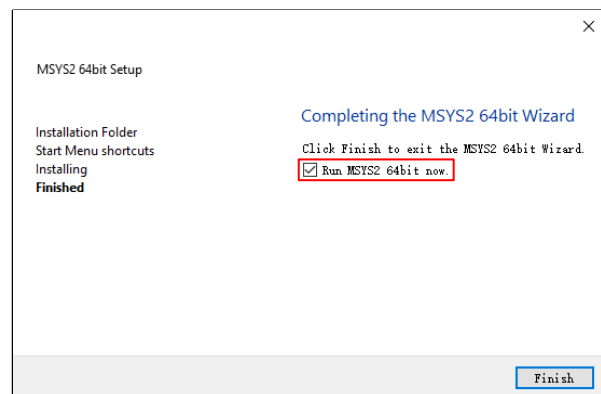
- MSYS2 is a collection of tools and libraries providing an easy-to-use environment for building, installing, and running native Windows software.
- MinGW is an advancement of the original mingw.org project. It is created to support the GCC compiler on Windows system.

The steps to prepare GCC environment are as follows:

- (1) Download MSYS2 from its official website <https://www.msys2.org/>.
- (2) Run the installer. MSYS2 requires 64-bit Windows 7 or newer.
- (3) Enter your desired **Installation Folder** (ASCII, no accents, spaces nor symlinks, short path)
- (4) When done, tick **Run MSYS2 now**.



Step (3)



Step (4)

- (5) Update the package database and base packages with:

```
pacman -Syu
```

When "Proceed with installation? [Y/n]" is displayed, type "Y" and continue until the package installation is done.

```

MSYS ~
$ pacman -Syu
:: Synchronizing package databases...
mingw32             1467.6 KiB   178 KiB/s 00:08 [#####] 100%
mingw64             1474.7 KiB   87.9 KiB/s 00:17 [#####] 100%
ucrt64              1625.5 KiB   186 KiB/s 00:09 [#####] 100%
clang64             1463.4 KiB   83.4 KiB/s 00:18 [#####] 100%
msys                 388.0 KiB   83.0 KiB/s 00:05 [#####] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (5) filesystem-2021.06-2 mintty-1~3.5.1-1 msys2-runtime-3.2.0-15
              pacman-6.0.1-3  pacman-mirrors-20210902-1

Total Download Size:   9.38 MiB
Total Installed Size: 45.57 MiB
Net Upgrade Size:      0.21 MiB

:: Proceed with installation? [Y/n]

```

CAUTION

After installation of MSYS2, there will be four start modes:

- MSYS2 MinGW 32-bit
- MSYS2 MinGW 64-bit
- MSYS2 MinGW UCRT 64-bit
- MSYS2 MSYS

Because the toolchain release will base on 64-bit MinGW, choose **MSYS2 MinGW 64-bit** when starting the MinGW terminal.

- (6) Run **MSYS2 MinGW 64-bit** from **Start** menu. Update the rest of the base packages with:

```
pacman -Syu
```

When "Proceed with installation? [Y/n]" is displayed, type "Y" and continue until the package installation is done.

- (7) Install the necessary software packages with the commands below in order:

```
pacman -S make
pacman -S unzip
pacman -S gcc
pacman -S python
pacman -S ncurses-devel
pacman -S openssl-devel
pacman -S mingw-w64-x86_64-gcc-libs
```

When "Proceed with installation? [Y/n]" is displayed, type "Y" and continue until each software package installation is done.

- (8) Search the needed packages (used to compile TF-M) in <https://packages.msys2.org/package/> and install them as you need with the commands below.

```
pacman -S diffutils
pacman -S vim
pacman -S python-pip
pacman -S cmake
pip install Jinja2
```

- (9) Remove the file path length limit by editing the registry to allow the file paths longer than 260 characters.

- a) Press **Win+R** keys to open the **Run** dialog box, then type "**regedit**" and press **Enter** to open the **Registry Editor**.
- b) Navigate to the registry key: **Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem**.
- c) Search and check if the "LongPathsEnabled" item exists. If not, continue to step **d**); otherwise, go to step **e**).
- d) Right-click on an empty space in the right pane, then select **New > DWORD (32-bit) Value**, and name it "**LongPathsEnabled**".
- e) Double-click on "**LongPathsEnabled**" and set its value to 1, then click **OK** to save.

1.2.2 Linux

On Linux, 32-bit Linux is not supported because of the toolchain.

The packages listed below should be installed for the GCC environment:

- gcc
- lib32ncurses5 (32-bit terminal handling for 64-bit platform. If you are using 32-bit platform, install **libncurses5** instead. If you cannot find this package, such as Ubuntu Server 20.04 LTS, install **lib32ncurses6** or **lib32ncurses5-dev** instead.)
- bash
- make
- libssl-dev
- binutils
- python3

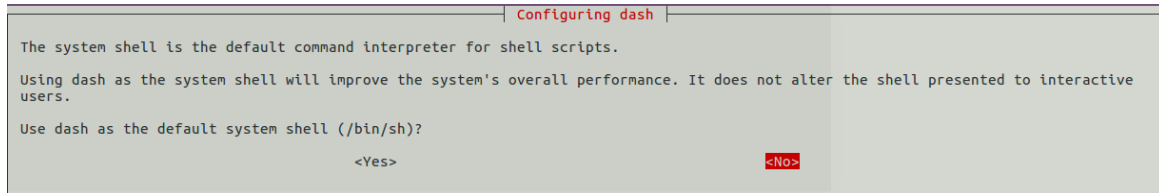
Some of the packages above may have been pre-installed in your operating system. You can either use package manager or type the corresponding version command on terminal to check whether these packages have already existed. If not, make them installed.

- **\$ echo \$SHELL**

Starting from Ubuntu 6.10, dash is used by default instead of bash. You can check by using **\$ echo \$SHELL** command.

```
ububu@ububu-Vostro-3400:~$ echo $SHELL
/bin/bash
```

To switch from dash to bash, you can use **\$sudo dpkg-reconfigure dash** command and choose No.



- **\$ make -v**

```
ububu@ububu-Vostro-3400:~$ make -v
GNU Make 4.1
Built for i686-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.
```

- **\$ sudo apt-get install libssl-dev**

```
ububu@ububu-Vostro-3400:~$ sudo apt-get install libssl-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libssl-doc zlib1g-dev
The following NEW packages will be installed:
  libssl-dev libssl-doc zlib1g-dev
0 upgraded, 3 newly installed, 0 to remove and 291 not upgraded.
Need to get 2,380 kB of archives.
After this operation, 8,929 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 zlib1g-dev i386 1:1.2.8.dfsg-2ubuntu4.3 [166 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 libssl-dev i386 1.0.2g-1ubuntu4.20 [1,137 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 libssl-doc all 1.0.2g-1ubuntu4.20 [1,077 kB]
Fetched 2,380 kB in 16s (142 kB/s)
Selecting previously unselected package zlib1g-dev:i386.
(Reading database ... 206184 files and directories currently installed.)
Preparing to unpack .../zlib1g-dev_1%3a1.2.8.dfsg-2ubuntu4.3_i386.deb ...
Unpacking zlib1g-dev:i386 (1:1.2.8.dfsg-2ubuntu4.3) ...
Selecting previously unselected package libssl-dev:i386.
Preparing to unpack .../libssl-dev_1.0.2g-1ubuntu4.20_i386.deb ...
Unpacking libssl-dev:i386 (1.0.2g-1ubuntu4.20) ...
Selecting previously unselected package libssl-doc.
Preparing to unpack .../libssl-doc_1.0.2g-1ubuntu4.20_all.deb ...
Unpacking libssl-doc (1.0.2g-1ubuntu4.20) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up zlib1g-dev:i386 (1:1.2.8.dfsg-2ubuntu4.3) ...
Setting up libssl-dev:i386 (1.0.2g-1ubuntu4.20) ...
Setting up libssl-doc (1.0.2g-1ubuntu4.20) ...
```

- **binutils**

Use **ld -v** command to check if binutils has been installed. If not, the following error may occur.

```
SE/project_kr4$ make all
make -C v sdk image2
make[1]: Entering directory '/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_ameba
Lite_va0_example/GCC-RELEASE/project_kr4/vsdk'
/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaLite_va0_example/GCC-RELEASE/
project_kr4/vsdk/build exist
/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaLite_va0_example/GCC-RELEASE/
project_kr4/vsdk/image exist
/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaLite_va0_example/GCC-RELEASE/
project_kr4/vsdk/lib/r16842_rom_acut exist
ToolChain Had Existed
make[2]: Entering directory '/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_ameba
Lite_va0_example/GCC-RELEASE/project_kr4/vsdk/make'
make -C bootloader all
make[3]: Entering directory '/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_ameba
Lite_va0_example/GCC-RELEASE/project_kr4/vsdk/make/bootloader'
CC /home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaLite_va0_example/GC
C-RELEASE/project_kr4/vsdk/../../../../component/soc/amebalite/bootloader/bootloader_kr4.c
riscv32-none-elf-gcc: fatal error: cannot execute 'as': execvp: No such file or directory
compilation terminated.
make[3]: *** [/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaLite_va0_exampl
e/GCC-RELEASE/project_kr4/vsdk/Makefile.include.gen:443: bootloader_kr4.o] Error 1
make[3]: Leaving directory '/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaL
ite_va0_example/GCC-RELEASE/project_kr4/vsdk/make/bootloader'
make[2]: *** [Makefile:73: bootloader-all] Error 2
make[2]: Leaving directory '/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaL
ite_va0_example/GCC-RELEASE/project_kr4/vsdk/make'
make[1]: *** [Makefile:95: make_subdirs] Error 2
make[1]: Leaving directory '/home/sandyeyow/Documents/sdk-amebalite_v10.0a-beta/project/realtek_amebaL
ite_va0_example/GCC-RELEASE/project_kr4/vsdk'
make: *** [Makefile:23: all] Error 2
```

1.2.3 Troubleshooting

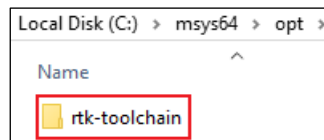
- MSYS2 pacman is responsible for managing and installing software, which is similar to apt-get in ubuntu. When "bash:XXX:command not found" appears, you can try instruction "pacman -S <package_name>" to install.
- For detailed information of one package, try "pacman -Si <package_name>".
- If system head files are not found when building tool, "No such file or directory" error will show up. You can try "pacman -Fy <FILE_NAME>" to check which package is lost, and install the lost package. If too many packages are lost, look for detailed information about the packages to decide which to install.
- For multi-version python host, command "**update-alternatives --install /usr/bin/python python /usr/bin/python3.x 1**" can be used to select python of specific version 3.x, where x represents a desired version number.

1.3 Installing Toolchain

1.3.1 Windows

This section introduces the steps to prepare the toolchain environment.

- (1) Acquire the zip files of RTL8721Dx toolchain from Realtek.
- (2) Create a new directory **rtk-toolchain** under the path **{MSYS2_path}\opt**.
For example, if your MSYS2 installation path is as set in section 1.2.1 step (3), the **rtk-toolchain** should be in **C:\msys64\opt**.



- (3) Unzip **asdk-10.3.x-mingw32-newlib-build-xxxx.zip** and place the toolchain folder **asdk-10.3.x** to the folder **rtk-toolchain** created in step (2).



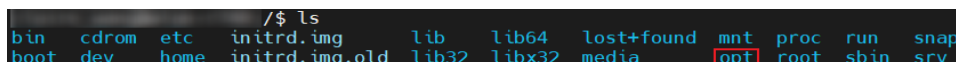
NOTE

- The unzip folders should stay the same with the figure above and do NOT change them, otherwise you need to modify the toolchain directory in makefile to customize the path.
- If an error of the toolchain, just like the log "Error: No Toolchain in /opt/rtk-toolchain/vsdk-10.3.1/mingw32/newlib" appears when building the project, find out if your toolchain files directory are not the same with the directory in the log. Place the toolchain files correctly and try again.

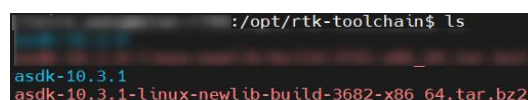
1.3.2 Linux

This section introduces the steps to prepare the toolchain environment.

- (1) Acquire the zip files of RTL8721Dx toolchain from Realtek.
- (2) Create a new directory **rtk-toolchain** under the path **/opt**.



- (3) Unzip **asdk-10.3.x-linux-newlib-build-xxxx.tar.bz2** to **/opt/rtk-toolchain**, then you can get the directory below:



NOTE

The unzip folders should stay the same with the figure above and do NOT change them, otherwise you need to modify the toolchain directory in makefile to customize the path.

1.4 Configuring SDK

This section illustrates how to change SDK configurations.

User can configure SDK options for KM0 and KM4 at the same time through **\$ make menuconfig** command.

- (1) Switch to the directory **{SDK}\amebadplus_gcc_project**
- (2) Run **\$ make menuconfig** command on MSYS2 MinGW 64-bit (Windows) or terminal (Linux)

NOTE

\$ make menuconfig command is only supported under {SDK}\amebadplus_gcc_project, but not supported under other paths.

The main configurable options are divided into four parts:

- General Config: the shared kernel configurations for KM4 and KM0. The configurations will take effect in both KM4 and KM0.
- Network Config: the shared kernel configurations for KM4 and KM0. The configurations will take effect in both KM4 and KM0.
- KM4 Config: the exclusive kernel configurations for KM4. The configurations will take effect only in KM4 but not in KM0.
- KM0 Config: the exclusive kernel configurations for KM0. The configurations will take effect only in KM0 but not in KM4.

Figure 1-1 is the menuconfig UI, and the options in red may be used frequently.

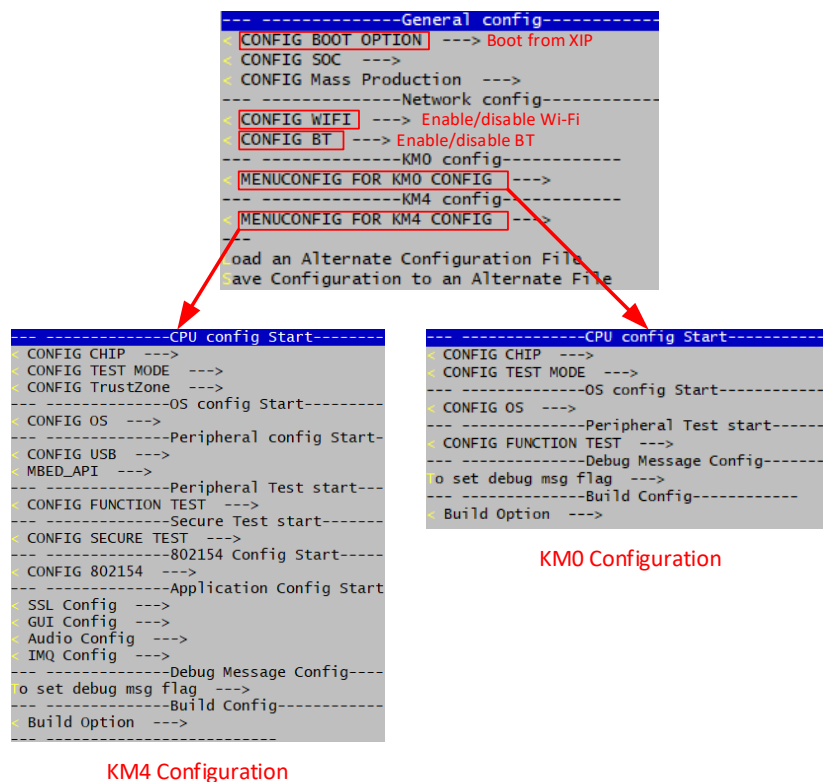


Figure 1-1 menuconfig UI

1.5 Building Code

This section illustrates how to build SDK for both Windows and Linux. Table 1-1 lists all the GCC project directories of SDK.

Table 1-1 GCC project directory

GCC project	Directory
KM4	{SDK}\amebadplus_gcc_project\project_km4
KM0	{SDK}\amebadplus_gcc_project\project_km0

NOTE

Replace the {SDK} with your own SDK directory.

There are two ways to build the SDK, you can choose either of them.

1.5.1 Build One by One

Follow these steps to build the SDK of KM4 and KM0 project one by one:

- (1) Use **\$ cd** command to switch to the project directories of SDK on Windows or Linux.
For example, you can type **\$ cd {SDK}\amebadplus_gcc_project\project_km4** to switch to the KM4 project, the same operation for the KM0 project.
- (2) Build SDK under the KM0 or KM4 project directory on Windows or Linux.
 - For normal image, simply use **\$ make all** command to build SDK.
 - For MP image, refer to Section [How to Build MP Image](#) to build SDK.
- (3) Check the command execution results. If somehow failed, type **\$ make clean** to clean and then redo the make procedure.
 - For KM4 project, if the terminal contains “target_img2.axf” and “Image manipulating end” message (see [Figure 1-2](#)), it means that KM4 images have been built successfully. You can find them under **\amebadplus_gcc_project\project_km4\asdk\image**, as shown in [Figure 1-3](#).
 - For KM0 project, if the terminal contains “target_img2.axf” and “Image manipulating end” message (see [Figure 1-4](#)), it means that KM0 image has been built successfully. You can find it under **\amebadplus_gcc_project\project_km0\asdk\image**, as shown in [Figure 1-5](#).

```

text      data      bss      dec      hex      filename
367976    32      55328    423336    675a8      /amebadplus_gcc_project/project_km4/asdk/image//target_img2.axf
367976    32      55328    423336    675a8      (TOTALS)
=====
Image Info DEC =====
=====
linker img2 ns end =====
=====
Image manipulating start =====
=====
Image manipulating end =====
=====
Image analyze start =====
=====
Image analyze end =====
make[1]: Leaving directory ' /amebadplus_gcc_project/project_km4/asdk'

```

Figure 1-2 KM4 project make all

amebadplus_gcc_project > project_km4 > asdk > image	
Name	Date modified
☐ km0_image2_all.bin	2/29/2024 10:55
☐ km0_image2_all_en.bin	2/29/2024 10:55
☒ km0_km4_app.bin	2/29/2024 10:55
☐ km0_km4_app_ns.bin	2/29/2024 10:55
☐ km0_km4_app_tmp.bin	2/29/2024 10:55
☐ km4_boot.bin	2/29/2024 10:40
☐ km4_boot_all.bin	2/29/2024 10:40

Figure 1-3 KM4 image generation

```

text      data      bss      dec      hex      filename
372872    64      36257    409193    63e69      /amebadplus_gcc_project/project_km0/asdk/image//target_img2.axf
372872    64      36257    409193    63e69      (TOTALS)
=====
Image Info DEC =====
=====
linker img2 end =====
=====
Image manipulating start =====
major: 1
=====
Image manipulating end =====
=====
Image analyze start =====
=====
Image analyze end =====
make[1]: Leaving directory ' /amebadplus_gcc_project/project_km0/asdk'

```

Figure 1-4 KM0 project make all

amebadplus_gcc_project > project_km0 > asdk > image	
Name	Date modified
☐ .gitignore	2/28/2024 5:12 PM
☐ code_size_image.map	2/29/2024 10:55 AM
☐ code_size_psram.map	2/29/2024 10:55 AM
☐ code_size_sram.map	2/29/2024 10:55 AM
☐ code_size_xip.map	2/29/2024 10:55 AM
☐ entry.bin	2/29/2024 10:55 AM
☐ entry_prepend.bin	2/29/2024 10:55 AM
☒ km0_image2_all.bin	2/29/2024 10:55 AM
☐ obj_list.txt	2/29/2024 10:55 AM

Figure 1-5 KM0 image generation

1.5.2 Build Together

In order to improve the efficiency of building SDK, you can also execute **\$ make all** command once under **\amebadplus_gcc_project**, instead of executing **\$ make all** command separately under the KM0 project and KM4 project.

- If the terminal contains “target_img2.axf” and “Image manipulating end” message (see [Figure 1-6](#)), it means that all the images have been built successfully. The image files are generated under \amebadplus_gcc_project, as shown in [Figure 1-7](#). You can also find them under \amebadplus_gcc_project\project_km0\asdk\image and \amebadplus_gcc_project\project_km4\asdk\image.
- If somehow failed, type \$ make clean to clean and then redo the make procedure.

```

text    data    bss    dec    hex filename
367976   32    55328  423336  675a8 /amebadplus_gcc_project/project_km4/asdk/image//target_img2.axf
367976   32    55328  423336  675a8 (TOTALS)
===== Image Info DEC =====
===== linker img2_ns end =====
===== Image manipulating start =====
major: 1
size = 759648
checksum 4abf092
===== Image manipulating end =====
===== Image analyze start =====
===== Image analyze end =====
make[2]: Leaving directory ' /amebadplus_gcc_project/project_km4/asdk'
make[1]: Leaving directory ' /amebadplus_gcc_project/project_km4'

```

Figure 1-6 KM4 & KM0 projects make all

amebadplus_gcc_project	
Name	Date modified
menuconfig	2/28/2024 5:12 PM
project_km0	2/28/2024 5:12 PM
project_km4	2/28/2024 5:12 PM
utils	2/28/2024 5:12 PM
.gitignore	2/28/2024 5:12 PM
amebaDplus_layout.ld	2/28/2024 5:12 PM
km0_km4_app.bin	2/29/2024 9:42 AM
km4_boot_all.bin	2/29/2024 9:42 AM
Makefile	2/28/2024 5:12 PM
manifest.json	2/28/2024 5:12 PM
OTA_All.bin	2/29/2024 9:42 AM

Figure 1-7 KM4 & KM0 image generation

NOTE

If you want to search some .map files for debugging, get them under the directory {SDK}\amebadplus_gcc_project\project_km0\asdk\image or {SDK}\amebadplus_gcc_project\project_km4\asdk\image, but not {SDK}\amebadplus_gcc_project.

1.6 Setting Debugger

The RTL8721Dx supports J-Link debugger. Before setting J-Link debugger, you need to do some hardware configuration and download images to the RTL8721Dx device first.

- (1) Connect J-Link to the SWD of RTL8721Dx.
 - a) Refer to [Figure 1-8](#) to connect SWCLK pin of J-Link to SWD CLK pin of RTL8721Dx, and SWDIO pin of J-Link to SWD DATA pin of RTL8721Dx.
 - b) Connect the RTL8721Dx device to PC after finishing these configurations.

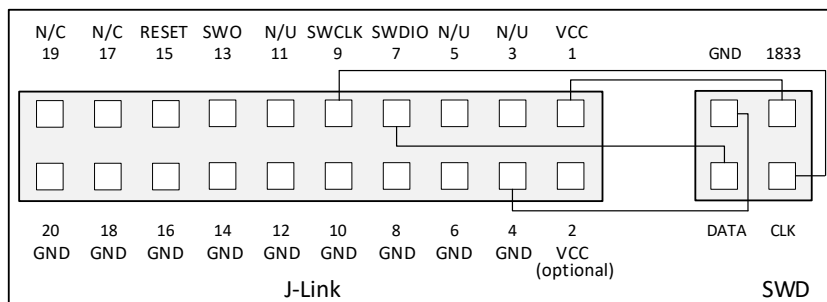


Figure 1-8 Wiring diagram of connecting J-Link to SWD

NOTE

For RTL8721Dx, the J-Link version must be v9 or higher. If Virtual Machine (VM) is used as your platform, make sure that the USB connection setting between VM host and client is correct, so that the VM host can detect the device.

- (2) Download images to the RTL8721Dx device via ImageTool.

ImageTool is a software tool provided by Realtek. For more information, refer to [Image Tool](#).

1.6.1 Windows

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.

i NOTE

The version of J-Link GDB server below is just an example, you can select the latest version to download.

1.6.1.1 KM4 Setup

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

Double-click the **cm4_jlink.bat** under **{SDK}\amebadplus_gcc_project\utils\jlink_script**. You may have to change the path of JLinkGDBServer.exe and JLink.exe in the **cm4_jlink.bat** script according to your own settings.

The started J-Link GDB server looks like [Figure 1-9](#). This window should NOT be closed if you want to download the image or enter debug mode.

! CAUTION

Keep this window active to download the images to target.

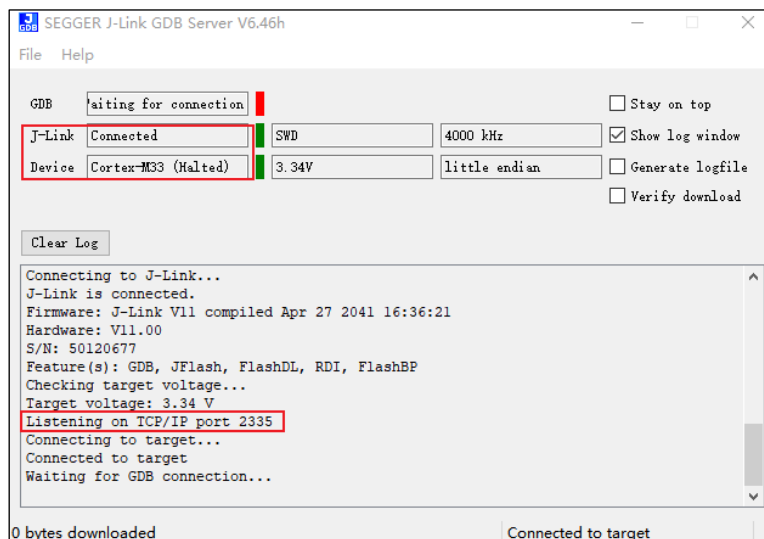


Figure 1-9 KM4 J-Link GDB server connection under Windows

(2) Setup J-Link for KM4

- Change the working directory to project_km4.
- On the MSYS2 terminal, type **\$ make setup GDB_SERVER=jlink** command before selecting J-Link debugger, as [Figure 1-10](#) shows.

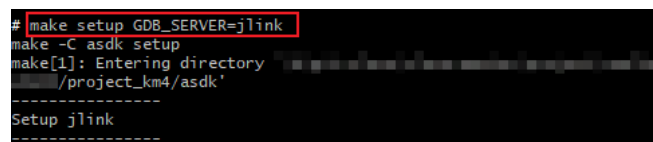


Figure 1-10 KM4 J-Link setup under Windows

1.6.1.2 KM0 Setup

(1) Execute the **cm0_jlink.bat**

Double-click the **cm0_jlink.bat** under **{SDK}\amebadplus_gcc_project\utils\jlink_script**, the same as executing the **cm4_jlink.bat**.

The started J-Link GDB server looks like [Figure 1-11](#). This window should NOT be closed if you want to download the image or enter debug mode. Because KM4 will download all the images, you don't need to connect J-Link to KM0 when downloading images. J-Link can connect to KM0 when debugging.

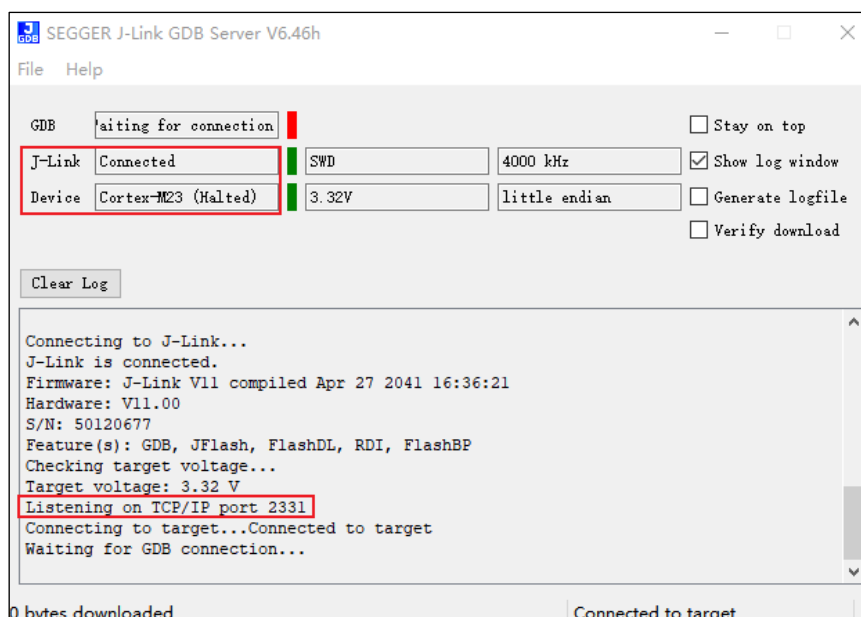


Figure 1-11 KM0 J-Link GDB server connection under Windows

- (2) Setup J-Link for KMO
 - a) Change working directory to project_km0.
 - b) On the Cygwin terminal, type **\$ make setup GDB_SERVER=jlink** command to select J-Link debugger.

```
# make setup GDB_SERVER=jlink
make -C asdk setup
make[1]: Entering directory '/home/jlink_u/Save/Save_server/project/realtime/ASDK/project_km0/asdk'
-----
Setup jlink
-----
```

Figure 1-12 KM0 J-Link setup under Windows

1.6.2 Linux

For J-Link GDB server, click <https://www.segger.com/downloads/jlink> and download the software in “J-Link Software and Documentation Pack”. It is suggested to use Debian package manager to install the Debian version.

Open a new terminal and type the following command to install GDB server. After the installation of the software pack, there is a tool named "JLinkGDBServer" under the J-Link directory. Take Ubuntu 18.04 as an example, the JLinkGDBServer can be found at **/opt/SEGGER/JLink**.

```
$ dpkg -i jlink 6.0.7 x86 64.deb
```

NOTE

The version of J-Link GDB server below is just an example, you can select the latest version to download.

1.6.2.1 KM4 Setup

- (1) Connect to KM4
 - a) Open a new terminal under directory `/amebadplus_gcc_project/utils/jlink_script`.
 - b) Type `$ /opt/SEGGER/JLink/JLinkGDBServer -select USB -device Cortex-M33 -if SWD -scriptfile AP2_KM4.JLinkScript port 2335`.

```

$ ./jlink_script /opt/SEGGER/JLink/JLinkGDBServer -select USB -device Cortex-M33 -if swd -scriptfile AP2_KM4.JLinkScript -port 2335
SEGGER J-Link GDB Server V7.60 Command Line Version

JLinkARM.dll V7.60 (DLL compiled Dec 14 2021 11:40:00)

WARNING: Unknown command line parameter USB found.
Command line: -select USB -device Cortex-M33 -if swd -scriptfile AP2_KM4.JLinkScript -port 2335
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2335
SWO raw output listening port: 2332
Terminal I/O port: 2333
Accept remote connection: yes
Generate logfile: off
Verify download: off
Init regs on start: off
Silent mode: off
Single run mode: off
Target connection timeout: 0 ms

```

Figure 1-13 KM4 J-Link GDB server connection setting under Linux

If the connection is successful, the log is shown as [Figure 1-14](#). This terminal should NOT be closed if you want to download software or enter GDB debugger mode.

```

-----J-Link related settings-----
J-Link Host interface: USB
J-Link script: AP2_KM4.JLinkScript
J-Link settings file: none
-----Target related settings-----
Target device: Cortex-M33
Target interface: SWD
Target interface speed: 4000kHz
Target endian: little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V11 compiled May 23 2023 14:44:38
Hardware: V11.00
S/N: 601015439
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.35 V
Listening on TCP/IP port 2335
Connecting to target...
Connected to target
Waiting for GDB connection...

```

Figure 1-14 KM4 J-Link GDB server connection success under Linux

- (2) Setup J-Link for KM4
 - a) Open a new terminal under project_km4 folder.
 - b) Type **\$ make setup GDB_SERVER=jlink** command before using J-Link to download software or enter GDB debugger.

```

$ cd /project_km4
$ make setup GDB_SERVER=jlink
make -C asdk setup
make[1]: Entering directory '/project_km4/asdk'
-----
Setup jlink
-----

```

Figure 1-15 KM4 J-Link terminal setup under Linux

1.6.2.2 KM0 Setup

- (1) Connect to KM0
 - a) Open a new terminal under directory /amebadplus_gcc_project/utis/jlink_script.
 - b) Type **\$ /opt/SEGGER/JLink/JLinkGDBServer -select USB -device Cortex-M23 -if SWD -scriptfile AP1_KM0.JLinkScript port 2331**.

```

#### jlink_script$ /opt/SEGGER/JLink/JLinkGDBServer -select USB -device Cortex-
M23 -if swd -scriptfile AP1_KM0.JLinkScript -port 2331
SEGGER J-Link GDB Server V7.60 Command Line Version

JLinkARM.dll V7.60 (DLL compiled Dec 14 2021 11:40:00)

Command line: -select USB -device Cortex-M23 -if swd -scriptfile AP1_KM0.JLinkS
cript -port 2331
-----GDB Server start settings-----
GDBInit file:          none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port:     2333
Accept remote connection: yes
Generate logfile:       off
Verify download:        off
Init regs on start:     off
Silent mode:            off
Single run mode:        off
Target connection timeout: 0 ms

```

Figure 1-16 KM0 J-Link connection setting under Linux

If the connection is successful, the log is shown below.

```

-----J-Link related settings-----
J-Link Host interface:  USB
J-Link script:          AP1_KM0.JLinkScript
J-Link settings file:   none
-----Target related settings-----
Target device:          Cortex-M23
Target interface:       SWD
Target interface speed: 4000kHz
Target endian:          little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V11 compiled May 23 2023 14:44:38
Hardware: V11.00
S/N: 601015439
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.35 V
Listening on TCP/IP port 2331
Connecting to target...
Connected to target
Waiting for GDB connection...

```

Figure 1-17 KM0 J-Link GDB server connection success under Linux

- (2) Setup J-Link for KM0
 - a) Open a new terminal under project_km0.
 - b) Type **\$ make setup GDB_SERVER=jlink** command before using J-Link to download software or enter GDB debugger.

```

####/project_km0$ make setup GDB_SERVER=jlink
make -C asdk setup
make[1]: Entering directory '
.../project_km0/asdk'
-----
Setup jlink
-----

```

Figure 1-18 KM0 J-Link terminal setup under Linux

1.7 Downloading Image to Flash

There are two ways to download image to Flash:

- (1) Image Tool, a software provided by Realtek (recommended). For more information, refer to [Image Tool](#).
- (2) GDB Server, mainly used for GDB debug user case.

This section illustrates the second method to download images to Flash.

To download software into Device Board, make sure the steps mentioned in Section 1.5 are done, and then type **\$ make flash** command on MSYS2 (Windows) or terminal (Linux).

Images are downloaded only under KM4 by this command. This command downloads the software into Flash and it will take several seconds to finish, as shown in [Figure 1-19](#).

```

TailSize = 0
global variables
FlashDatSrc:3000379c
FlashBlockWriteSize:800
FlashAddrForWrite:800000
Flash write start...
Flash_write FileName:a
Flash_write FileSize:0
Flash_write FlashAddrForWrite:c00000
FileSize: 0
Loopnumber = 0
TailSize = 0
global variables
FlashDatSrc:3000379c
FlashBlockWriteSize:800
FlashAddrForWrite:c00000
Flash write start...

Breakpoint 2, RtlFlashProgram () at /project_hp/asdk/flashloader/rtl_flash_download.c:131
131
dump for check
make[1]: Leaving directory '/project_hp/asdk'
FlashWriteComplete = 0;

```

```

ROM:[V1.0]
FLASHRATE:1
BOOT FROM NOR
=====
Flash Downloader Build Time: 2021/09/30-10:20:35
=====
Flash init start
Flash init done
Flash download start
Flash download done

```

Figure 1-19 Download codes success log

To check whether the image is downloaded correctly into memory, you can select "verify download" before downloading images, and during image download process, "verified OK" log will be shown.

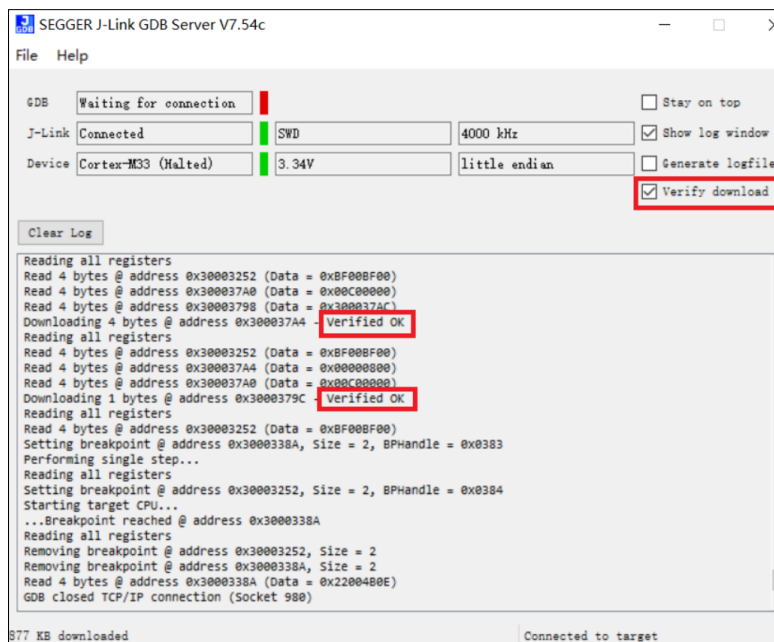


Figure 1-20 Verify download

After download is successful, press **Reset** button and you will see that the device boots with the new image.

NOTE

The command is only supported to use in KM4 project, and km4_boot_all.bin & KM0_km4_app.bin can be downloaded to Flash.

1.8 Entering Debug Mode

1.8.1 GDB Server

To enter GDB debugger mode, follow the steps below:

- (1) Make sure that the steps mentioned in Sections 1.4 to 1.6 are finished, then reset the device.
- (2) Change directory to target project which can be project_km4 or project_km0, and type **\$ make debug** command on MSYS2 (Windows) or terminal (Linux).

1.8.2 J-Link

1.8.2.1 Steps

- (1) Press **Win+R** on your keyboard. Hold down the Windows key on your keyboard, and press the "R" button. This will open the "Run" tool in a new pop-up window. Alternatively, you can find and click **Run** on the Start menu.
- (2) Type **cmd** in the Run window. This shortcut will open the Command Prompt terminal.
- (3) Click **OK** in the Run window. This will run your shortcut command, and open the Command Prompt terminal in a new window.
- (4) Copy the J-Link script command below for specific target:

For KM4:

```
"{Jlink_path}\JLink.exe" -device Cortex-M33 -if SWD -speed 4000 -autoconnect 1
```

For KM0:

```
"{Jlink_path}\JLink.exe" -device Cortex-M23 -if SWD -speed 4000 -autoconnect 1
```

NOTE

The J-Link connection command path mentioned above are:

- **{Jlink_path}**: the path your Segger J-Link installed, the default is "C:\Program Files (x86)\SEGGER\JLink".
- **{script path}**: {SDK}\amebadplus_gcc_project\utils\jlink_script.

1.8.2.2 Commands

The following commands are often used when the program is stuck. All commands are accepted case insensitive.

Command (long)	Command (short)	Syntax	Explanation
Halt	H		Halt CPU
Go	G		Start CPU if halted
Mem		Mem <Addr> <NumBytes>	Read memory and show corresponding ASCII values
SaveBin		SaveBin <FileName> <Addr> <NumBytes>	Save target memory range into binary file
Exit			Close J-Link connection and quit

For more information, you can visit https://wiki.segger.com/J-Link_Commander.

NOTE

- You can type "H" and "G" several times and record the PC, then look for the PC in which function in asm file. This function might be where you get stuck.
- You can also use "mem" to dump some address after "sp", from these addresses you can find the function call stack.

1.9 Command Lists

The commands mentioned above are listed in [Table 1-2](#).

Table 1-2 Command lists

Usage	Command	Description
all	\$ make all	Compile the project to generate ram_all.bin
setup	\$ make setup GDB_SERVER= jlink	Select GDB_SERVER
flash	\$ make flash	Download ram_all.bin to Flash
clean	\$ make clean	Remove compile file (*.bin, *.o, ...)
debug	\$ make debug	Enter debug mode

1.10 GDB Debugger Basic Usage

GDB, the GNU project debugger, allows you to examine the program while it executes, and it helps catch bugs. Section 1.8 has described how to enter GDB debugger mode, this section illustrates some basic usage of GDB commands. For further information about GDB debugger, click <https://www.gnu.org/software/gdb/>. [Table 1-3](#) describes commonly used instructions and their functions, and specific usage can be found in **GDB User Manual** of website <https://www.sourceware.org/gdb/documentation/>.

Table 1-3 GDB debugger command list

Usage	Command	Description
Breakpoint	\$ break	Breakpoints are set with the break command (abbreviated b). The usage can be found at Setting Breakpoints section.
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. The usage of these commands can be found at Setting Watchpoints section. NOTE <i>Keep the range of watchpoints less than 20 bytes.</i>
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 Deleting Breakpoints section.
Continue	\$ continue	To resume program execution, use the continue command (abbreviated c). The usage can be found at Continue and Stepping section.
Step	\$ step	To step into a function call, use the step command (abbreviated s). It will continue running your program until the control reaches a different source line. The usage can be found at Continue and Stepping section.
Next	\$ next	To step through the program, use the next command (abbreviated n). The execution will stop when the control reaches a different line of code at the original stack level. The usage can be found at Continue and Stepping section.
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 Quitting GDB section.
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 a Backtraces section.
Print source lines	\$ list	To print lines from a source file, use the list command (abbreviated l). The usage can be found at Printing Source Lines section.
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 Examining Data section.

2 SDK Architecture

The SDK consists of four parts, as shown in [Figure 2-1](#). The contents and descriptions of each part are summarized in the following sections.

- project
- component
- tools
- doc



Figure 2-1 SDK architecture

2.1 project

Item	Description	
amebadplus_gcc_project	Makefile	Compile the project with one command
	menuconfig	Used to configure the project
	project_km4	KM4 project files
	project_km0	KM0 project files
	utils	J-Link script for connecting KM4 and KM0
		● src/main.c ● inc: ■ FreeRTOSConfig.h: configure FreeRTOS ■ main.h ■ platform_autoconf.h: generated by “make menuconfig” ■ platform_opts_bt.h: some utilities for BT ● Makefile ● Link script ● Bin files ● Library ● ...

2.2 component

Items	Description
at_cmd	AT commands
bluetooth	BT related source code and library
example	Utility examples: audio/network/ota/peripheral example/...
file_system	File system: fatfs/littlefs/ftl/kv/vfs...
lwip	LWIP APIs and driver codes
network	● cJSON ● coap ● dhcp ● http2 ● httpc ● httpd ● mDNS ● mqtt ● ping ● iperf ● snmp ● websocket ● xml
os	FreeRTOS source codes
os_dep	● osdep_service.h: Realtek encapsulating interface header

	osdep_service.c: Realtek encapsulating interface for FreeRTOS
soc	- app: monitor and shell - bootloader - cmsis: Arm headers, including Arm CPU registers and operations - cmsis-dsp: Arm CMSIS-DSP source codes - fwlib: user configuration, low level drivers, such as Audio Codec/SPORTS/UART/I2C/SPI/Timer/PWM... - hal: mbed APIs source codes, header files - img3: files for image3 - misc: misc file like crashdump/ota/pmu... - swlib: standard software library supported by ROM code, such as _memcpy/_memcmp...
ssl	mbed TLS
utils	IPC: util for multicore communication
wifi	- Wi-Fi driver interface - Wi-Fi promisc mode interface - Wi-Fi fast connection

2.3 tools

Items	Description
TraceTool	Tools used to print logs and send commands
ImageTool	Image tool
DownloadServer	Used to send image to the device based on socket by OTA function
DownloadServer (HTTP)	Used to send image to the device based on HTTP by OTA function
iperf	iperf for Wi-Fi performance test
littlefs	Tools to make littlefs file system

2.4 Critical Header Files

Items	Description	Location
basic_types.h	- SUCCESS/FAIL - TRUE/FALSE - ENABLE/DISABLE - ON/OFF - NULL - u8/u16/u32/u64 - BOOL - BITx ...	{SDK}\component\os_dep
section_config.h	Section definition used in link script: - BOOT_RAM_DATA_SECTION - IMAGE2_RAM_TEXT_SECTION ...	{SDK}\component\soc\amebadplus\fwlib\include
mbed API headers	Peripheral header files for mbed APIs. If you want to use mbed APIs, related headers must be included.	{SDK}\component\soc\amebadplus\hal
ameba_soc.h	Peripheral header files for raw APIs Raw APIs have more features than mbed APIs, which just have basic features. If you want to use raw APIs, this header must be included.	{SDK}\component\soc\amebadplus\fwlib\include

3 GCC Makefile

3.1 Makefile Architecture

Figure 3-1 and Figure 3-2 summary the makefile architectures of each project.

```
project_km0\Makefile-->
  project_km0\asdk\Makefile-->
    project_km0\asdk\make\Makefile-->
      project_km0\asdk\make\api\Makefile
      project_km0\asdk\make\bootloader\Makefile
      project_km0\asdk\make\cmsis\Makefile
      project_km0\asdk\make\mbedtls\Makefile
      project_km0\asdk\make\monitor\Makefile
      project_km0\asdk\make\network\Makefile
      project_km0\asdk\make\os\Makefile
      project_km0\asdk\make\project\Makefile
      project_km0\asdk\make\target\Makefile
      project_km0\asdk\make\utilities\Makefile
      project_km0\asdk\make\wlan\Makefile
      project_km0\asdk\make\wlan_encrypt\Makefile
      project_km0\asdk\make\wps\Makefile
```

Figure 3-1 KM0 makefile architecture

```
project_km4\Makefile-->
  project_km4\asdk\Makefile-->
    project_km4\asdk\make\Makefile-->
      project_km4\asdk\make\api\Makefile
      project_km4\asdk\make\app\Makefile
      project_km4\asdk\make\application\Makefile
      project_km4\asdk\make\audio\Makefile
      project_km4\asdk\make\bootloader\Makefile
      project_km4\asdk\make\cmsis\Makefile
      project_km4\asdk\make\project\Makefile→
        project_km4\asdk\make\project\sram\Makefile: how to build code into RAM
        project_km4\asdk\make\project\xip\Makefile: how to build code into Flash
        project_km4\asdk\make\project\library\Makefile: how to build library
      project_km4\asdk\make\example\Makefile
      project_km4\asdk\make\file_system\Makefile
      project_km4\asdk\make\flashloader\Makefile
      project_km4\asdk\make\mbed\Makefile
      project_km4\asdk\make\mbedtls\Makefile
      project_km4\asdk\make\monitor\Makefile
      project_km4\asdk\make\network\Makefile
      project_km4\asdk\make\os\Makefile
      project_km4\asdk\make\RT_xmodem\Makefile
      project_km4\asdk\make\rtl_bluetooth\Makefile
      project_km4\asdk\make\target\Makefile
      project_km4\asdk\make\ui\Makefile
      project_km4\asdk\make\utilities\Makefile
      project_km4\asdk\make\utilities_example\Makefile
      project_km4\asdk\make\wlan\Makefile
      project_km4\asdk\make\wlan_encrypt\Makefile
      project_km4\asdk\make\wps\Makefile
```

Figure 3-2 KM4 makefile architecture

3.2 How to Build Library

The Makefile in {SDK}\amebadplus_gcc_project\project_km4\asdk\make\project\library is an example to show how to build user library. As shown below, lib_user.a will be generated in {SDK}\amebadplus_gcc_project\project_km4\asdk\lib\application.

```
include $(MAKE_INCLUDE_GEN)

.PHONY: all clean

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

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

TARGET_LIB = lib_user

#-----
#                               Source FILE LIST
#-----
#add your source code here
CSRC += \
    $(CUSTOMER_DIR)/lib_user_test.c\
    $(TARGET_LIB)_git_version.c

#-----
#                               Object FILE LIST
#-----
OBJS = $(notdir $(CSRC:.c=.o))

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

#-----
#                               RULES TO GENERATE TARGETS
#-----
$(CSRC): GEN_GIT_VERSION_FILE
# Define the Rules to build the core targets
COPY_RAM_OBJS: CORE_TARGETS
all: COPY_RAM_OBJS
    $(AR) rvs lib_user.a $(OBJS) $(OBJS_SRAM)
    $(MOVE) -f lib_user.a $(ROOTDIR)/lib/application
    $(REMOVE) $(TARGET_LIB)_git_version.*
```

3.3 How to Add Library

Open the file {SDK}\amebadplus_gcc_project\project_km4\asdk\Makefile, and append lib_user.a to LINK_APP_LIB.

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

4 SDK Examples

There are two kinds of examples in the SDK of RTL8721Dx: application examples and peripheral examples. This chapter illustrates the contents of examples and how to build example source code. The path and description of SDK examples are listed in [Table 4-1](#).

Table 4-1 Examples in SDK

Items	Path	Description
Application example	{SDK}\component\example	xml, ssl, http, ...
Peripheral example	{SDK}\component\example\peripheral	ADC, UART, I2C, SPI, Timer, ...

4.1 Application Example

In each folder of application example, there are C source files, header files and README.txt. You should check for detailed configurations of the example according to README.txt.

i NOTE

The application examples are shared by all Realtek SoC, so you need to refer to README.txt for detailed information of RTL8721Dx.

The application examples normally run on KM4. The entry function of application example is `app_example()` in `main.c` under `{SDK}\amebadplus_gcc_project\project_km4\src`. Each application example has its own `app_example()`, and the `app_example()` in `main.c` will be replaced automatically when the application example is built.

```
// default main
int main(void)
{
    .....

    app_example();

    .....

    /* Enable Schedule, Start Kernel */
    vTaskStartScheduler();
} < end main >
```

Figure 4-1 Entry function of application example

To run application example, you only need to:

- (1) Check software and hardware settings in README.txt of the example.
- (2) Add compile options "**EXAMPLE={example folder name}**" when building the project, and replace **{example folder name}** with the specific folder name of this example.

For example, if you want to build xml example to start an xml example thread, you need to set the macro in SDK according to README.txt in `{SDK}\component\example\xml`. Then enter "**make EXAMPLE=xml**" for KM4 on MSYS2 MinGW 64-bit (Windows) or terminal (Linux).

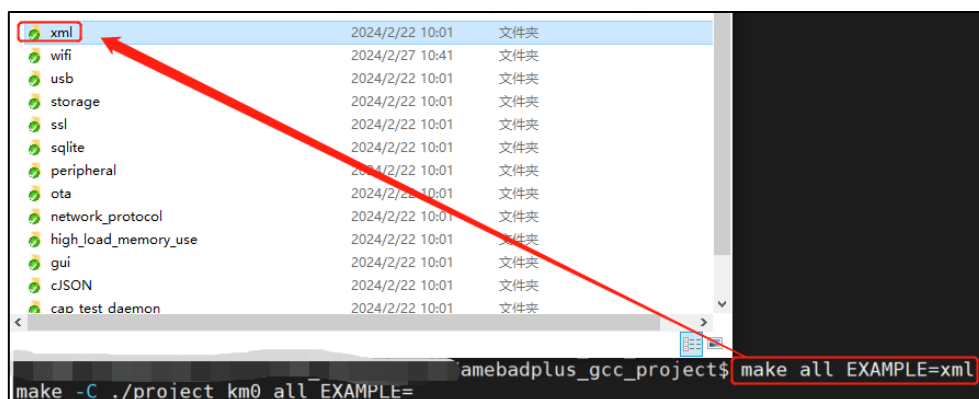


Figure 4-2 Building XML application example

4.2 Peripheral Example

The peripheral examples are demos of peripherals. Most examples consist of raw and mbed folders, you can choose raw or mbed demos as you like.

Table 4-2 Comparison of raw and mbed examples

Items	Path	Description
mbed	{SDK}\component\example\peripheral\{peripheral}\mbed	mbed APIs are used.
raw	{SDK}\component\example\peripheral\{peripheral}\raw	Low-level driver APIs are used.

Each example folder has main.c and README.txt. There are example descriptions, required components, HW connection and expected behavior in README.txt.

The peripheral examples normally run on KM4. To run peripheral examples, you only need to:

- (1) Check software and hardware settings in README.txt of the example.
- (2) Replace the original **main.c** under {SDK}\amebadplus_gcc_project\project_km4\src with **main.c** in the example directory.
- (3) (Optional) Copy other header files that are depicted in the README.txt to {SDK}\amebadplus_gcc_project\project_km4\src if needed.
- (4) Re-build the project by command "**make**".

5 Device Profile Editor

5.1 Introduction

The Device Profile Editor is the official device profile generation tool developed by Realtek. It is released along with ImageTool, and can be used to create new device profiles or edit existing device profiles for ImageTool to download images.

Device profile is a type of encrypted file with extension file name ".rdev". It is defined by Realtek to represent SoC specific information, which can be loaded by ImageTool for image download.

The UI of Device Profile Editor is shown in [Figure 5-1](#).

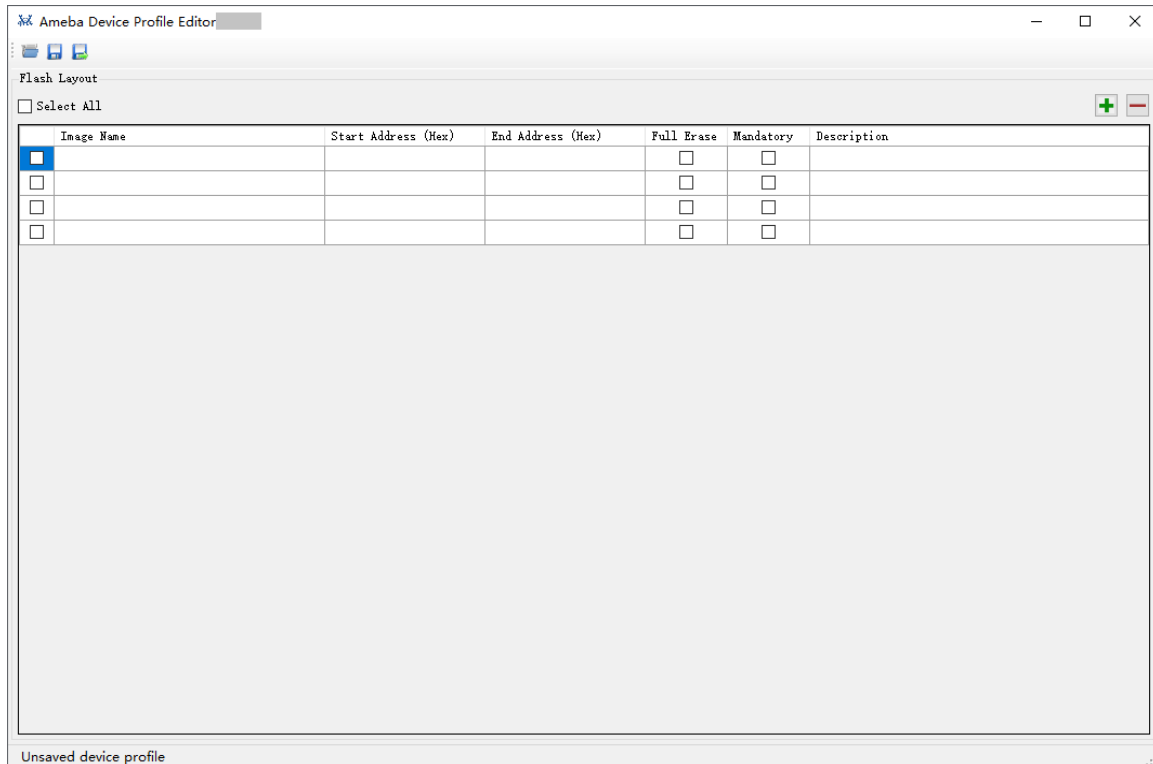


Figure 5-1 Device Profile Editor UI

5.2 Environment Setup

- Environment Requirements: Windows XP or later, Microsoft .NET Framework 4.0.
- Device Profile Editor location: <SDK>/tools/Ameba/DeviceProfileEditor/AmebaDeviceProfileEditor.exe

5.3 Modify a Device Profile

Steps to modify an existing device profile are listed below:

- (1) Launch Device Profile Editor.
- (2) Click **Open** button to load an existing device profile.
- (3) Change the configuration of **Flash Layout** as required.
 - **Image Name**: the image name built by SDK
 - **Start Address**: start address in hex format. For NAND Flash, the value shall be aligned to block size.
 - **End Address**: end address in hex format. For NAND Flash, the value shall be aligned to block size and the partition size shall be a multiple of block size with proper percent of spare blocks (at least one) for bad block management.
 - **Full Erase**: flag indicating ImageTool to erase the entire partition or not before image download
 - ◆ Checked: full erase, normally for file system partitions; for NAND Flash, all the partitions will be checked as default and not allowed to uncheck.

- ◆ Unchecked: not full erase, only the actual size of the image file will be erased, only for NOR Flash non-file-system partitions.
 - **Mandatory:** flag indicating ImageTool to enable the partition to download as default.
 - ◆ Checked: mandatory partition, enabled as default.
 - ◆ Unchecked: optional partition, disabled as default.
 - **Description:** the description text to describe the image, this information will be used as mouse hover tips for images.
- (4) Click **Save** button to overwrite the existing device profile or click **Save As** button to save the modified device profile to a new file.

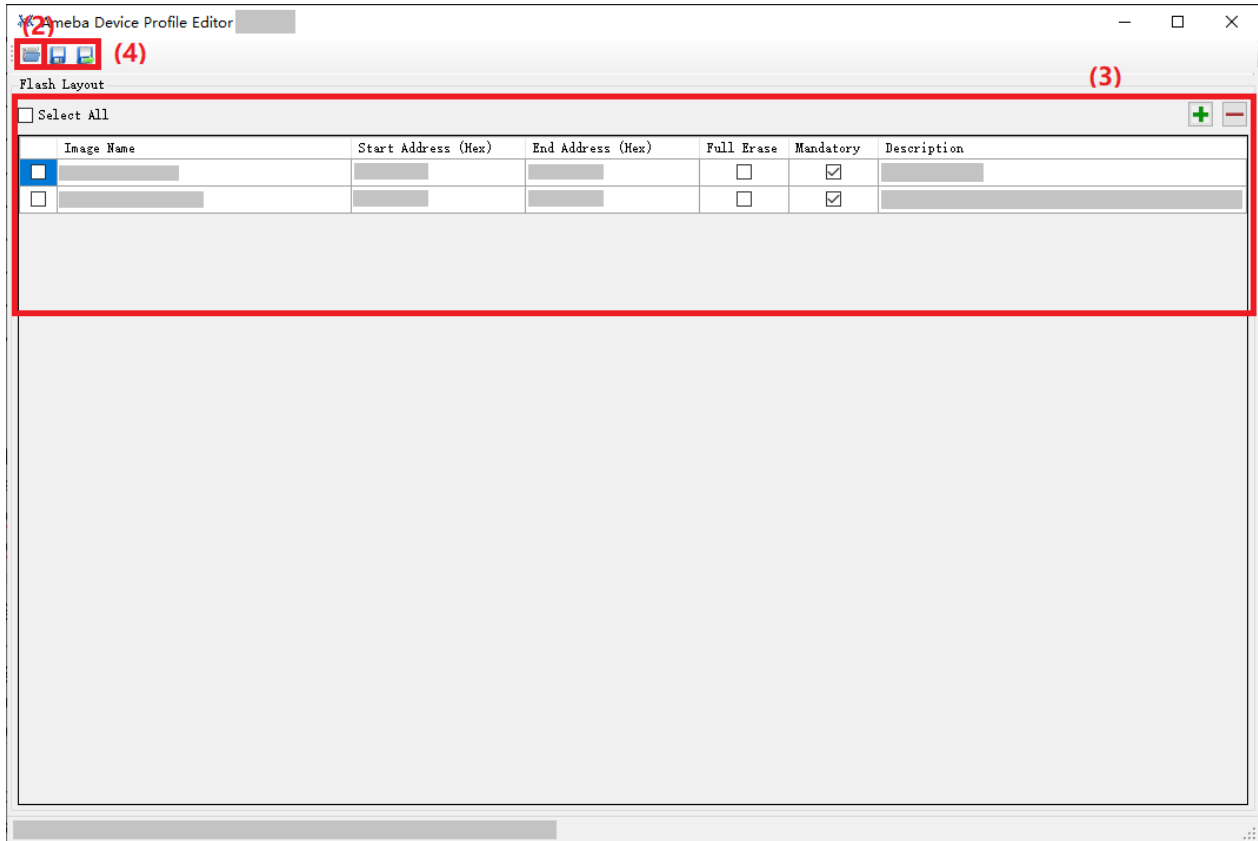


Figure 5-2 Editing an existing device profile

6 Image Tool

6.1 Introduction

The Image Tool is the official image download tool developed by Realtek for Ameba series SoC. It can be used to download images to the Flash of RTL8721Dx device through the interface in [Table 6-1](#).

Table 6-1 RTL8721Dx download interface

Flash type	UART download interface	USB download interface
NOR Flash	✓	✓

The UI of Image Tool is shown in [Figure 6-1](#).

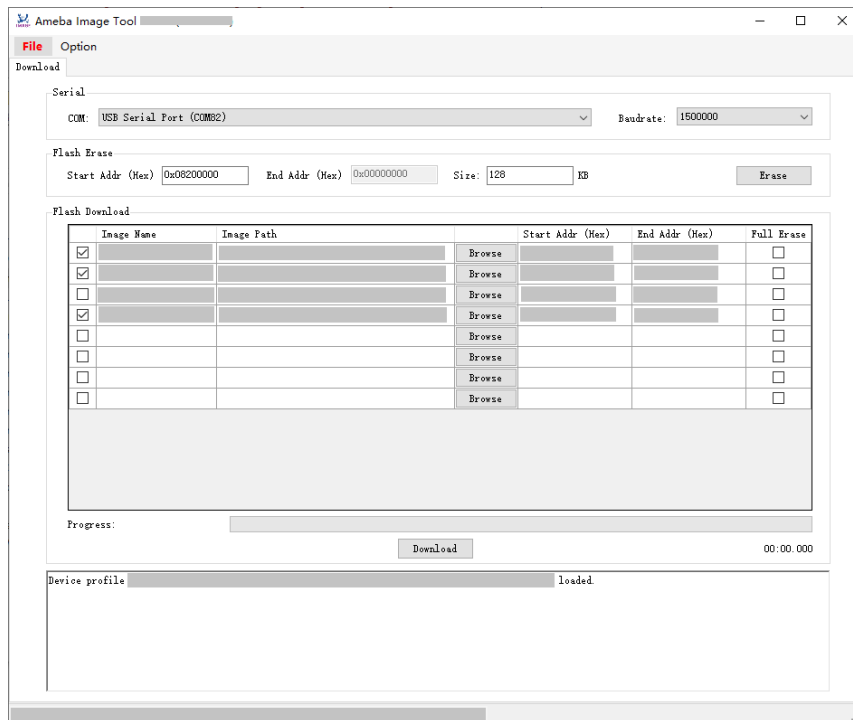


Figure 6-1 Image Tool UI

6.2 Environment Setup

6.2.1 Hardware Setup

The hardware setup for image download is shown in [Figure 6-2](#).

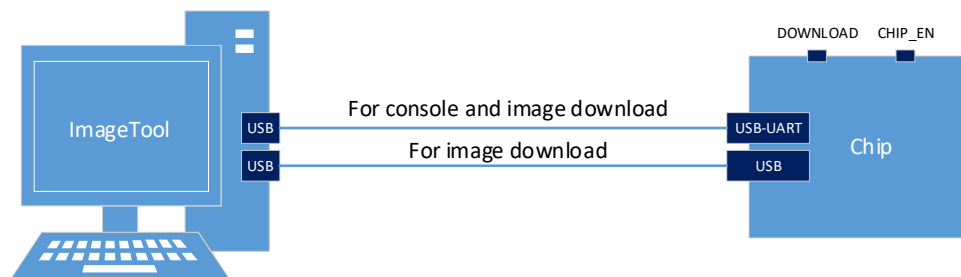


Figure 6-2 Hardware setup for image download

6.2.2 Software Setup

- Environment Requirements: EX. WinXP, Win 7 or later, Microsoft .NET Framework 4.0.
- Software location:
 - Image Tool: <SDK>/tools/Ameba/ImageTool/AmebaImageTool.exe
 - Device profiles: <SDK>/tools/Ameba/DeviceProfiles

Device profiles provide the necessary device information required for image download, with the naming rules:

```
<RTL number>_<OS type>_<Flash type>[_<Extra info>].rdev
```

Where:

- ◆ RTL number: the unique RTL number for a Realtek Ameba SoC
- ◆ OS type: FreeRTOS or Linux
- ◆ Flash type: NOR or NAND
- ◆ Extra info: extra information like Flash size, application, etc.

NOTE

- <ImageTool> will be used for short of <SDK>/tools/Ameba/ImageTool in the following sections.
- For WinXP or Win7 only, install the following USB driver if there is a need to download images through USB interface:
<ImageTool>/RtkUsbCdcAcmSetup.INF

6.3 Image Download

For an empty chip, the following mandatory images shall be downloaded:

Image name	Description	Mandatory?
km4_boot_all.bin	KM4 bootloader	√
km0_km4_app.bin	KM0/KM4 applications	√

The image download steps are illustrated below:

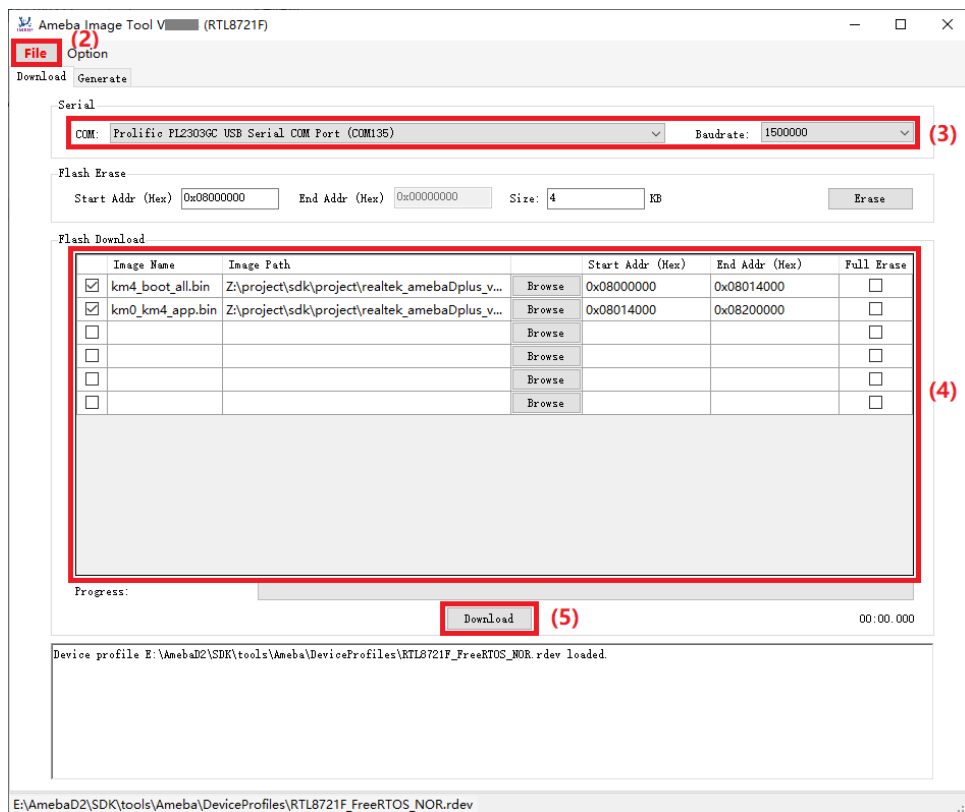


Figure 6-3 Image download operation

- (1) Enter into download mode.
There are two ways to enter into download mode.
 - The first and recommended way is to push the hardware Download and CHIP_EN buttons.

- i. Push the DOWNLOAD button and keep it pressed.
- ii. Re-power on the device or press the CHIP_EN button.
- iii. Release the DOWNLOAD button.

■ The alternate way is to type the “reboot uartburn” command from the UART console if this command is not removed from SDK and AP is running normally.

Now, RTL8721Dx device goes into download mode and is ready to receive data.

- (2) Open Image Tool, click **File > Open** menu and select device profile RTL8721F_FreeRTOS_NOR.rdev.
- (3) Select the corresponding serial port and transmission baud rate. The default baud rate is 1500000.

NOTE

The baud rate will be ignored for USB download interface.

- (4) Click the **Browse** button to select the images to be programmed.

NOTE

Flash layout is allowed to be changed via Image Tool if indeed necessary. However, to formally change the Flash layout, it is suggested to use Device Profile Editor other than Image Tool and the Flash layout in SDK shall be changed accordingly. Refer to Chapter [Device Profile Editor](#) for more details.

- (5) Click the **Download** button to start. The progress bar will show the download progress of each image and the log widget will show the operation status.

6.4 Flash Erase

Steps to erase Flash:

- (1) Enter into download mode as introduced above.
- (2) Open Image Tool, click **File > Open** menu and select the proper device profile.
- (3) Select the corresponding serial port and baud rate.

NOTE

The baud rate will be ignored for USB download interface.

- (4) Input erase start address.
 - For NOR Flash, the value shall be 4KB aligned.
- (5) Input erase size.
 - For NOR Flash, the value shall be cast to a multiple of 4KB.
- (6) Click the **Erase** button, and erase operation begins. You would get the operation result from the log window.

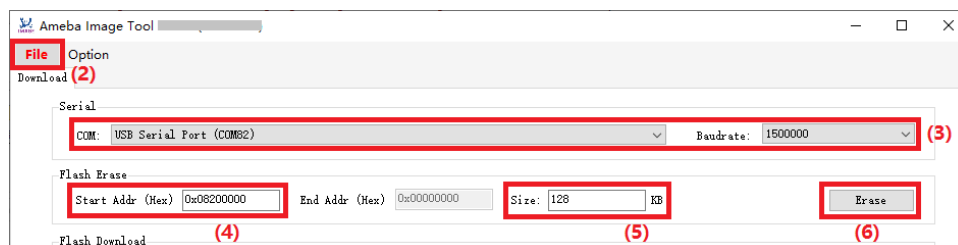


Figure 6-4 NOR Flash erase operation

NOTE

- No need to erase Flash manually before image download since Flash will be automatically erased during image download process.
- If Flash block protection is detected at step (6), refer to section 6.6 for details.

6.5 Flash Register Access

This function is for internal usage only, used to read/write Flash status/feature registers.

CAUTION

Any Flash register operations, especially write operations, shall refer to the datasheet of the Flash, otherwise it may cause irreversible damage to the Flash.

Common pre-steps to access Flash register:

- (1) Make sure Image Tool is closed.
- (2) Enter **expert mode** by editing <ImageTool>/Setting.json, setting **ExpertMode** value to none-zero integer (such as 1).
- (3) Enter into download mode as introduced above.
- (4) Open Image Tool, click **File > Open** menu and select the proper device profile.
- (5) Select the corresponding serial port and baud rate.

6.5.1 NOR Flash Register Access

Besides the common pre-steps, click **Advanced** menu and select **NOR Flash Register Access** menu item to lunch the NOR Flash Register Access dialog for further operations:

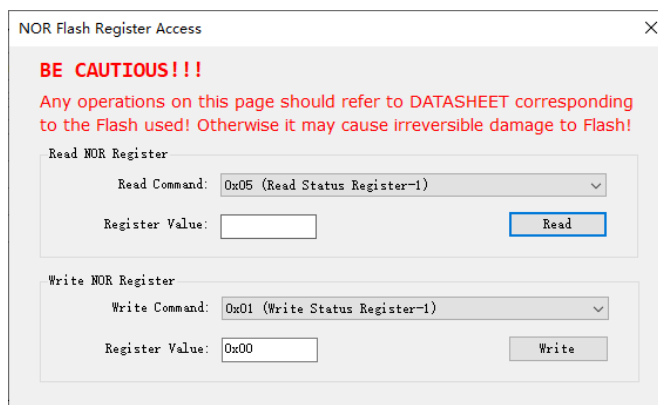


Figure 6-5 NOR Flash Register Access dialog

6.5.1.1 Read NOR Flash Register

After the common pre-steps, next steps to read NOR Flash register:

- (1) Select the read command to read specific register.
- (2) Click the **Read** button, the register value will show up in the Register Value text box.

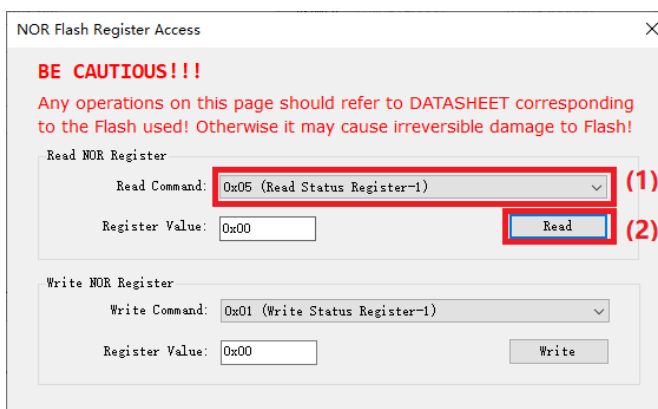


Figure 6-6 Read NOR Flash register operation

6.5.1.2 Write NOR Flash Register

After the common pre-steps, next steps to write NOR Flash register:

- (1) Select the write command to write specific register.
- (2) Input the register value.
- (3) Click the **Write** button.
- (4) Read back the register value for verification, refer to section [6.5.1.1](#).

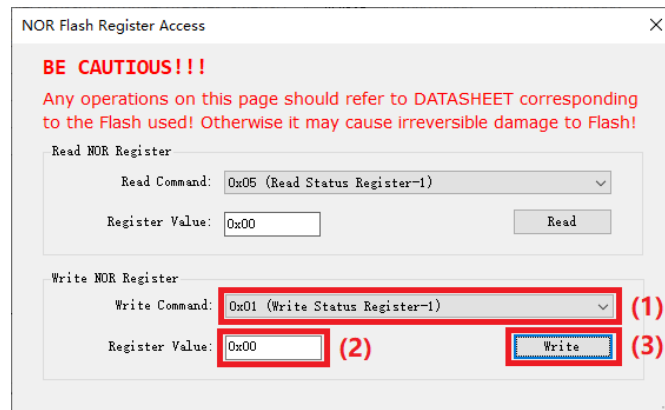


Figure 6-7 Write NOR Flash register operation

6.6 Flash Block Protection Process

During image download or Flash Erase operation, if Flash block protection configuration is detected on RTL8721Dx device, Image Tool will pop up a dialog to guide user for the follow-up actions.

For NOR Flash, only Flash type and protection register value will be shown as [Figure 6-8](#).

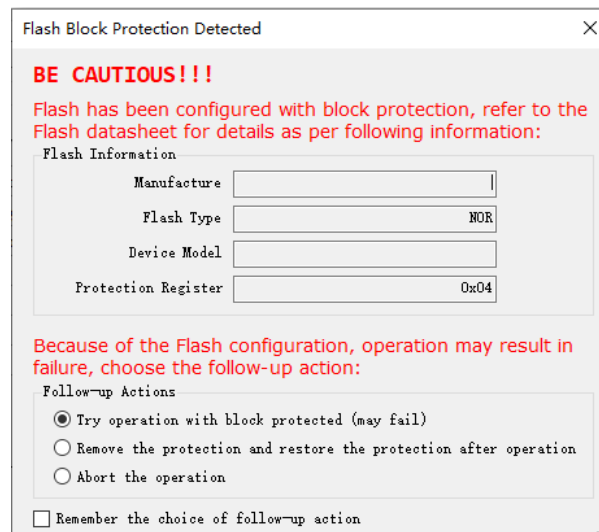


Figure 6-8 Flash block protection detected dialog for NOR Flash

Following follow-up actions are provided for user to choose:

- Try operation with block protected (may fail)
- Remove the protection and restore the protection after operation
- Abort the operation

Additionally, user can check the **Remember the choice of follow-up action** check box to remember the choice for further operations, and uncheck the menu item **Option > Remember Flash Protection Process** to forget the remembered choice.

7 Trace Tool

7.1 Introduction

The Trace Tool is the official serial port debug tool developed by Realtek. It can be used to communicate with the device over a standard RS-232 port. You can directly input commands into the terminal and view or record logs in real time through Trace Tool.

This chapter illustrates how to use Trace Tool to print logs and send commands. The UI of Trace Tool is shown in [Figure 7-1](#). The Trace Tool can display logs when the AGG function of loguart is enabled or disabled. There are two configuration tabs in the upper left corner of UI.

- **Edit**: the history logs clear, history commands clear, load and save.
- **Option: Global Settings** including row count and timestamp, and **Tag Filter** is used to attach different tag before logs to distinguish logs from different cores when the AGG function is enabled.

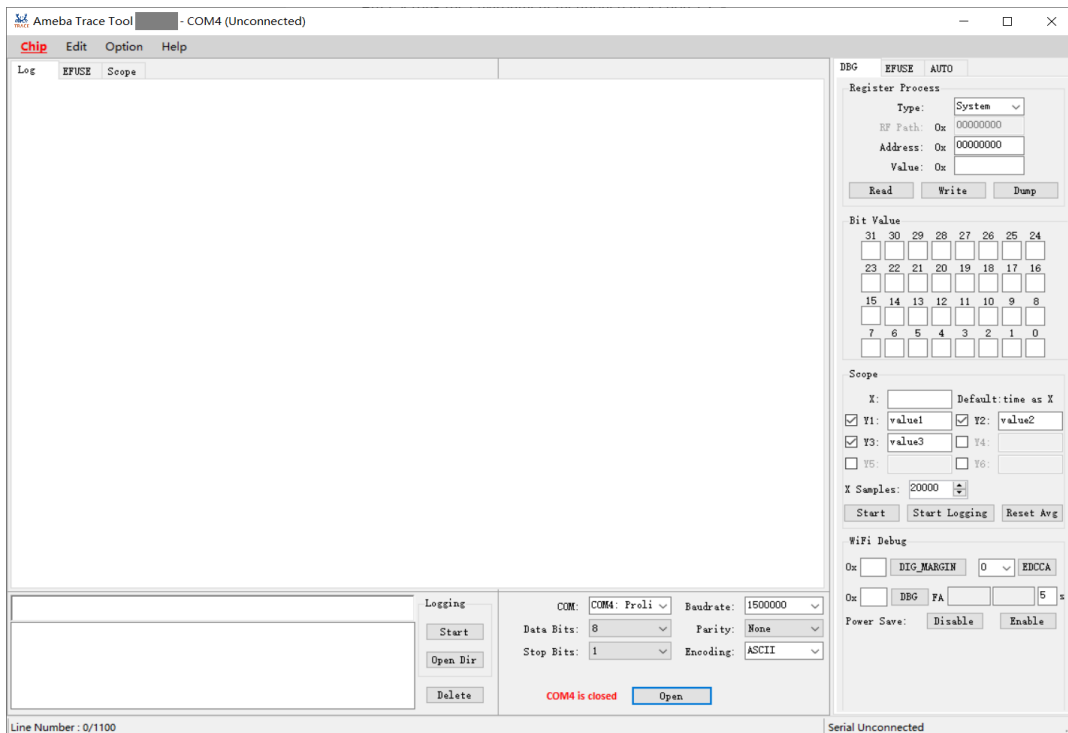


Figure 7-1 Trace tool UI

By default, the AGG (aggregation) function of loguart is disabled so any serial tool can be used to display logs. When the AGG function is enabled, every packet will be send from UART Tx module with AGG header, which is used for Trace Tool to distinguish multi-path data that from different cores.

The AGG function is used for multi-paths to print logs at the same time, which are KM0, KM4, BT trace and BT FW logs.

- When the AGG function is enabled, hardware will attach AGG header automatically. TraceTool can separate logs from different paths according to AGG header. In this case, logs from BT trace and BT FW will be saved into files separately and other logs will print on screen. Therefore, users must use Trace Tool instead of any other serial port tools because the latter will print disordered logs.
- When the AGG function is disabled, no AGG header is attached. In this case, users can use other tools to print log as long as there is no BT trace and BT FW logs. But disordered logs may appear when more than one CPU are printing logs.

7.2 Environment Setup

7.2.1 Hardware Connection

The hardware connection is illustrated in [Figure 7-2](#).

NOTE

If external UART is used to download images, the USB to UART dongle must be used.

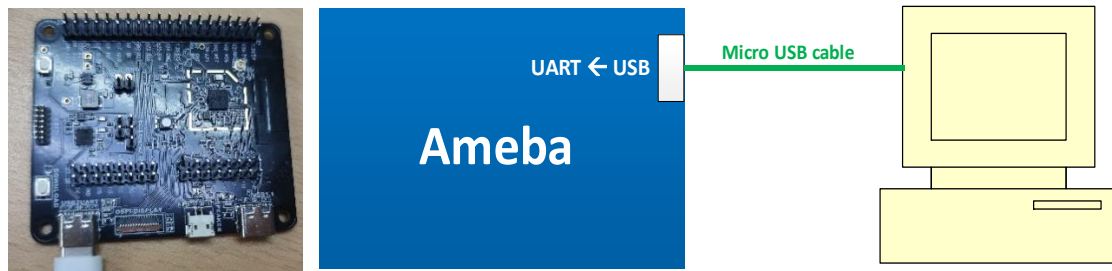


Figure 7-2 Hardware connection

7.2.2 Software Setup

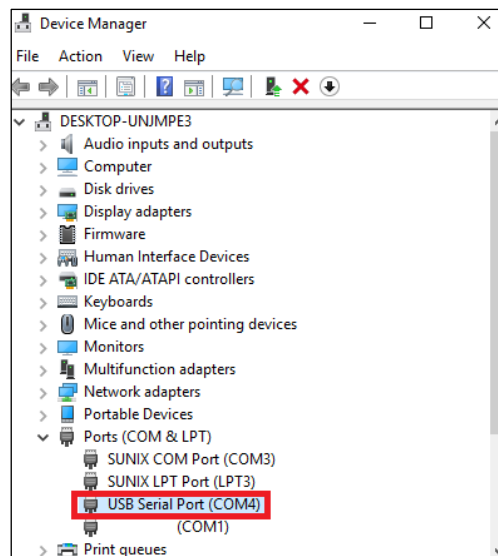
- Environment requirements: WinXP above, .NET Framework 4.0
- Location: <SDK>/tools/Ameba/TraceTool/AmebaTraceTool.exe

7.3 Usage

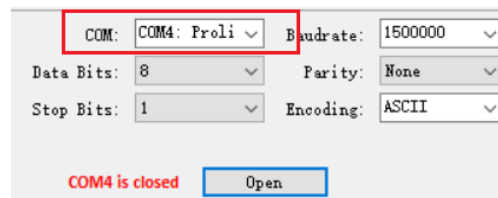
7.3.1 Log Print

After setting the environment mentioned in section 7.2:

- (1) Select COM port.
 - a) Check your COM through Device Manager in your computer, as shown below. In COM & LPT item, all the COM ports connected to the computer are listed.



- b) Select COM port in "Port Setting" box.



- (2) Set the format.
 - Baudrate: default is 1500000bps.
 - Data Bits: default is 8.
 - Parity: default is "None".
 - Stop Bits: default is 1.
 - Encoding: default is UTF8.

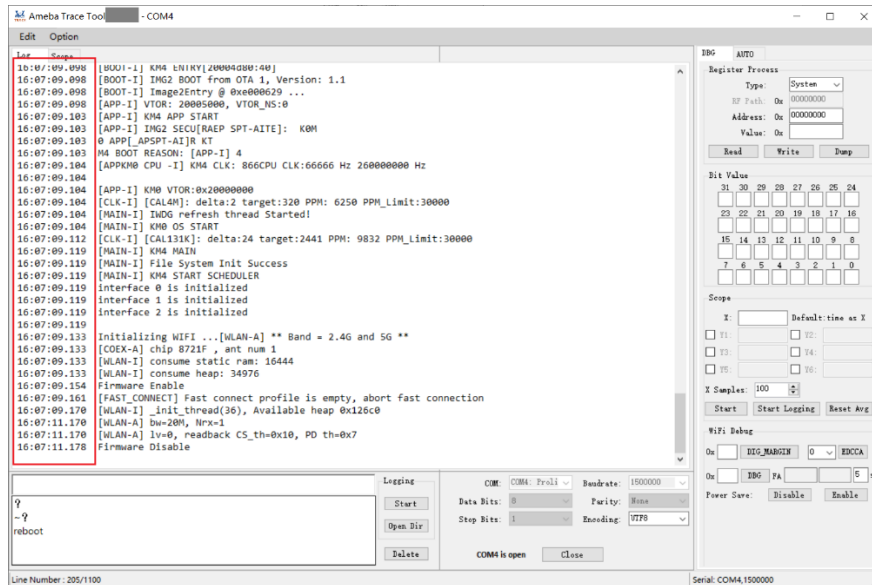
These settings should not be changed freely. If users have to change the format setting for some reasons, the loguart format needs to be changed accordingly to make sure that the format of loguart matches that of Trace Tool.

(3) Click the **Open** button.

- When COM port is open, the Trace Tool starts to receive logs from the device. Log is shown in Log window. By default, the timestamp is inserted at the start of each log.

Click **Option** button, then choose **Global Settings** to select whether add timestamp. Note that the timestamp is not very accurate because log processing takes time so there is a slight time interval between receiving logs and displaying logs.

- If COM port is closed, the Trace Tool will not display logs and the commands can't be sent.



The Trace Tool supports log saving function.

- After clicking **Start** button in "Logging" box, all the receiving logs can be saved in a .txt file in the log folder of the same directory for .exe file.

NOTE

For segmented log, click **Option** and **Global Settings** to set log size.

- After clicking **Open Dir** button in "Logging" box, the folder where the log is stored can be opened directly.

The Trace Tool can print logs when the AGG function of loguart is enabled or disabled, because the Trace Tool can automatically detect whether the AGG function is enabled or not and can handle it accordingly. When the AGG function is enabled, click **option** button and choose **Tag Filter**, log from different cores will be added corresponding tag. There may be a period of disorder logs when the AGG function switches from enable to disable or from disable to enable.

7.3.2 Command Send

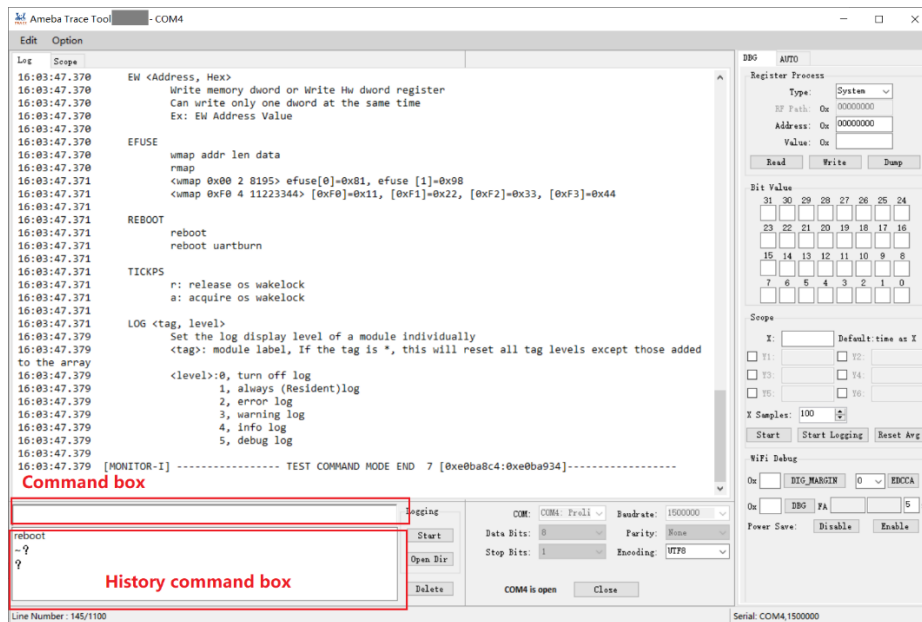
When COM port is open, you can send commands to the device through Trace Tool. The steps are shown below:

- (1) Input a command in command block as shown in the red block.

NOTE

Refer to Section 7.3.3 to decide whether a command prefix is needed to add before the command.

- (2) Press the **"Enter"** key.



The history command box records the commands have been sent before.

- Click the command, it will be displayed in the command box.
- Double-click the command, it will be sent to the chip.
- Click the command then click **delete**, the command will be deleted from the history command box.

7.3.3 Command Prefix

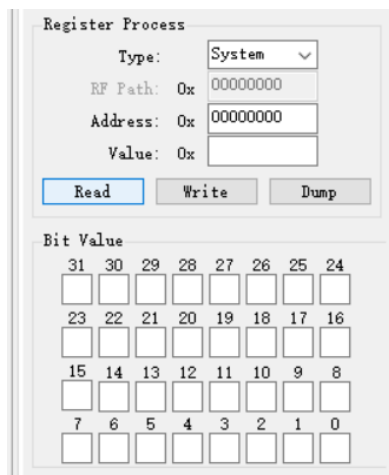
Core	Role	Command prefix
KM4	AP	None
KM0	NP	@

7.3.4 Debug

7.3.4.1 Register Access

The REG function is used to read and write registers by register address.

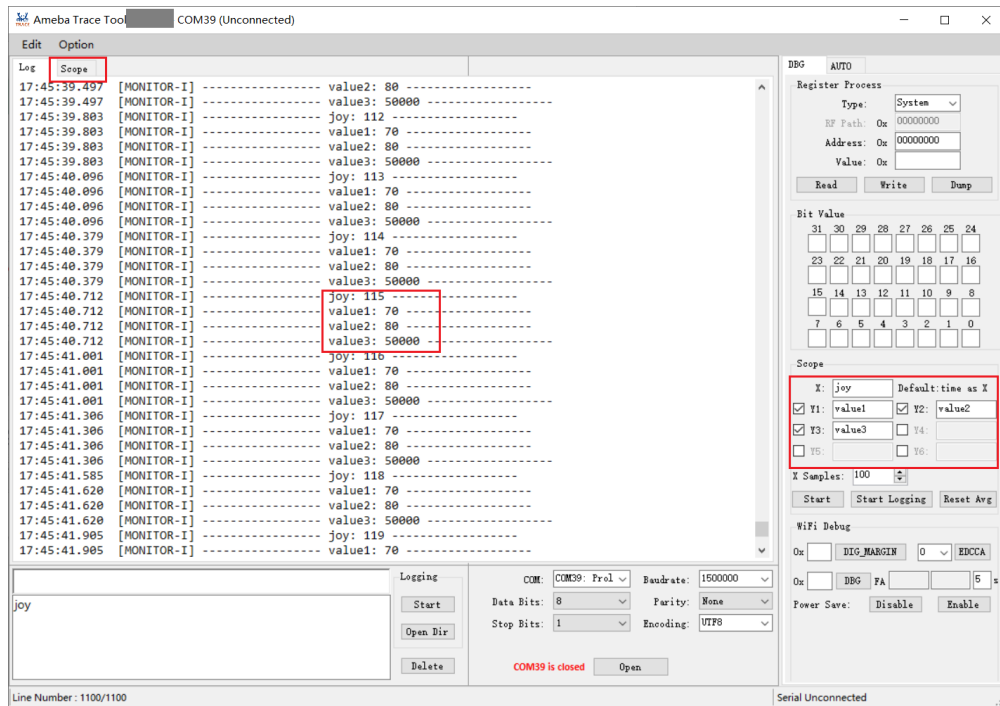
- **Type**: used to select the register type. Wifi MAC/Wifi BB/Wifi RF options are used for Wi-Fi function, which have different base addresses. Select the corresponding options according to your needs.
- **Read/Write Register**: enter the register address to **Read** or **Write** the register value. **Dump** means batch printing register values, only Wifi MAC/Wifi BB/Wifi RF registers are supported to be dump.
- **Bit Value**: bitwise accessing the register specified by address.



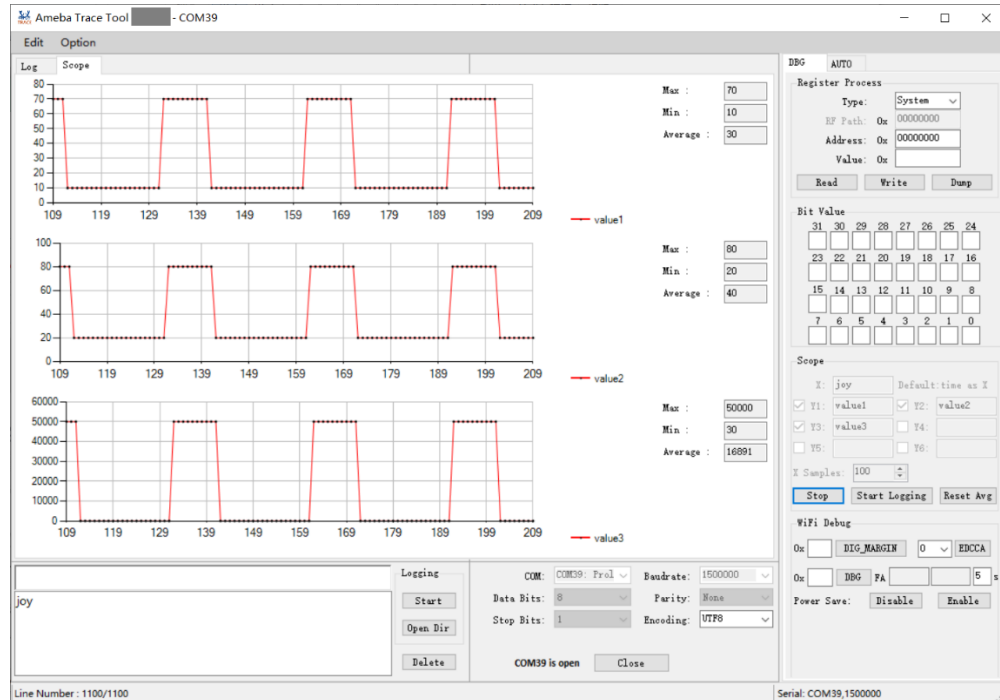
7.3.4.2 Scope Illustration

The scope function is used to capture specific data in log and illustrate waveform dynamically.

- Enter X and Y pattern, X default value is time.
- Click Start button

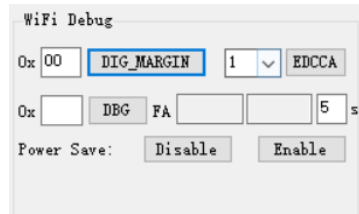


In the scope interface, waveform will be illustrated synchronously.



7.3.4.3 WiFi Debug

- **DIG_MARGIN**: set WiFi DIG margin, available address: [0x00,0x3c].
- **EDCCA**: set MAC EDCCA mode, available value: 0/1/9.
- **DBG**: set WiFi RA debug, available address: [0,0xff]. And illustrate CCK_FA and OFDM_FA average value.
- **Power Save**: enable or disable WiFi power saving mode.

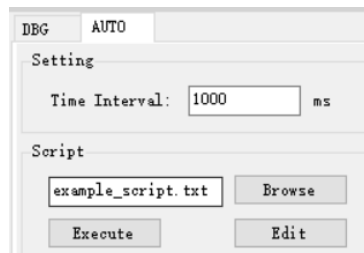


7.3.5 AUTO Script

7.3.5.1 Usage

The AUTO feature is used for automatic script execution.

- (1) Click **Browse** button to choose the script you want to execute.
- (2) Click **Execute** button to execute the script.



7.3.5.2 Script Format

The format of script.txt which is used in auto mode is as follows:

```
CMD1
CMD2
CMD3
...
```

- If commands are required to be repeated multiple times, `loop` can be used:

```
loop=10
loop_start
  CMD1
  CMD2
  sleep 1000
  ...
loop_end
```

- For a loop, three key words are necessary:

- ◆ **loop**: The number after "loop=" means loop times.
- ◆ **loop_start**: Used to mark the beginning of the loop.
- ◆ **loop_end**: Used to mark the end of the loop.

loop_start and **loop_end** have to appear in pairs.

- **sleep**: Used to delay some time between commands, and unit is millisecond. "sleep 1000" means to delay 1000ms. There should be a blank space between "sleep" and sleep time.

- Nested loop is supported as below:

```
loop=2
loop_start
  CMD1
  sleep 1000
  loop=3
  loop_start
    CMD2
    sleep 2000
  loop_end
loop_end
```

- Catching certain patterns, like `pass_pattern` or `fail_pattern`, to indicate the result of specific CMD execution is supported, the format is as below:

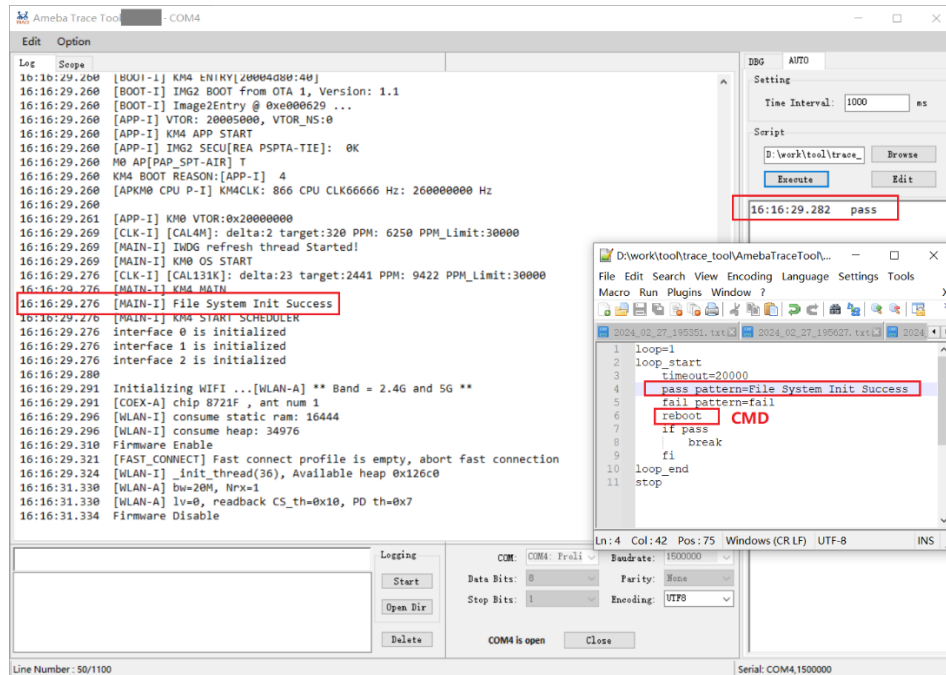
```
loop=10
```

```

loop_start
    timeout=1000
    pass_pattern=xxx
    fail_pattern=xxx
    CMD1
    if fail/pass/timeout
        break
    fi
    CMD2
    ...
loop_end

```

- The key word **pass_pattern** and **fail_pattern** and **timeout** are only valid for the next command CMD1, used to catch patterns in CMD1 execution. When catching the patterns, key word **if...fi** can be used to perform subsequent operation, now only the **break** operation is supported, which is used to jump out of the loop.



- The number after "**timeout**=" means the time frame you want to catch log to match the pattern. It can be set to different values according to needs (unit: millisecond), and default value is 5000.
- The string after "**pass_pattern**=" means the pass_pattern, and the string after "**fail_pattern**=" means the fail_pattern, which is used to indicate the result of the CMD execution.

When the corresponding pattern is matched or not during CMD execution, there will be three results:

- **Pass:** pass_pattern matched within timeout in CMD execution results.
- **Fail:** pass_pattern matched within timeout in CMD execution results.
- **Timeout:** no pass_pattern/fail_pattern matched within timeout in CMD execution results.

CAUTION

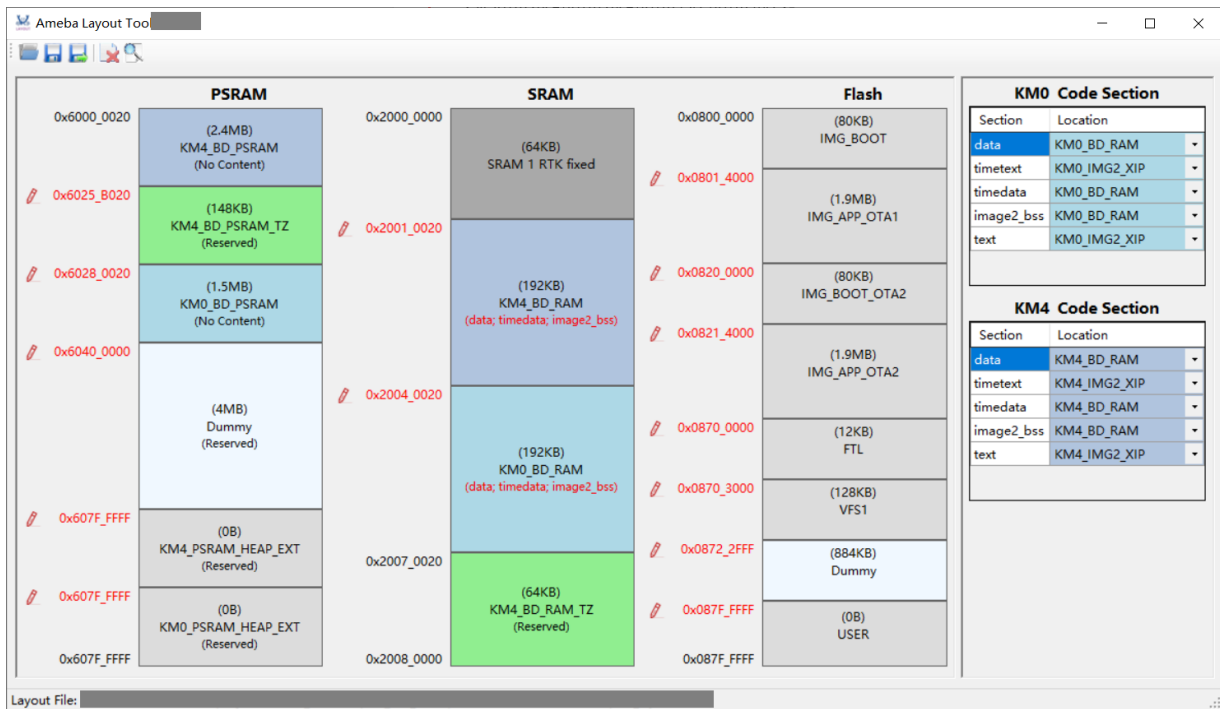
- One command in a single line.
- TAB is used to indent a line. Only TAB can be used, WHITE SPACE is not allowed.
- WHITE SPACE before or after "=" is not allowed.

8 Layout Tool

8.1 Introduction

The Layout Tool is the official layout adjustment tool developed by Realtek for Ameba series SoC, used to adjust chip memory layout and configure link script files.

As shown below, the UI of Layout Tool contains two parts: the memory layout and the link configuration.



8.2 Environment Setup

8.2.1 Software Setup

- Environment Requirements: EX. WinXP, Win 7 or later, Microsoft .NET Framework 4.0.
- Software location: <SDK>/tools/Ameba/LayoutTool/AmebaLayoutTool.exe

8.3 Open

8.3.1 Select SDK Path

The SDK select steps are illustrated below:

- (1) Click the **Open** button.
- (2) Click the **Browse** button to select the SDK path.

The following paths are supported:

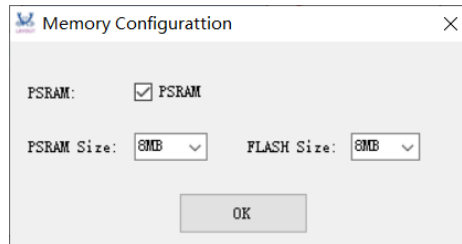
```
SDK
SDK\amebadplus_gcc_project
```

- (3) Click the **OK** button to confirm the selection.

8.3.2 Configure Memory

- (1) Select PSRAM.

- (2) PSRAM size: 4MB/8MB/16MB/32MB, default 8MB.
- (3) FLASH size: 4MB/8MB/16MB/32MB/64MB, default 8MB.
- (4) Click the **OK** button to confirm selection.



8.4 Edit

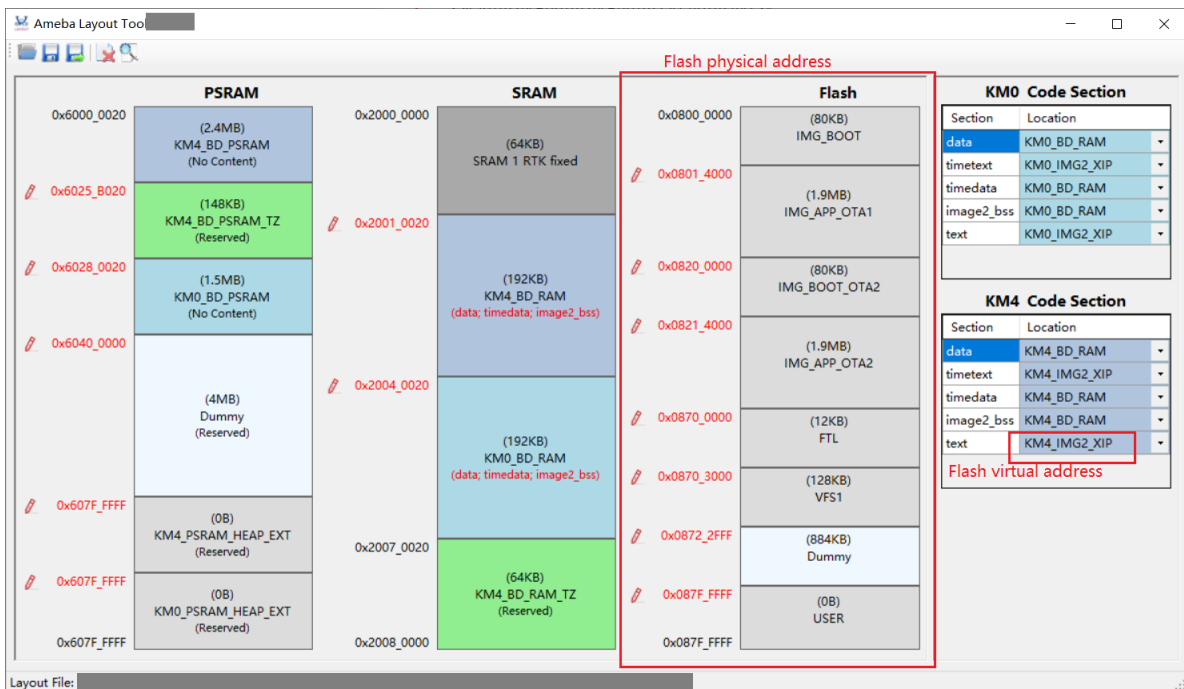
8.4.1 Configure Link

Link configuration is to adjust the location of code section. The adjustment steps are illustrated below:

- (1) Click the value of the **Location** column which needs to be edited.
 - (2) Click the drop-down box to switch the location.
- The memory layout will be changed accordingly when code section is moved.

NOTE

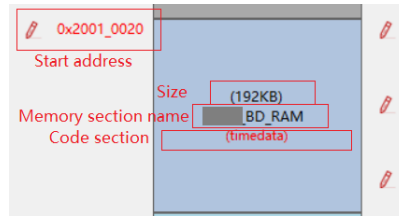
- Location value containing "IMG2_XIP" means **FLASH** virtual address.
- In order to adjust the memory address conveniently, the memory layout shows **FLASH** physical address.



8.4.2 Resize Memory Section

Memory layout contains PSARM, SRAM, and Flash. Every memory consists of several memory sections. The structure of memory section is illustrated below:

- Start address: value in red means can be adjusted.
- Size: in MB/KB/B.
- Memory section name.
- Code section: value in red means code section can be moved, **Reserved** means this section did not contain code section.



The steps of adjusting start address are:

- (1) Click the start address value.
- (2) Enter the required address in keyboard.

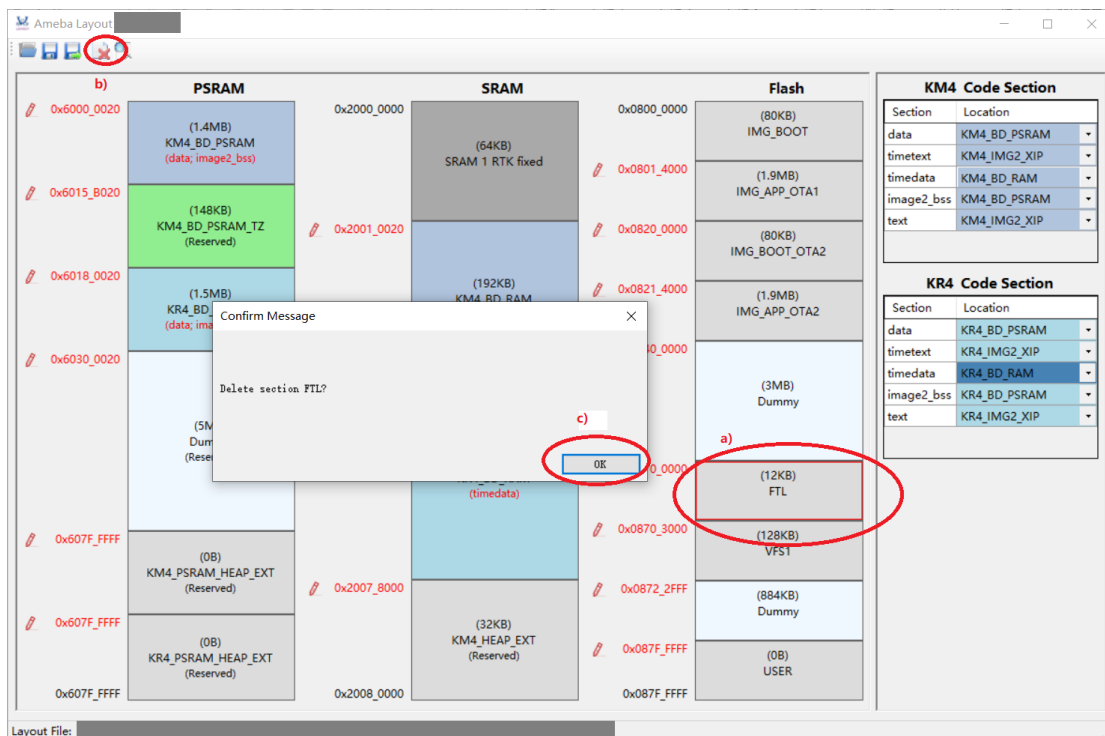
NOTE

- If not **Dummy** section, range of start address: pre-section start address (not contained) - current section end address (not contained).
 - If **Dummy** section, range of start address: pre-section start address (not contained) - current section end address (contained).
- (3) Press **Enter** to enable it, size will be changed accordingly.

8.4.3 Delete Section

There are two ways to delete a memory section:

- Use **Delete** button in tool strip
 - a) Click the section, the border will turn red to indicate selected.
 - b) Click the **Delete** button.
 - c) Click the **OK** button to confirm deletion.



- Use right mouse button
 - a) Right-click on the section.
 - b) Select **Delete**.
 - c) Click the **OK** button to confirm deletion.

When memory section is deleted, its name turns **Dummy**, and its color changes to light blue. In the case its space can be allocated freely.

8.5 Preview

- (1) Click the **Preview** button to preview the link scripts.
- (2) Use **Core** combo-box to preview different core's link script.

8.6 Save and Save As

8.6.1 Save

Click the **Save** button to save layout file and link script files.

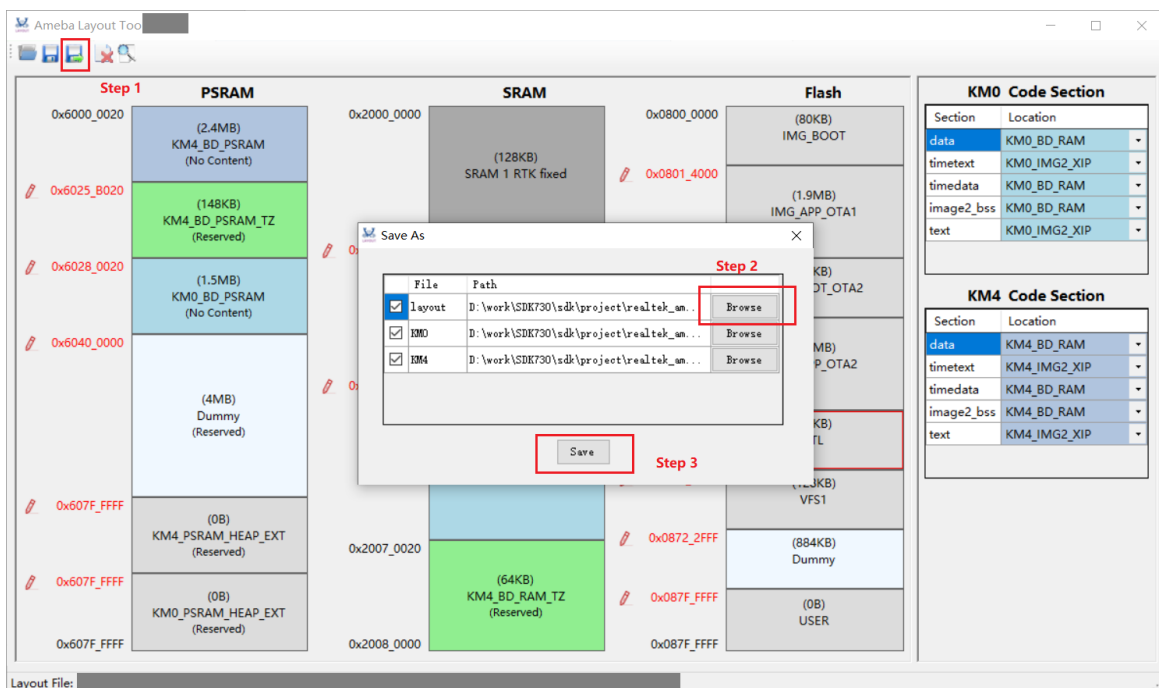
8.6.2 Save As

The steps of save as are illustrated below:

- (1) Click the **Save As** button in menu bar.
- (2) Click the **Browse** button to edit the file name.
- (3) Click the **OK** button.

NOTE

Path of the layout file and link script file cannot be changed, only allowed to save as another file.



9 Flash Layout

This chapter introduces the default Flash layout of RTL8721Dx and how to modify the Flash layout if needed.

9.1 Introduction

The default Flash layout used in SDK is illustrated in [Figure 9-1](#) and [Table 9-1](#). The layout takes the 8MB Flash as an example. The start address of boot manifest is fixed to 0x0800_0000, and other start addresses can be configured by users flexibly.

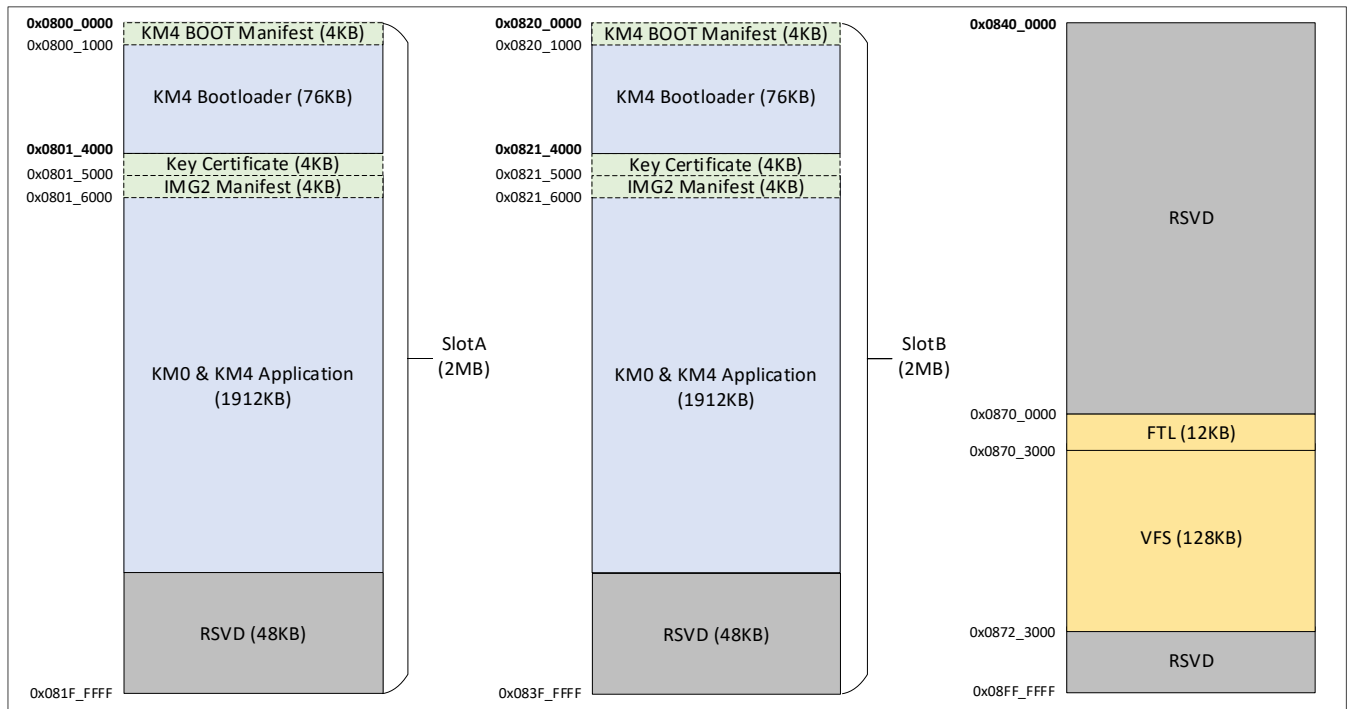


Figure 9-1 Flash layout

Table 9-1 Flash layout

Item	Physical address	Size (KB)	Description	Mandatory?
KM4 Bootloader manifest	0x0800_0000	4	KM4 bootloader manifest	✓
KM4 Bootloader	0x0800_1000	76	KM4 bootloader (code/data), containing KM4 bootloader IMG, mapped to the virtual address 0x0F80_0000.	✓
Key Certificate	0x0801_4000	4	Public key hash information for other images.	✓
IMG2 manifest	0x0801_5000	4	KM0 & KM4 Application & KM4 IMG3 manifest	✓
KM0 & KM4 Application	0x0801_6000	1912	Combines KM0 image, KM4 image2 and KM4 image3 ● KM0_IMG: KM0 image (code/data), mapped to the virtual address 0x0C00_0000. ● KM4_IMG2: KM4 non-secure image (code/data), mapped to the virtual address 0x0E00_0000. ● KM4_IMG3: secure image (code/data)	✓
KM4 Bootloader manifest OTA2	0x0820_0000	4	KM4 bootloader manifest	✓
KM4 Bootloader OTA2	0x0820_1000	76	KM4 bootloader (code/data), containing KM4 bootloader IMG, mapped to the virtual address 0x0F80_0000.	✓
Key Certificate OTA2	0x0821_4000	4	Public key hash information for other images.	✓
IMG2 manifest OTA2	0x0821_5000	4	KM0 & KM4 Application & KM4 IMG3 manifest	✓
KM0 & KM4 Application OTA2	0x0821_6000	1912	Combines KM0 image, KM4 image2 and KM4 image3 ● KM0_IMG: KM0 image (code/data), mapped to the virtual address 0x0C00_0000. ● KM4_IMG2: KM4 non-secure image (code/data), mapped to the virtual address 0x0E00_0000. ● KM4_IMG3: secure image (code/data)	✓
FTL	0x0870_0000	12	For Bluetooth	×
VFS	0x0870_3000	128	For file system	×

NOTE

The reserved space can be used by users.

9.2 Memory Management Unit (MMU)

To achieve flexibility of image and for image encryption when RSIP is enabled (the fixed address is needed by IV when doing image encryption, refer to RSIP for more information), Flash MMU is applied by default. The default MMU layout used in SDK is illustrated in [Figure 9-2](#).

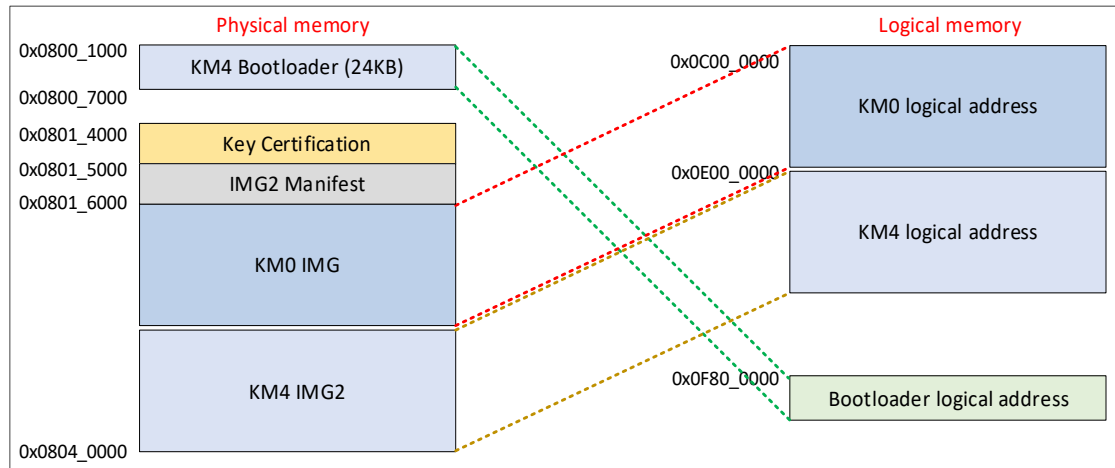


Figure 9-2 Flash MMU layout

9.3 How to Modify Flash Layout

The following locations in the Flash can be modified through the Layout Tool provided by Realtek. Refer to Chapter [Layout Tool](#) to modify the Flash layout.

- Bootloader OTA2 location
- APP location
 - APP OTA1
 - APP OTA2
- FTL/VFS Location

9.4 Flash Protect Enable

Before loading APP IMG, the Bootloader will read the Status Register from Flash. If only Quad Enable (QE) Bit is set in the output of bitwise AND between Status Register of Flash and status_mask in Flash_AVL (component\soc\amebadplus\usrsrc\ameba_flashcfg.c), do nothing, or the output of bitwise AND will be written to the Flash Status Register.

NOTE

By default, setting the QE bit will unlock all the Block Protect Bits. To avoid this operation, set Block Protect bits corresponding to Status_mask in Flash_AVL to 0. For example, change the Status_mask of winbond in the Flash_AVL to 0x000043C0.

```

/**
 * @brief Flash_AVL maintains the flash IC supported by SDK.
 * If users want to adopt new flash, add item in the following AVL.
 * Note that, if new flash can be classified as one of the Realtek-defined categories according to Classification SPEC,
 * filling in the defined class index is necessary. Otherwise, FlashClassUser can be used to indicate new class.
 */
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_*/
    {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,      FlashClass1,      0x000043FC,      NULL}, /*XMC: MANUFACTURER_ID_WINBOND*/
    //{0x20,          0x000000FF,      FlashClass5,      0x000000FC,      NULL}, /*Micron: 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},
};

```

In order to avoid the image being destroyed due to improper operation when writing the user data using LittleFS, it is recommended to modify the location of FTL/LittleFS to the last 64KB area of Flash, and set the Block Protect Bit in the Status Register of Flash at the same time.

NOTE

- Only the last 64KB area of Flash can be modified, and the other areas are protected. Remember to unlock the Flash during OTA upgrade, and keep it locked when OTA is completed.
- For some Flashes, you cannot set the Flash to allow only the last block to be modified through Block Protect Bit. In this case, it is recommended to enable the Flash block protection of the first half part.

10 Memory Layout

This chapter introduces the default memory layout of RTL8721Dx and how to modify the memory layout if needed.

10.1 RAM Layout

In total, there are 512KB SRAM on chip, and the size of PSRAM can be 0MB/4MB/8MB/16MB..., which is decided by users. The RAM layout is illustrated in [Figure 10-1](#).

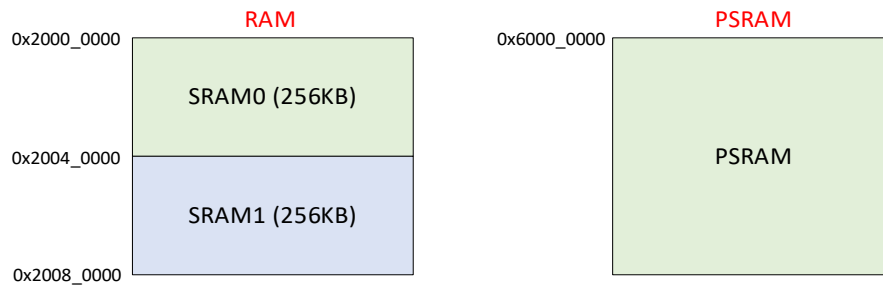


Figure 10-1 RAM layout

10.1.1 SRAM0 (First 64KB) Layout

The first 64KB SRAM0 layout is illustrated in [Table 10-1](#) and [Figure 10-2](#). It is the same for all situations.

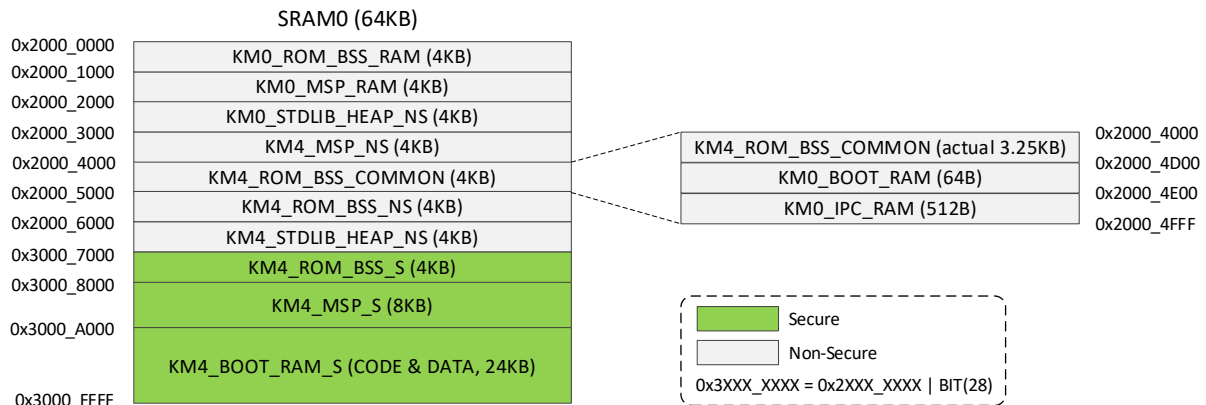


Figure 10-2 SRAM0 (first 64KB) layout

Table 10-1 SRAM0 (first 64KB) layout

Items	Start address	Size	Description	Mandatory?
KM0_ROM_BSS_RAM	0x2000_0000	4KB	KM0 ROM BSS	✓
KM0_MSP_RAM	0x2000_1000	4KB	KM0 Main Stack Pointer	✓
KM0_STDLIB_HEAP_NS	0x2000_2000	4KB	KM0 ROM STDLIB heap	✓
KM4_MSP_NS	0x2000_3000	4KB	KM4 non-secure Main Stack Pointer	✓
KM4_ROM_BSS_COMMON	0x2000_4000	3.25KB	KM4 ROM secure and non-secure common BSS	✓
KM0_BOOT_RAM	0x2000_4D00	64B	KM0 IMG2 entry	✓
KM0_IPC_RAM	0x2000_4E00	512B	Exchange messages between cores	✓
KM4_ROM_BSS_NS	0x2000_5000	4KB	KM4 ROM non-secure common BSS	✓
KM4_STDLIB_HEAP_NS	0x2000_6000	4KB	KM4 ROM non-secure STDLIB heap	✓
KM4_ROM_BSS_S	0x3000_7000	4KB	KM4 ROM secure-only BSS	✓
KM4_MSP_S	0x3000_8000	8KB	KM4 secure Main Stack Pointer	✓
KM4_BOOT_RAM_S	0x3000_A000	24KB	KM4 secure bootloader, including code and data	✓

10.1.2 RAM & PSRAM Layout

There are 192KB SRAM for KM4 and 192KB SRAM for KM0, which can be used for Power Management Controller (PMC) code and performance-cared text and data. [Figure 10-3](#) and [Table 10-2](#) illustrate the RAM layout with PSRAM.

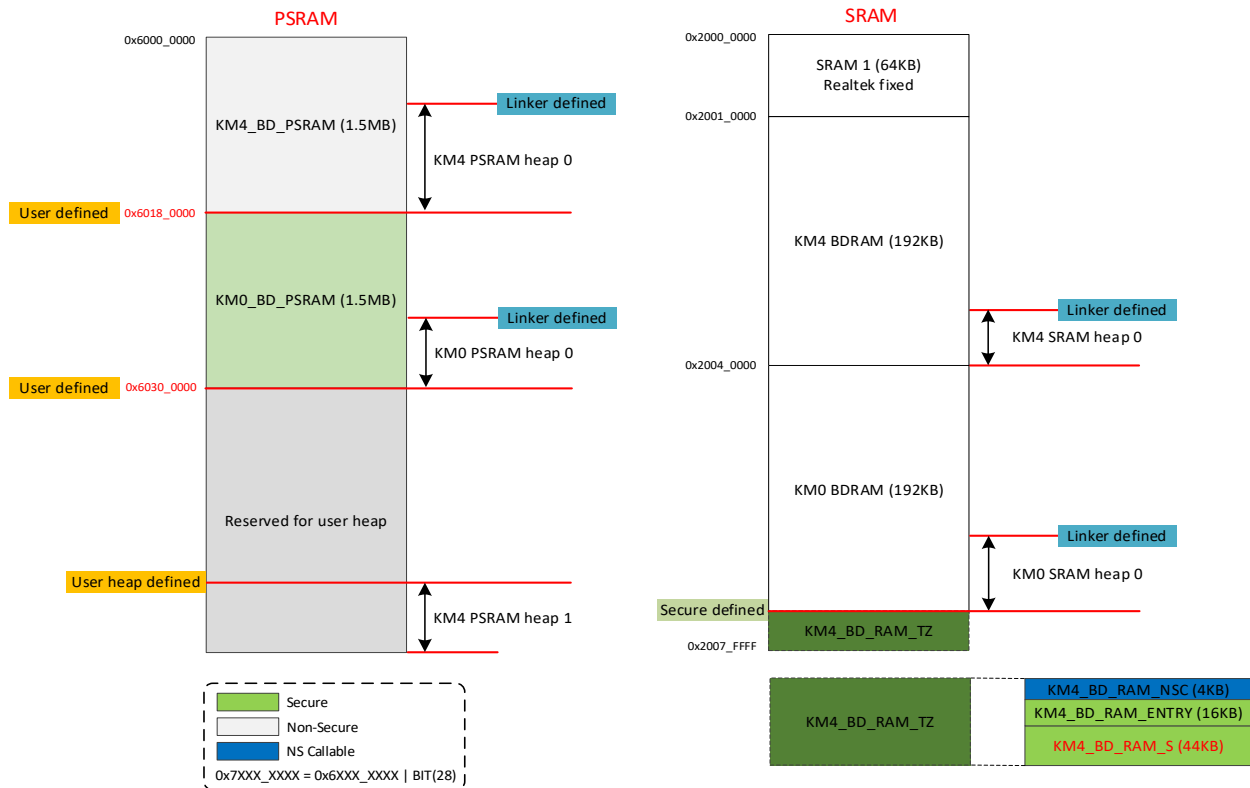


Figure 10-3 RAM layout (with PSRAM)

Table 10-2 RAM layout (with PSRAM)

Item	Start address	Size (KB)	Description	Mandatory?
SRAM0	0x2000_0000	64	For Bootloader, ROM BSS, MSP, ...	✓
KM4 BDRAM	0x2001_0000	192	KM4 BDRAM data, BSS and heap	✓
KM0 BDRAM	0x2004_0000	192	KM0 BDRAM data, BSS and heap	✓
KM4 IMG3	0x2007_0000	64	KM4 IMG3, can be recycled if IMG3 is not needed	✓
KM4 PSRAM	0x6000_0000	1536	KM4 PSRAM code, can be empty	✓
KM0 PSRAM	0x6018_0000	1536	KM0 PSRAM code, can be empty	✓
KM4 HEAP EXT	0x6FFF_FFFF	0	If KM4 heap is not enough, it can be used to extend the heap size	x
KM0 HEAP EXT	0x6FFF_FFFF	0	If KM0 heap is not enough, it can be used to extend the heap size	x

10.2 How to Modify Memory Layout

The following memory size can be modified through the Layout Tool provided by Realtek. Refer to Chapter [Layout Tool](#) to modify the memory layout.

- Bootloader
- BD_RAM
- BD_PSRAM
- Heap

11 MPU

11.1 Functional Description

The Memory Protection Unit (MPU) is a component provided by Arm and is used to provide hardware protection by software definition. The code in SDK provides the `mpu_region_config` struct to set the region memory attribute of MPU.

Table 11-1 shows the member variables of the `mpu_region_config` struct.

Table 11-1 `mpu_region_config` struct

Member variable name	Type	Description
<code>region_base</code>	<code>uint32_t</code>	MPU region base, 32 bytes aligned
<code>region_size</code>	<code>uint32_t</code>	MPU region size, 32 bytes aligned
<code>xn</code>	<code>uint8_t</code>	Execute Never attribute ● <code>MPU_EXEC_ALLOW</code>: Allows program execution in this region ● <code>MPU_EXEC_NEVER</code>: Does not allow program execution in this region
<code>ap</code>	<code>uint8_t</code>	Access permissions ● <code>MPU_PRIV_RW</code>: Read/write by privileged code only ● <code>MPU_UN_PRIV_RW</code>: Read/write by any privilege level ● <code>MPU_PRIV_R</code>: Read only by privileged code only ● <code>MPU_PRIV_W</code>: Read only by any privilege level
<code>sh</code>	<code>uint8_t</code>	Share ability for Normal memory ● <code>MPU_NON_SHAREABLE</code>: Non-shareable ● <code>MPU_OUT_SHAREABLE</code>: Outer shareable ● <code>MPU_INR_SHAREABLE</code>: Inner shareable
<code>attr_idx</code>	<code>uint8_t</code>	Memory attribute indirect index This parameter can be a value of 0 ~ 7, the detailed attribute is defined in <code>mpu_init()</code> and is customized. The typical definition is as follows: ● 0: <code>MPU_MEM_ATTR_IDX_NC</code>, defines memory attribute of Normal memory with non-cacheable. ● 1: <code>MPU_MEM_ATTR_IDX_WT_T_RA</code>, defines memory attribute of Normal memory with write-through transient, read allocation. ● 2: <code>MPU_MEM_ATTR_IDX_WB_T_RWA</code>, defines memory attribute of Normal memory with write-back transient, read and write allocation. ● 3 ~ 7: <code>MPU_MEM_ATTR_IDX_DEVICE</code>, defines memory attribute of Device memory with non-gathering, non-recording, non-early Write Acknowledge.

11.2 MPU APIs

11.2.1 `mpu_init`

Items	Description
Introduction	Initialize MPU region memory attribute to typical value
Parameters	None
Return	None

11.2.2 `mpu_set_mem_attr`

Items	Description
Introduction	Change MPU region memory attribute
Parameters	● <code>attr_idx</code>: region memory attribute index, which can be 0 ~ 7. ● <code>mem_attr</code>: region memory attributes.
Return	None

11.2.3 mpu_region_cfg

Items	Description
Introduction	Configure MPU region memory attribute.
Parameters	● region_num: ■ KM4_NS: 0 ~ 7 ■ KM4_S: 0 ~ 3 ● pmpu_cfg: point to the <code>mpu_region_config</code> struct which has been configured
Return	None

11.2.4 mpu_entry_free

Items	Description
Introduction	Free MPU entry
Parameters	MPU entry index: ● KM4_NS: 0 ~ 7 ● KM4_S: 0 ~ 3
Return	None

11.2.5 mpu_entry_alloc

Items	Description
Introduction	Allocate a free MPU entry
Parameters	None
Return	MPU entry index: ● KM4_NS: 0 ~ 7 ● KM4_S: 0 ~ 3 ● Fail: -1

11.3 Usage

Follow these steps to set a MPU region:

- (1) Define a new variable and struct
 - Variable to store MPU entry index
 - Struct `mpu_region_config` to store the region memory attribute
- (2) Call `mpu_entry_alloc()` to allocate a free MPU entry
- (3) Set the struct of region memory attribute
- (4) Call `mpu_region_cfg()` to configure MPU region memory attribute

12 Cache

12.1 Functional Description

The Cache of RTL8721Dx supports Enable/Disable, Flush and Clean operation, as [Table 12-1](#) lists.

Table 12-1 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	x	✓

NOTE

In the ROM code, the default states of Cache are:

- KM4 Cache: enabled by default
- KM0 Cache: disabled by default

12.2 Cache APIs

12.2.1 ICache_Enable

Items	Description
Introduction	Enable I-Cache
Parameters	None
Return	None

12.2.2 ICache_Disable

Items	Description
Introduction	Disable I-Cache
Parameters	None
Return	None

12.2.3 ICache_Invalidate

Items	Description
Introduction	Invalidate I-Cache
Parameters	None
Return	None

12.2.4 DCache_IsEnabled

Items	Description
Introduction	Check D-Cache enabled or not
Parameters	None
Return	D-Cache enable status: ● 1: Enable ● 0: Disable

12.2.5 DCache_Enable

Items	Description
Introduction	Enable D-Cache
Parameters	None
Return	None

12.2.6 DCache_Disable

Items	Description
Introduction	Disable D-Cache
Parameters	None
Return	None

12.2.7 DCache_Invalidate

Items	Description
Introduction	Invalidate D-Cache by address
Parameters	- Address: Invalidated address (aligned to 32-byte boundary) - Bytes: Size of memory block (in number of bytes)
Return	None

12.2.8 DCache_Clean

Items	Description
Introduction	Clean D-Cache by address
Parameters	- Address: Clean address (aligned to 32-byte boundary) - Bytes: size of memory block (in number of bytes) - Note: Address set 0xFFFFFFFF is used to clean all D-Cache
Return	None

12.2.9 DCache_CleanInvalidate

Items	Description
Introduction	Clean and invalidate D-Cache by address
Parameters	- Address: Clean and invalidated address (aligned to 32-byte boundary) - Bytes: size of memory block (in number of bytes) - Note: Address set 0xFFFFFFFF is used to clean and flush all D-Cache
Return	None

12.3 How to Define a Non-cacheable Data Buffer

Add `SRAM_NOCACHE_DATA_SECTION` before the buffer definition to define a data buffer with non-cacheable attribute.

```
SRAM_NOCACHE_DATA_SECTION u8 noncache_buffer[DATA_BUFFER_SIZE];
```

12.4 Cache Consistency When Using DMA

When DMA is used to migrate data from/to memory buffers, the start and end address of the buffer must be aligned with the cache line to avoid inconsistencies between cache data and memory data. For example, if the start address of a buffer is in the middle of the cache line and the first half is occupied by other programs, when other programs invalidate or clean the current cache line, this operation will affect the entire cache line, resulting in inconsistent cache and memory data of the current buffer.

CAUTION

The DMA operation address requires exclusive ownership of a complete cache line. You can define the buffer using `malloc()` or `ALIGNMT0(CACHE_LINE_SIZE) u8 op_buffer[CACHE_LINE_ALIGNMENT(op_buffer_size)]`.

DMA Tx Flow:

- (1) CPU allocates Tx buffer
- (2) CPU writes Tx buffer
- (3) Realtek recommends: DCache_Clean
- (4) DMA Tx Config
- (5) DMA Tx Interrupt

DMA Rx Flow:

- (1) CPU allocates Rx buffer
- (2) DCache_Clean (if the Rx buffer is in a clean state, this step can be skipped)

CAUTION

If the Rx buffer is in a dirty state in the cache, the CPU may write the Rx buffer back to memory from the cache when CPU's D-Cache becomes full, which could overwrite content that DMA Rx has already written.

- (3) DMA Rx Config
- (4) DMA Rx interrupt
- (5) DCache_Invalidate (this step is mandatory)

CAUTION

Prevents the CPU from reading old values into the cache during DMA processing.

- (6) CPU read Rx buffer (the value returned by DMA RX).

13 MP Image

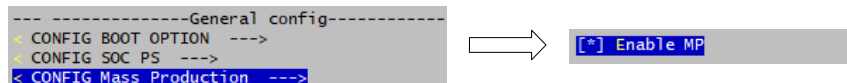
The MP image is used for Wi-Fi & BT performance verification and Wi-Fi & BT parameters calibration in massive production. This chapter mainly illustrates how to build and use the MP image.

Refer to the MP document for more details about MP flow.

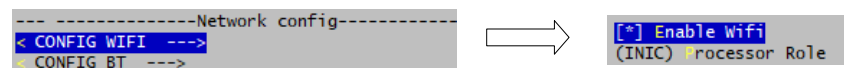
13.1 How to Build MP Image

The steps of building MP image are depicted below:

- (1) Switch to work directory **{SDK}\amebadplus_gcc_project**.
- (2) Use command **\$ make menuconfig** to modify the configurations.
 - a) Enable MP



- b) Enable Wi-Fi, and configure KM4 as AP core and KM0 as NP core.



- c) Enable BT.



- d) Save and exit the menuconfig.
- (3) Use command **\$ make all** to rebuild the projects of KM0 and KM4.

The MP image (km0_km4_app_mp.bin) will be generated in **{SDK}\amebadplus_gcc_project** and all the images related to the MP image can be found in the paths below:

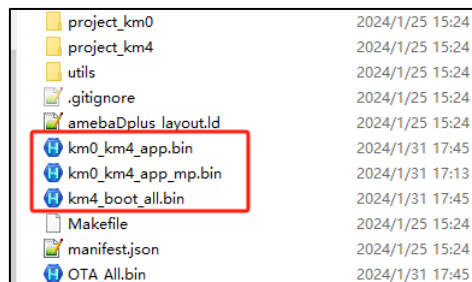
- {SDK}\amebadplus_gcc_project\project_km4\asdk\image_mp
- {SDK}\amebadplus_gcc_project\project_km0\asdk\image_mp

13.2 How to Combine Images

Before downloading images into the chip, the MP image needs to be combined with normal image. In massive production, the MP image is used to calibrate the parameters first. After calibration, you can use the command illustrated in Section 13.4 to switch to the normal image and boot from it. Since the chip boots from OTA1 by default when both OTA1 and OTA2 images are valid and with the same version number, the MP image should be located in the OTA1 field.

The steps of combining images are depicted below:

- (1) Check if both the normal image and MP image are valid.
- (2) Use the Image Tool to combine all the images, including km4_boot_all.bin, km0_km4_app_mp.bin and km0_km4_app.bin, which are located in **{SDK}\amebadplus_gcc_project**.



- a) Set the image offsets in Image Tool according to the Flash layout, which can be found in **{SDK}\component\soc\amebadplus\usrcfg\ameba_flashcfg.c**.

- ◆ Set km0_km4_app_mp.bin in **IMG_APP_OTA1** section
 - ◆ Set km0_km4_app.bin in **IMG_APP_OTA2** section
- b) Click **Generate** button in Image Tool, and save the combined image named Image_All.bin.

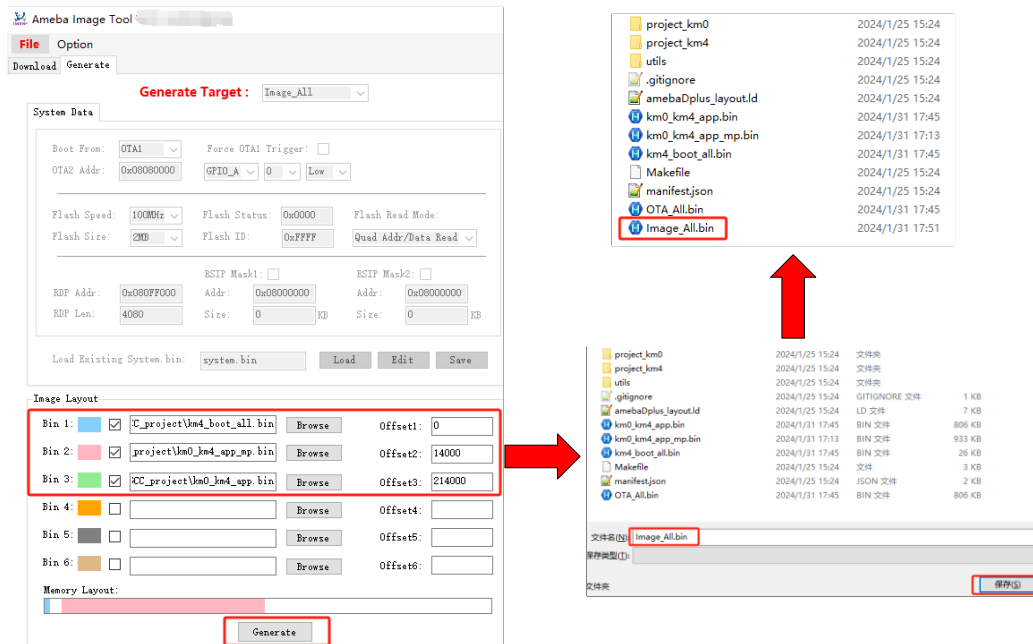
```

/*
 * @brief Flash layout is set according to Flash Layout in User Manual
 * In each entry, the first item is flash region type, the second item is start address, the second item is end address */
const FlashLayoutInfo_TypeDef Flash_Layout[] = {
    /*Region Type, [StartAddr, EndAddr] */
    {IMG_BOOT, 0x08000000, 0x08013FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K)
    //Users should modify below according to their own memory
    {IMG_APP_OTA1, 0x08014000, 0x081F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA1 + Manifest(4K) + RDP IMG OTA1
    {IMG_BOOT_OTA2, 0x08200000, 0x08213FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K) OTA
    {IMG_APP_OTA2, 0x08214000, 0x083F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA2 + Manifest(4K) + RDP IMG OTA2

    {FTL, 0x08700000, 0x08702FFF}, //FTL for BT(>=12K), The start offset of flash pages which is allocated to FTL physical map.
    {VFS1, 0x08703000, 0x08722FFF}, //VFS region 1 (128K)
    {USER, 0xFFFFFFFF, 0xFFFFFFFF}, //reserve for user

    /* End */
    {0xFF, 0xFFFFFFFF, 0xFFFFFFFF},
};

```



- (3) Download the Image_All.bin into the device after the combination is finished.

NOTE

The normal image (km0_km4_app.bin) is built with MP disabled. Check the normal image in {SDK}\amebadplus_gcc_project. All the images related to normal image can be found in paths below:

- {SDK}\amebadplus_gcc_project\project_km4\asdk\image
- {SDK}\amebadplus_gcc_project\project_km0\asdk\image

13.3 Boot Flow

- (1) Reset the device after downloading image is finished.
- (2) Check if the device boots from MP image successfully.

```
[MODULE_BOOT-LEVEL_INFO]:KM4 SRAM[20010000:1200]
[MODULE_BOOT-LEVEL_INFO]:KM4 PSRAM[0e045a20:20]
[MODULE_BOOT-LEVEL_INFO]:IMG2 BOOT from OTA 1
[MODULE_BOOT-LEVEL_INFO]:Start NonSecure @ 0xe000115 ...
[MODULE_BOOT-LEVEL_INFO]:VTOR: 20005000, VTOR_NS:0
[MODULE_BOOT-LEVEL_INFO]:KM4 APP START
[MODULE_BOOT-LEVEL_INFO]:IMG2 SEC[MODULE_URE_STATBOOT-LEVE: 0
[MODULE_BOOT-LEVEL_INFO]:BOOT-LEVEL UP
[MODULE_BOOT-LEVEL_INFO]:KM4 BOOT-LEVEL REASON: L_INFO]:0
[MODULE_BOOT-LEVEL_INFO]:KM4 BOOT-LEVEL UP
[MODULE_BOOT-LEVEL_INFO]:KM4 BOOT-LEVEL CPU CLKEL_INFO]: 260000:KM4 CPU000 Hz
CLK: 86666666 Hz
[MODULE_BOOT-LEVEL_INFO]:KM4 VTOR:0x20000000
[MODULE_PMC-LEVEL_INFO]:AP wake event 410 80000008
[MODULE_PMC-LEVEL_INFO]:NP wake event 804 41008308
[MODULE_BOOT-LEVEL_INFO]:KM4 OS START
[MODULE_BOOT-LEVEL_INFO]:delta:22 target:2441 PPM: 9012 PPM_Limit:30000
[MODULE_BOOT-LEVEL_INFO]:delta:0 target:320 PPM: 0 PPM_Limit:30000
[MODULE_BOOT-LEVEL_INFO]:KM4 MAIN
Register flash disk driver to Fatfs.
FATFS Register: disk driver 0
Flash drive path: 0:/
Test flash drive (file: flash.txt)

flash mount OK
[MODULE_BOOT-LEVEL_INFO]:File System Init Success
[MODULE_BOOT-LEVEL_INFO]:KM4 START SCHEDULER
interface 0 is initialized
interface 1 is initialized
interface 2 is initialized

Initializing WIFI ...
RTL8721F[Driver]: The driver is for MP
Firmware enable
```

- (3) Start the massive production flow if the device boots from MP image successfully.

13.4 How to Switch Image

When MP is finished, you need to switch the image from MP image to the normal image to verify the application.

- (1) Use command "ATSC" from serial terminal to clear the signature of MP image in order to assign the MP image invalid.

```
ATSC
[ATSC]: _AT_SYSTEM_CLEAR_OTA_SIGNATURE_
[sys_clear_ota_signature] IMGID: 1, current OTA1 Address: 0x00015000, target OTA2 Address: 0x00215000
[MEM] After do cmd, available heap 1310976
```

- (2) Reset the device, then the device will boot from the normal image located in OTA2 field.

```
[MODULE_BOOT-LEVEL_INFO]:KM4 XIP_IMG[0c000000:4c020]
[MODULE_BOOT-LEVEL_INFO]:KM4 SRAM[20040000:3960]
[MODULE_BOOT-LEVEL_INFO]:KM4 PSRAM[0c04f980:20]
[MODULE_BOOT-LEVEL_INFO]:KM4 BOOT[20004d00:40]
[MODULE_BOOT-LEVEL_INFO]:KM4 XIP_IMG[0e000000:43200]
[MODULE_BOOT-LEVEL_INFO]:KM4 SRAM[20010000:11e0]
[MODULE_BOOT-LEVEL_INFO]:KM4 PSRAM[0e0443e0:20]
[MODULE_BOOT-LEVEL_INFO]:IMG2 BOOT from OTA 2
[MODULE_BOOT-LEVEL_INFO]:Start NonSecure @ 0xe000115 ...
[MODULE_BOOT-LEVEL_INFO]:VTOR: 20005000, VTOR_NS:0
[MODULE_BOOT-LEVEL_INFO]:KM4 APP[MODU START
LE_BOOT-[MODULELEVEL_IN_BOOT-LEF0]:KM4 VEL_INFOBOOT UP ]:IMG2 S
```

14 Virtual File System

14.1 Introduction

This section introduces how to run Virtual File System (VFS) on RTL8721Dx, including FatFS and LittleFS as the underlying implementation. Users can ignore the differences between different file systems by using VFS. The VFS provides common file operation interfaces like fopen, fclose, fwrite, fread, etc. The Key-Value (KV) interfaces are based on these common file operations. The architecture of VFS is illustrated in Figure 14-1.

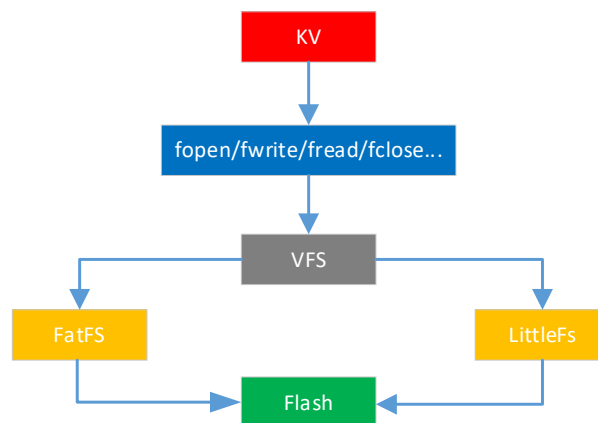


Figure 14-1 Architecture of VFS

14.2 VFS Initialization

By default, the VFS has been initialized in **main.c** as follows:

```

void app_filesystem_init(void)
{
    int ret = 0;
    vfs_init();
    ret = vfs_user_register("lfs", VFS_LITTLEFS, VFS_INF_FLASH, VFS_FLASH_R1, VFS_RW);
    if (ret == 0) {
        ret = rt_kv_init();
        if (ret == 0) {
            RTK_LOGI(TAG, "File System Init Success\n");
            return;
        }
    }
    RTK_LOGE(TAG, "File System Init Fail \n");
    return;
}
  
```

The `vfs_user_register()` API will mount VFS to the Flash by users' configuration. If failed to mount, this API will check whether the Flash is clean (0xFF). And if the Flash is clean, it will program the Flash to initialize VFS.

```

int vfs_user_register(const char *prefix, int vfs_type, int interface, char region,
char flag)
  
```

Where:

- `prefix`: defined by users, used to distinguish different file systems
- `vfs_type`: file system type of the underlying implementation (FatFS or LittleFS)
- `interface`: memory type (Flash only)
- `region`: Flash partition (VFS1 or USER, described in section 14.3.1)
- `flag`: operation authority of file system (read-write or read-only)

i NOTE

vfs_type is set to LittleFS by default for read-write balance and power failure protection.

14.3 Usage of VFS

14.3.1 VFS on Flash

Adjust the Flash partitions appropriately if the VFS interfaces are set to the Flash, and modify VFS1 in `Flash_Layout[]` in `{SDK}\component\soc\amebadplus\usrcfg\ameba_flashcfg.c`.

```
const FlashLayoutInfo_TypeDef Flash_Layout[] = {
    /*Region_Type, [StartAddr, EndAddr] */
    {IMG_BOOT, 0x08000000, 0x08013FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K)
    //Users should modify below according to their own memory
    {IMG_APP_OTA1, 0x08014000, 0x081F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA1 + Manifest(4K) + RDP IMG OTA1

    {IMG_BOOT_OTA2, 0x08200000, 0x08213FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K) OTA
    {IMG_APP_OTA2, 0x08214000, 0x083F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA2 + Manifest(4K) + RDP IMG OTA2

    {FTL, 0x08700000, 0x08702FFF}, //FTL for BT(>=12K), The start offset of flash pages which is allocated to FTL physical map.
    {VFS1, 0x08703000, 0x08722FFF}, //VFS region 1 (128K)
    {USER, 0xFFFFFFFF, 0xFFFFFFFF}, //Reserved for users

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

NOTE

The VFS1 region must exist, and its size should always be larger than 128KB.

14.3.2 Common File Operation

The common file operation interfaces used in VFS are listed below:

API	Parameter	Description
fopen	- const char *filename - const char *mode	Open the filename pointed to, by filename using the given mode
fclose	FILE *stream	Close the stream
fread	- void *ptr - size_t size - size_t count - FILE *stream	Read data from the given stream by ptr into the array pointed to
fwrite	- const void *ptr - size_t size - size_t count - FILE *stream	Write data from the array pointed to by ptr to the given stream
fseek	- FILE *stream - long int offset - int origin	Set the file position of the stream to the given offset
rewind	FILE *stream	Set the file position to the beginning of the file of the given stream
fgetpos	- FILE *stream - fpos_t *p	Get the current file position of the stream and writes it to pos
fsetpos	- FILE *stream - fpos_t *p	Set the file position of the given stream to the given position
fflush	FILE *stream	Flush the output buffer of a stream
remove	const char *filename	Delete the given filename so that it is no longer accessible
rename	- const char *oldname - const char *newname	Cause the filename referred to from old_filename to new_filename
feof	FILE *stream	Test the end-of-file indicator for the given stream
ferror	FILE *stream	Test the error indicator for the given stream
ftell	FILE *stream	Return the current file position of the given stream
ftruncate	- FILE *stream - off_t length	Truncate a file to a specified length
opendir	const char *name	Open a directory
readdir	DIR *pdir	Read a directory
closedir	DIR *dirp	Close a directory
rmdir	const char *path	Remove a directory

mkdir	- const char *pathname - mode_t mode	Make a directory
access	- const char *pathname - int mode	Determine accessibility of a file
stat	- const char *path - struct stat *buf	Get file status

Users can rebuild the project by "make all EXAMPLE=vfs" to test how common file operations work. Test logs should be like below:

```
[example_vfs_thread] fwrite success!!!
[example_vfs_thread] fread success!!!
```

NOTE

There are some interfaces whose return value is different from standard interfaces. If successful, fwrite/fread returns 1 and fseek returns offset according to the beginning of file.

14.3.3 Key-Value Operation

Simple KV interfaces are also provided for users. All KV APIs are placed in {SDK}\component\file_system\kv\kv.c. Users can rebuild the project by "make all EXAMPLE=kv" to test how KV APIs work. Test logs should be like below:

```
rt_kv_set success, write 28 letters.
rt_kv_get success, read 28 letters.
rt_kv_delett success.
```

14.3.4 Code Conversion

The conversion between Unicode and other codes is not supported on FatFS by default.

Modify the macro FF_CODE_PAGE in {SDK}\component\file_system\fatfs\r0.14b\include\ffconf.h to enable the code conversion function, where FF_CODE_PAGE should be chosen as code page number which is desired.

```
#define FF_CODE_PAGE 999
/* This option specifies the OEM code page to be used on the target system.
/ Incorrect code page setting can cause a file open failure.
/ 437 - U.S.
/ 720 - Arabic
/ 737 - Greek
/ 771 - KBL
/ 775 - Baltic
/ 850 - Latin 1
/ 852 - Latin 2
/ 855 - Cyrillic
/ 857 - Turkish
/ 860 - Portuguese
/ 861 - Icelandic
/ 862 - Hebrew
/ 863 - Canadian French
/ 864 - Arabic
/ 865 - Nordic
/ 866 - Russian
/ 869 - Greek 2
/ 932 - Japanese (DBCS)
/ 936 - Simplified Chinese (DBCS)
/ 949 - Korean (DBCS)
/ 950 - Traditional Chinese (DBCS)
/ 999 - Realtek defined for code size
/ 0 - Include all code pages above and configured by f_setcp()
*/
```

14.3.5 VFS Bin File Generation

If data needs to be placed in the Flash in advance, VFS bin file can be generated on PC. After generating the bin file, it should be downloaded to VF51 region according to the Flash layout.

14.3.5.1 LittleFS Bin File Generation

- (1) Prepare a needed object folder including files before generating LittleFS bin files. For example:

```
root:~$ ls -R test
test:
AUDIO  KV

test/AUDIO:
AUDIO1

test/KV:
TEST1
```

AUDIO and KV directories will be LittleFS directory in the Flash.

- (2) Use the command `./mklittlefs -b 4096 -p 256 -c test image_littlefs.bin` in `mklittlefs` tool located at `\tools\littlefs` to generate LittleFS bin files.

Where:

- `-b`: block size decided by Flash
- `-p`: page size
- `-s`: bin file size
- `-c`: object folder
- `<Image_littlefs.bin>`: LittleFS bin file name

```
root:~$ ./mklittlefs -b 4096 -p 256 -s 0x20000 -c test image_littlefs.bin
/AUDIO/AUDIO1
/KV/TEST1
```

NOTE

"-b 4096" and "-p 256" are default configurations, users should adapt the configuration according to "block_size" and "cache_size" of `lfs_config` in `{SDK}\component\file_system\littlefs\littlefs_adapter.c`. "-s 0x20000" is according to VFS1 region mentioned in section 14.3.1.

- (3) Download the image to the Flash.
The start address of image should be VFS1 Flash region address mentioned in section 14.3.1. Test logs are shown below:

```
=====mklittlefs example=====
[TEST1]: This is a test file for mklittle ...
[AUDIO1]: Copyright (c) 2013 Realtek ...
```

14.3.5.2 FatFS Bin File Generation

The steps to generate FatFS bin files are listed below:

- (1) Use command `root@ubuntu # dd if=/dev/zero of=test.bin count=64 bs=1KB` to create `test.bin` that has 64 blocks and each block is 1KB.
- (2) Use command `root@ubuntu # mkfs.fat -S 512-F 12 test.bin` to build a FAT file system.
- (3) Use command `root@ubuntu # sudo mount test.bin ./fs` to mount `test.bin` to file folder `fs`.
- (4) Use command `root@ubuntu # sudo cp hello.txt ./fs` to copy the files that users want to store into `test.bin`.
In this step, `hello.txt` is stored in `test.bin`.
- (5) Use command `root@ubuntu # sudo umount ./fs` to generate the FatFS file after unmounting `test.bin`.

Users should find other related information from the internet, and copy `test.bin` into user data area of Flash finally.

15 OTA Firmware Update

15.1 Introduction

Over-the-air (OTA) programming provides a methodology of updating device firmware remotely via TCP/IP network. For OTA via TCP/IP network, the RTL8721Dx provides solutions to implement OTA firmware upgrade from local server or cloud.

15.1.1 Image Slot

There are two slots for all the images in the Flash layout as shown in [Figure 15-1](#), which named OTA1 and OTA2 respectively. Each image can be chosen to boot from OTA1 or OTA2.

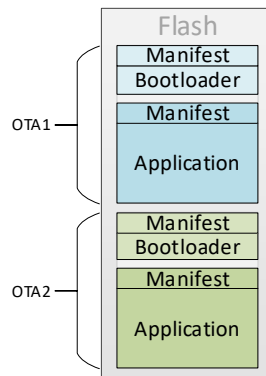


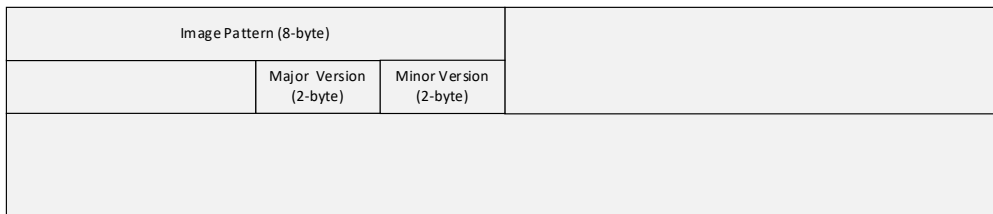
Figure 15-1 OTA1 and OTA2 position

15.1.2 Version Number

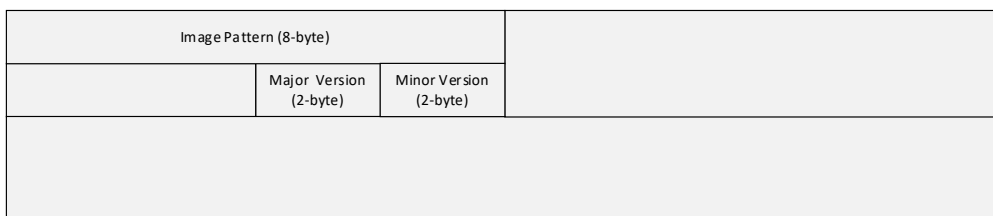
The device boot from OTA1 or OTA2 mainly depends on the version number in certificate and manifest. As shown in [Figure 15-2](#), there is a 2-byte major version and 2-byte minor version in manifest and certificate.

The combination of major version and minor version is the 4-byte version number. OTA select flow checks the whole version number.

Version number = (Major version << 16) | Minor version



(a) Certificate



(b) Manifest

Figure 15-2 Major and minor version

i NOTE

For bootloader, version number can be 0 to 32767; for application, version number can be 0 to 65535.

As described in [15.1.1](#), there are two slots (OTA1 and OTA2) for all the images in the Flash layout. When reboot after OTA upgrade finished, the device would check the image to determine to boot from OTA1 or OTA2.

The general principle of the OTA scheme is checking the image pattern first and then comparing the version number of OTA1 and OTA2.

The following items must be checked for each image:

- (1) Check the image pattern for both OTA1 and OTA2.
 - If only one image pattern is valid, boot from the valid image.
 - If both the image pattern are invalid, boot fail.
- (2) Compare the version number of OTA1 and OTA2.
 - If both OTA1 and OTA2 are valid and with different version numbers, the device will boot from the image with a bigger version.
 - If both OTA1 and OTA2 are valid but with the same version number, the device will boot from OTA1.

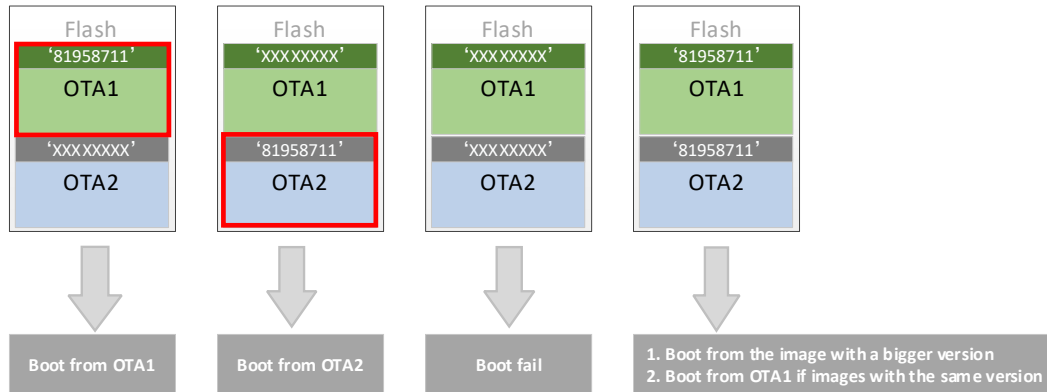


Figure 15-3 OTA select diagram

15.1.3 Anti-rollback

Anti-rollback is the function to prevent version rollback attack. When the anti-rollback is enabled, the version number in certificate or manifest must not be smaller than the anti-rollback version stored in OTP. Otherwise, this image will be regarded as invalid and the chip will not boot from invalid image. Normally, if OTA update is security-related, user can program a bigger anti-rollback version number in OTP and update image with a bigger major version at the same time to prevent rollback attack.

The anti-rollback flow is shown in Figure 15-4. Once the anti-rollback is enabled, the device will compare the major version numbers got from OTA1 and OTA2 images respectively with the anti-rollback version number in OTP. If the major version number in the image is smaller than the anti-rollback version number, this image will be regarded as invalid.

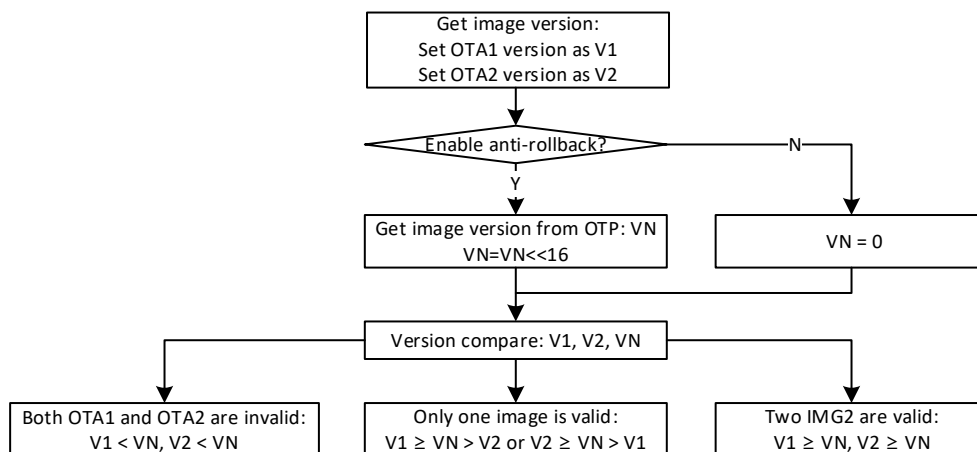


Figure 15-4 Anti-rollback flow

15.2 Bootloader

15.2.1 OTA Image

The KM4 bootloader image named km4_boot_all.bin can be updated through OTA, which can be chosen to boot from OTA1 or OTA2. The layout of KM4 bootloader image is illustrated in Figure 15-5.

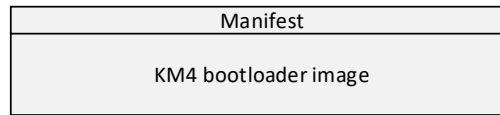


Figure 15-5 Layout of KM4 bootloader image

15.2.2 OTA Select Flow

The KM4 ROM will select OTA image according to the image version number in bootloader manifest.

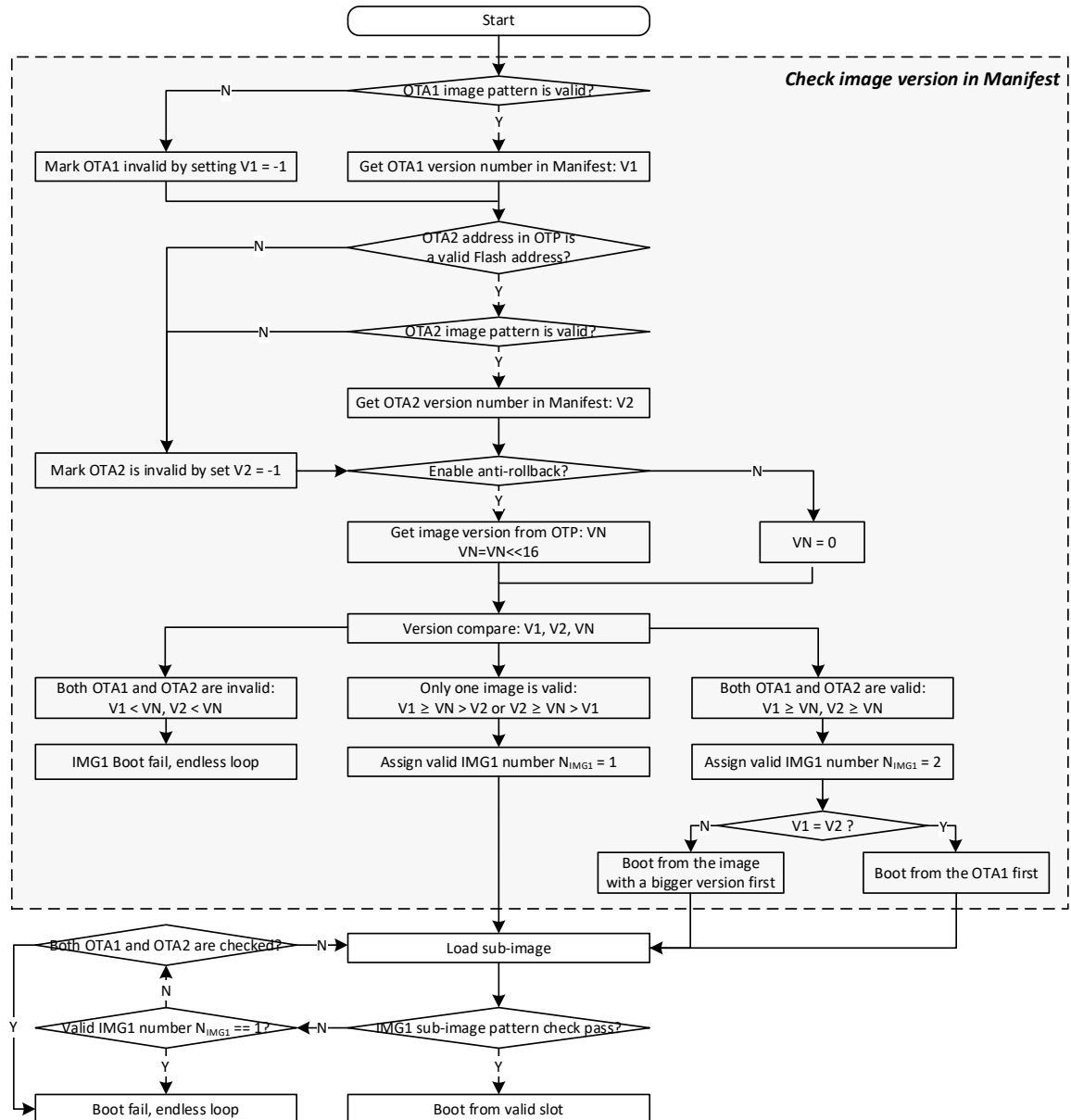


Figure 15-6 KM4 bootloader OTA select flow

15.3 Application

15.3.1 OTA Image

The application image named km0_km4_app.bin, including KM0, KM4 non-secure application image and KM4 secure image, can be updated through OTA, which can be chosen to boot from OTA1 or OTA2. The layout of the whole application image is illustrated in [Figure 15-7](#).



Figure 15-7 Layout of application image

15.3.2 OTA Select Flow

The application image OTA select flow is illustrated in [Figure 15-8](#).

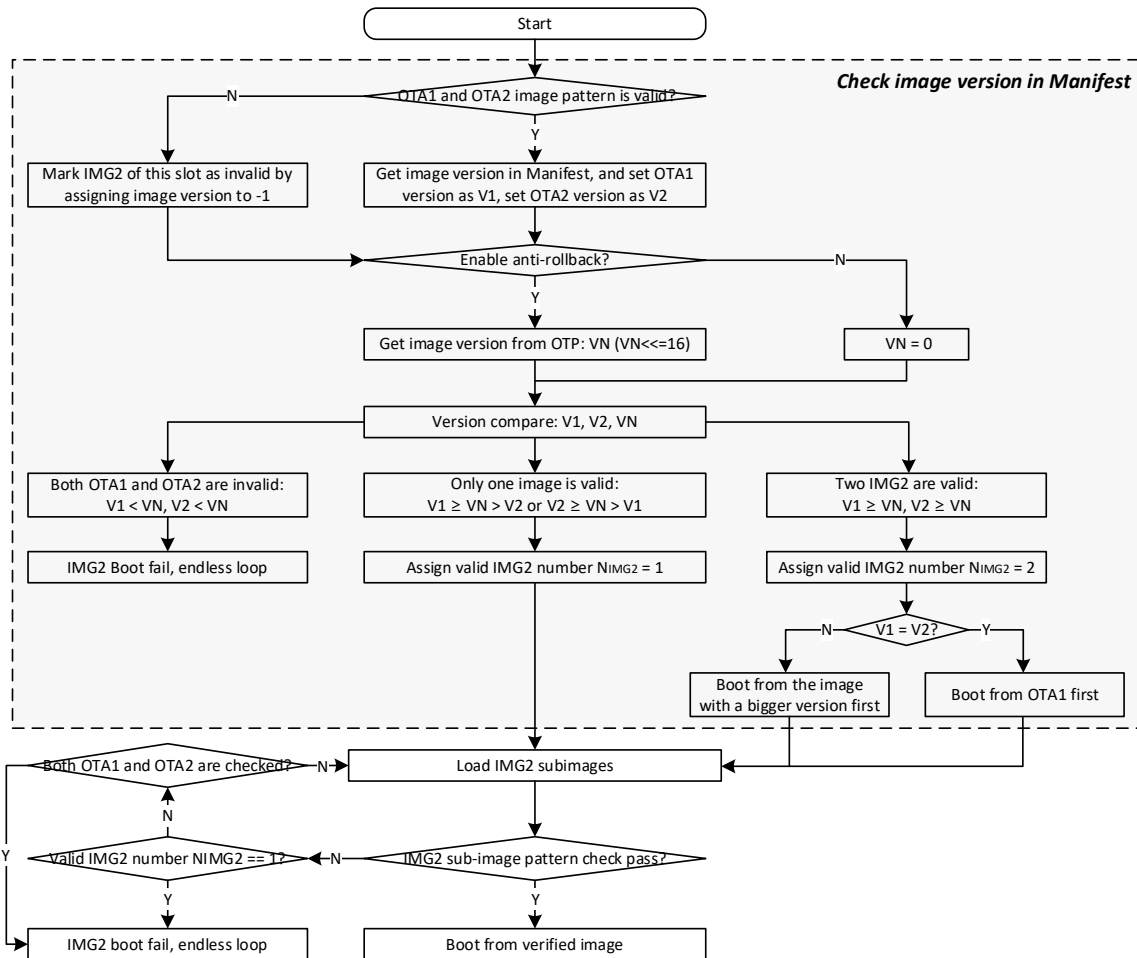


Figure 15-8 Application image OTA select flow

15.4 Build OTA Image

15.4.1 Modify Configurations

- (1) Modify the version number in configuration file: **manifest.json**.

File	Tag	Description	Path
manifest.json	boot	Configure major and minor version for KM4 bootloader	{SDK}\amebadplus_gcc_project
	app	Configure major and minor version for certificate and application	

- a) Modify the version number for bootloader.

```
"boot":
{
  "IMG_ID": "0",
  "IMG_VER_MAJOR": 1,
  "IMG_VER_MINOR": 1,
  "SEC_EPOCH": 1,
  "HASH_ALG": "sha256",
  "RSIP_IV": "01020304050607080000000000000000"
},
```

- b) Modify the version number for certificate and application.

```
"app":
{
  "IMG_ID": "1",
  "IMG_VER_MAJOR": 1,
  "IMG_VER_MINOR": 1,
  "SEC_EPOCH": 1,
  "HASH_ALG": "sha256",
  "RSIP_IV": "213253647586a7b80000000000000000"
},
```

- (2) Change the bootloader version of anti-rollback and enable anti-rollback if necessary.

- a) Change the bootloader version of anti-rollback

By default, all images use the same anti-rollback version in OTP as threshold to prevent anti-rollback attack.

Name	OTP address	Length	Description
BOOTLOADER_VERSION	Physical 0x36E~0x36F	16 bits	The bootloader version of anti-rollback

The bootloader version of anti-rollback is 0 by default. Users can change the number of '0' bit to enlarge the bootloader version. For example, users can program the bootloader version of anti-rollback to 1 by the following command:

```
EFUSE wraw 36E 2 FFFE
```

- b) Enable anti-rollback

Users can program OTP by the following command to enable anti-rollback.

```
EFUSE wraw 368 1 BF
```

NOTE

- Once anti-rollback is enabled, it cannot be disabled.
- If bootloader and application do not use the same anti-rollback version, modify `BOOT_OTA_GetCertRollbackVer()` in `{SDK}\component\soc\amebadplus\bootloader\boot_ota_km4.c` and define another anti-rollback version in OTP for the application.

- (3) Write the bootloader OTA2 address into OTP if users need to upgrade the bootloader, which sets the bootloader OTA2 address to 0x08260000.

```
EFUSE wraw 36C 2 6082
```

NOTE

- The address of bootloader OTA2 is the value of OTP 0x36C with 12-bit left shifted, or is the value of OTP 0x36C * 4K.
- If the address of bootloader OTA2 is 0xFFFFFFFF by default, the bootloader won't be upgraded when in OTA upgrade and the device always boots from bootloader OTA1.
- The above commands are used in the serial terminal tool.

- (4) Rebuild the project using "make all" command to generate the signed images.

- (5) Download the images into Flash, and reset the board.

```
[MODULE_BOOT-LEVEL_INFO]:KM0 PSRAM[0c051b60:20]
[MODULE_BOOT-LEVEL_INFO]:KM0 BOOT[20004d00:40]
[MODULE_BOOT-LEVEL_INFO]:KM4 XIP IMG[0e000000:5ad80]
[MODULE_BOOT-LEVEL_INFO]:KM4 SRAM[20010000:13c0]
[MODULE_BOOT-LEVEL_INFO]:KM4 PSRAM[0e05c140:20]
[MODULE_BOOT-LEVEL_INFO]:IMG2 BOOT from OTA 1
[MODULE_BOOT-LEVEL_INFO]:Start NonSecure @ 0xe001715 ...
```

15.4.2 Generate OTA Image Automatically

Users can modify the makefile to generate OTA image automatically both in KM0 and KM4 projects. The makefile is located at `{SDK}\amebadplus_gcc_project\project_km4\asdk`.

- (1) `km0_km4_app.bin` is included in `OTA_All.bin` by default.

```
@if [ -f $(IMAGE_TARGET_FOLDER)/km0_km4_app.bin ]; then \
  $(OTAPREPENDTOOL) $(IMAGE_TARGET_FOLDER)/km0_km4_app.bin; \
fi
```

- (2) Add `km4_boot_all.bin` behind `km0_km4_app.bin` if the bootloader is needed to be upgraded.

- (3) Rebuild the project by command "make all" under `{SDK}\amebadplus_gcc_project`. The OTA image file called `OTA_All.bin` will be generated in the same directory.

NOTE

- `km4_boot_all.bin` must be added behind `km0_km4_app.bin`.
- The address of bootloader OTA2 must be set in OTP as described in 15.4.1 step (3) if users need to upgrade the bootloader.

15.5 Update from Local Server

This section introduces the design principles and usage of OTA from local server. It has well-transportability to porting to OTA applications from cloud.

The OTA from local server shows how the device updates the image from a local download server. The local download server sends the image to the device based on the network socket, as Figure 15-9 shows.

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

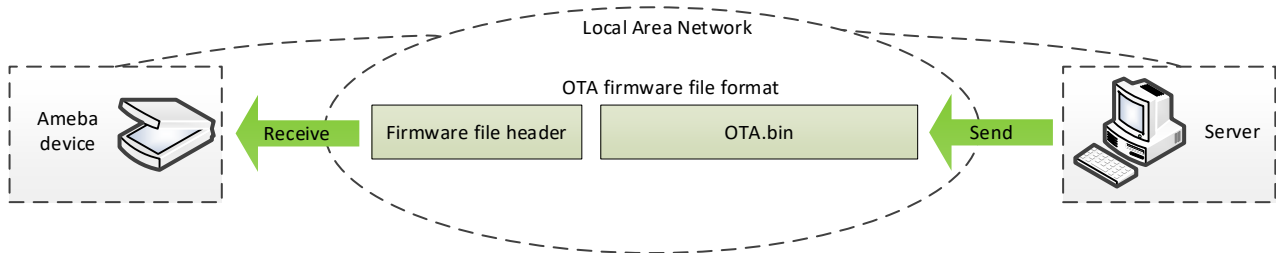


Figure 15-9 OTA update diagram via network

15.5.1 Firmware Format

The firmware format is illustrated in Figure 15-10.

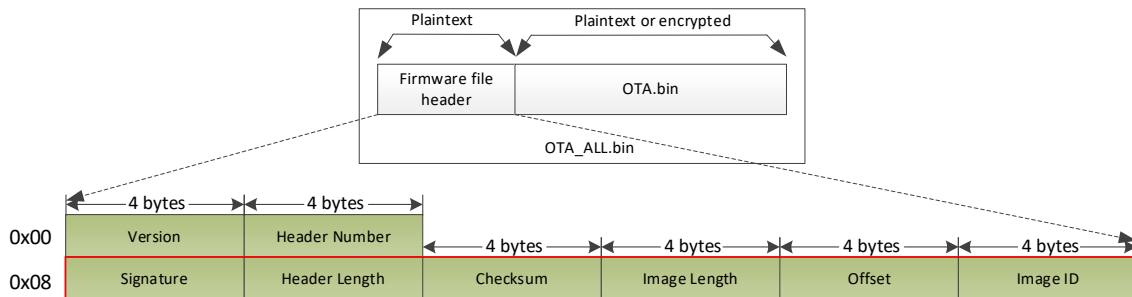


Figure 15-10 Firmware format

Table 15-1 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 RTL8721Dx, this value can be 1, 2
Signature	0x08	4 bytes	OTA Signature is string. For RTL8721Dx, this value is "OTA"
Header Length	0x0C	4 bytes	The length of OTA header. For RTL8721Dx, 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
Image ID	0x1C	4 bytes	The image ID of current image ● OTA_IMGID_BOOT: 0x0 ● OTA_IMGID_APP: 0x1

15.5.2 OTA Flow

The OTA demo locates in {SDK}\component\soc\amebadplus\misc\ameba_ota.c. The image upgrade is implemented in the following steps:

- (1) Connect to the server. The IP address, port and OTA type are needed.
- (2) Acquire the older firmware address to be upgraded according to the MMU setting. If the address is re-mapping to OTA1 space by MMU, the OTA2 address would be selected to upgrade. Otherwise, the OTA1 address would be selected.
- (3) Receive the firmware file header to get the target OTA image information, such as image number, image length and image ID.
- (4) Download the new firmware from server.
- (5) Erase the Flash space for new firmware and write it into Flash except Manifest structure.
- (6) Verify the checksum. If the checksum is error, OTA fails.
- (7) If the checksum is ok, write Manifest structure to the upgraded firmware region to indicate boot from a new firmware next time.
- (8) OTA is finished and reset the device. Then it would boot from the new firmware.

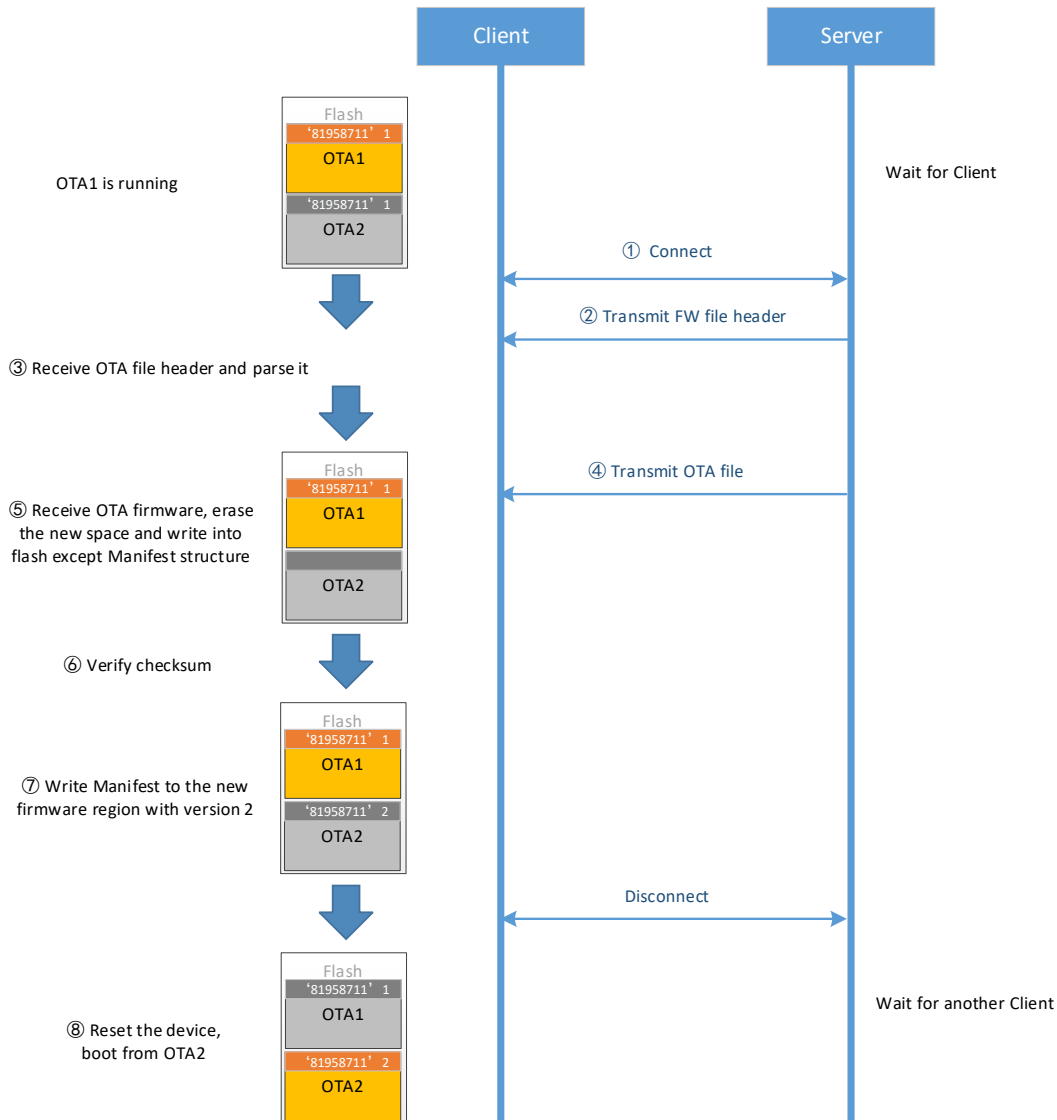


Figure 15-11 OTA operation flow

15.6 Run OTA Demo

Follow these steps to run the OTA demo to update from local server:

- (1) Edit {SDK}\component\example\ota\example_ota.c.
 - a) Edit the host according to the server IP address.

```
#define PORT 8082
static const char *host = "192.168.31.193"; // "m-apps.oss-cn-shenzhen.aliyuncs.com"
static const char *resource = "OTA_All.bin"; // "051103061600.bin"
```

- b) Edit the OTA type to OTA_LOCAL.

```
ret = ota_update_init(ctx, (char *)host, PORT, (char *)resource, OTA_LOCAL);
```

- (2) Rebuild the project with the command "make all EXAMPLE=ota" and download the images to the device.
 (3) Modify the major and minor version number in Manifest to a bigger version as described in Section 15.1.2.

NOTE

The bootloader will select OTA image with a bigger version number by default. If users don't want to modify the version number, modify `OTA_CLEAR_PATTERN` to 1 defined in `ameba_ota.h` before step (2). It should only be used in the development stage.

- (4) Rebuild the project and copy **OTA_All.bin** into the folder `{SDK}\tools\DownloadServer`.
 (5) Edit `{SDK}\tools\DownloadServer\start.bat`.

c) port = 8082

d) file name = OTA_All.bin

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

- (6) Click the **start.bat**, and start the download server program.
 (7) Reboot the DUT and connect the device to the AP which the OTA Server in.
 (8) Reboot DUT to execute the new firmware after finishing image download.

15.7 OTA Firmware Swap

The Figure 15-12 shows the firmware swap procedure after OTA upgrade.

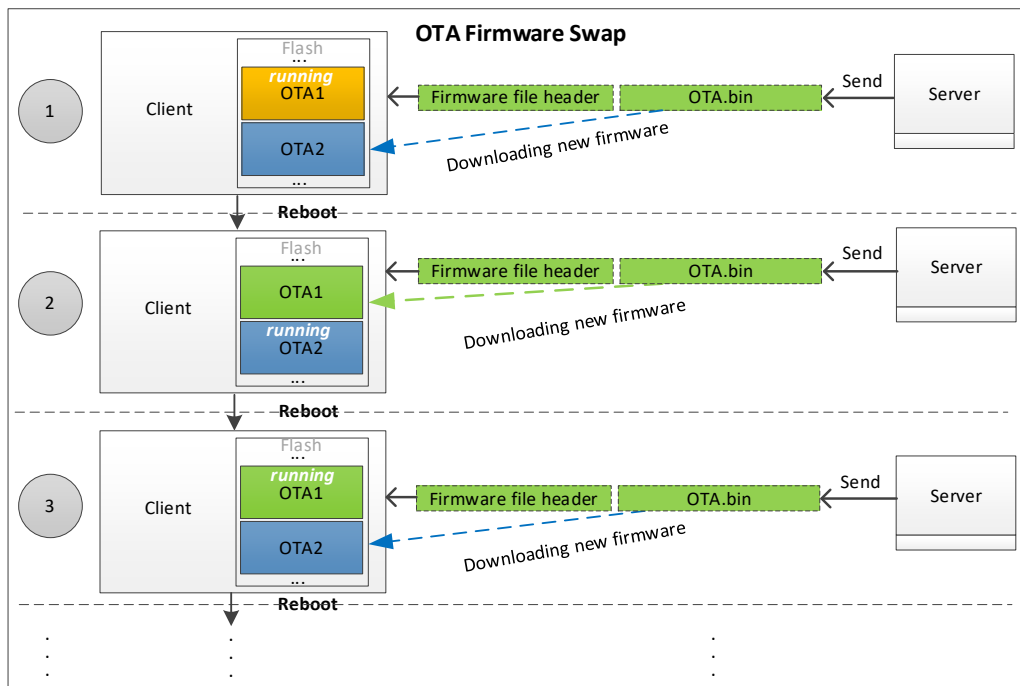


Figure 15-12 OTA firmware swap procedure

15.8 User Configuration

Modify the memory layout in `{SDK}\component\soc\amebadplus\usrcfg\ameba_flashcfg.c` if needed.

```

/*
 * @brif Flash layout is set according to Flash Layout in User Manual
 * In each entry, the first item is flash region type, the second item is start address, the second item is end address */
const FlashLayoutInfo_TypeDef Flash_Layout[] = {
    /*Region_Type, [StartAddr, EndAddr] */
    {IMG_BOOT, 0x08000000, 0x08013FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K)
    //Users should modify below according to their own memory
    {IMG_APP_OTA1, 0x08014000, 0x081F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA1 + Manifest(4K) + RDP IMG OTA1
    {IMG_BOOT_OTA2, 0x08200000, 0x08213FFF}, //Boot Manifest(4K) + KM4 Bootloader(76K) OTA
    {IMG_APP_OTA2, 0x08214000, 0x083F3FFF}, //Certificate(4K) + Manifest(4K) + KM4 Application OTA2 + Manifest(4K) + RDP IMG OTA2

    {FTL, 0x08700000, 0x08702FFF}, //FTL for BT(>=12K), The start offset of flash pages which is allocated to FTL physical map.
    {VFS1, 0x08703000, 0x08722FFF}, //VFS region 1 (128K)
    {USER, 0xFFFFFFFF, 0xFFFFFFFF}, //reserve for user

    /* End */
    {0xFF, 0xFFFFFFFF, 0xFFFFFFFF},
};

```

Figure 15-13 Flash layout

16 Boot Process

16.1 Features

- On-chip boot ROM
- Contains the bootloader with In-System Programming (ISP) facility
- Secure boot process with multiple cryptographic algorithms of hardware or software engine
- Suspend resume process
- Boot from NOR flash
- PSRAM as a memory

16.2 Boot Address

After reset, CPU will boot from the vector table start address, which is fixed by hardware. Both KM4 and KM0 boot from address 0x0000_0000.

Table 16-1 Boot address

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

16.3 Pin Description

The RTL8721Dx supports ISP via LOGUART (PB4 & PB5). The ISP mode, given in [Table 16-2](#), is determined by the state of PB5 when boot.

Table 16-2 ISP mode

Boot Mode	PB5 (UART_DOWNLOAD)	Description
No ISP	HIGH	ISP bypassed. Part attempts to boot from Flash.
ISP	LOW	Part enters ISP via LOGUART.

16.4 Boot Flow

The boot flow of RTL8721Dx is illustrated in [Figure 16-1](#). After a power-up or hardware reset, the hardware will boot KM4 at 150MHz. The boot process is handled by the on-chip boot ROM and is always executed by the KM4. After the KM4 bootloader code, the KM4 will set up the environment for the KM0.

- KM4 boots ROM
- KM4 secure boot (optional)
- KM4 boots to SRAM
- KM4 helps KM0 load images and check the signature (optional)

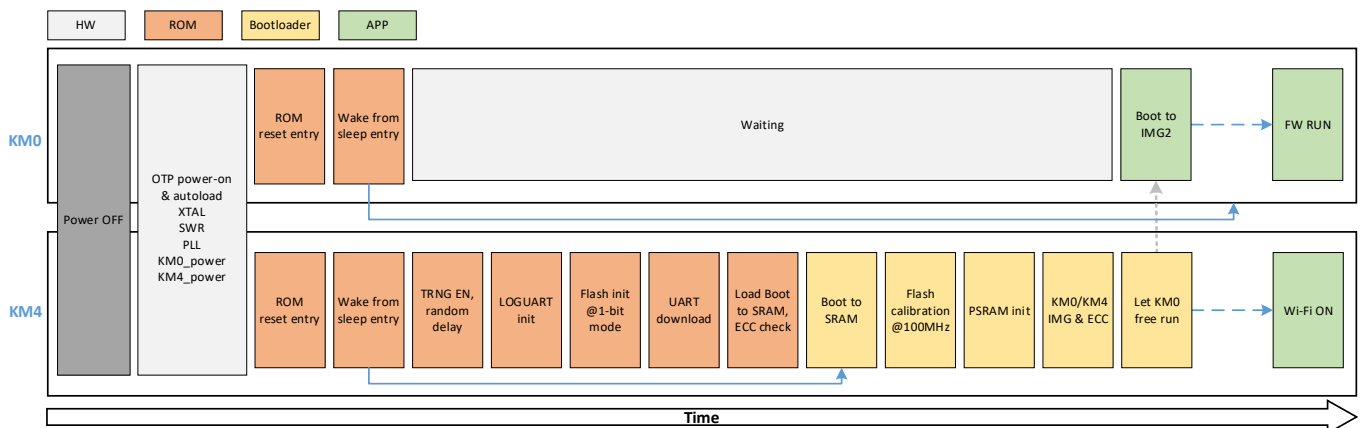


Figure 16-1 Boot flow

The immutable ROM provides ISP service, which could be initial programming of a blank device, erasing and re-programming of a previously

programmed device.

When the rising edge on RESET pin is generated, the trap pin (PB5) will be sampled to determine whether to continue the boot process or ISP service. If the trap pin is sampled LOW, the external hardware requests to start the ISP service. Otherwise, the chip boots normally.

16.5 Boot API

The boot API is used to obtain the cause of the chip boot, and the function prototype is:

```
u32 BOOT_Reason(void);
```

The default return value of this API is 0 when initially powered on, and return value of re-boot caused by other reasons can be found in the following table. Users can find macro-definitions about return value in file **sysreg_aon.h**.

Items	Description
Introduction	Get boot reason
Parameters	None
Return	Boot reason. It can be any of the following values or combinations: ● AON_BIT_RSTF_OCP: entering deep-sleep mode when OCP happens ● AON_BIT_RSTF_KM0_SYS: KM0 system reset ● AON_BIT_RSTF_KM4_SYS: KM4 system reset ● AON_BIT_RSTF_IWDG: KM0 Independent watchdog reset ● AON_BIT_RSTF_WDG0: KM0 watchdog reset ● AON_BIT_RSTF_WDG1: KM4 secure watchdog reset ● AON_BIT_RSTF_WDG2: KM4 non-secure watchdog reset ● AON_BIT_RSTF_WARM_KM42PERI: KM4 warm reset ● AON_BIT_RSTF_WARM_KM02PERI: KM0 warm reset ● AON_BIT_RSTF_DSLP: Wakeup from deep-sleep mode ● AON_BIT_RSTF_BOR: BOR Reset

17 SoC Clock Modification

17.1 Introduction

The KM4 in RTL8721Dx device boots at 200MHz at the BootRom Stage, and switches to a higher frequency during the BootLoader Stage. There are some limits when changing SoC clock:

Clock	Cut	Frequency	Core voltage
PLL		300MHz ~ 600MHz	-
KM0	A-Cut	≤115MHz	-
KM4	A-Cut	≤260MHz	0.9V
		≤345MHz	1.0V

NOTE

The maximum operating speed of Flash with Wide Range VCC 1.65V~3.6V should use the speed limit of 1.65V~2.3V power supply.

17.2 SoC Clock Switch

17.2.1 Flow

- (Optional) Find out the speed limit of PSRAM device embedded in RTL8721Dx if not sure.
 - Print the value of `ChipInfo_BDNum()` function, which will get the chip info from OTP.
 - Refer to PSRAM type in `Chip_Info[]` in `{SDK}\component\soc\amebadplus\lib\ram_common\ameba_chipinfo_lib.c`.
For Now, This step can be skipped because wb955 is only used.
- Check the value of `Boot_SocClk_Info_Idx` and `SocClk_Info[]` in `{SDK}\component\soc\amebadplus\usrfcg\ameba_bootcfg.c`.

```
// for km4, max 330MHz under 1.0v, max 260MHz under 0.9v
// for km0, max 100MHz under all core power
// PLL can be 300MHz~600MHz
// KM4_CKD range is [1, 8], KM0_CKD range is [1, 16] or USEXTAL
BOOT_RAM_DATA_SECTION
SocClk_Info_TypeDef SocClk_Info[] = {
    /* PLL_CLK, Vol_Type, KM4_CKD, KM0_CKD, PSRAM_CKD */
    {PLL_520M, CORE_VOL_0P9, CLKDIV(2), CLKDIV(6), CLKDIV(2)},
    {PLL_330M, CORE_VOL_1P0, CLKDIV(1), CLKDIV(4), CLKDIV(1)},
    {PLL_400M, CORE_VOL_0P9, CLKDIV(2), CLKDIV(10), CLKDIV(1)},
    {PLL_600M, CORE_VOL_0P9, CLKDIV(3), CLKDIV(15), CLKDIV(2)},
};

/**
 * @brief SocClk_Info select
 * Boot_SocClk_Info_Idx valid value is [0, 3] and 0xFF
 * when Boot_SocClk_Info_Idx is 0xFF, set socclk by chipinfo Automatically
 * when Boot_SocClk_Info_Idx is [0, 3], set socclk by SocClk_Info[Boot_SocClk_Info_Idx]
 */
BOOT_RAM_DATA_SECTION
u8 Boot_SocClk_Info_Idx = 0;
```

- Check the `BOOT_ChipInfo_ClkInfoIdx()` function in `{SDK}\component\soc\amebadplus\bootloader\bootloader_km4.c`.

```
BOOT_RAM_TEXT_SECTION
u32 BOOT_ChipInfo_ClkInfoIdx(void)
{
    /*psram die is wb955 which can run up to 200MHz*/

    /*Boot_SocClk_Info_Idx is valid, use user config socclk*/
    return Boot_SocClk_Info_Idx;
}
```

No Limitation by PSRAM Device, so BootLoader will set the SoC clock defined by `SocClk_Info[Boot_SocClk_Info_Idx]`.

PSRAM type	PSRAM speed	SocClk_Info[x]	Description	Clock Info
No PSRAM	-	Boot_SocClk_Info_Idx = 0;	BootLoader will set the Soc clock according to <code>SocClk_Info[0]</code>	PLL: 520MHz KM4: 260MHz

WB955	<=200M	Boot_SocClk_Info_Idx = 1;	BootLoader will set the Soc clock according to SocClk_Info[1]	KM0: 86.6MHz PLL: 330MHz KM4: PLL/1 KM0: PLL/4 PSRAM: PLL/1/2
-------	--------	---------------------------	---	---

- (4) Refer to one of the following methods to change the SoC clock if needed.
- Modify SocClk_Info[0] in {SDK}\component\soc\amebadplus\usrcfg\ameba_bootcfg.c, refer to 17.2.2 step (2) for details.
 - Modify Boot_SocClk_Info_Idx to [0, sizeof(SocClk_Info)), and then define your own clock info in SocClk_Info [Boot_SocClk_Info_Idx].

NOTE

Consider the limitations of the hardware and do not set the clock info illogically.

- (5) Rebuild the project and download the new image again.

17.2.2 Example

- (1) Refer to 17.2.1 step (1) to find out the speed limit of PSRAM device if not sure (suppose the maximum speed is 200MHz).
- (2) Change KM4_CKD of SocClk_Info[0] to CLKDIV(3) if KM4 is wanted to run at 520MHz/3.

```
// for km4, max 330MHz under 1.0v, max 260MHz under 0.9v
// for km0, max 100MHz under all core power
// PLL can be 300MHz~600MHz
// KM4_CKD range is [1, 8], KM0_CKD range is [1, 16] or USEXTAL
BOOT_RAM_DATA_SECTION
SocClk_Info_TypeDef SocClk_Info[] = {
    /* PLL_CLK, Vol_Type, KM4_CKD, KM0_CKD, PSRAM_CKD */
    {PLL_520M, CORE_VOL_0P9, CLKDIV(2), CLKDIV(6), CLKDIV(2)},
    {PLL_330M, CORE_VOL_1P0, CLKDIV(1), CLKDIV(4), CLKDIV(1)},
    {PLL_400M, CORE_VOL_0P9, CLKDIV(2), CLKDIV(10), CLKDIV(1)},
    {PLL_600M, CORE_VOL_0P9, CLKDIV(3), CLKDIV(15), CLKDIV(2)},
};

/**
 * @brief SocClk_Info select
 * Boot_SocClk_Info_Idx valid value is [0, 3] and 0xFF
 * when Boot_SocClk_Info_Idx is 0xFF, set socclk by chipinfo Automatically
 * when Boot_SocClk_Info_Idx is [0, 3], set socclk by SocClk_Info[Boot_SocClk_Info_Idx]
 */
BOOT_RAM_DATA_SECTION
u8 Boot_SocClk_Info_Idx = 0;
```

- (3) Rebuild the project and download the new image.
- Now, the clock of KM4 is 173.3MHz, KM0 is 86.6MHz, PSRAM controller is 260MHz (twice the PSRAM), and core power is 0.9V. The clocks of left modules in RTL8721Dx will be set to a reasonable value by software automatically based on their maximum speeds.

17.3 Flash Clock Switch

Flash runs half as fast as the SPI Flash controller. By default, the speed of the SPI Flash controller is divided by the PLL, and the speed of the SPI Flash controller shall be less than SPIC_CLK_LIMIT (208MHz). If the Flash needs to run slower, change the value of Flash_Speed (SPIC0) or Data_Flash_Speed (SPIC1) in {SDK}\component\soc\amebadplus\usrcfg\ameba_flashcfg.c.

```
/**
 * @brief Indicate the flash baudrate. It can be one of the following value:
 * CLKDIV(10): => SPIC clock = 1/10 pll
 * CLKDIV(9): => SPIC clock = 1/9 pll
 * CLKDIV(8): => SPIC clock = 1/8 pll
 * CLKDIV(7): => SPIC clock = 1/7 pll
 * CLKDIV(6): => SPIC clock = 1/6 pll
 * CLKDIV(5): => SPIC clock = 1/5 pll
 * CLKDIV(4): => SPIC clock = 1/4 pll
 * CLKDIV(3): => SPIC clock = 1/3 pll
 * CLKDIV(2): => SPIC clock = 1/2 pll
 * other value is not support.
 */
const u16 Flash_Speed = CLKDIV(2);
const u16 Data_Flash_Speed = CLKDIV(2);
```

18 Hardware Crypto Engine

18.1 Introduction

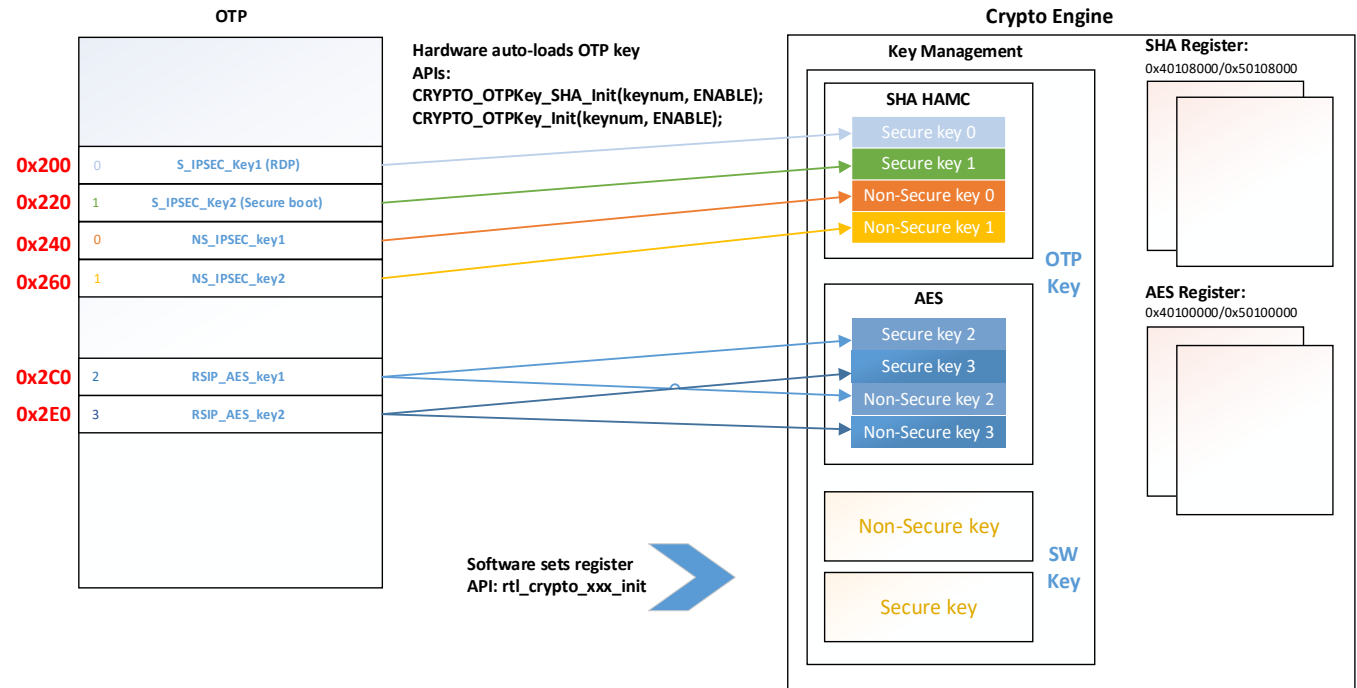
The Hardware Crypto Engine (also called IPsec) accelerates applications that need cryptographic functions, such as authentication, encryption and decryption. Hardware crypto engine executing these functions cannot only reduce software overhead but also save CPU and memory resources, and the processing of it is more secure and faster than software. The key pair can be passed in by software or automatically loaded from OTP by hardware without software access the key.

18.1.1 OTP Keys for Crypto

Crypto engine loads the secret key in two ways: one is that the user passes the secret key into the engine (software accessible key), the other way is that Crypto automatically loads keys from OTP (software inaccessible key).

OTP physical map can store six keys for Crypto to use as secret keys, which can only be accessed by crypto engine and will not be tampered with or read by attackers. The premise is that the secret key needs to be programmed into OTP physical map.

Among these six keys, there are two secure keys and two non-secure keys available for SHA HMAC to choose from, and four secure keys and non-secure keys available for AES to choose from. Refer to the table below for specific information.



Engine	Engine type	Key index	OTP key
SHA HMAC	Secure	0	S_IPSEC_Key1
		1	S_IPSEC_Key2
	Non-Secure	0	NS_IPSEC_Key1
		1	NS_IPSEC_Key2
AES	Secure	0	S_IPSEC_Key1
		1	S_IPSEC_Key2
		2	RSIP_AES_key1
		3	RSIP_AES_key2
	Non-Secure	0	NS_IPSEC_Key1
		1	NS_IPSEC_Key2
		2	RSIP_AES_key1
		3	RSIP_AES_key2

The detailed functions of OTP key are as follows:

OTP key name	Address	Size	Default	Description
S_IPSEC_Key1 (RDP)	Logic Map 0x200	32 bytes	Each Byte 0xFF	Secure crypto engine will auto-load this key for HMAC or AES function when OTPKey_init function is enabled.
S_IPSEC_Key2 (Secure boot HMAC)	Logic Map 0x220	32 bytes	Each Byte 0xFF	
NS_IPSEC_Key1	Logic Map 0x240	32 bytes	Each Byte 0xFF	Non-Secure crypto engine will auto-load this key for HMAC or AES function when OTPKey_init function is enabled.
NS_IPSEC_Key2	Logic Map 0x260	32 bytes	Each Byte 0xFF	
RSIP_AES_key1	Logic Map 0x2c0	32 bytes	Each Byte 0xFF	Non-Secure AES engine and secure AES engine will auto-load this key for AES function when OTPKey_init function is enabled.
RSIP_AES_key2	Logic Map 0x2e0	32 bytes	Each Byte 0xFF	
S_IPSEC_Key1 Read Protection	Physical Map 0x365[3]	1 bit	1	0: Enable read protection for S_IPSEC_Key1 to prevent from being read out. 1: Disable read protection for S_IPSEC_Key1.
S_IPSEC_Key1 Write Protection	Physical Map 0x365[4]	1 bit	1	0: Enable write protection for S_IPSEC_Key1 to prevent from being programmed to all 0 by hacker. 1: Disable write protection for S_IPSEC_Key1.
S_IPSEC_Key2 Read Protection	Physical Map 0x365[5]	1 bit	1	0: Enable read protection for S_IPSEC_Key2 to prevent from being read out. 1: Disable read protection for S_IPSEC_Key2.
S_IPSEC_Key2 Write Protection	Physical Map 0x365[6]	1 bit	1	0: Enable write protection for S_IPSEC_Key2 to prevent from being programmed to all 0 by hacker. 1: Disable write protection for S_IPSEC_Key2.
NS_IPSEC_Key1 Read Protection	Physical Map 0x365[7]	1 bit	1	0: Enable read protection for NS_IPSEC_Key1 to prevent from being read out. 1: Disable read protection for NS_IPSEC_Key1.
NS_IPSEC_Key1 Write Protection	Physical Map 0x366[0]	1 bit	1	0: Enable write protection for S_IPSEC_Key1 to prevent from being programmed to all 0 by hacker. 1: Disable write protection for NS_IPSEC_Key1.
NS_IPSEC_Key2 Read Protection	Physical Map 0x366[1]	1 bit	1	0: Enable read protection for NS_IPSEC_Key2 to prevent from being read out. 1: Disable read protection for NS_IPSEC_Key2.
NS_IPSEC_Key2 Write Protection	Physical Map 0x366[2]	1 bit	1	0: Enable write protection for NS_IPSEC_Key2 to prevent from being programmed to all 0 by hacker. 1: Disable write protection for NS_IPSEC_Key2.
RSIP_AES_Key1 Read Protection	Physical Map 0x366[7]	1 bit	1	0: Enable read protection for RSIP_AES_Key1 to prevent from being read out. 1: Disable read protection for RSIP_AES_Key1.
RSIP_AES_Key1 Write Protection	Physical Map 0x367[0]	1 bit	1	0: Enable write protection for RSIP_AES_Key1 to prevent from being programmed to all 0 by hacker. 1: Disable write protection for RSIP_AES_Key1.
RSIP_AES_Key2 Read Protection	Physical Map 0x367[1]	1 bit	1	0: Enable read protection for RSIP_AES_Key2 to prevent from being read out. 1: Disable read protection for RSIP_AES_Key2.
RSIP_AES_Key2 Write Protection	Physical Map 0x367[2]	1 bit	1	0: Enable write protection for RSIP_AES_Key2 to prevent from being programmed to all 0 by hacker. 1: Disable write protection for RSIP_AES_Key2.

NOTE

For the two secure keys of IPsec, if the system enables RDP, SDK will use S_IPSEC_Key1 for secure data protection by default. And if boot uses HMAC as hash algorithm, it will use S_IPSEC_Key2. Users need to reasonably allocate the use of keys according to the above contents.

18.1.2 Crypto Key Order

The key order for Crypto Engine is the same with mbedtls, it uses little endian mode. For example, if the key is:

```
0x0123456789abcdef0123456789abcdef00112233445566778899aabbccddeeff
```

When the key is put in an array and pass to HW engine using RTK API or mbedtls API, the array should like:

```
u8 key1[32]={
    0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22,
    0x11, 0x00,
    0xef, 0xcd, 0xab, 0x89, 0x67, 0x45, 0x23, 0x01, 0xef, 0xcd, 0xab, 0x89, 0x67, 0x45,
    0x23, 0x01
};
```

When program the key into OTP, customer should use the following commands, Take NS_SHA_key2 as an example:

```
Efuse wraw 0x260 20 ffeeddccbbaa99887766554433221100efcdab8967452301efcdab8967452301
```

The contents in OTP is:

0x260	ff	ee	dd	cc	bb	aa	99	88	77	66	55	44	33	22	11	00
0x270	ef	cd	ab	89	67	45	23	01	ef	cd	ab	89	67	45	23	01

18.1.3 Crypto Key Flash Procedure

The process of programming OTP physical map is as follows:

- (1) Generate the key.
- (2) Write into OTP physical map by command "Efuse wraw <address> <length> <data>"

```
Efuse wraw 0x260 20 0123456789abcdef0123456789abcdef00112233445566778899aabbccddeeff
```
- (3) Use the following command to read OTP key back to check if it is written correctly. If not, re-write it.

```
Efuse rraw
```
- (4) Enable Key Read Protection and Write Protection to prevent key exposure and tampering after the written IPSEC Key is confirmed.

```
Efuse wraw 0x366 1 f9
```

NOTE

After read protection and write protection are programmed, the key can never be read out again. Please maintain the key pair carefully.

18.2 Usage

18.2.1 Start Hardware Crypto Engine

Call initialization APIs to initialize the engine before using and any calculation

NOTE

The power of the engine will be turn off when the system enters low power status, so the settings will be rested .A new round initialization is suggested during resume.

18.2.2 Use Auto-loaded OTP Key from OTP

- (1) Initialize AES OTP key function:

```
CRYPTO_OTPKey_Init(keynum, ENABLE);
```
- (2) Initialize HMAC OTP key function:

```
CRYPTO_OTPKey_SHA_Init(keynum, ENABLE);
```

18.2.3 Start Crypto Engine Calculation

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

18.2.3.1 Hash Algorithm

If a hash algorithm is selected, and the message length doesn't exceed 245745 (=15 * (214 -1)) bytes, Hash APIs (rtl_crypto_xxx) can be used to calculate the digest. If the message length exceeds 245745 bytes or you want to divide this message into many particular size blocks to process them, you could use Sequential Hash Mechanism to verify the hash function.

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.

NOTE

- Considering that the crypto engine moves data through DMA and bypasses the D-Cache, if the destination array is placed in stack and the start address is not 32-byte aligned, the cache line would be dirty during function call in some cases. To avoid reading wrong digest back, users should choose one of the following methods:
 - ◆ The digest array is a global variable.
 - ◆ Or the start address of digest array should be 32-byte aligned if it's a local variable.

- ◆ Or call the following line to restore data from memory before reading digest when calculation is finished if it's a local variable.
`DCache_Invalidate(((u32)digest&CACHE_LINE_ADDR_MSK), (sizeof(digest)+CACHE_LINE_SIZE));`
- After KM4 power gating, IPsec needs to be initialized again to work normally.

18.2.3.2 Cipher Algorithm

Steps to encrypt or decrypt message are as follows:

- (1) Call Cipher APIs (rtl_crypto_aes_XXX_init) to initialize
- (2) Encrypt or decrypt the message
 - Call Cipher APIs (rtl_crypto_aes_XXX_encrypt) to encrypt source message.
 - Call Cipher APIs (rtl_crypto_aes_XXX_decrypt) to decrypt source message.

18.3 Demo Code

The demo code of hardware crypto engine locates in {SDK}\component\example\peripheral\raw\Crypto.

19 One-time Programmable Controller (OTPC)

19.1 Introduction

Antifuse one-time programmable (OTP) is the most secure embedded non-volatile memory (eNVM). The default value is '1' and can only be change from '1' to '0'.

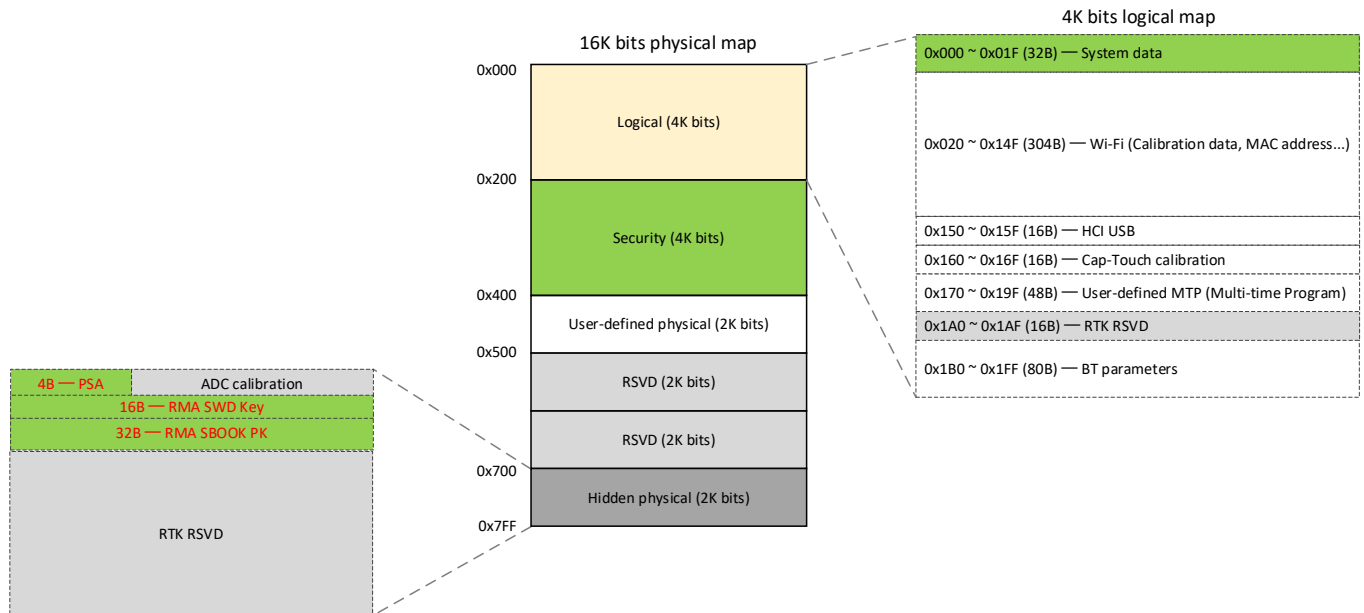


Figure 19-1 OTP Layout

NOTE

For detailed layout of each OTP zone, refer to UM1000.

19.2 OTP Programming APIs

API	Description	Operation zone
OTP_Read8	Read OTP physical zone one byte	Physical zone
OTP_Write8	Write OTP physical zone one byte	Physical zone
OTP_LogicalMap_Read	Read OTP logical zone by length	Logical zone
OTP_LogicalMap_Write	Write OTP logical address by length	Logical zone
OTP_RemainLength	Get OTP remain available length used for logical map	Logical zone
OTPGetCRC	Get the CRC of security area	Physical zone

19.2.1 OTP_Read8

Items	Description
Introduction	Read OTP physical zone one byte
Parameters	- Addr: OTP physical zone address to be read - Data: one byte data buffer for OTP data read
Return	Result of read operation 1: SUCCESS 0: FAIL

19.2.2 OTP_Write8

Items	Description
Introduction	Write OTP physical zone one byte
Parameters	● Addr: OTP physical zone address to be written ● Data: one byte data to be written
Return	Result of write operation ● 1: SUCCESS ● 0: FAIL

19.2.3 OTP_LogicalMap_Read

Items	Description
Introduction	Read OTP logical map
Parameters	● pbuf: buffer used for OTP logical map ● addr: OTP logical zone start address to be read ● len: OTP logical zone byte length to be read
Return	Result of read operation ● 1: SUCCESS ● 0: FAIL

19.2.4 OTP_LogicalMap_Write

Items	Description
Introduction	Write OTP logical address by length
Parameters	● addr: OTP logical zone start address to be written ● cnts: OTP logical byte length to be written ● data: data buffer to be write
Return	Result of write operation ● 1: SUCCESS ● 0: FAIL

19.2.5 OTP_RemainLength

Items	Description
Introduction	Get OTP remain available length used for logical map
Parameters	None
Return	Remain available length

19.2.6 OTPGetCRC

Items	Description
Introduction	Get the CRC value of security area
Parameters	None
Return	CRC value

19.3 OTP Programming Commands

You can only access OTP by the following commands from serial port.

19.3.1 Logical Zone

You can read and write the logical zone by commands in [Table 19-1](#).

Table 19-1 Read/write commands of logical zone

Operation	Command	Description
Read	EFUSE rmap	Read the whole logical zone.
Write	EFUSE wmap <address> <length> <data>	Write to specific address of logical zone. ● address: start logical address to be written to, in hex ● length: bytes of data needed to be written, in hex ● data: data to be written, in hex i NOTE <i>The string length of data to be written must be even.</i>

For example:

- By command "EFUSE wmap 0 2 3087", you can write 0x3087 that is 2 bytes into logical address 0x0.
- By command "EFUSE rmap", the logical zone is all shown immediately.

```

EFUSE wmap 0 2 3087
efuse wmap write len:2, string len:4
EFUSE rmap
EFUSE[000]: 30 87 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[010]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[020]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[030]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[040]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[050]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[060]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[070]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[080]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[090]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[0a0]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[0b0]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
EFUSE[0c0]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```

In the massive production (MP) stage, another command to program logical zone is "iwpriv", which has been integrated into RF calibration tools. This command is only recommended to be used to program Wi-Fi calibration zone.

19.3.2 Physical Zone

You can read and write the physical zone by commands in [Table 19-2](#). The value of physical zone can only be written from "1" to "0", please program it carefully.

Table 19-2 Read/write commands of physical zone

Operation	Command	Description
Read	EFUSE rraw	Read the whole physical zone.
Write	EFUSE wraw <address> <length> <data>	Write to specific address of physical zone. ● address: start physical address to be written to, in hex ● length: bytes of data needed to be written, in hex ● data: data to be written, in hex i NOTE <i>The string length of data to be written must be even.</i>

For example:

- By command "EFUSE wraw 366 1 FE", you can write 0xFE that is 1 byte into physical address 0x366 to enable the NS_IPSEC_Key2_R_Forbidden_EN bit.
- By command "EFUSE rraw", the physical zone is all shown immediately.

```

EFUSE wraw 366 1 FE
efuse wraw write len:1, string len:2
wraw: 366 fe
EFUSE rraw
efuse rraw
RawMap[000]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[010]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[020]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[030]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[040]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[050]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[060]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
RawMap[070]: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```

... ..

RawMap[320]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
RawMap[330]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
RawMap[340]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
RawMap[350]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
RawMap[360]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
RawMap[370]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
RawMap[380]:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff

In the MP stage, you can also use Wi-Fi command "iwpriv" as mentioned in section 19.3.1.

19.4 Usage

19.4.1 Logical Zone

The OTP can only be programmed once, however some data needs to be overwritten in some reason. Therefore, the logical data can be overwritten after format conversion defined by Realtek, as described in User Manual (Section: Mapping Relationship of Physical OTP and Logical OTP).

The logical zone can be programmed multi-times, in case the remain length of physical zone 0x0~0x1FF is enough.

NOTE

The logical zone is programmed in bytes instead of bits. Therefore, to avoid writing incorrectly that would cause wasting the physical zone for logical mapping, you should read logical map to check the original value before programming new value.

19.4.1.1 System Data

The logical area 0x000 ~ 0x01F (32Bytes) will be auto-loaded to system registers by hardware when the system boots. If the system data hasn't been programmed, system registers keep initial value that are 0x00, however reading logical map will get 0xFF.

The procedure of programming the system data is described in Figure 19-2.

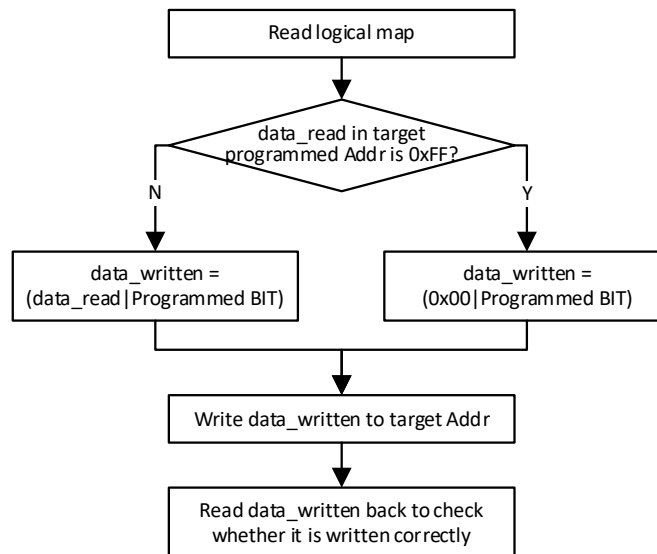


Figure 19-2 Programming the system data

About the target address that you want to program, there're two cases:

- One is that the system data in the target address has been programmed before. In this case, you can refer to Example1.
- The other is that the system data hasn't been programmed never. In this case, you can refer to Example2.

NOTE

When programming the system data, the start address must be 4-byte aligned at 4-byte length.

Example1

Program the value of logical address 0x02[1] to 1, you should follow these steps:

- (1) Read the logical map to check the original value in logical address 0x00~0x03.

```
efuse rmap
```

OR

```
u8 data_read[4];
OTP_LogicalMap_Read(&data_read,0,4);
```

- (2) Assume the data is 0x12A03456 in logical address 0x00~0x03 in step (1). Let 0xA0 makes 'OR' operation with programmed bit[1], and other data keeps default value. So the new value to be written is 0x12A23456.

- (3) Write the new value 0x12A23456 to logical address 0x00~0x03.

```
efuse wmap 0 4 5634A212
```

OR

```
u8 data_written[4]={0x56,0x34,0xA2,0x12};
OTP_LogicalMap_Write(0,4,(u8 *)data_written);
```

- (4) Read the data_written again to check whether the value is written correctly.

```
efuse rmap
```

OR

```
u8 data_read[4];
OTP_LogicalMap_Read(&data_read,0,4);
```

Example2

Program the value of logical address 0x08[0] to 1, you should follow following steps:

- (1) Read the logical map to check the original value.

```
efuse rmap
```

OR

```
u8 data_read[4];
OTP_LogicalMap_Read(&data_read,8,4);
```

- (2) Assume the data is 0xFFFFFFFF in logical address 0x08~0x0B in step (1). Let 0x00 makes 'OR' operation with programmed bit[0], and other data keeps default value. So the new value to be written is 0x00000001.

- (3) Write the new value 0x00000001 to logical address 0x08~0x0B.

```
efuse wmap 8 4 01000000
```

OR

```
u8 data_written[4]={0x01,0x00,0x00,0x00};
OTP_LogicalMap_Write(8,4,(u8 *)data_written);
```

- (4) Read the data_written again to check whether the value is written correctly.

```
efuse rmap
```

OR

```
u8 data_read[4];
OTP_LogicalMap_Read(&data_read,8,4);
```

19.4.1.2 Wi-Fi Calibration Data

For detailed information about Wi-Fi Calibration Data, refer to *WS_MP_FLOW.pdf*.

19.4.1.3 Programming Scenarios

Usually, system data has their initial value, and you can program specific bits according to your demands. [Table 19-3](#) list some scenarios that specific bits need to be programmed at your requirements.

Table 19-3 Programming scenarios

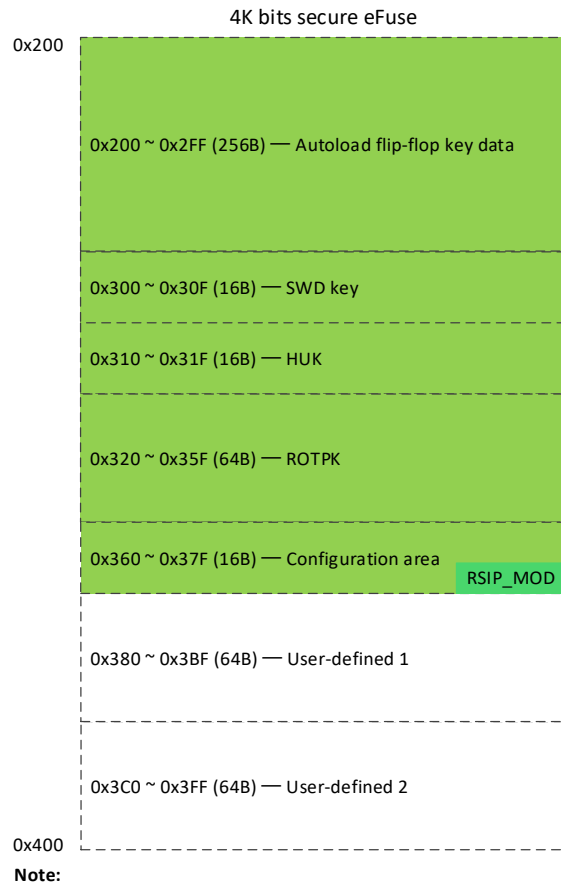
Offset	Bit	Symbol	INI	Description	Scenarios
0x02	[1]	SPIC_ADDR_4BYTE_EN	0	SPI Flash controller address 4-byte enable 0: Disable 1: Enable	- If embedded Flash is used, ignore it. - If external Flash is used, moreover, its size is larger than 16M bytes, program it.
0x03	[1]	LOW_BAUD_LOG_EN	0	LOGUART baud rate selection 0: 1.5Mbps 1: 115200bps	If the LOGUART baud rate needs to be changed from 1.5Mbps to 115200bps, program it.
0x03	[0]	DIS_BOOT_LOG_EN	0	Boot ROM log disable 0: Enable 1: Disable	If boot ROM log needs to be disabled, program it.

19.4.2 Security Zone

The security zone is divided into three parts, as illustrated in [Figure 19-3](#).

- Key area: 0x200~0x35F

- Configuration area: 0x360~0x37F
- User-defined area: 0x380~0x3FF


Note:

- The zone filled with green color will be auto-loaded to internal memory by hardware.
- The RW access to each smaller zone is controlled by configuration area.

Figure 19-3 Security area layout

19.4.2.1 Key Area

Contents in key area are listed in [Table 19-4](#). For more detailed usage about the keys, refer to the corresponding chapters.

Table 19-4 Key area

Function	Name	Size (bits)	Start offset	End offset	Usage
IPSEC	S_IPSEC_Key1 (RDP)	256	0x200	0x021F	<i>Hardware Crypto Engine</i>
IPSEC	S_IPSEC_Key2 (Secure boot HMAC)	256	0x220	0x023F	
IPSEC	NS_IPSEC_Key1	256	0x240	0x025F	
IPSEC	NS_IPSEC_Key2	256	0x260	0x027F	
USER PRI	USER_PRI_KEY1	256	0x280	0x029F	
USER PRI	USER_PRI_KEY2	256	0x2A0	0x02BF	
RSIP	RSIP_ECB_KEY	256	0x2C0	0x02DF	
RSIP	RSIP_CTR_KEY	256	0x2E0	0x02FF	
SWD	SWD_PASSWORD	128	0x300	0x030F	
PSA	HUK	128	0x310	0x031F	
Secure Boot	PK1 (ROTPK hash)	256	0x320	0x033F	
Secure Boot	PK2 (ROTPK hash)	256	0x340	0x035F	

19.4.2.2 Configuration Area

Contents in configuration area are listed in [Table 19-5](#). About field's usage in this area, you can get detailed information in the corresponding

chapters.

Table 19-5 Configuration area

Offset	Bit	Symbol	Description	Usage	
0x360	[31:0]	SWD_ID	SWDID used to mapping the real SWD Key		
0x364	[0]	SWD_PWD_EN	SWD password enable		
	[1]	SWD_DBGEN	SWD external debug authentication		
	[2]	SWD_NIDEN			
	[3]	SWD_SPIDEN			
	[4]	SWD_SPNIDEN			
	[5]	SWD_PWD_R_Protection_EN	Key write protection and read protections		
	[6]	SWD_PWD_W_Forbidden_EN			
	[7]	HUK_W_Forbidden_EN			
0x365	[0]	RSVD			
	[1]	PK1_W_Forbidden_EN			
	[2]	PK2_W_Forbidden_EN			
	[3]	S_IPSEC_Key1_R_Protection_EN	Hardware Crypto Engine		
	[4]	S_IPSEC_Key1_W_Forbidden_EN			
	[5]	S_IPSEC_Key2_R_Protection_EN			
	[6]	S_IPSEC_Key2_W_Forbidden_EN			
	[7]	NS_IPSEC_Key1_R_Protection_EN			
0x366	[0]	NS_IPSEC_Key1_W_Forbidden_EN			
	[1]	NS_IPSEC_Key2_R_Protection_EN			
	[2]	NS_IPSEC_Key2_W_Forbidden_EN			
	[3]	USER_PRI_KEY1_R_Protection_EN			
	[4]	USER_PRI_KEY1_W_Forbidden_EN			
	[5]	USER_PRI_KEY2_R_Protection_EN			
	[6]	USER_PRI_KEY2_W_Forbidden_EN			
	[7]	RSIP_KEY1_R_Protection_EN			
0x367	[0]	RSIP_KEY1_W_Forbidden_EN			
	[1]	RSIP_KEY2_R_Protection_EN			
	[2]	RSIP_KEY2_W_Forbidden_EN			
	[3]	RSIP_MODE_W_Forbidden_EN			
	[4]	SIC_SECURE_EN	Permit SIC to access secure zone 1: Permit 0: Forbid	Program it or keep it default value according to your requirements.	
	[5]	CPU_PC_DBG_EN	Enable to get KM4/KM0 PC value through debug port 1: Enable 0: Disable	Program it or keep it default value according to your requirements.	
	[6]	UDF1_TRUSTZONE_EN	User-defined 1 area (0x380~0x3BF) TrustZone protection enable 0: Enable 1: Disable	By default, this area can be accessible from both secure world and non-secure world. To make this area only be accessible from secure world, program this bit.	
	[7]	UDF2_TRUSTZONE_EN	User-defined 2 area (0x3C0~0x3FF) TrustZone protection enable 0: Enable 1: Disable	By default, this area can be accessible from both secure world and non-secure world. To make this area only be accessible from secure world, program this bit.	
0x368	[0]	UART_DOWNLOAD_DISABLE	Used in ROM to disable power on latch UART download 0: Disable 1: Enable (default)	To disable power on latch UART download, program this bit.	
	[1]	RSVD	-		
	[2]	RSIP_EN	Enable/Disable RSIP control		
	[3]	SECURE_BOOT_EN			
	[4]	SECURE_BOOT_HW_DIS			
	[5]	RDP_EN			
	[6]	ANTI_ROLLBACK_EN		OTA Firmware Update	
	[7]	FAULT_LOG_PRINT_DIS	Used in ROM to disable ROM hard fault log 0: Disable 1: Enable (default)	To disable ROM hard fault log, program this bit.	
0x369	[1:0]	RSIP_MODE	RSIP Mode		
	[2]	HUK_DERIV_EN	Enable/Disable HUK derive		
	[3]	USER_PHYSICAL_TZ1_EN	-		
	[4]	USER_PHYSICAL_TZ2_EN	-		
	[5]	SW_RSVD0	-		

	[6]	SWTRIG_UART_DOWNLOAD_DISABLE	Used in ROM to disable SW trigger UART download 0: Disable 1: Enable (default)	To disable SW trigger UART download, program this bit.
	[7]	SPIC_PINMUX_TESTMODE_DISABLE	-	
0x36A	[7:0]	RSVD	-	
0x36B	[3:0]	SECURE_BOOT_AUTH_LOG	Secure boot Auth Algorithm	
	[7:4]	SECURE_BOOT_HASH_LOG	Secure boot Hash Algorithm	
0x36C	[15:0]	OTA_ADDR	OTA address, 4K aligned	OTA Firmware Update
0x36E	[15:0]	BOOTLOADER_VERSION	Bootloader version	
0x370	[31:0]	CRC0	CRC check	19.4.2.2.1
0x374	[31:0]	CRC1		
0x378	[31:0]	CRC2		
0x37C	[31:0]	CRC3		

19.4.2.2.1 CRC

CRC is used for defending against injection attacks, and this function is accomplished by comparing the valid CRC entry that you programmed into OTP with the one calculated by hardware for security zone (0x200~0x36B). If you want to ensure the secure zone un-attacked, then CRC field needs to be programmed.

One CRC entry takes 4 bytes, including a 2-byte magic number and a 2-byte valid CRC value.

There are 4 CRC entries in total in physical OTP and you can only use one entry at one time. You must use entry 0 first, and then entry1, entry2 and use entry3 at last. CRC check cannot be disabled once enabled. Rom will enter endless loop if magic number or valid CRC check fail.

When you use CRC validation function for the first time, please follow the following steps:

- Program CRC entry after you make sure that security zone has been programmed done. Because any modification for the security zone (0x200~0x36B) will cause CRC value changed, then you have to re-program an new CRC entry, which will result in wasting one CRC entry
- Get the valid CRC value without actually enabling the CRC function by using command:

```
EFUSE getcrc
```

```
#EFUSE getcrc
new crc value is 0xb4c5
```

- Program valid CRC value calculated in previous step and magic number (0x8730) of the entry 0.

- Magic number is 0x8730:

```
EFUSE wraw 370 2 3087
```

- Assuming that CRC value is 0xB4C5:

```
EFUSE wraw 372 2 C5B4
```

- Read the CRC entry back, to check whether it's been written correctly

```
EFUSE rraw
```

CAUTION

Pay attention to the order of data.

- Reset the chip
 - If the CRC entry is checked pass, the boot process will be successfully.
 - If the CRC entry is checked fail, the following log will show up, and the chip enters endless loop.

```
ROM: [V1.0]
FLASHRATE:1
BOOT FROM NOR
Expect OTP CRC 0xb5c4 PG CRC 0x74ac
PG CRC Entry 0 Magic 0x8730
```

If security zone (0x200~0x36B) has been changed, a new CRC entry is needed.

- Make sure that security zone has programmed done.
- Get the new CRC value.

```
EFUSE getcrc
```

- Program the previous used entry to all 0x00 to invalidate this entry, that means both CRC and magic number are programmed into 0x00.

For example, assuming entry 0 is the previous entry:

```
EFUSE wraw 370 4 00000000
```

- Program the next CRC entry with valid CRC and magic number to validate the next entry.

For example, if entry 0 is the previous entry, entry 1 should be used now:

- Magic number is 0x8730:
EFUSE wraw 374 2 3087

- Assuming that CRC value is 0xB4C5:
EFUSE wraw 376 2 C5B4

(5) Read the CRC entry back to check whether it's been written correctly

```
EFUSE rraw
```

(6) Reset the chip to check if CRC entry is ok.

CAUTION

- We suggest users programming CRC entry in factory.
- Once CRC entry is programmed, and you need to modify secure zone. Please remember to invalidate current CRC entry and program the correct CRC value and magic number in the next CRC entry before re-boot. Otherwise, chip will enter endless loop and cannot boot successfully again.

19.4.3 Hidden Physical Zone

The hidden physical zone contains some RMA keys and the Realtek's calibration data. Users can only program the RMA-related area: In this area, two keys have their own read protection and write protection. These two keys will be auto-loaded to the HW:

- For SWD Key, a non-programmed value means that the key is all 0xFF.
- For secure boot public key hash, a non-programmed value means that the secure boot is disabled in RMA mode.

Contents in hidden physical area and usage is listed in [Table 19-6](#).

Table 19-6 Hidden physical area

Offset	Bit	Symbol	Description	Usage
0x700	[7:0]	RMA (Life State)	Define which mode device works in. ● If the number of 1 is odd, it will go to RMA mode ● If the number of 1 is even, it will go to Normal mode HW will auto-load the work mode first when boot. In RMA mode, the secure 4K bits should be protected and return all "1" when read.	● To make the device go to RMA mode, you should program this field to make sure the number of 1 is odd. ● To make the device go to Normal mode, you should program this field to make sure the number of 1 is even. ● By default, the value is 0xFF and it's in normal mode.
0x701	[1:0]	ROM_PATCH_EN	Defined by Realtek	Used by Realtek
	[2]	ROM_PATCH_LWE1		
	[3]	ROM_PATCH_LWE2		
	[4]	ROM_PATCH_LWE3		
	[5]	ROM_PATCH_LWE4		
	[6]	ROM_PATCH_LWE5		
	[7]	ROM_PATCH_HWE		
0x702	[0]	RMA_SWD_PWD_R_Protection_EN	Key read protection and write protection.	
	[1]	RMA_SWD_PWD_W_Forbidde n_EN		
	[2]	RMA_PK_W_Forbidden_EN		
	[7:3]	RSVD	Reserved	
0x704	[63:0]	ADC calibration	Defined by Realtek	Used by Realtek
0x710	128	RMA SWD Key	SWD Key in RMA Mode	
0x720	256	RMA SBOOT KEY HASH	SBOOT Key Hash in RMA Mode	

NOTE

After Read protection and Write protection programmed, the key can never be read out again. Please maintain the key carefully.

20 Power Saving

20.1 Power-Saving Mode

20.1.1 Summary

The RTL8721Dx has an advanced Power Management Controller (PMC), which can flexibly power up different power domains of the chip, to achieve the best balance between chip performance and power consumption. AON, SYSON, SOC are three main power domains in digital system. Functions in different power domains will be turned off differently in different power-saving modes.

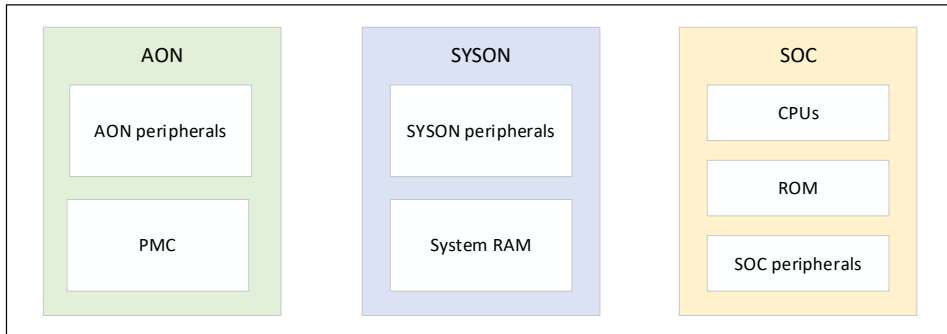


Figure 20-1 FreeRTOS tickless in an idle task

To simplify the power management for typical scenarios, RTL8721Dx supports two low-power modes, which are sleep mode and deep-sleep mode. Tickless is a FreeRTOS low power feature, which just gates the CPU (no clock or power be turned off) when it has nothing to do. Sleep mode flow and deep-sleep mode flow are based on Tickless. [Table 20-1](#) explains power-saving related terms.

Table 20-1 Power-saving mode

Mode	AON domain	SYSON domain	SOC domain	Description
Tickless	ON	ON	ON	- FreeRTOS low power feature - CPU periodically enters WFI, and exits WFI when interrupts happen. - Radio status can be configured off/periodically on/always on, which depends on the application.
Sleep	ON	ON	Clock-gated or power-gated	- A power saving mode on chip level, including clock-gating mode and power-gating mode. - CPU can restore stack status when the system exits from sleep mode. - The system RAM will be retained, and the data in system RAM will not be lost.
Deep-sleep	ON	OFF	OFF	- A more power-saving mode on chip-level. - CPU cannot restore stack status. When the system exits from deep-sleep mode, the CPU follows the reboot process. - The system RAM will not be retained.

Peripherals are also divided into different power domains in order to achieve better power consumption.

- SOC power domain: including peripherals such as GDMA, SPI, I2C, SDIO, PWM, QSPI, PPE, LEDC, etc.
 - SYSON power domain: mainly including peripherals which support wakeup, such as Basic Timer, GPIO, UART, Key-Scan, Cap-Touch, etc.
- For peripherals in SOC domain, note that they need to be reinitialized after wakeup from PG since the power domain they are on has been power gated during sleep.

FreeRTOS supports a low-power feature called tickless. It is implemented in an idle task which has the lowest priority. That is, it is invoked when there is no other task under running. Note that unlike the original FreeRTOS, the RTL8721Dx does not wake up based on the "xExpectedIdleTime" in [Figure 20-2](#).

[Figure 20-2](#) shows idle task code flow. In idle task, it will check sleep conditions (wakelock, sysactive_time, details in [20.6.1](#), [20.6.2](#)) to determine whether needs to enter sleep mode or not.

- If not, the CPU will execute an ARM instruction "WFI" (wait for interrupt) which makes the CPU suspend until the interrupt happens. Normally systick interrupt resumes it. This is the software tickless.
- If yes, it will execute the function `freertos_pre_sleep_processing()` to enter sleep mode or deep-sleep mode.

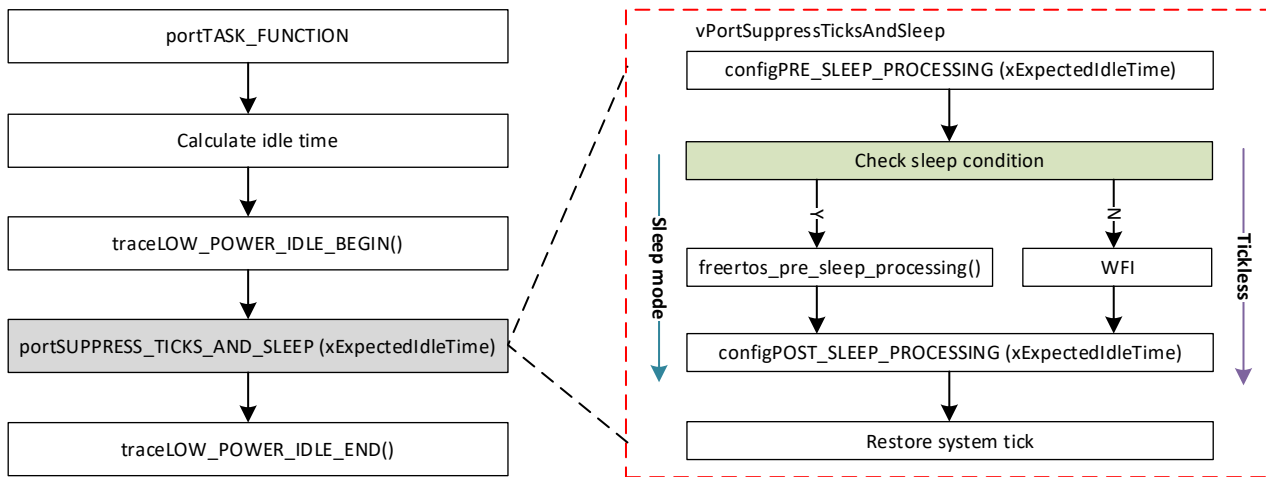


Figure 20-2 FreeRTOS tickless in an idle task

NOTE

- Even FreeRTOS time control like software timer or `vTaskDelay` is set, it still enters the sleep mode if meeting the requirement as long as the idle task is executed.
- `configUSE_TICKLESS_IDLE` must be enabled for power-saving application because sleep mode flow is based on tickless.

20.1.2 Sleep Mode

There are two sleep mode types, sleep CG and sleep PG. CG means to turn off the clock of the SOC domain, while PG means to turn off both the power and clock of the SOC domain. So PG has lower power consumption.

Since the CPU is in SOC domain, sleep PG will need to back up and restore the CPU status, thus PG will consume a bit more time during sleep and wakeup flow than CG.

KM0 and KM4 support both CG or both PG since they are in the same power domain. KM4 is normally used as Application Processor (AP), and KM0 is normally used as Network Processor (NP) for Wi-Fi driver and Wi-Fi firmware, thus KM0 can enter sleep mode only if KM4 requests to enter sleep mode first.

- For sleep PG, if a wakeup event occurs, PMC will turn on the SOC domain's power and clock. KM0 will first start from the reset handler, and check the flag to see if it wakes from PG. If so, KM0 will restore the CPU status, continue to execute from where it sleeps, and then check wakeup reasons to see if this wake source is for KM4, then decide whether to release KM4's clock to resume KM4. [Figure 20-3](#) shows the sleep and wake flow of PG.
- For sleep CG, if a wakeup event occurs, PMC will turn on the SOC domain's clock. Since KM0 is not power-gated in sleep CG, it will wake up and continue to execute from where it sleeps, and then check wakeup reasons to see if need to resume KM4. [Figure 20-4](#) shows the sleep and wake flow of CG.

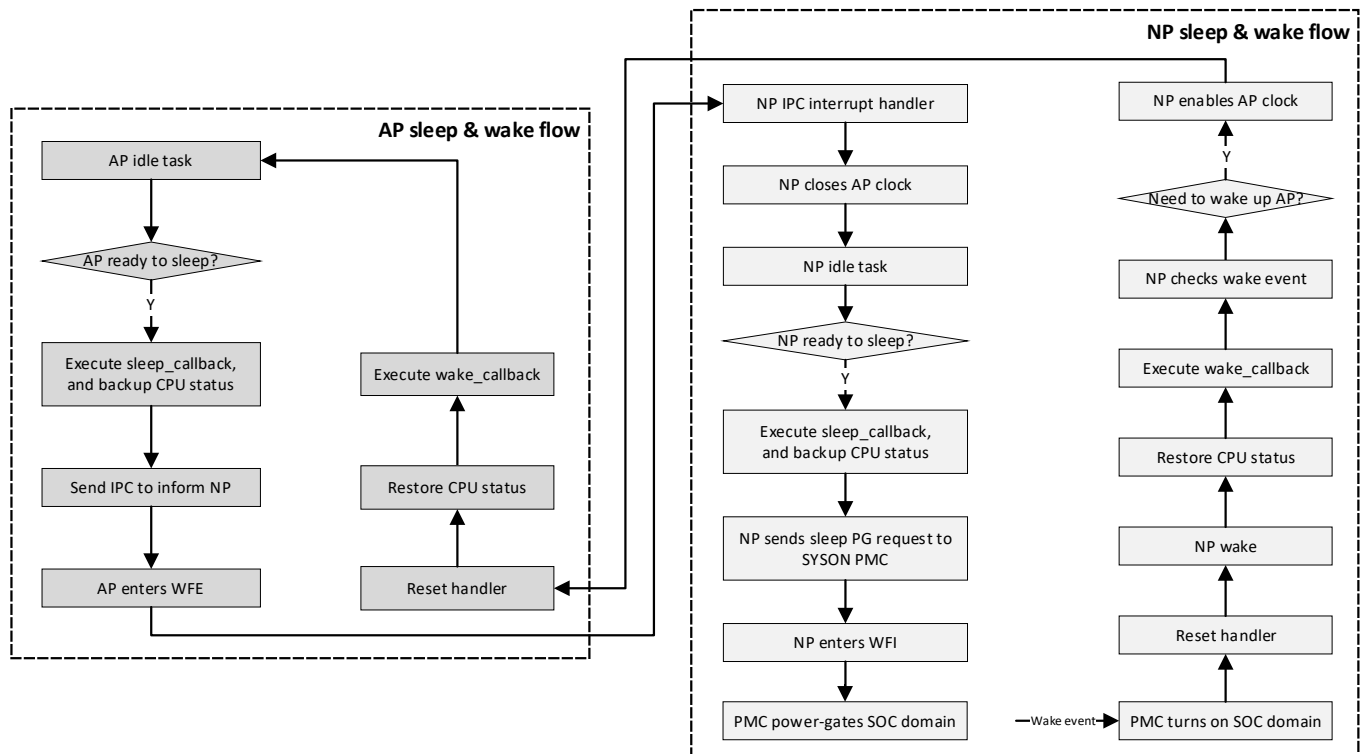


Figure 20-3 Sleep and wake flow of PG

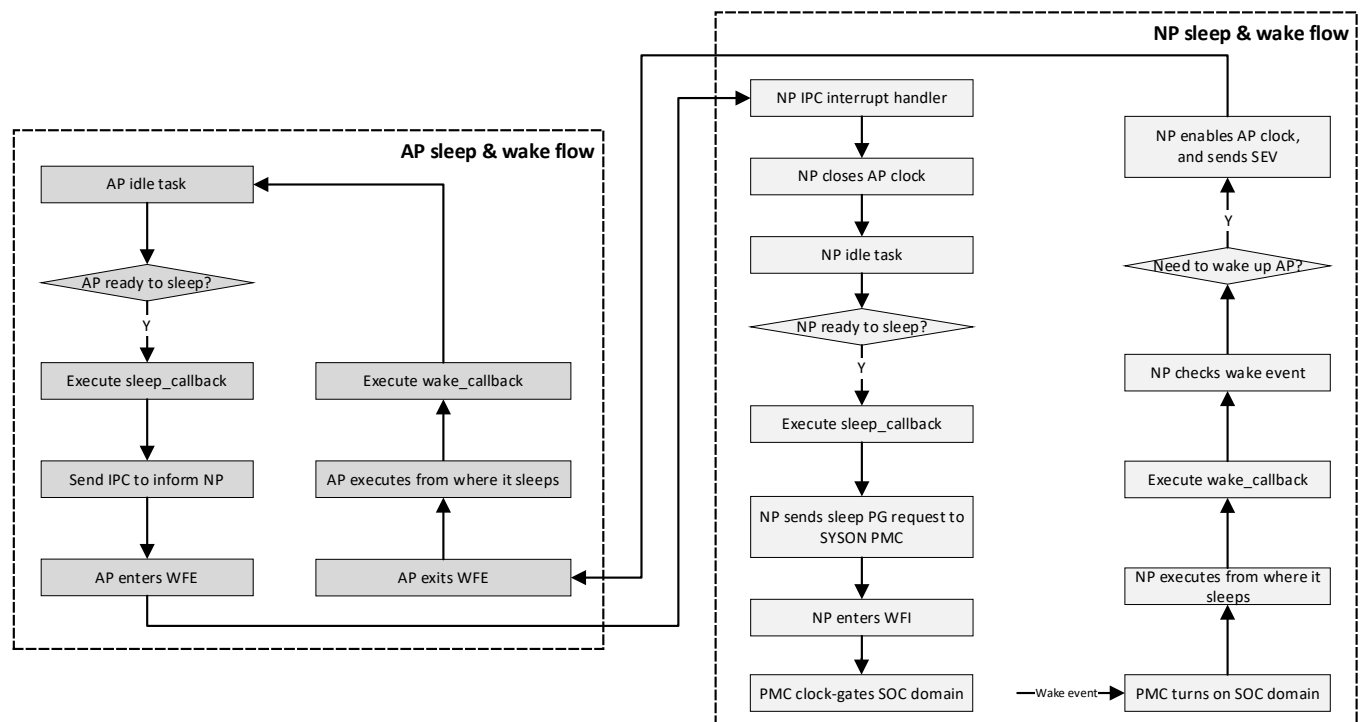


Figure 20-4 Sleep and wake flow of CG

20.1.3 Deep-Sleep Mode

Deep-sleep mode has a lower power consumption as only the AON domain is on while the SYSON and SOC domains are off. So only peripherals in the AON domain can wake up the chip.

When the chip wakes up from deep-sleep mode, it will do the boot process. As system SRAM and CPU are shut down in deep-sleep mode, the corresponding interrupt of the peripheral which is set as the wake source should be registered again after wakeup to process the interrupt

handler. Figure 20-5 shows deep-sleep mode flow.

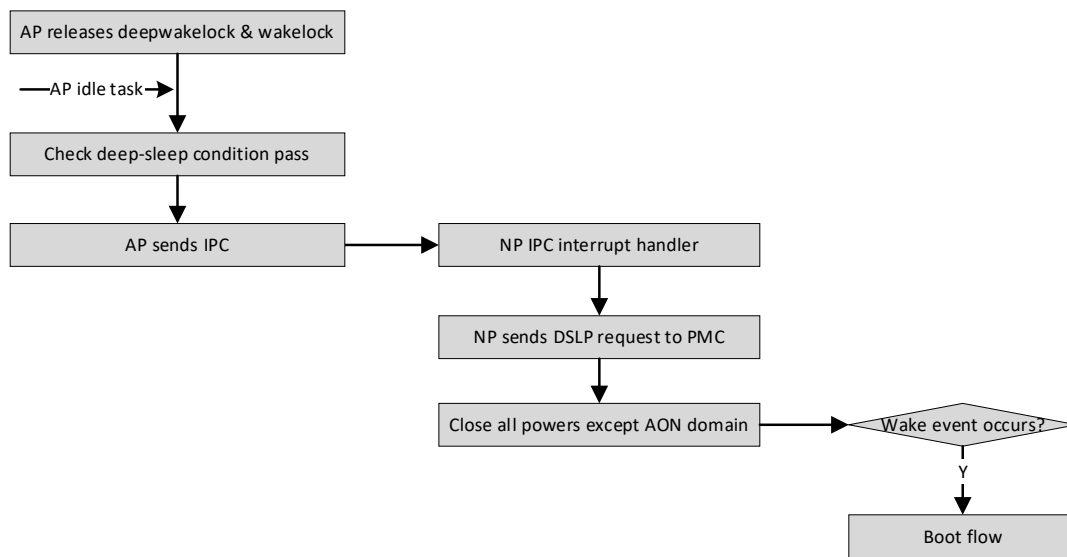


Figure 20-5 Deep-sleep mode flow

20.2 Wakeup Source

Table 20-2 lists the wakeup sources that can be used to wake up the system under different power modes.

Table 20-2 Wakeup source

Wakeup source	Sleep CG	Sleep PG	Deep-sleep	Comment
WLAN	√	√	X	
BT	√	√	X	
IWDG	√	√	X	
IPC	√	√	X	Only KM0 can use IPC to wake up KM4
Basic Timer4~7	√	√	X	
PMC Timer	√	√	X	For internal usage
UART0~2	√	√	X	Need to keep OSC4M on during sleep, not recommended to use when the baudrate is larger than 115200
LOGUART	√	√	X	Need to keep XTAL on during sleep
GPIO	√	√	X	
I2C	√	√	X	
Cap-Touch	√	√	X	
ADC	√	√	X	
ADC comparator	√	√	X	
SDIO	√	√	X	
Key-Scan	√	√	X	
BOR	√	√	X	
PWR_DOWN	√	√	√	
AON_TIMER	√	√	√	
AON_WAKEPIN	√	√	√	
RTC	√	√	√	

20.3 Entering Sleep Mode

Sleep mode is based on FreeRTOS tickless, thus it is recommended to enter sleep mode by releasing the wakelock.

- (1) Initialize the specific peripheral.
- (2) Enable and register the peripheral's interrupt.
- (3) Set `sleep_wevent_config[]` in `ameba_sleepcfg.c`, and the interrupt should be registered on the same CPU selected by `sleep_wevent_config[]`.
- (4) For peripherals that need special clock settings, set `ps_config[]` in `ameba_sleepcfg.c` if needed.

- (5) Register sleep/wakeup callback if needed.
- (6) Enter sleep mode by releasing the wakelock in KM4 (PMU_OS needs to be released since it is acquired by default when boot).
- (7) Clear the peripheral's interrupt when wakeup.

For peripherals that need specific clock settings, such as UART and LOGUART, their setting flows are described in [20.3.1](#) and [20.3.2](#).

20.3.1 UART

- When using UART as a wakeup source, clock OSC4M should not be closed when the system is in sleep mode.
- When the baudrate is larger than 115200, it is not recommended to use UART as a wakeup source.

Configuration:

- (1) Initialize UART and enable its interrupt.
- (2) Set the related wakeup source (WAKE_SRC_UART0/WAKE_SRC_UART1/WAKE_SRC_UART2_BT) in `sleep_wevent_config[]` to WAKEUP_KM4 or WAKEUP_KM0 (based on which CPU you want to wake). The interrupt should be registered on the same CPU selected by `sleep_wevent_config[]`.
- (3) Set the corresponding entry of `uart_config[]` in `ameba_sleepcfg.c` to "ENABLE".
- (4) Set `keep_OSC4M_on` in `ps_config[]` to "TRUE" to keep OSC4M enabled during sleep mode.
- (5) Enter sleep mode by releasing the wakelock in KM4 (PMU_OS needs to be released since it is acquired by default when boot).
- (6) Clear the UART interrupt when wakeup.

20.3.2 LOGUART

When using LOGUART as a wakeup source, XTAL should not be closed during sleep.

Configuration:

- (1) Initialize LOGUART and enable its interrupt.
- (2) Set WAKE_SRC_UART_LOG in `sleep_wevent_config[]` to WAKEUP_KM4 or WAKEUP_KM0 (based on which CPU you want to wake). The interrupt should be registered on the same CPU selected by `sleep_wevent_config[]`.
- (3) Set `xtal_mode_in_sleep` to XTAL_Normal in `ps_config[]`.
- (4) Enter sleep mode by releasing the wakelock in KM4 (PMU_OS needs to be released since it is acquired by default when boot).
- (5) Clear the LOGUART interrupt when wakeup.

20.4 Entering Deep-Sleep Mode

Deep-sleep can also be entered from FreeRTOS tickless flow.

When the system boots, KM4 holds the deepwakelock PMU_OS, thus `freertos_ready_to_dsleep()` will be checked fail and the system does not enter deep-sleep mode in idle task by default. Since `freertos_ready_to_dsleep()` will be checked only after `freertos_ready_to_sleep()` is checked pass, both the wakelock and deepwakelock need to be released for entering deep-sleep mode.

Configuration:

- (1) Initialize the related peripheral and enable its interrupt.
- (2) Set `sleep_wakepin_config[]` in `ameba_sleepcfg.c` when using AON wakepin as a wakeup source.
- (3) Enter deep-sleep mode by releasing the deepwakelock and wakelock in KM4.

20.5 Power-Saving Configuration

20.5.1 Wakeup Mask Setup

For sleep mode, only one CPU is required to wake up to execute the program in some situations. The wakeup mask module is designed to implement this function. By setting a wakeup mask, you can choose to wake up only KM0, or KM4. If KM4 is chosen, KM0 will be waked up first and then KM0 will resume KM4.

Users can set the wakeup attribute in `sleep_wevent_config[]` in `ameba_sleepcfg.c` to choose which CPU you want to wake up. The wakeup attribute of each wakeup source can be set to WAKEUP_KM4 or WAKEUP_KM0 or WAKEUP_NULL, respectively indicating that this wakeup source is only to wake up KM4, or wake up KM0, or not used as a wakeup source.

```

/* Wakeup entry can be set to WAKEUP_NULL/WAKEUP_KM4/WAKEUP_KM0 */
WakeEvent_TypeDef sleep_wevent_config[] = {
//    Module                                Wakeup
{WAKE_SRC_SDIO,                            WAKEUP_NULL},
{WAKE_SRC_AON_WAKEPIN,                     WAKEUP_NULL},
{WAKE_SRC_AON_TIM,                         WAKEUP_NULL},
{WAKE_SRC_Keyscan,                         WAKEUP_NULL},
{WAKE_SRC_PWR_DOWN,                       WAKEUP_NULL},
{WAKE_SRC_BOR,                             WAKEUP_NULL},
{WAKE_SRC_ADC_COMP,                       WAKEUP_NULL},
{WAKE_SRC_ADC,                             WAKEUP_NULL},
{WAKE_SRC_RTC,                             WAKEUP_NULL},
{WAKE_SRC_CTOUCH,                         WAKEUP_NULL},
{WAKE_SRC_I2C1,                           WAKEUP_NULL},
{WAKE_SRC_I2C0,                           WAKEUP_NULL},
{WAKE_SRC_GPIOB,                           WAKEUP_NULL},
{WAKE_SRC_GPIOA,                           WAKEUP_NULL},
{WAKE_SRC_UART_LOG,                       WAKEUP_NULL},
{WAKE_SRC_UART2_BT,                       WAKEUP_NULL},
{WAKE_SRC_UART1,                           WAKEUP_NULL},
{WAKE_SRC_UART0,                           WAKEUP_NULL},
{WAKE_SRC_pmc_timer1,                     WAKEUP_KM0},    /* Internal use, do not change it*/
{WAKE_SRC_pmc_timer0,                     WAKEUP_KM4},    /* Internal use, do not change it*/
{WAKE_SRC_Timer7,                         WAKEUP_NULL},
{WAKE_SRC_Timer6,                         WAKEUP_NULL},
{WAKE_SRC_Timer5,                         WAKEUP_NULL},
{WAKE_SRC_Timer4,                         WAKEUP_NULL},
{WAKE_SRC_IWDG0,                           WAKEUP_NULL},
{WAKE_SRC_IPC_KM4,                         WAKEUP_KM4},    /* IPC can only wake up KM4, do not
change it*/
{WAKE_SRC_BT_WAKE_HOST,                   WAKEUP_NULL},
{WAKE_SRC_KM4_WAKE_IRQ,                   WAKEUP_KM0},    /* Internal use, do not change it*/
{WAKE_SRC_WIFI_FTSR_MAILBOX,              WAKEUP_KM0},    /* Wi-Fi wakeup, do not change it*/
{WAKE_SRC_WIFI_FISR_FESR_IRQ,             WAKEUP_KM0},    /* Wi-Fi wakeup, do not change it*/
{0xFFFFFFFF,                             WAKEUP_NULL},
};

```

20.5.2 AON Wakepin Configuration

AON wakepin is one of the peripherals that can be set as a wakeup source. The RTL8721Dx has two AON wakepins (PB30 and PB31), which can be configured in `sleep_wakepin_config[]` in `ameba_sleepcfg.c`. The config attribute can be set to `DISABLE_WAKEPIN` or `HIGH_LEVEL_WAKEUP` or `LOW_LEVEL_WAKEUP`, meaning not wake up, or GPIO level high will wake up, or GPIO level low will wake up respectively.

```

/* can be used by sleep mode & deep sleep mode */
/* config can be set to DISABLE_WAKEPIN/HIGH_LEVEL_WAKEUP/LOW_LEVEL_WAKEUP */
Wakepin_TypeDef sleep_wakepin_config[] = {
//    wakepin                                config
{WAKEPIN_0,    DISABLE_WAKEPIN},    /* WAKEPIN_0 corresponding to _PB_30 */
{WAKEPIN_1,    DISABLE_WAKEPIN},    /* WAKEPIN_1 corresponding to _PB_31 */
{0xFFFFFFFF,    DISABLE_WAKEPIN},
};

```

NOTE

- By default, `AON_WAKEPIN_IRQ` will not be enabled in `sleep_wakepin_config[]`, and users need to enable it by themselves.
- The wakeup mask will not be set in `sleep_wakepin_config[]`. If wakepin is used for sleep mode, `WAKE_SRC_AON_WAKEPIN` entry needs to be set in `sleep_wevent_config[]`.

20.5.3 Clock and Voltage Configuration

The XTAL, OSC4M state, and sleep mode voltage are configurable in `ps_config[]` in `ameba_sleepcfg.c`. Users can use this configuration for peripherals that need XTAL or OSC4M on in sleep mode.

```

PSCFG_TypeDef ps_config = {
    .keep_OSC4M_on = FALSE,    /* Keep OSC4M on or off for sleep */
    .xtal_mode_in_sleep = XTAL_OFF,    /* Set XTAL mode during sleep mode, see enum
xtal_mode_sleep for details */
};

```

```
.sleep_to_08V = FALSE, /* Default sleep to 0.7V, setting this option to
TRUE will sleep to 0.8V */
};
```

20.5.4 Sleep Type Configuration

KM4 can set sleep mode to CG or PG by calling the function `pmu_set_sleep_type(uint32_t type)`, and users can get CPU's sleep mode by calling the function `pmu_get_sleep_type()`.

NOTE

- *KM0 and KM4 are in the same power domain, so they will have the same sleep type, thus `pmu_set_sleep_type()` should be set to KM4, and KM0 will follow KM4's sleep mode type.*
- *Sleep mode is set to PG by default. If users want to change the sleep type, `pmu_set_sleep_type()` needs to be called before sleep.*

20.6 Power-Saving Related APIs

20.6.1 Wakelock APIs

In some situations, the system needs to keep awake to receive certain events; otherwise, events may be missed when the system is in sleep mode. An idea of wakelock is introduced in that the system cannot enter sleep mode if some module is holding the wakelock.

Wakelock is a 32-bit map. Each module has its own bit in the wakelock bit map (see enum. `PMU_DEVICE`). Users can also add the wakelock in the enum. If the wakelock bit map equals zero, it means that there is no module holding the wakelock. If the wakelock bit map is larger than zero, it means that some modules are holding the wakelock, and the system is not allowed to enter sleep mode.

Wakelock is a judging condition in the function `freertos_ready_to_sleep()`. When the system boots, KM4 holds the wakelock `PMU_OS`, and KM0 holds the wakelock `PMU_OS` and `PMU_KM4_RUN`. Only if all wakelocks are released, KM4 or KM0 is permitted to enter sleep mode. The function `freertos_ready_to_sleep()` will judge the value of wakelock.

```
typedef enum {
    PMU_OS = 0,
    PMU_WLAN_DEVICE,
    PMU_LOGUART_DEVICE,
    PMU_KM4_RUN,
    PMU_KM0_RUN,

    PMU_UART0_DEVICE = 5,
    PMU_UART1_DEVICE,
    PMU_I2C0_DEVICE,
    PMU_TOUCH_DEVICE,
    PMU_RTC_DEVICE,
    PMU_CONSOL_DEVICE,
    PMU_ADC_DEVICE,
    PMU_WAKWLOCK_TIMEOUT,
    PMU_PSRAM_DEVICE,
    PMU_WLAN_FW_DEVICE,
    PMU_BT_DEVICE,
    PMU_DEV_USER_BASE = 16, /* Number 16 ~ 31 is reserved for customer use*/
    PMU_MAX = 31
} PMU_DEVICE;
```

It is recommended to enter sleep mode by releasing the wakelock. After the wakelock `PMU_OS` of KM4 is released, KM4 will enter sleep mode in idle task and send IPC to KM0. KM0 will gate KM4's clock first and then release `PMU_OS` and `PMU_KM4_RUN`.

When the system wakes, it will enter sleep mode again quickly unless it acquires the wakelock.

Similar to the wakelock for sleep mode, there is a 32-bit deepwakelock map for deep-sleep mode. If the deepwakelock bit map is larger than zero, it means that some modules are holding the deepwakelock, and the system is not allowed to enter deep-sleep mode. When the system boots, KM4 holds the deepwakelock `PMU_OS`.

Deepwakelock is a judging condition in the function `freertos_ready_to_dsleep()`. After the deepwakelock `PMU_OS` of KM4 is released, and all wakelocks of KM4 are released, KM4 will be allowed to enter deep-sleep mode and send IPC to KM0 in idle task. KM0 will

send a deep-sleep request and let the chip finally enter deep-sleep mode.

APIs in the following sections are provided to control the wakelock or deepwakelock.

20.6.1.1 pmu_acquire_wakelock

Items	Description
Introduction	Acquire the wakelock for one module
Parameter	nDeviceId: Device ID of the corresponding module <pre>typedef enum { PMU_OS = 0, ... PMU_MAX = 31 } PMU_DEVICE;</pre>
Return	None

20.6.1.2 pmu_release_wakelock

Items	Description
Introduction	Release the wakelock for one module
Parameter	nDeviceId: Device ID of the corresponding module <pre>typedef enum { PMU_OS = 0, ... PMU_MAX = 31 } PMU_DEVICE;</pre>
Return	None

20.6.1.3 pmu_acquire_deepwakelock

Items	Description
Introduction	Acquire the deepwakelock for one module
Parameter	nDeviceId: Device ID of the corresponding module <pre>typedef enum { PMU_OS = 0, ... PMU_MAX = 31 } PMU_DEVICE;</pre>
Return	None

20.6.1.4 pmu_release_deepwakelock

Items	Description
Introduction	Release the deepwakelock for one module
Parameter	nDeviceId: Device ID of the corresponding module <pre>typedef enum { PMU_OS = 0, ... PMU_MAX = 31 } PMU_DEVICE;</pre>
Return	None

20.6.2 pmu_set_sysactive_time

In some situations, the system needs to keep awake for a period of time to complete a certain process while the system is active.

Items	Description
Introduction	Set a period of time that the system will keep active
Parameter	timeout: time value, unit is ms.

	The system will keep active for this time value from now on.
Return	0

20.6.3 Sleep/Wake Callback APIs

These APIs are used to register suspend/resume callback function for <nDeviceId>. The suspend callback function will be called in idle task before the system enters sleep mode, and the resume callback function will be called after the system resumes.

It is a good way to use the suspend and resume function if there is something to do before the chip sleeps or after the chip wakes. A typical application of the resume function is to acquire the wakelock to prevent the chip from sleeping again. Also, if the CPU chooses PG, some peripherals will lose power so they need to be reinitialized. This can be implemented in the resume function.

NOTE

- Yield OS is not permitted in the suspend/resume callback functions, thus RTOS APIs which may cause OS yield such as `rtos_task_yield`, `rtos_time_delay_ms`, or mutex, semaphore related APIs are not recommended to use.
- `pmu_set_sysactive_time` is not permitted in the suspend callback function, but permitted in the resume callback function.

20.6.3.1 pmu_register_sleep_callback

Items	Description
Introduction	Register the suspend/resume callback function for one module
Parameter	- nDeviceId: Device ID needs suspend/resume callback <pre>typedef enum { PMU_OS = 0, ... PMU_MAX = 31 } PMU_DEVICE;</pre> - sleep_hook_fun: Suspend callback function - sleep_param_ptr: Suspend callback function parameter - wakeup_hook_fun: Resume callback function - wakeup_param_ptr: Resume callback function parameter
Return	None

20.6.3.2 pmu_unregister_sleep_callback

Items	Description
Introduction	Register the suspend/resume callback function for one module
Parameter	- nDeviceId: Device ID needs suspend/resume callback <pre>typedef enum { PMU_OS = 0, ... PMU_MAX = 31 } PMU_DEVICE;</pre> - sleep_hook_fun: Suspend callback function - sleep_param_ptr: Suspend callback function parameter - wakeup_hook_fun: Resume callback function - wakeup_param_ptr: Resume callback function parameter
Return	None

20.6.4 pmu_set_max_sleep_time

Items	Description
Introduction	Set the maximum sleep time
Parameter	timer_ms: system maximum sleep timeout, unit is ms.
Return	None

NOTE

- The system will be woken up after the timeout.
- The system may be woken up by other wake events before the timeout.
- This setting only works once. The timer will be cleared after the system wakes up.

20.6.5 Wakeup Reason APIs

20.6.5.1 SOCPS_AONWakeReason

Items	Description
Introduction	Get the deep-sleep wakeup reason
Parameter	None
Return	Bit[0]: CHIP_EN short press Bit[1]: CHIP_EN long press Bit[2]: BOR Bit[3]: AON Timer Bit[5:4]: AON GPIO Bit[8]: RTC

20.6.5.2 WAK_STATUS0

The following register can be used to get the sleep wakeup reason.

Register	Parameters
WAK_STATUS0	Bit[1:0]: WLAN Bit[2]: KM4_WAKE Bit[3]: BT_WAKE_HOST Bit[4]: IPC_KM4 Bit[5]: IWDG0 Bit[9:6]: BASIC_TIMER4~7 Bit[11:10]: PMC_TIMER0~1 Bit[14:12]: UART0~2 Bit[15]: UART_LOG Bit[16]: GPIOA Bit[17]: GPIOB Bit[19:18]: I2C0~1 Bit[20]: Cap-Touch Bit[21]: RTC Bit[22]: ADC Bit[23]: ADC_COMP Bit[24]: BOR Bit[25]: PWR_DOWN Bit[26]: Key-Scan Bit[27]: AON_Timer Bit[28]: AON_Wakepin Bit[29]: SDIO

Note that when wakeup, the corresponding peripheral interrupt will be raised; when clearing the interrupt, the corresponding bit in wakeup reason will also be cleared. Thus it is not possible to get the wakeup reason after the interrupt is cleared.

20.7 Wi-Fi Power Saving

IEEE 802.11 power save management allows the station to enter its own sleep state. It defines that the station needs to keep awake at a certain timestamp and enter a sleep state otherwise.

WLAN driver acquires the wakelock to avoid the system entering sleep mode when WLAN needs to keep awake. And it releases the wakelock when it is permitted to enter the sleep state.

IEEE 802.11 power management allows the station to enter power-saving mode. The station cannot receive any frame during power saving. Thus AP needs to buffer these frames and requires the station to periodically wake up to check the beacon which has the information of buffered frames.

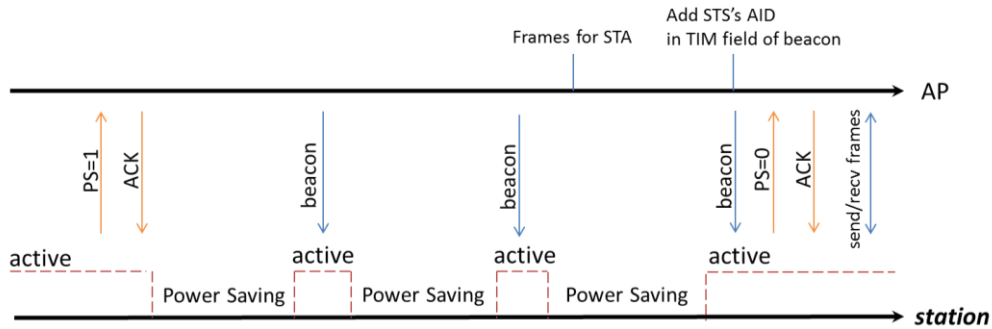


Figure 20-6 Timeline of power saving

When the system is active, and Wi-Fi is connected and enters IEEE 802.11 power management mechanism, this is called LPS in SDK; if the system enters sleep mode when Wi-Fi is connected, we call it WoWLAN mode.

In WoWLAN mode, a timer with a period of about 102ms will be set in the suspend function, and KM0 will wake up every 102ms to receive the beacon to maintain the connection.

Except LPS and WoWLAN modes, there is also an IPS mode, which can be used when Wi-Fi is not connected. [Table 20-3](#) lists all three power-saving modes for Wi-Fi. [Table 20-4](#) describes the relationship between power modes of the system and Wi-Fi.

Table 20-3 Wi-Fi power-saving modes

Mode	Wi-Fi status		Description	SDK
IPS	Not associated	RF/BB/MAC OFF	Wi-Fi driver automatically turns Wi-Fi off to save power.	IPS mode is enabled in SDK by default and is not recommended to be disabled.
LPS	Associated	- RF periodically ON/OFF - MAC/BB always ON	LPS mode is used to implement IEEE 802.11 power management. NP will control RF ON/OFF based on TSF and TIM IE in the beacon.	LPS mode is enabled in SDK by default but can be disabled through API in Table 20-5 .
WoWLAN	Associated	- RF and BB periodically ON/OFF - MAC periodically enters/exits CG/PG	NP is waked up at each beacon early interrupt to receive a beacon from the associated AP. NP will wake up AP when receiving a data packet.	WoWLAN mode is enabled in SDK by default.

Table 20-4 Relationship between power modes of system and Wi-Fi

System power mode	Wi-Fi power mode	Description
Active	IPS	Wi-Fi is on, but not connected
	LPS	Wi-Fi is connected and enters IEEE 802.11 power management mechanism
Sleep	Wi-Fi OFF/IPS	
	WoWLAN	Wi-Fi keeps associating.
Deep-sleep	Wi-Fi OFF	Deep-sleep is not recommended if Wi-Fi needs to keep on or associated.

Table 20-5 API to enable/disable LPS

API	Parameters
int wifi_set_lps_enable(u8 enable)	Parameter: enable - TRUE: enable LPS - FALSE: disable LPS

When Wi-Fi is connected and the system enters sleep mode, WoWLAN mode will be entered automatically, and KM0 will periodically wake up to receive the beacon to maintain the connection, this will consume some power. If you are more concerned about the system power consumption during sleep mode, and Wi-Fi is not a necessary function in your application, it is recommended to set Wi-Fi off or choose Wi-Fi IPS mode.

21 CHIP Enable

21.1 Introduction

The CHIP_EN (chip enable) is an external reset input pin that can be used to control the reset status of the whole SoC. This pin can work in level reset mode, interrupt reset mode or pulse reset mode. By default, it works in level reset mode. This pin always has the function of resetting system no matter in which mode.

In RTL8721Dx, the button with "CHIP_EN" is connected to the CHIP_EN pin, as [Figure 19-1](#) shows.



Figure 21-1 CHIP_EN on board

21.2 Three Working Modes

There are three working modes of CHIP_EN for different usages.

21.2.1 Level Reset Mode

Level reset mode is the default mode of CHIP_EN. In this mode, a low level on this pin longer than debounce time will trigger RESET directly and the SoC will reboot when this pin goes from low to high.

User can reset the whole chip by simply pressing down and releasing the CHIP_EN button. When low level is detected longer than debounce time, the chip will be reset after the CHIP_EN button is released. So user can adjust the sensitivity of CHIP_EN by setting different debounce time from 100us to 16ms.

21.2.2 Interrupt Reset Mode

In interrupt reset mode, a low level on this pin longer than the sum of debounce and short press time can trigger an interrupt instead of resetting the system directly. Thus, software can handle the interrupt and choose whether to reset the system.

The interrupt reset mode is mainly designed for the power save scenario. The CHIP_EN button just likes the power button of the smartphone: a short press will trigger the system to enter low power mode or wake up the system; a long press will trigger an interrupt to remind the user whether to reset the system. The reboot will happen if user chooses reboot or the system crashed.

21.2.3 Pulse Reset Mode

In pulse reset mode, a low level on this pin longer than debounce time will trigger RESET directly and the SoC will reboot immediately. User can read the status of the CHIPEN pin in boot code to distinguish short/long press. This mode usually can be found in router. Short press will cause the system cold boot; long press will cause the system code boot and restore factory settings.

Once the chip is configured working in Pulse Reset Mode, software cannot change the mode anymore. Only power off will take the chip to Level Reset Mode.

21.3 How to Choose Work Mode

- For device does not need any control on this pin, tie this pin to high.
- For simply reset requirement, use the default level reset mode.
- If both reset and re-storage of factory settings are needed, use the pulse reset mode.
- For low power scenario, power control and reset are needed on a single pin, use the interrupt reset mode.

21.4 CHIP_EN Driver APIs

21.4.1 CHIPEN_WorkMode

Items	Description
Introduction	Configure CHIP_EN work mode CHIP_EN works in HW reset mode by default.
Parameters	mode: new work mode of CHIPEN, which can be ● CHIPEN_HW_RESET_MODE ● CHIPEN_INT_RESET_MODE ● CHIPEN_PULSE_RESET_MODE
Return	None

21.4.2 CHIPEN_DebounceSet

Items	Description
Introduction	Set the CHIP_EN debounce Time, which works in all work mode
Parameters	debounce: new debounce counter, which can be ● CHIPEN_DBC_0US ● CHIPEN_DBC_100US ● CHIPEN_DBC_500US ● CHIPEN_DBC_1MS ● CHIPEN_DBC_2MS ● CHIPEN_DBC_4MS ● CHIPEN_DBC_8MS ● CHIPEN_DBC_16MS
Return	None

21.4.3 CHIPEN_ThresHoldSet

Items	Description
Introduction	Set long press and short press threshold, which only works in interrupt reset mode
Parameters	● Thres_LP: long press threshold ■ CHIPEN_LP_1S ■ CHIPEN_LP_1P5S ■ CHIPEN_LP_2S ■ CHIPEN_LP_2P5S ■ CHIPEN_LP_3S ■ CHIPEN_LP_3P5S ■ CHIPEN_LP_4S ■ CHIPEN_LP_4P5S ● Thres_SP: short press threshold ■ CHIPEN_SP_50MS ■ CHIPEN_SP_100MS ■ CHIPEN_SP_150MS ■ CHIPEN_SP_200MS ■ CHIPEN_SP_250MS ■ CHIPEN_SP_300MS ■ CHIPEN_SP_350MS
Return	None

21.4.4 CHIPEN_AckTimeSet

Items	Description
Introduction	Set the ACK threshold for long press interrupt. If long press interrupt can't be cleared within Tack, the system will reboot
Parameters	Tack: ● CHIPEN_ACK_50MS ● CHIPEN_ACK_100MS ● CHIPEN_ACK_200MS ● CHIPEN_ACK_400MS
Return	None

21.4.5 CHIPEN_ClearINT

Items	Description
Introduction	Clear CHIP_EN interrupt status
Parameters	INTrBit: interrupt to be cleared
Return	None

21.4.6 CHIPEN_GetINT

Items	Description
Introduction	Get CHIP_EN interrupt status
Parameters	None
Return	Interrupt status

22 Inter-Processor Communication (IPC)

22.1 Introduction

There are two-CPU's named KM4 (AP) and KM0 (NP) integrated in the RTL8721Dx. The inter-processor communication (IPC) hardware is designed to make these CPU's communicate with each other. Also, a KM0 shared SRAM is used to transmit information to each other. The block diagram is shown in [Figure 22-1](#).

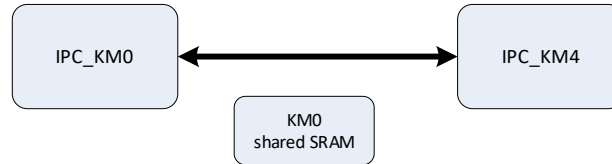


Figure 22-1 IPC block diagram

22.2 General Principle

There are 2 directions for 2 cores to communicate with each other: KM0 $\leftrightarrow$ KM4. There are 16 TX channels and 16 RX channels for each direction. All the channels are processed independently. That means one core can send different information to another core through different channels at any time, the channels will not affect each other.

Each channel has one transmit side and one receive side, the transmit side and the receive side of the same channel is a pair. For example, KM0 sends an IPC to KM4 through channel 7, the transmit side is KM0 channel 7, and receive side is KM4 channel 7, and information is sent from KM0 channel 7 to KM4 channel 7.

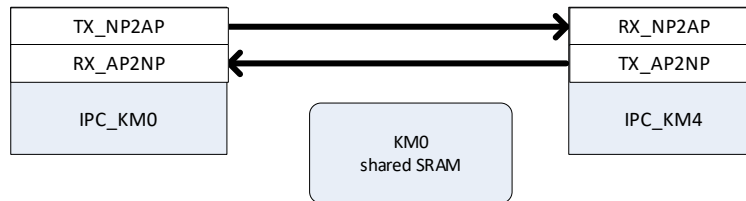


Figure 22-2 IPC schematic diagram

22.3 How to Use IPC

22.3.1 IPC Usage Procedure

For example, KM0 sends an IPC to KM4 through channel 8.

- Register the receiver IRQ handler function of the selected channel in receiver IPC.
In this case, add a new IPC_INIT_TABLE in the KM4 for channel 8 of KM0 to KM4. Rx IRQ handler, Rx IRQ data, Tx IRQ handler, Tx IRQ data, IPC direction and IPC channel should be set and defined. If message exchange is needed, you should specify the message type: data or point.

```

IPC_TABLE_DATA_SECTION
Const IPC_INIT_TABLE ipc_channel8_table[] = {
    {IPC_USER_DATA, IPC_CHANNEL8_ipc_int, (VOID *)NULL, IPC_TXHandler, (VOID *)NULL,
    IPC_KM0_TO_KM4, IPC_N2A_Channel8},
};
  
```

- Uncomment the corresponding channel in **ameba_ipccfg.h** and add some description as shown below. This macro is used in step (5).

```

/** @defgroup IPC_KM0_Tx_Channel
 * @{
 */
#define IPC_N2A_TICKLESS_INDICATION 0 /*!< KM0 -> KM4 Tickless indicate */
#define IPC_N2A_WAKE_AP 1 /*!< KM0 -> KM4 Wakeup*/
#define IPC_N2A_WIFI_FW_INFO 2 /*!< KM0 -> KM4 FW Info*/
#define IPC_N2A_FLASHPG_REQ 3 /*!< KM0 -> KM4 Flash Program REQUEST*/
#define IPC_N2A_LOGUART_RX_SWITCH 4 /*!< KM0 -> KM4 Loguart Rx Switch*/
#define IPC_N2A_BT_API_TRAN 5 /*!< KM0 -> KM4 BT API Exchange */
  
```

```
// #define IPC_N2A_BT_DATA_TRAN 5 /*!< KM0 -> KM4 BT DATA Exchange */
#define IPC_N2A_WIFI_TRX_TRAN 6 /*!< KM0 -> KM4 WIFI Message Exchange */
#define IPC_N2A_WIFI_API_TRAN 7 /*!< KM0 -> KM4 API WIFI Message Exchange */
#define IPC_N2A_Channel8 8
// #define IPC_N2A_Channel9 9
// #define IPC_N2A_Channel10 10
// #define IPC_N2A_Channel11 11
// #define IPC_N2A_Channel12 12
// #define IPC_N2A_Channel13 13
// #define IPC_N2A_Channel14 14
// #define IPC_N2A_Channel15 15
/**
 * @}
 */
```

- (3) Register the transmit IRQ handler function of the selected channel in transmit IPC and uncomment the corresponding channel. This step is **optional**, because this step is for register Tx interrupt, which is for transmit IPC to know that the receiver IPC has received this IPC. In this case, add a new IPC_INIT_TABLE in the KM0 for channel 8 of KM0 to KM4.
- (4) SDK will enable the IPC receiver interrupt of KM4 and transmit interrupt of KM0 (if configured in step (3)) according to IPC_INIT_TABLE, and register the corresponding IRQ handler and data for the channel.
- (5) When KM0 sends an IPC request to KM4 through channel 8, it should call **ipc_send_message()** and specify the channel number and message. If no message is needed, just input NULL for the third parameter of **ipc_send_message()**.

```
IPC_MSG_STRUCT ipc_msg_temp;
// init ipc_msg
ipc_msg_temp.msg_type = IPC_USER_POINT;
ipc_msg_temp.msg = (u32)&tmp_np_log_buf;
ipc_msg_temp.msg_len = 1;
ipc_msg_temp.rsvd = 0;
//send ipc message
ipc_send_message(IPC_KM0_TO_KM4, IPC_N2A_Channel8, & ipc_msg_temp);
```

- (6) After receiving IPC from KM0 channel8, KM4 will enter IPC interrupt handler and the corresponding receive IRQ handler will be executed, call **ipc_get_message()** to get the message if needed.
- (7) If you have configured in step (3), after KM4 receives the IPC, KM0 also enters IPC interrupt handler and executes the corresponding transmit IRQ handler.

NOTE

Several channels are already used by Realtek, you can use the remaining channels.

22.3.2 Suggested Usage: ipc_get_message()

- Use **ipc_get_message()** in IPC interrupt handle or user interrupt handler.

```
void IPC_CHANNEL8_ipc_int(void *Data, u32 IrqStatus, u32 ChanNum)
{
    /* To avoid gcc warnings */
    (void) Data;
    (void) IrqStatus;
    (void) ChanNum;

    PIPC_MSG_STRUCT ipc_msg_temp = (PIPC_MSG_STRUCT)ipc_get_message(IPC_KM0_TO_KM4,
IPC_N2A_Channel8);

    u32 addr = ipc_msg_temp->msg;
    DCache_Invalidate(addr, sizeof(UART_LOG_BUF));
}
```

```
IPC_TABLE_DATA_SECTION
const IPC_INIT_TABLE ipc_channel8_table[] = {
    {IPC_USER_DATA, IPC_CHANNEL8_ipc_int, (VOID *)NULL, IPC_TXHandler, (VOID *)NULL,
IPC_KM0_TO_KM4, IPC_N2A_Channel8},
};
```

- IPC_MSG_STRUCT is no need to cache invalidation any more after **ipc_get_message()**.

```
PIPC_MSG_STRUCT p_ipc_rcv_msg = ipc_get_message(BT_IPC_DIR_EVENT_RX, \
BT_IPC_H2D_API_TRAN);
DCache_Invalidate((uint32_t)p_ipc_rcv_msg, sizeof(IPC_MSG_STRUCT));
```

no need!

- Forcing IPC_MSG_STRUCT type conversion has risk.

```
void inic_ipc_dev_event_int_hdl(VOID *Data, u32 IrqStatus, u32 ChanNum)
{
    UNUSED(Data);
    UNUSED(IrqStatus);
    UNUSED(ChanNum);

    inic_ipc_ex_msg_t *p_ipc_msg = NULL;
    sint ret = FAIL;

    #ifdef IPC_DIR_MSG_RX
        PIPC_MSG_STRUCT p_ipc_rcv_msg = ipc_get_message(IPC_DIR_MSG_RX, \
        p_ipc_msg = (inic_ipc_ex_msg_t *)p_ipc_rcv_msg->msg; suggest do like this: inic_ipc_ex_msg store in p_ipc_rcv_msg->msg
    #else
        p_ipc_msg = (inic_ipc_ex_msg_t *)ipc_get_message(IPC_INT_CHAN_WIFI_TRX_TRAN); X dangerous!!!
    #endif /* IPC_DIR_MSG_RX */

    #ifdef CONFIG_ENABLE_CACHE
        DCache_Invalidate((u32)p_ipc_msg, sizeof(inic_ipc_ex_msg_t));
    #endif /* CONFIG_ENABLE_CACHE */
}
```

- There are certain risks in using ipc_get_message() in task.

```
void bt_ipc_dev_api_task(void *context)
{
    (void)context;
    // uint8_t ret;
    struct bt_ipc_dev_api_task_struct *pcmdtask = &(btIpcDevApiTask);
    ble_audio_ipc_host_request_message *p_ipc_msg = NULL;

    DBG_BT_IPC("%s: Enter BT_IPC_DEV_API_TASK\r\n");
    while (1) {
        if (osif_sem_take(pcmdtask->wakeup_sema, 0xFFFFFFFF) == _FAIL) {
            DBG_BT_IPC("%s: download sema fail\r\n", __func__);
            break;
        }
        if (pcmdtask->blocked) {
            DBG_BT_IPC("%s: blocked(%d) break at line %d\r\n",
                __func__, (int)pcmdtask->blocked, __LINE__);
            break;
        }
        PIPC_MSG_STRUCT p_ipc_rcv_msg = ipc_get_message(BT_IPC_DIR_EVENT_RX, \
        BT_IPC_H2D_API_TRAN); X
        DCache_Invalidate((uint32_t)p_ipc_rcv_msg, sizeof(IPC_MSG_STRUCT));
        p_ipc_msg = (ble_audio_ipc_host_request_message *)p_ipc_rcv_msg->msg;
        DCache_Invalidate((uint32_t)p_ipc_msg, sizeof(ble_audio_ipc_host_request_message));
    }
}
```

- If ipc_get_message() needs to be used in task, do as follows:
 - Task takes the semaphore.
 - In IPC Rx user interrupt handle, using ipc_get_message() to get message.
 - Copy the message to another memory after get message in the same Rx user interrupt handle.
 - Give semaphore.
 Then task can use the message.

```

VOID shell_loguratRx_ipc_int(VOID *Data, u32 IrqStatus, u32 ChanNum)
{
    /* To avoid gcc warnings */
    (void) Data;
    (void) IrqStatus;
    (void) ChanNum;

    PIPC_MSG_STRUCT ipc_msg_temp = (PIPC_MSG_STRUCT)ipc_get_message(IPC_KM0_TO_KM4, IPC_N2A_LOGUART_RX_SWITCH); b)

    u32 addr = ipc_msg_temp->msg;
    DCache_Invalidate(addr, sizeof(UART_LOG_BUF));
    memcpy(shell_ctl.pTmpLogBuf, (u32 *)addr, sizeof(UART_LOG_BUF)); c)

    PUART_LOG_BUF addr_temp = (PUART_LOG_BUF)addr;
    addr_temp->BufCount = 0;
    shell_array_init((u8 *)addr, sizeof(UART_LOG_BUF), '\0');
    DCache_CleanInvalidate(addr, sizeof(UART_LOG_BUF));

    shell_ctl.ExecuteCmd = _TRUE;
    if (shell_ctl.shell_task_rdy) {
        shell_ctl.GiveSema(); d)
    }
} « end shell_loguratRx_ipc_int »

static VOID shell_task_ram(VOID *Data)
{
    /* To avoid gcc warnings */
    (void) Data;

    portALLOCATE_SECURE_CONTEXT(configMINIMAL_SECURE_STACK_SIZE);

    do {
        xSemaphoreTake(shell_sema, RTW_MAX_DELAY); a)

#ifdef ARM_CORE_CM0 ...
#endif
        if (shell_ctl.ExecuteCmd) {
            u8 argc = 0;
            u8 **argv;
            PUART_LOG_BUF pUartLogBuf = shell_ctl.pTmpLogBuf;

```

23 Direct Memory Access Controller (DMAC)

23.1 Introduction

The global direct memory access (GDMA) controller is mainly used to transfer data from/to memory or peripherals over the AXI/OCF bus without CPU intervention, thereby offloading CPU. There is a GDMA instance with 8 channels, where channel 0 and channel 1 have a FIFO size of 128 bytes, and the other channels' FIFO have 32 bytes.

The GDMA (or DMAC) is a dual AXI/OCF master bus architecture with a slave interface for programming, and supports hardware priority and programmable priority between DMA requests.

23.2 GDMA Performance

The data-transmission efficiency of GDMA is affected by clock synchronization, channel FIFO depth, transfer types, handshake efficiency, GDMA interface setting of slave and other factors. The following data is based on the results of the experiment with the transmission type of single block and the transmission channel is zero.

Slave	Clock (Hz)	Writing 64 bytes	Reading 64 bytes
SRAM	250M	$(64 \times 8) / (280\text{ns}) = 1828.57\text{Mbps}$	$(64 \times 8) / (240\text{ns}) = 2133.33\text{Mbps}$
PSRAM	250M	$(64 \times 8) / (350\text{ns}) = 1462.86\text{Mbps}$	$(64 \times 8) / (360\text{ns}) = 1422.22\text{Mbps}$
Audio	40M	$(64 \times 8) / (1050\text{ns}) = 487.62\text{Mbps}$	$(64 \times 8) / (470\text{ns}) = 1089.36\text{Mbps}$
SPI	100M	$(64 \times 8) / (710\text{ns}) = 721.13\text{Mbps}$	$(64 \times 8) / (670\text{ns}) = 764.18\text{Mbps}$

NOTE

The time of GDMA turn-around is not included.

23.3 Usage

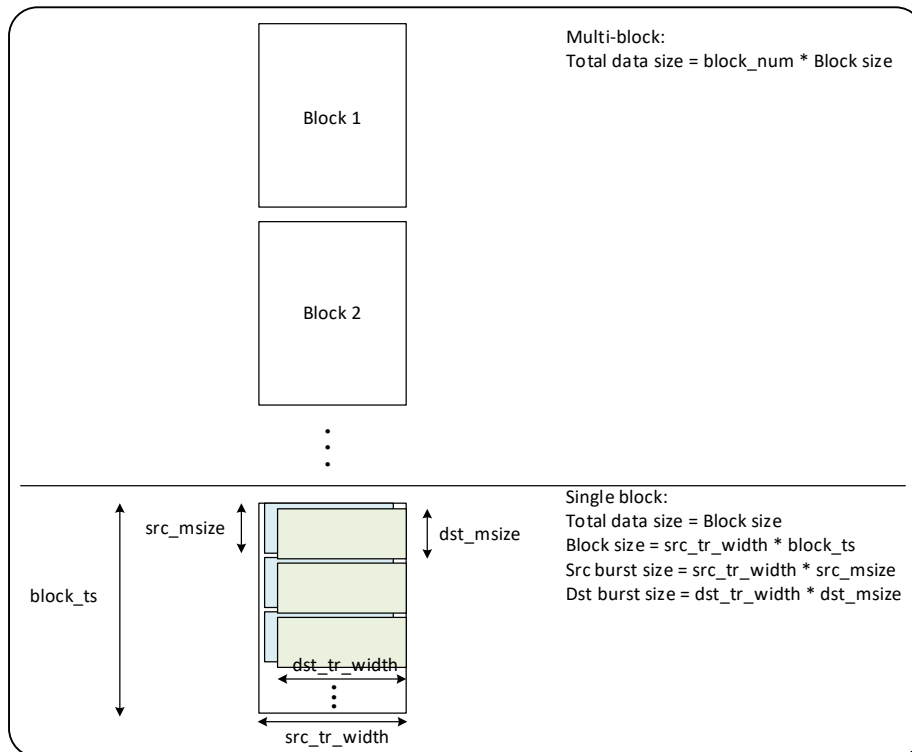


Figure 23-1 DMA block size diagram

23.3.1 GDMA Configuration

23.3.1.1 Data Size

Figure 23-1 illustrates the setting of GDMA transmission data size. The `block_ts` indicates the amount of data that will be transferred in a single data block. It needs to be set to the total number of data/SRC_TR_WIDTH, and max value is 0xFFFF.

23.3.1.2 Transfer Direction and Flow Controller

TT_FC[2:0] field of CTLx register (x is channel)	Direction	Flow controller
000	Memory to Memory	DMAC
001	Memory to Peripheral	DMAC
010	Peripheral to Memory	DMAC
011	Peripheral to Peripheral	DMAC
100	Peripheral to Memory	Peripheral
101	Peripheral to Peripheral	Source Peripheral
110	Memory to Peripheral	Peripheral
111	Peripheral to Peripheral	Destination Peripheral

There are currently four transmission directions and two flow controller settings, with a total of eight available configurations. There are the following differences between a DMAC acting as a flow controller and a peripheral acting as a flow controller:

- When the peripheral acts as a flow controller, , and the DMA transfers data according to the peripheral issues a single/burst request;
- When the DMAC acts as a flow controller, all requests from the peripheral will be processed according to the configured requests.

i NOTE

The `block_ts` parameters can only be set when the DMAC is used as a flow controller.

23.3.1.3 Transfer msize

The length of each transaction can be configured.

- `msize > 1`: burst transaction.
- `msize = 1`: single transaction.

SRC_MSIZ[2:0]/DEST_MSIZ[2:0] field of CTLx register	Transfer msize
000	1
001	4
010	8
011	16
100 and above	Not support

23.3.1.4 Transfer Width

GDMA supports the following transmission width.

SRC_TR_WIDTH[2:0]/DST_TR_WIDTH[2:0] field of CTLx register	Transfer width (byte)
000	1
001	2
010	4
011 and above	Not support

i NOTE

- When reading and writing peripherals, the `SRC_TR_WIDTH/ DST_TR_WIDTH` is completely determined by the width of the peripherals.
- When reading and writing memory:
 - ◆ If cache is disabled, the address does not need to be aligned to any value. It only needs to be `SRC_TR_WIDTH` divisible by the total amount of data so that the `block_ts` is an integer.
 - ◆ If cache is enabled, buffer boundary addresses and cache line alignment are necessary.
- If Memory is destination (P2M, M2M), `DST_TR_WIDTH` parameter will be ignored, and writes are always based on the bus width (the bus width is typically 32 bits, 4 bytes).

23.3.1.5 Transfer Types

23.3.1.5.1 Single block

Single-block DMA transfer – Consists of a single block.

23.3.1.5.2 Multi-block

Multi-block DMA transfer – DMA transfer may consist of multiple RTK_DMAL blocks. Multi-block transfer types include:

- Auto-reloading mode
- Linked list mode

Auto-reloading mode

In auto-reloading mode, the source and destination can independently select which method to use.

Auto-reloading transfer types	Setting	Introduction
Src auto reload	PGDMA_InitTypeDef->GDMA_ReloadSrc = 1; PGDMA_InitTypeDef->GDMA_ReloadDst = 0;	For multi-block transfers, the SAR register can be auto-reloaded from the initial value at the end of each block, and DST address is contiguous, as shown in Figure 23-2 .
Dst auto reload	PGDMA_InitTypeDef->GDMA_ReloadSrc = 0; PGDMA_InitTypeDef->GDMA_ReloadDst = 1;	For multi block transfers, the DAR register can be auto-reloaded from its initial value at the end of each block, and the SRC address is contiguous.
Src & Dst auto reload	PGDMA_InitTypeDef->GDMA_ReloadSrc = 1; PGDMA_InitTypeDef->GDMA_ReloadDst = 1;	For multi block transfers, the SAR and DAR register can be auto-reloaded from its initial value at the end of each block, as shown in Figure 23-3 .

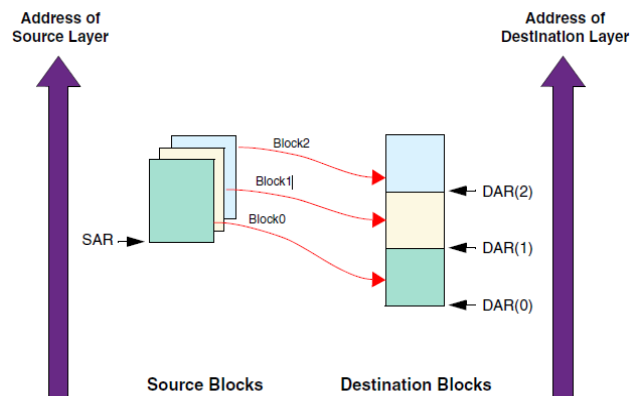


Figure 23-2 Multi-block DMA transfer with source address auto-reloaded and contiguous destination address

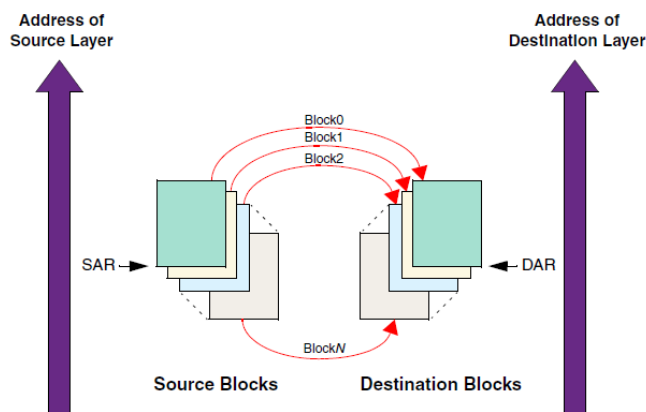


Figure 23-3 Multi-block DMA transfer with source and destination address auto-reloaded

Linked list mode

In linked list mode, the addresses between data blocks do not have to be consecutive.

Link list transfer types	Setting	Introduction
Src: Continue address Dst: Link list	PGDMA_InitTypeDef->GDMA_SrcAddr = pSrc; PGDMA_InitTypeDef->GDMA_LlpDstEn = 1;	Source memory is a continuous data block, while destination data blocks are organized in linked list.
Src: Auto-reloading Dst: Link list	PGDMA_InitTypeDef->GDMA_ReloadSrc = 1; PGDMA_InitTypeDef->GDMA_SrcAddr = pSrc; PGDMA_InitTypeDef->GDMA_LlpDstEn = 1;	In source, SAR register can be auto-reloaded from the initial value at the end of each block, as shown in Figure 23-4 .
Src: Link list Dst: Continue address	PGDMA_InitTypeDef->GDMA_LlpSrcEn = 1; PGDMA_InitTypeDef->GDMA_DstAddr = pDst;	Source memory is organized in the form of a linked list, and destination memory is a continuous data block, as shown in Figure 23-5 .
Src: Link list Dst: Auto-reloading	PGDMA_InitTypeDef->GDMA_LlpSrcEn = 1; PGDMA_InitTypeDef->GDMA_DstAddr = pDst; PGDMA_InitTypeDef->GDMA_ReloadDst = 1;	The source data blocks are organized in a linked list, and the destination data blocks are auto-reloading.
Src: Link list Dst: Link list	PGDMA_InitTypeDef->GDMA_LlpSrcEn = 1; PGDMA_InitTypeDef->GDMA_LlpDstEn = 1;	Both source and destination data blocks are organized in linked lists, and show in Figure 23-6 .

If both the destination and the source are continue data blocks, multi-block transmission should not be used, and single-block transmission is more appropriate.

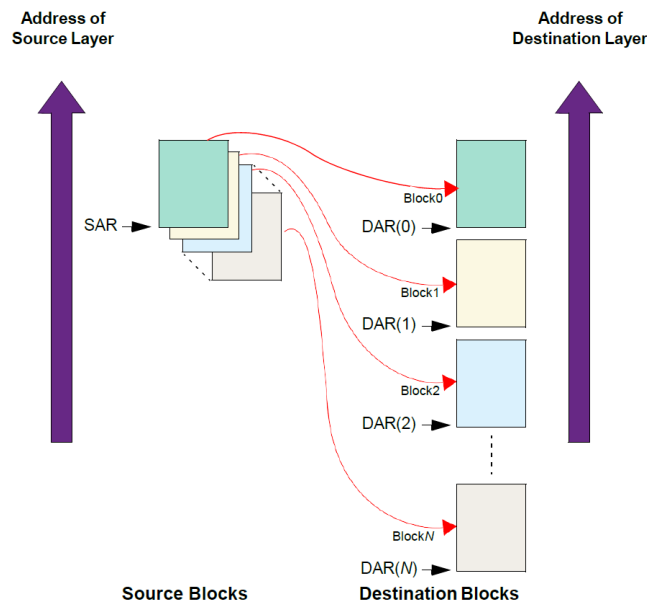


Figure 23-4 Multi-Block DMA transfer with source address auto-reloaded and linked list destination address

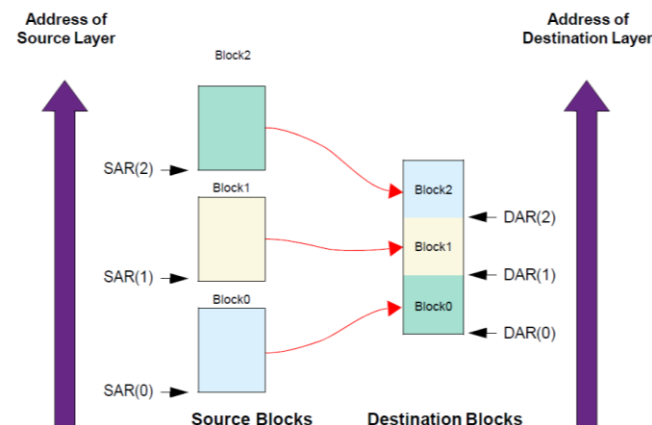


Figure 23-5 Multi-Block DMA transfer with linked list source address and contiguous destination address

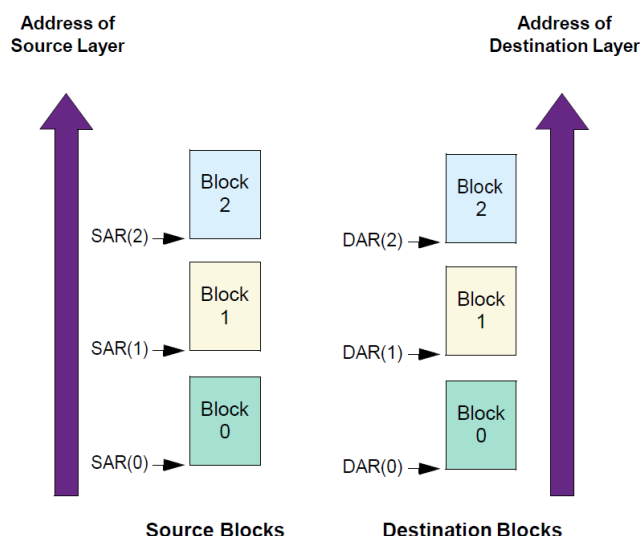


Figure 23-6 Multi-Block with linked address for source and destination

23.3.1.6 Address Increment Type

23.3.1.6.1 Source Address Increment

There are two modes:

- **Increment:** Indicates whether to increment the source address on every source transfer. Incrementing is done for alignment to the next CTLx.SRC_TR_WIDTH boundary.
- **No change:** If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to “No change”.

23.3.1.6.2 Destination Address Increment

There are two modes:

- **Increment:** indicates whether to increment destination address on every destination transfer. Incrementing is done for alignment to the next CTLx.DST_TR_WIDTH boundary.
- **No change:** If the device is writing data to a destination peripheral FIFO with a fixed address, then set this field to “No change”.

23.3.1.7 Real-time Status Acquisition

GDMA supports real-time acquisition of the current transmission source address, destination address and the data size that has been transmitted. Call the corresponding APIs to read.

NOTE

To get the amount of data that has been transferred, the block_ts must be greater than 768 at least, and cannot be read in an interrupt function, otherwise the value obtained is always 0.

23.3.1.8 Interrupt Type

There are several supported interrupt type, these interrupts can be used independently or in combination.

Interrupt type	Introduction
block interrupt	Triggered by the completion of a data block transfer
transfer interrupt	Occurs when all data blocks have been transferred
error interrupt	There was a transfer error

In particular, the transfer-completed condition of the linked list mode is that the pointer of the last data block pointing to the next data block is null.

NOTE

- *In multi-block, when the block in auto-reload mode is interrupted, the data will be transmitted after the interrupt processing function.*
- *In linked-list mode, when the block interruption comes, the data will still continue to be transmitted.*

23.3.1.9 Secure

To start secure transfer, you need to configure the security channel control bit in the register. Access for master interface and slave interface are secure when the secure bit is set. Secure channel can only be configured in secure world. Secure channel can access secure memory and non-secure memory. Non-secure channel can only access non-secure memory.

```
PGDMA_InitTypeDef->SecureTransfer = 1;
```

23.3.1.10 Suspend and Abort

GDMA supports channel suspend resume and termination.

- To suspend a channel, just configure CFGx.CH_SUSP, but there is no guarantee that the current data transaction is completed. Combined with CFGx.INACTIVE, the channel can be safely paused without losing data.
- To resume data transmission after suspension, clear CFGx.CH_SUSP.
- To terminate data transfer, CFGx.INACTIVE must be continuously polled until this bit is set to 1, then the data transfer can be aborted.

NOTE

The following is situation that channels is inactive:

- CFGx.INACTIVE can only be activated after Memory has been written, and then canceled.
- The data of peripheral is 4bytes, but the FIFO of DMAC is only 2 bytes. There is no writing at this time and CFGx.INACTIVE is activated directly.

23.3.1.11 Priority

GDMA supports two kinds of channel priority:

- Software: the priority of each channel can be configured in the CFGx.CH_PRIOR. The valid value is 0 ~ (DMAC_NUM_CHANNELS-1), where 0 is the highest priority value and (DMAC_NUM_CHANNELS-1) is the lowest priority value.
- Hardware: if two channel requests have the same software priority level, or if no software priority is configured, the channel with the lower number takes priority over the channel with the higher number. For example, channel 2 takes priority over channel 4.

23.3.1.12 Cache

When DMA slave type is memory, you need to pay attention to cache operation. DCache_CleanInvalidate() should be called every time before DMA transmission starts.

The following steps should be added when executing DMA Rx/Tx.

Operation	Step
DMA Rx	(1) Prepare DST buffer (2) Do DCache_CleanInvalidate() to avoid cache flush during DMA Rx (3) Do DMA Rx configuration (4) Trigger DMA Rx interrupt (5) Do DCache_Invalidate() in Rx Done Handler to clean the old data, to avoid the problem that the data in the cache is inconsistent with the data in the memory after Rx done if the CPU read or write allocate the DST buffer during GDMA transmission. NOTE During GDMA transmission, it is forbidden to write or cache flush DST buffer. (Taking {SDK}\component\example\peripheral\raw\uart\uart_dma_stream\src\main.c for example, uart_rcv_string_done is DMA Rx Done Interrupt Handler) <pre>u32 uart_rcv_string_done(void *data) { UNUSED(data); /* To solve the cache consistency problem, DMA mode needs it */ DCache_Invalidate((u32)rx_buf, SRX_BUF_SZ); dma_free(); rx_done = 1; return 0; }</pre> (6) CPU reads DST buffer
DMA Tx	(1) CPU prepares SRC buffer data (2) Do DCache_CleanInvalidate() for SRC buffer to synchronize the data

- | | |
|-----|--------------------------|
| (3) | Do DMA Tx configuration |
| (4) | Trigger DMA Tx interrupt |

Aligning the buffer address with the cache line will reduce the problem of inconsistent cache and memory data, and details for referring to [Cache Consistency When Using DMA](#).

23.3.2 Demo for Single Block

- (1) Allocate a free channel

```
ch_num = GDMA_ChnlAlloc(gdma.index, (IRQ_FUN) Dma_memcpy_int, (u32)(&gdma), 3);
```

This function also includes the following operation:

- Register IRQ handler if use interrupt mode
- Enable NVIC interrupt
- Register the GDMA channel to use

- (2) Configure interrupt type

```
PGDMA_InitTypeDef->GDMA_IsrType = (TransferType | ErrType);
```

- (3) Configure interrupt handling function

Clear the pending interrupt in the interrupt processing function.

```
GDMA_ClearINT(0, PGDMA_InitTypeDef->GDMA_ChNum);
```

- (4) Configure transfer settings

```
PGDMA_InitTypeDef->GDMA_SrcMsize = MsizeEight;
PGDMA_InitTypeDef->GDMA_SrcDataWidth = TrWidthFourBytes;
PGDMA_InitTypeDef->GDMA_DstMsize = MsizeEight;
PGDMA_InitTypeDef->GDMA_DstDataWidth = TrWidthFourBytes;
PGDMA_InitTypeDef->GDMA_BlockSize = DMA_CPY_LEN >> 2;
PGDMA_InitTypeDef->GDMA_DstInc = IncType; // if dst type is peripheral:no change
PGDMA_InitTypeDef->GDMA_SrcInc = IncType; // if src type is peripheral:no change
```

- (5) Configure hardware handshake interface if slave is peripheral

```
GDMA_InitStruct->GDMA_SrcHandshakeInterface= GDMA_HANDSHAKE_INTERFACE_AUDIO_RX;
```

or

```
GDMA_InitStruct->GDMA_DstHandshakeInterface = GDMA_HANDSHAKE_INTERFACE_AUDIO_TX;
```

- (6) Configure the transfer address

```
PGDMA_InitTypeDef->GDMA_SrcAddr = (u32)BDSrcTest;
```

```
PGDMA_InitTypeDef->GDMA_DstAddr = (u32)BDDstTest;
```

- (7) Program GDMA index, GDMA channel, data width, Msize, transfer direction, address increment mode, hardware handshake interface, reload control, interrupt type, block size, multi-block configuration and the source and destination address using the GDMA_Init() function.

```
GDMA_Init(gdma.index, gdma.ch_num, PGDMA_InitTypeDef);
```

- (8) Cache clean invalidate

```
DCache_CleanInvalidate();
```

- (9) Enable GDMA channel

```
GDMA_Cmd(gdma.index, gdma.ch_num, ENABLE);
```

23.3.3 Demo for Multi-block

This example is SRC auto reload, compared with single block, multi-block is different in steps (2) ~ (4).

- (1) Allocate a free channel

```
ch_num = GDMA_ChnlAlloc(gdma.index, (IRQ_FUN) Dma_memcpy_int, (u32)(&gdma), 3);
```

This function also includes the following operation:

- Register IRQ handler if use interrupt mode
- Enable NVIC interrupt
- Register the GDMA channel to use

- (2) Configure interrupt type

```
PGDMA_InitTypeDef->GDMA_IsrType = (BlockType | TransferType | ErrType);
```

- (3) Configure interrupt handling function

- a) Clear the interrupt.

```
GDMA_ClearINT(0, GDMA_InitStruct->GDMA_ChNum);
```

- b) Clear the auto reload mode before the last block starts.

```
GDMA_ChCleanAutoReload(0, GDMA_InitStruct->GDMA_ChNum, CLEAN_RELOAD_SRC);
```

- (4) Configure transfer settings

```
PGDMA_InitTypeDef->GDMA_SrcMsize = MsizeEight;
PGDMA_InitTypeDef->GDMA_SrcDataWidth = TrWidthFourBytes;
PGDMA_InitTypeDef->GDMA_DstMsize = MsizeEight;
```

```
PGDMA_InitTypeDef->GDMA_DstDataWidth = TrWidthFourBytes;
PGDMA_InitTypeDef->GDMA_BlockSize = DMA_COPY_LEN >> 2;
PGDMA_InitTypeDef->GDMA_DstInc = IncType; // If DST type is peripheral: no change
PGDMA_InitTypeDef->GDMA_SrcInc = IncType; // If SRC type is peripheral: no change
PGDMA_InitTypeDef->GDMA_ReloadSrc = 1;
PGDMA_InitTypeDef->GDMA_ReloadDst = 0;
```

- (5) Configure hardware handshake interface if slave is peripheral.

```
GDMA_InitStruct->GDMA_SrcHandshakeInterface = GDMA_HANDSHAKE_INTERFACE_AUDIO_RX;
```

or

```
GDMA_InitStruct->GDMA_DstHandshakeInterface = GDMA_HANDSHAKE_INTERFACE_AUDIO_TX;
```

- (6) Configure the transfer address

```
PGDMA_InitTypeDef->GDMA_SrcAddr = (u32)BDSTest;
PGDMA_InitTypeDef->GDMA_DstAddr = (u32)BDDstTest;
```

- (7) Program GDMA index, GDMA channel, data width, Msize, transfer direction, address increment mode, hardware handshake interface, reload control, interrupt type, block size, multi-block configuration and the source and destination address using the GDMA_Init() function.

```
GDMA_Init(gdma.index, gdma.ch_num, PGDMA_InitTypeDef);
```

- (8) Cache clean invalidate

```
DCache_CleanInvalidate();
```

- (9) Enable GDMA channel

```
GDMA_Cmd(gdma.index, gdma.ch_num, ENABLE);
```

24 Pseudo-Static Random Access Memory (PSRAM)

24.1 Introduction

Pseudo-Static Random Access Memory (PSRAM) is used for high-speed transmission of the data stream. The RTL8721Dx communicates with PSRAM via PSRAM controller (PSRAMC). PSRAM can be accessed by KM4 and KM0, and supports execution on PSRAM.

The features of PSRAM are:

- Clock rate: up to 200MHz
- Double Data Rate (DDR)
- Read-Write Data Strobe (DQS)
- Supports Half sleep and deep power-down mode
- Programmable drive strength
- Temperature Compensated Refresh
- 16/32/64/128 bytes wrap burst access
- Distributed refresh interval varies with temperature
- Address mapping: 0x6000_0000~0x6040_0000

24.2 Throughput

PSRAM supports direct access and DMA access. The throughput of PSRAM is listed in [Table 24-1](#).

Table 24-1 Throughput of PSRAM (200MHz)

Access mode	Writing 32 bytes		Reading 32 bytes	
	Theory	Test on the KM4	Theory	Test on the KM4
Direct access (write back)	1523.81Mbps	$(32 \times 8) / (199.68\text{ns}) = 1282.05\text{Mbps}$	1454.55Mbps	$(32 \times 8) / (212.16\text{ns}) = 1204.14\text{Mbps}$
DMA access	2206.9Mbps	1641.03Mbps	2133.33Mbps	1172.16Mbps

NOTE

- **Throughput theoretical calculation:**
 - ◆ The test data above takes variable initial latency, so there will be 1 or 2 times initial latency depending on RWDS.
 - ◆ The header overlaps with delay by 1T.
 - ◆ Since it is DDR PSRAM, 16T is used to transmit 32 bytes.
- **Direct access:**
 - ◆ Since the efficiency of accessing PSRAM will be reduced when cache is off, it is recommended to use cache on to access PSRAM. Write back only writes to the cache during write operation (when cache miss, data will be loaded from PSRAM into cache line first). Only when cache replacement or manual cache clearing occurs, the modified cache data (marked with dirty) will be written to PSRAM. In [Table 24-1](#), after the 32bit operation is written, the cache clean operation is performed.
 - ◆ Read operation reads 32 bytes (cache line) once.
 - ◆ Write operation writes 32 bits once, but the interval between two write operations is greatly reduced.
 - ◆ Instruction execution time also needs to take into consideration.
- **DMA sets burst length 64 bytes and disables the cache.**
 - ◆ For DMA write, DMA 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, a 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.

Table 24-2 PSRAM throughput theoretical calculation

Item	Writing 32 bits	Reading 32 bits
Header + delay	$[3 + 22] \times 4\text{ns} = 100\text{ns}$	$[3 + 23] \times 4\text{ns} = 104\text{ns}$
Data transmit period	$2 \times 4\text{ns} = 8\text{ns}$	$16 \times 4\text{ns} = 64\text{ns}$
Hardware hold	$1 \times 4\text{ns} = 4\text{ns}$	$2 \times 4\text{ns} = 8\text{ns}$
Total without considering instruction execution time	$100\text{ns} + 8\text{ns} + 4\text{ns} = 112\text{ns}$	$104\text{ns} + 64\text{ns} + 8\text{ns} = 176\text{ns}$
Throughput theoretical value	$32 / 112\text{ns} = 285.71\text{Mbps}$	$(32 \times 8) / 176\text{ns} = 1454.55\text{Mbps}$

24.3 Boot from PSRAM

If the PSRAM is embedded in the chip, follow these steps to boot from PSRAM in the SDK.

- (1) Enable the power supply of PSRAM in the bootloader
- (2) Initialize the PSRAM controller and PSRAM device to synchronize the relevant parameters
- (3) Calibrate the PSRAM

```
RCC_PeriphClockCmd(APBPeriph_PSRAM, APBPeriph_PSRAM_CLOCK, ENABLE);
DBG_PRINT(MODULE_BOOT, LEVEL_INFO, "Init PSRAM\r\n");
BOOT_PSRAM_Init();
```

24.4 PSRAM Cache “Write Back” Policy

When a cache hit occurs on a store access, the data is only written to the cache. Data in the cache can therefore be more up-to-date than data in memory. Any such data is written back to memory when the cache line is cleaned or reallocated. Another common term for a write-back cache is a copy-back cache.

24.4.1 Row Hammer

With the increasing density of DRAM, its memory cells become smaller and smaller, and the stored charge decreases. As a result, the noise tolerance between memory cells is reduced, resulting in the interaction of charges between two independent memory cells. Row hammer is caused by this defect in the design of memory hardware chip. Its principle is to repeatedly read and write the peer address in DRAM memory unit, so that the charge leakage occurs in adjacent rows, and the bit reversal phenomenon occurs in adjacent rows, that is, 0 is reversed to 1, and 1 is reversed to 0.

Therefore, when a large number of accesses are made to PSRAM in a short time, if the refresh frequency is not enough, the MEM space of every 2K (i.e. two rows) will affect each other. When we perform a large number of continuous write operations on a line, the charges of adjacent lines will be affected and the value will change.

24.4.2 Notice

24.4.2.1 Cache Operation

On the “Write Back” policy, the synchronization operations need to be taken between cache and PSRAM to keep content consistency, especially for multiple access by different sources, e.g. CPU, serial ports and peripherals.

As the cache line of KM4/KM0 cache is 32 bytes, and cache operations are all based on the cache line. So the buffer size and buffer starting address are recommended to be 32/64 bytes aligned to avoid synchronization issues.

24.4.2.2 DMA Operation

The following steps should be added when executing DMA Rx/Tx.

Operation	Step
DMA Rx	(1) Prepare Rx buffer (2) Do DCache_CleanInvalidate() to avoid cache data write back during DMA Rx (3) Do DMA Rx config (4) Trigger DMA Rx interrupt (5) Do DCache_Invalidate() in Rx Done Handler to clean the old data NOTE During GDMA transmission, it is forbidden to write or cache flush DST buffer. (Taking {SDK}\component\example\peripheral\raw\uart\uart_dma_stream\src\main.c for example, uart_recv_string_done is DMA Rx Done Interrupt Handler) <pre>u32 uart_recv_string_done(void *data) { UNUSED(data); /* To solve the cache consistency problem, DMA mode needs it */ DCache_Invalidate((u32)rx_buf, SRX_BUF_SZ); dma_free(); }</pre>

		<pre> rx_done = 1; return 0; } </pre>
	(6)	CPU reads Rx Buffer
DMA Tx	(1)	CPU prepares Tx buffer data
	(2)	Do DCache_CleanInvalidate() for Tx buffer to synchronize the data
	(3)	Do DMA Tx configuration
	(4)	Trigger DMA Tx interrupt

In SDK, only the example of one-time xxx_GDMA_Init one-time transmission is illustrated. Step (2) is included in xxx_GDMA_Init by default.

If you need multi-time DMA Tx/Rx with only one-time xxx_GDMA_Init, DCache_CleanInvalidate() should be called every time before DMA transmission starts.

```

BOOL UART_TXGDMA_Init(
    u8 UartIndex,
    GDMA_InitTypeDef *GDMA_InitStruct,
    void *CallbackData,
    IRQ_FUN CallbackFunc,
    u8 *pTxBuf,
    int TxCount
)
{
    u8 GdmaChnl;

    assert_param(GDMA_InitStruct != NULL);

    DCache_CleanInvalidate((u32)pTxBuf, TxCount);
}

```

```

BOOL UART_RXGDMA_Init(
    u8 UartIndex,
    GDMA_InitTypeDef *GDMA_InitStruct,
    void *CallbackData,
    IRQ_FUN CallbackFunc,
    u8 *pRxBuf,
    int RxCount
)
{
    u8 GdmaChnl;
    UART_TypeDef *UARTx;

    assert_param(GDMA_InitStruct != NULL);

    DCache_CleanInvalidate((u32)pRxBuf, RxCount);
}

```

24.5 TCEM Setting

The TPR0[24:31] (CS_TCEM) provides the function that when the CSN low pulse width is equal to (CS_TCEM * 32)*busclk, the SPI Flash Controller will automatically chop the current transmission and pull CS up.

24.5.1 Winbond

When the temperature is less than 85°C, PSRAM refresh the intern cell array using normal rate (4us). When the temperature is greater than 85°C and less than 125°C, PSRAM refresh the internal cell array using faster rate (1us). This sets an upper limit on the length of read and write transactions so that the automatic distributed refresh operation can be done between transactions. This limit is called the CS# low maximum time (tCSM) and the tCSM will be equal to the maximum distributed refresh interval.

So when the temperature is less than 85°C, for higher performance, we recommend that CS_TCEM should be equal to 4us/busclk/32. When the temperature is greater than 85°C, the value should be equal to 1us/busclk/32.

24.5.2 APM

APM is in extended mode by default, so it always keeps fast refresh (1us). Here, CS_TCEM is recommended equal to 1us/busclk/32.

25 General Purpose Input/Output (GPIO)

25.1 Output

To do GPIO output test, the following steps are mandatory.

- (1) Select GPIO pin, and set GPIO mode to output.

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
```
- (2) Initialize hardware using the parameters in step (1).

```
GPIO_Init(GPIO_InitTypeDef *GPIO_InitStruct);
```
- (3) Write a logic value to a specified output port pin.

```
GPIO_WriteBit(u32 GPIO_Pin, u32 Pin_State);
```

25.2 Input

To do GPIO input test, the following steps are mandatory.

- (1) Select GPIO pin, and set GPIO mode to input.

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
```
- (2) Initialize hardware using the parameters in step (1).

```
GPIO_Init(GPIO_InitTypeDef *GPIO_InitStruct);
```
- (3) Read a specified output port pin.

```
GPIO_ReadDataBit(u32 GPIO_Pin);
```

25.3 Interrupt

To do GPIO interrupt test, the following steps are mandatory.

- (1) Select GPIO pin, set GPIO pin mode to interrupt, no pull, and configure interrupt trigger type, polarity, debounce.

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_INT;
GPIO_InitStruct.GPIO_ITTrigger = trigger_type;
GPIO_InitStruct.GPIO_ITPolarity = polarity;
GPIO_InitStruct.GPIO_ITDebounce=GPIO_INT_DEBOUNCE_ENABLE(or GPIO_INT_DEBOUNCE_DISABLE);
```
- (2) Initialize hardware using the parameters in step (1).

```
GPIO_Init(GPIO_InitTypeDef *GPIO_InitStruct);
```
- (3) Register and enable the GPIO interrupt.

```
InterruptRegister(GPIO_INTHandler, GPIO_DEV_TABLE[port].IrqNum, (u32)GPIO_DEV_TABLE[port].GPIOx, 10);
InterruptEn(GPIO_DEV_TABLE[port].IrqNum, 10);
```
- (4) (Optional) Configure the GPIO interrupt mode.

```
GPIO_INTMode(u32 GPIO_Pin, u32 NewState, u32 GPIO_ITTrigger, u32 GPIO_ITPolarity, u32 GPIO_ITDebounce);
```
- (5) Register user handle.

```
GPIO_UserRegIrq(GPIO_Pin, gpio_test_irq_handler, (&GPIO_InitStruct_Temp));
```
- (6) Enable the specified GPIO pin interrupt.

```
GPIO_INTConfig(GPIO_Pin, ENABLE);
```

26 Pin Controller (Pinctrl)

To configure the pin multiplexing function, follow the steps below:

- (1) Turn off the SWD if the pin default function is SWD, and configure it to other functions.

```
Pinmux_Swdoff();
```

- (2) Configure pinmux function.

```
Pinmux_Config(u8 PinName, u32 PinFunc);
```

- (3) Set pin pull type.

```
PAD_PullCtrl(u8 PinName, u8 PullType); //normal mode
```

```
PAD_SleepPullCtrl(u8 PinName, u8 PullType); //sleep and deep-sleep mode
```

- (4) Set driving strength if needed.

```
PAD_DrvStrength(u8 PinName, u32 DrvStrength);
```

27 Pin Multiplexing Instructions

27.1 Introduction

The RTL8721Dx provides a pin multiplexing (pinmux) circuit to maximize the user's freedom under limited pin-out conditions. Each pin can be connected to different internal IP circuits through configuration. For the specific correspondence between each pin and IP circuit, refer to the provided pinmux table.

Before using the chip for further development, pay attention to the following precautions about pinmux to avoid inconvenience to your use due to unexpected behavior.

27.2 Trap Pins

During the process of powering on the chip, the internal circuit will latch several pins' conditions to decide whether entering into different modes. The trap pins and descriptions are listed in [Table 27-1](#).

Table 27-1 Description of trap pins

Pin name	Symbol	Active level	Description
PB31	TM_DIS	Low	Test Mode Disable, default internal pull up. It is for internal test only and should be logical high for normal operation. 1: Normal operation mode 0: Test mode
PB5	UD_DIS	Low	UART Download Disable, default internal pull up 1: Enter into normal boot mode 0: Enter into UART download mode

i NOTE

The trap pin needs to select the external pull-up and pull-down voltages according to the I/O power supply.

27.3 Wake Pins

PB30 and PB31 are directly connected to the wake up circuit which is used to wake up system from deep-sleep state. If you want to use other functions on this pin, disable the wake up function first.

27.4 SWD Pins

PA30 and PA31 are forced to SWD/JTAG function by default. If you want to multiplex these two pins to other functions, call `sys_jtag_off()` or `pinmux_Swdoff()` before switching.

27.5 Function Multiplexing

27.5.1 Function ID 0-18

For functions whose ID number is among 0-18, each pin can only be connected to a fixed signal of a certain IP. The functions that can be configured on a pin are very limited, but a dedicated design can maximize the performance of each IP.

i NOTE

For example, function ID 8 and function ID 29-32 are both SPI functions. Since function ID 8 is a dedicated pin, the maximum speed of the SPI function reaches 50MHz (master mode); while the maximum speed of the pins (full-cross pins) corresponding to function ID 29-32 is only 12.5MHz (master mode).

Take PB30 as an example. If you configure function ID of PB30 to 1, then the pin will be directly connected to the UART1_RXD signal of the UART1 via pinmux.

Refer to the pinmux table for the specific function distribution available on each pin.

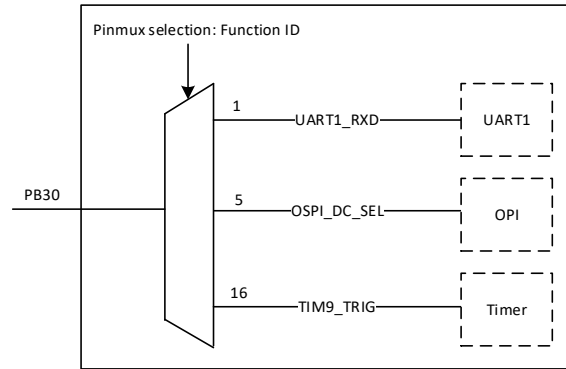


Figure 27-1 Schematic diagram of pinmux connection of PB30

27.5.2 Function ID 19-81

For functions whose ID number is after 19, each pin can be connected to different signals of a certain IP. This method maximizes the freedom of use, but the scope of use and some IPs' performance (maximum transfer speed) is limited.

Take PA12 as an example. According to the pinmux table, you can connect PA12 with the UART0_TXD signal of UART0 by configuring the PA11 function ID to 19. You can also configure the PA12 function ID to 20, and connect PA12 with the UART0_RXD signal of UART0. For details, refer to the pinmux table.

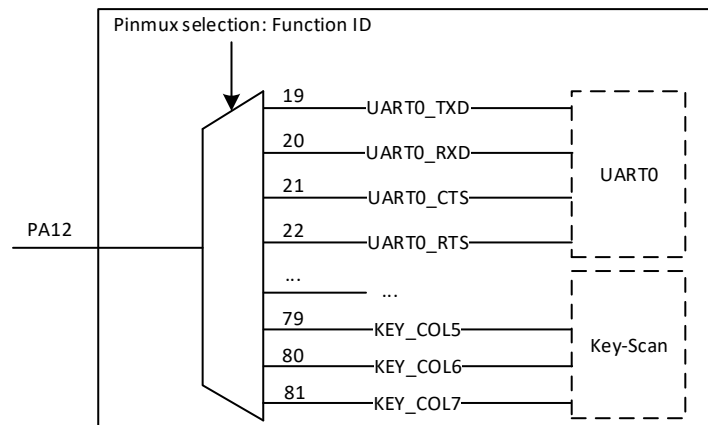


Figure 27-2 Schematic diagram of pinmux connection of PA12

28 RTC_IO

28.1 Introduction

The RTC_IO module serves to temporarily store RTC time data when the chip is powered off. Additionally, it records the duration of time the chip remains powered off. This stored time data and duration can be read back when the chip is powered on again.

Software can transmit data to the RTC_IO module, with support limited to LDO_RCAL currently, representing the calibration value of OSC32K. The RTC_IO module maintains the OSC32K calibration function even during power off.

i NOTE

The RTC module will send RTC time data to RTC_IO module periodically. The RTC module's register PREDIV_A[8:0] can't be set to be greater than the default value (0x80).

28.2 Usage

- (1) (Optional) Enable work mode to rtc_mode. Perform this step, RTC_IO can receive command.

```
RTCIO_SetWorkMode(MODE_RTC_START);
```

i NOTE

When RTC module is enabled, the RTC_IO module will enter rtc_mode automatically.

- (2) Check whether is RTC_IO function enabled or not. If not, go to step (5) directly.

```
RTCIO_IsEnabled();
```

- (3) Read out the stored time data and duration during power off. The info has been placed into struct RTCIO_TimeInfo.

```
RTCIO_GetTimeInfo(RTCIO_TimeInfo *pDataOut);
```

- (4) Calculate the new time according to struct RTCIO_TimeInfo from step (3), and set the new time to the device.

- (5) Reset LDO_RCAL in RTC_IO module, and calibrate OSC131K clock source.

```
RTCIO_SetRValue(RTCIO_RECV_RVAL_RST);
OSC131K_Calibration(30000);
```

- (6) Update new calibrated value LDO_RCAL to RTC_IO module.

```
RTCIO_SetRValue(RTCIO_RECV_RVAL_CAL);
```

- (7) (Optional) Initialize RTC module and set RTC time. If RTC module has been enabled before, this step can be ignored.

29 Infrared Radiation (IR)

29.1 Introduction

The Infrared Radiation (IR) provides hardware modulation for Infrared Radiation sending and hardware auto capture for receiving.

This chapter introduces how to use IR.

29.2 IR Sending

29.2.1 Tx Polling Mode

To use IR sending function, the following steps are mandatory.

- (1) Configure the IR pin according to the pinmux table.
- (2) For example, in order to use PB4 as IR Tx pin, call the following function. It is the same for other IR pins.

```
Pinmux_Config(_PB_4, PINMUX_FUNCTION_IR_TX);
```

- (3) Call `IR_Cmd()` to disable IR.
 - (4) Set parameters, and modify some parameters if needed.
- ```
IR_StructInit(IR_InitTypeDef *IR_InitStruct);
```
- (5) Initialize the hardware using the parameters in step (4).
- ```
IR_Init(IR_InitTypeDef *IR_InitStruct);
```
- (6) Write Tx data to FIFO by using `IR_SendBuf()` or `IR_SendData()`.
 - (7) Call `IR_Cmd()` to enable IR to start transmission.
 - (8) Write more data to FIFO if needed.

NOTE

- In step (3) to step (7), it is suggested that disabling IR at first, and then enabling IR after writing data to FIFO.
- In step (6), pay attention to convert the data into the appropriate format that Tx FIFO register can recognize before writing data to FIFO.

29.2.2 Special Notes

29.2.2.1 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`.

29.2.2.2 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 write some data packets to Tx FIFO before enabling IR Tx. Refer to step (2) and step (6) in Section 29.2.1.

29.3 IR Receiving

29.3.1 Rx Interrupt Mode

To use IR receiving function, the following steps are mandatory.

- (1) Configure the IR pin according to the pinmux table.
- For example, in order to use PB5 as IR Rx pin, call the following function. It is the same for other IR pins.
- ```
Pinmux_Config(_PB_5, PINMUX_FUNCTION_IR_RX);
```
- (2) Set parameters, such as sampling frequency, Rx FIFO threshold level, Rx counter threshold type, Rx counter threshold level, and Rx trigger mode if needed.
- ```
IR_StructInit(IR_InitTypeDef *IR_InitStruct);
```
- (3) Initialize the hardware using the parameters in step (2).
- ```
IR_Init(IR_InitTypeDef *IR_InitStruct);
```
- (4) Configure the interrupt if needed and register the 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) Call `IR_ClearRxFIFO()` to clear Rx FIFO.
- (7) When Rx FIFO threshold interrupt triggers, read data from Rx FIFO with `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.
- 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`.

## 29.3.2 Rx Learning

The process of Rx learning is similar to common Rx. As shown in [Figure 29-1](#), 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.

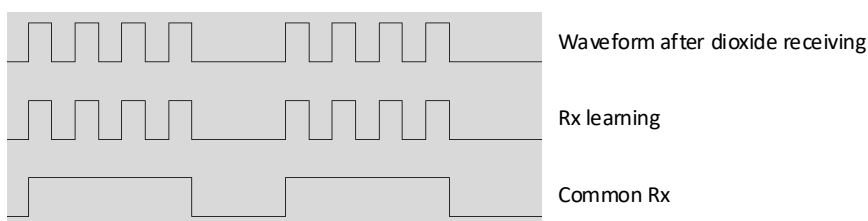


Figure 29-1 Difference of waveform between Rx learning and common Rx

**NOTE**

- It is advised that putting the interrupt handle function code in RAM, and close other peripheral interrupts to avoid the interfere.
- If the carrier frequency of learning waveform is larger than 400kHz, the hardware may cannot respond to the interrupt in time, which will result in decoding carrier frequency failed.

# 30 Light Emitting Diode Controller (LEDC)

## 30.1 Introduction

LEDC is used to drive external smart LEDs, like WS2812B. LEDC supports DMA mode and interrupt mode, with 32\*24bit LEDC FIFO. LEDC also supports RGB888 display mode and configurable refresh time.

## 30.2 LEDC Control Application

Taking WS2812C-2020 LED for example, before using this LED, we have to configure the logical character '0' and character '1' according to the LEDC data input code of typical LED datasheet, as shown in [Table 30-1](#).

Table 30-1 Typical LED modulation datasheet

| Modulate code | Description                     | Time range | Unit |
|---------------|---------------------------------|------------|------|
| T0H           | Digital 0 code, high-level time | 220 ~ 380  | ns   |
| T0L           | Digital 0 code, low-level time  | 580 ~ 1000 | ns   |
| T1H           | Digital 1 code, high-level time | 580 ~ 1000 | ns   |
| T1L           | Digital 1 code, low-level time  | 220 ~ 420  | ns   |
| RESET         | Frame unit, low-level time      | > 280      | μs   |

Logical “0” coding time is T0-code: 800ns ~ 1980ns, and logical “1” coding time is T1-code: 800ns ~ 2020ns. The supported LED number is:

$$LED\_NUM = \frac{\frac{1}{LED \text{ fresh rate}(\frac{frame}{s})} - 280us}{(logical \ 0 \ or \ 1 \ coding \ time) * 24}$$

- When LED's refresh rate is 30 frames/s, the supported LED number is 681 to 1023.
- When LED's refresh rate is 60 frames/s, the supported LED number is 337 to 853.

The related registers of LED setting are LED\_T0&T1\_TIMING\_CTRL\_REG, LEDC\_DATA\_FINISH\_CNT\_REG, LED\_RESET\_TIMING\_CTRL\_REG, LEDC\_WAIT\_TIME\_CTRL\_REG, LEDC\_DMA\_CTRL\_REG, and LEDC\_INTERRUPT\_CTRL\_REG.

WS2812 serious LED data transfer protocol uses a single NRZ (non-return zero) communication mode. After the LED power-on reset, the DI port receives data from the controller, the first LED collects initial 24-bit data then sends 24-bit data to the internal data latch, the other data reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission of each pixel, the signal reduces 24 bits. Pixel adopts auto reshaping transmit technology, making the pixel cascade number is not limited by the signal transmission, only depending on the speed of signal transmission.

## 30.3 Operation Flow

The flow of LEDC normal configuration process is shown in [Figure 30-1](#).

- (1) Configure LEDC data mode through LED\_RGB\_MODE field, LED\_MSB\_TOP bit, LED\_MSB\_BYTE2 bit, LED\_MSB\_BYTE1 bit and LED\_MSB\_BYTE0 bit in the LEDC\_CTRL\_REG register.
- (2) Configure LEDC control mode through the registers: LED\_T01\_TIMING\_CTRL\_REG, LEDC\_DATA\_FINISH\_CNT\_REG, LED\_RESET\_TIMING\_CTRL\_REG, LEDC\_WAIT\_TIME0\_CTRL\_REG, LEDC\_WAIT\_TIME1\_CTRL\_REG, LEDC\_DMA\_CTRL\_REG and LEDC\_INTERRUPT\_CTRL\_REG.
- (3) Enable the LEDC by writing 1 to the LEDC\_EN bit in LEDC\_CTRL\_REG register.
- (4) If LEDC\_DMA\_EN bit in LEDC\_DMA\_CTRL\_REG register is set to 1, go to (5). If not, go to (7).
- (5) LEDC sends dma\_req to DMAC.
- (6) DMAC transfers data from memory to LEDC FIFO by APB bus and sends dma\_ack signal following the last data. Go to (8).
- (7) LEDC sends CPU\_REQ\_INT signal to CPU. DMAC is required to transfer data from memory to LEDC. Go to (8).
- (8) LEDC transfers data from LEDC FIFO to LED lamps.
- (9) When LED\_DATA\_FINISH\_CNT field in LEDC\_DATA\_FINISH\_CNT\_REG register equals TOTAL\_DATA\_LENGTH set in LEDC\_CTRL\_REG register, LED\_TRANS\_FINISH\_INT interrupt is generated, and hardware resets LEDC\_EN to 0.
- (10) Clear LED\_TRANS\_FINISH\_INT interrupt by software.
- (11) It is suggested that asserting LEDC\_SOFT\_RESET bit in LEDC\_CTRL\_REG register to clear all interrupt status registers.
- (12) If software wants to scan LED lamps again, go to (1).

### NOTE

For more details, refer to the document of mbed APIs.

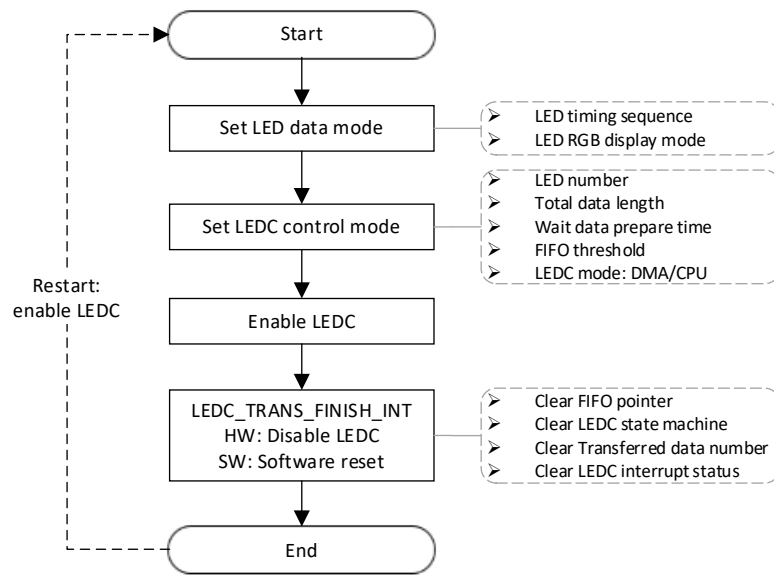


Figure 30-1 LEDC normal configuration process

# 31 Universal Serial Bus On-The-Go (USB OTG)

## 31.1 Introduction

### 31.1.1 Features

The features of Realtek USB stack are listed below:

- USB 2.0 full-speed device mode
- Unified HAL API for all Realtek Ameba SoC series
- Device-only API for class and application development
- Device classes: CDC ACM, Composite, HID and Vendor
- Device descriptor full customizable
- NOT supported: OTG/host mode, high/low-speed device mode

### 31.1.2 Architecture

The software architecture of USB stack is as below:

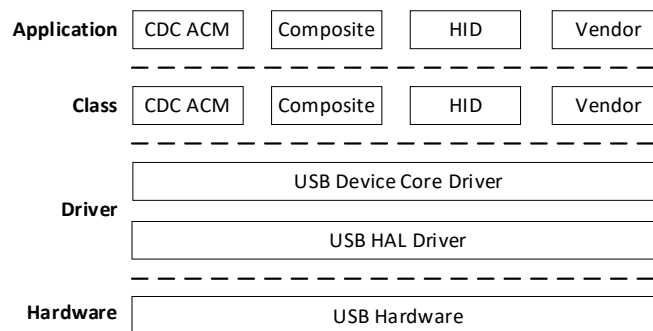


Figure 31-1 USB stack architecture

The function of each layer:

- USB HAL driver: implements the SoC-specific USB functions and provides unified USB HAL API for USB device core drivers
- USB device core driver: implements the USB device-specific functions and provides unified USB device API for USB device classes
- USB class: implements the USB classes as per USB IF specifications
- USB application: implements the USB applications corresponding with the USB classes

### 31.1.3 Implementation

The USB stack is implemented in the following files:

| Class     | Transfer                                    | Class directory                | Application directory                |
|-----------|---------------------------------------------|--------------------------------|--------------------------------------|
| CDC ACM   | CTRL, BULK IN/OUT, INTR IN                  | component\usb\device\cdc_acm   | component\example\usb\usbd_cdc_acm   |
| Composite | CTRL, BULK IN/OUT, INTR IN/OUT              | component\usb\device\composite | component\example\usb\usbd_composite |
| HID       | CTRL, INTR IN/OUT                           | component\usb\device\hid       | component\example\usb\usbd_hid       |
| Vendor    | CTRL, BULK IN/OUT, INTR IN/OUT, ISOC IN/OUT | component\usb\device\vendor    | component\example\usb\usbd_vendor    |

## 31.2 Configuration

- (1) Type "make menuconfig" command under {SDK}\amebadplus\_gcc\_project, then select MENUCONFIG FOR KM4 CONFIG > CONFIG USB:

```

< MENUCONFIG FOR KM4 CONFIG
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help.
Legend: [*] built-in [] excluded <M> module < > module capable

-----CPU config Start-----
< CONFIG CHIP --->
-----OS config Start-----
< CONFIG OS --->
-----Peripheral config Start-----
< CONFIG USB --->
< MBED_API --->
-----Peripheral Test start-----
< CONFIG FUNCTION TEST --->
-----Secure Test start-----
< CONFIG SECURE TEST --->
-----802154 Config Start-----
< CONFIG 802154 --->
v(+)

<Select> < Exit > < Help >

```

Figure 31-2 USB configuration menu entry

- (2) Select Enable USB Device, then choose the desired USB class:

```

< CONFIG USB
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help.
Legend: [*] built-in [] excluded <M> module < > module capable

[*] Enable USB Device
[*] CDC_ACM
[] Composite
[] HID
[] Vendor

<Select> < Exit > < Help >

```

Figure 31-3 USB configuration

- (3) Type "make all EXAMPLE=<example>" command under GCC auto build project to build image with USB application.

**NOTE**

The <example> is the folder name under component/example/usb, and shall correspond to the mode and class configuration in step (2).

## 31.3 USB HAL APIs

### 31.3.1 Overview

The USB HAL APIs provide unified interfaces for upper layer USB device core drivers to access SoC-specific USB hardware.

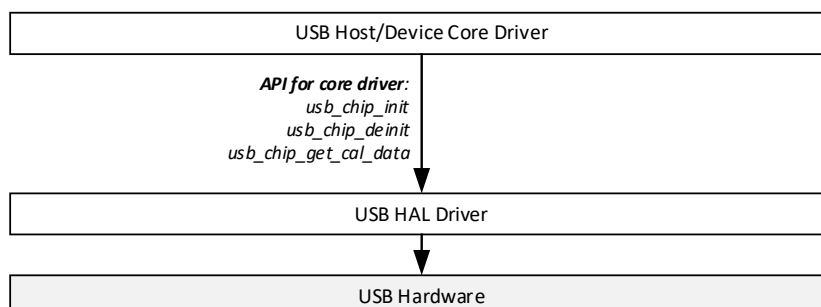


Figure 31-4 USB HAL APIs

### 31.3.2 APIs for Core Driver

Header file: {SDK}\component\soc\amebadplus\fwlib\include\ameba\_usb.h

| API                   | Description                        |
|-----------------------|------------------------------------|
| usb_chip_init         | SoC-specific USB initialization    |
| usb_chip_deinit       | SoC-specific USB de-initialization |
| usb_chip_get_cal_data | SoC-specific USB calibration data  |

## 31.4 USB Device APIs

### 31.4.1 Overview

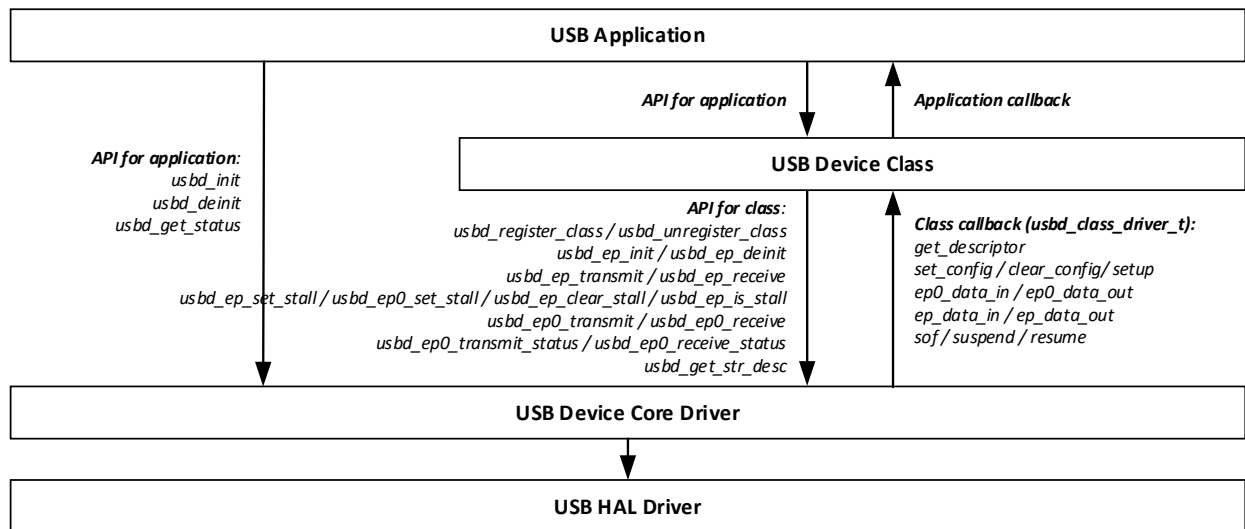


Figure 31-5 USB device APIs

### 31.4.2 Core APIs

Header file: {SDK}\component\usb\device\core\usbd.h

#### 31.4.2.1 APIs for Class

| API                      | Description                                                                                                      |
|--------------------------|------------------------------------------------------------------------------------------------------------------|
| usbd_register_class      | Register a class, the class is defined by type <code>usbd_class_driver_t</code> , refer to 31.4.2.2 for details. |
| usbd_unregister_class    | Unregister a class                                                                                               |
| usbd_ep_init             | Initialize an endpoint                                                                                           |
| usbd_ep_deinit           | De-initialize an endpoint                                                                                        |
| usbd_ep_transmit         | Transmit data to an endpoint                                                                                     |
| usbd_ep_receive          | Prepare to receive data from an endpoint                                                                         |
| usbd_ep_set_stall        | Set an endpoint to STALL state                                                                                   |
| usbd_ep_clear_stall      | Clear the STALL state of an endpoint                                                                             |
| usbd_ep_is_stall         | Check whether the endpoint is in STALL state                                                                     |
| usbd_ep0_set_stall       | Set endpoint 0 to STALL state                                                                                    |
| usbd_ep0_transmit        | Transmit data to endpoint 0, i.e. control endpoint                                                               |
| usbd_ep0_receive         | Prepare to receive data from endpoint 0, i.e. control endpoint                                                   |
| usbd_ep0_transmit_status | Transmit status to endpoint 0, i.e. control endpoint                                                             |
| usbd_ep0_receive_status  | Prepare to receive status from endpoint 0, i.e. control endpoint                                                 |
| usbd_get_str_desc        | Used for class to transfer ASCII string to USB string descriptor format in UNICODE 16                            |

### 31.4.2.2 Class Callback

The USB device class is defined by type `usbd_class_driver_t` as a group of callbacks:

```
typedef struct _usbd_class_driver_t {
 u8 (*get_descriptor)(usb_dev_t *dev, usb_setup_req_t *req,
 usb_speed_type_t speed, u16 *len);

 u8 (*set_config)(usb_dev_t *dev, u8 config);
 u8 (*clear_config)(usb_dev_t *dev, u8 config);
 u8 (*setup)(usb_dev_t *dev, usb_setup_req_t *req);

 u8 (*sof)(usb_dev_t *dev);
 u8 (*suspend)(usb_dev_t *dev);
 u8 (*resume)(usb_dev_t *dev);

 u8 (*ep0_data_in)(usb_dev_t *dev, u8 status);
 u8 (*ep0_data_out)(usb_dev_t *dev);

 u8 (*ep_data_in)(usb_dev_t *dev, u8 ep_addr, u8 status);
 u8 (*ep_data_out)(usb_dev_t *dev, u8 ep_addr, u16 len);
} usbd_class_driver_t;
```

Description of the callbacks:

| API                         | Description                                                                                                                        |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <code>get_descriptor</code> | Get device descriptor                                                                                                              |
| <code>set_config</code>     | Called when device core sets configuration, e.g. SET_CONFIGURATION request received at addressed state                             |
| <code>clear_config</code>   | Called when device core clears configuration, e.g. SET_CONFIGURATION request with a new configuration received at configured state |
| <code>setup</code>          | Called at setup phase of a control transfer, used for class-specific request handling                                              |
| <code>ep_data_in</code>     | Called at data in phase of a transfer, used to inform the class that the data transmit is done                                     |
| <code>ep_data_out</code>    | Called at data out phase of a transfer, used to inform the class to handle the received data                                       |
| <code>ep0_data_in</code>    | Called at data in phase of a control transfer, used to inform the class that the control data transmit is done                     |
| <code>ep0_data_out</code>   | Called at data out phase of a control transfer, used to inform the class to handle the received control data                       |
| <code>sof</code>            | Called at SOF interrupt, used for class-specific SOF handling                                                                      |
| <code>suspend</code>        | Called at suspend interrupt, used for class-specific suspend handling                                                              |
| <code>resume</code>         | Called at resume interrupt, used for class-specific resume handling                                                                |

### 31.4.2.3 APIs for Application

| API                    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>usbd_init</code> | <p>Initialize USB device stack with configuration defined by type <code>usbd_config_t</code>:</p> <pre>typedef struct {     u8 speed; /* USB speed:                 USB_SPEED_HIGH: high speed                 USB_SPEED_HIGH_IN_FULL: full speed */     u8 dma_enable; /* Enable USB internal DMA mode,                     0-Disable, 1-Enable */     u8 isr_priority; /* USB ISR thread priority */     u8 intr_use_ptx_fifo; /* Use Periodic Tx FIFO for INTR IN                            transfer */     u32 rx_fifo_depth; /* RX FIFO depth */     u32 nptx_fifo_depth; /* Non-Periodical TX FIFO depth */     u32 ptx_fifo_depth; /* Periodical TX FIFO depth */     u32 ext_intr_en; /* Enable extra USB interrupts:                      BIT0: USBD_SOF_INTR, GINTSTS.bit3                      BIT1: USBD_EOPF_INTR, GINTSTS.bit15                      BIT2: USBD_EPMIS_INTR, GINTSTS.bit17                      BIT3: USBD_ICII_INTR, GINTSTS.bit20                      BIT16: USBD_ITTXFE_INTR, DIEPINTn.bit4                      */     u8 nptx_max_epmis_cnt; /* Max Non-Periodical TX transfer EPMIS                            interrupt count allowed, EPMIS interrupt                            will be handled only if the EPMIS</pre> |

|                 |                                                                              |                                                                                                                                                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 |                                                                              | <pre> interrupt count is higher than this value and USBD_EPMIS_INTR is enabled in ext_intr_en */ u8 nptx_max_err_cnt[USB_MAX_ENDPOINTS];    /* Max Non-Periodical TX transfer error count allowed for each endpoint, if endpoint transfer error count is higher than this value, the transfer status will be determined as failed */ } usbd_config_t; </pre> |
| usbd_deinit     | De-initialize USB device stack                                               |                                                                                                                                                                                                                                                                                                                                                              |
| usbd_get_status | Get attach status, the return value is defined by type usbd_attach_status_t: |                                                                                                                                                                                                                                                                                                                                                              |
|                 |                                                                              | <pre> typedef enum {     USBD_ATTACH_STATUS_INIT      = 0U,    // Initialized     USBD_ATTACH_STATUS_ATTACHED  = 1U,    // Attached to host     USBD_ATTACH_STATUS_DETACHED  = 2U     // Detached from host } usbd_attach_status_t; </pre>                                                                                                                   |

### 31.4.2.4 Application Callback

N/A

## 31.4.3 Class APIs

### 31.4.3.1 CDC ACM

Header file: {SDK}\component\usb\device\cdc\_acm\usbd\_cdc\_acm.h

#### 31.4.3.1.1 API for Application

| API                              | Description                                                                                                                                                                                                                                                                                           |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| usbd_cdc_acm_init                | Initialize the class with parameters: <ul style="list-style-type: none"> <li>● RX buffer length (rx_buf_len): BULK OUT buffer length</li> <li>● TX buffer length (tx_buf_len): BULK IN buffer length</li> <li>● Application callback (cb): refer to <a href="#">31.4.3.1.2</a> for details</li> </ul> |
| usbd_cdc_acm_deinit              | De-initialize the class                                                                                                                                                                                                                                                                               |
| usbd_cdc_acm_transmit            | Transmit BULK IN data to host, the data length shall not be larger than the TX buffer length                                                                                                                                                                                                          |
| usbd_cdc_acm_receive             | Prepare to receive BULK OUT data, the data length will be limited to RX buffer length                                                                                                                                                                                                                 |
| usbd_cdc_acm_notify_serial_state | Send INTR IN data to notify device serial state to host                                                                                                                                                                                                                                               |

#### 31.4.3.1.2 Application Callback

CDC ACM class provides callbacks for user application, the callbacks are defined by type usbd\_cdc\_acm\_cb\_t:

```

typedef struct {
 u8(* init)(void);
 u8(* deinit)(void);
 u8(* setup)(usb_setup_req_t *req, u8 *buf);
 u8(* receive)(u8 *buf, u32 len);
} usbd_cdc_acm_cb_t;

```

Description of the callbacks:

| API     | Description                                                                                                |
|---------|------------------------------------------------------------------------------------------------------------|
| init    | Called at the end of class initialization flow, for application-specific initialization                    |
| deinit  | Called at the beginning of class de-initialization flow, for application-specific de-initialization        |
| setup   | Called at setup phase or data out phase of class-specific control requests, for application-specific setup |
| receive | Called at data out phase of BULK OUT transfer, for application to handle the received data                 |

### 31.4.3.1.3 Example

An example is provided for users to use CDC ACM device class. The example turns RTL8721Dx into a virtual serial port for PC, common serial port tools such as Tera Term can be used to communicate with RTL8721Dx, and RTL8721Dx will echo back the message sent to it.

Refer to the readme.txt file of the example for details.

### 31.4.3.2 Composite

Header files:

- {SDK}\component\usb\device\composite\usbd\_composite.h
- {SDK}\component\usb\device\composite\usbd\_composite\_cdc\_acm.h
- {SDK}\component\usb\device\composite\usbd\_composite\_hid.h

#### 31.4.3.2.1 API for Application

General API:

| API                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| usbd_composite_init   | Initialize the class with parameters: <ul style="list-style-type: none"> <li>● cdc_bulk_out_xfer_size: CDC ACM BULK OUT transfer length</li> <li>● cdc_bulk_in_xfer_size: CDC ACM BULK IN transfer length</li> <li>● cdc_cb: CDC ACM application callback, refer to <a href="#">31.4.3.2.2</a> for details</li> <li>● hid_intr_in_xfer_size: HID INTR IN transfer length</li> <li>● hid_cb: HID application callback, refer to <a href="#">31.4.3.2.2</a> for details</li> </ul> |
| usbd_composite_deinit | De-initialize the class                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

CDC ACM interface API:

| API                              | Description                                                                                  |
|----------------------------------|----------------------------------------------------------------------------------------------|
| usbd_cdc_acm_transmit            | Transmit BULK IN data to host, the data length shall not be larger than the TX buffer length |
| usbd_cdc_acm_receive             | Prepare to receive BULK OUT data, the data length will be limited to RX buffer length        |
| usbd_cdc_acm_notify_serial_state | Send INTR IN data to notify device serial state to host                                      |

HID interface API:

| API                | Description                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------|
| usbd_hid_send_data | Transmit INTR IN data to host, the data length shall not be larger than the TX buffer length |

#### 31.4.3.2.2 Application Callback

Composite class provides callbacks for user application, the callbacks are defined by two types:

- usbd\_cdc\_acm\_cb\_t:

```
typedef struct {
 u8(* init)(void);
 u8(* deinit)(void);
 u8(* setup)(usb_setup_req_t *req, u8 *buf);
 u8(* receive)(u8 *buf, u32 len);
} usbd_cdc_acm_cb_t;
```

Description of the callbacks:

| API     | Description                                                                                                |
|---------|------------------------------------------------------------------------------------------------------------|
| init    | Called at the end of class initialization flow, for application-specific initialization                    |
| deinit  | Called at the beginning of class de-initialization flow, for application-specific de-initialization        |
| setup   | Called at setup phase or data out phase of class-specific control requests, for application-specific setup |
| receive | Called at data out phase of BULK OUT transfer, for application to handle the received data                 |

- usbd\_hid\_usr\_cb\_t:

```
typedef struct {
 u8(* init)(void);
 void(* deinit)(void);
```

```

u8(* setup)(usb_setup_req_t *req, u8 *buf);
void(* transmit_complete)(u8 status);
} usbd_hid_usr_cb_t;

```

Description of the callbacks:

| API               | Description                                                                                                |
|-------------------|------------------------------------------------------------------------------------------------------------|
| init              | Called at the end of class initialization flow, for application-specific initialization                    |
| deinit            | Called at the beginning of class de-initialization flow, for application-specific de-initialization        |
| setup             | Called at setup phase or data out phase of class-specific control requests, for application-specific setup |
| transmit_complete | Called at data in phase of INTR IN transfer to inform the application that the INTR IN transfer is done    |

### 31.4.3.2.3 Example

An example is provided for user to use Composite device class. The example turns RTL8721Dx into a CDC ACM and HID composite device. Refer to the readme.txt file of the example for details.

### 31.4.3.3 HID

Header file: {SDK}\component\usb\device\hid\usbd\_hid.h

#### 31.4.3.3.1 APIs for Application

| API                | Description                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| usbd_hid_init      | Initialize the class with parameters: <ul style="list-style-type: none"> <li>TX buffer length (tx_buf_len): INTR IN buffer length</li> <li>Application callback (cb): refer to <a href="#">31.4.3.3.2</a> for details</li> </ul> |
| usbd_hid_deinit    | De-initialize the class                                                                                                                                                                                                          |
| usbd_hid_send_data | Transmit INTR IN data to host, the data length shall not be larger than the TX buffer length                                                                                                                                     |

#### 31.4.3.3.2 Application Callback

HID class provides callbacks for user application, the callbacks are defined by type usbd\_hid\_usr\_cb\_t:

```

typedef struct {
 void(* init)(void);
 void(* deinit)(void);
 void(* setup)(void);
 void(*transmit_complete)(void);
#if HID_DEVICE_TYPE == HID_KEYBOARD_DEVICE
 void(* receive)(u8 *buf, u32 len);
#endif
} usbd_hid_usr_cb_t;

```

Description of the callbacks:

| API               | Description                                                                                                |
|-------------------|------------------------------------------------------------------------------------------------------------|
| init              | Called at the end of class initialization flow, for application-specific initialization                    |
| deinit            | Called at the beginning of class de-initialization flow, for application-specific de-initialization        |
| setup             | Called at setup phase or data out phase of class-specific control requests, for application-specific setup |
| transmit_complete | Called at data in phase of INTR IN transfer to inform the application that the INTR IN transfer is done    |
| receive           | Called at data out phase of INTR OUT transfer to inform the application that the INTR OUT data is received |

### 31.4.3.3.3 Example

An example is provided for user to use HID device class. The example turns RTL8721Dx into a mouse for PC, simulates the mouse move, scroll, button pressed events.

Refer to the readme.txt file of the example for details.

### 31.4.3.4 Vendor

Header file: {SDK}\component\usb\device\vendor\usbd\_vendor.h

#### 31.4.3.4.1 APIs for Application

| API                            | Description                  |
|--------------------------------|------------------------------|
| usbd_vendor_init               | Initialize the class         |
| usbd_vendor_deinit             | De-initialize the class      |
| usbd_vendor_transmit_ctrl_data | Transmit CTRL data           |
| usbd_vendor_transmit_bulk_data | Transmit BULK data           |
| usbd_vendor_receive_bulk_data  | Prepare to receive BULK data |
| usbd_vendor_transmit_intr_data | Transmit INTR data           |
| usbd_vendor_receive_intr_data  | Prepare to receive INTR data |
| usbd_vendor_transmit_isoc_data | Transmit ISOC data           |
| usbd_vendor_receive_isoc_data  | Prepare to receive ISOC data |

#### 31.4.3.4.2 Application Callback

Vendor class provides callbacks for user application, the callbacks are defined by type usbd\_cdc\_acm\_cb\_t:

```
typedef struct {
 u8(* init)(void);
 u8(* deinit)(void);
 u8(* setup)(usb_setup_req_t *req, u8 *buf);
 u8(* set_config)(void);
 u8(* bulk_received)(u8 *buf, u32 len);
 u8(* intr_received)(u8 *buf, u32 len);
 u8(* isoc_received)(u8 *buf, u32 len);
 void(* bulk_transmitted)(u8 status);
 void(* intr_transmitted)(u8 status);
 void(* isoc_transmitted)(u8 status);
} usbd_cdc_acm_cb_t;
```

Description of the callbacks:

| API              | Description                                                                                                |
|------------------|------------------------------------------------------------------------------------------------------------|
| init             | Called at the end of class initialization flow, for application-specific initialization                    |
| deinit           | Called at the beginning of class de-initialization flow, for application-specific de-initialization        |
| setup            | Called at setup phase or data out phase of class-specific control requests, for application-specific setup |
| set_config       | Called when device core sets configuration, e.g. SET_CONFIGURATION request received at addressed state     |
| bulk_received    | Called at data out phase of BULK OUT transfer, for application to handle the received data                 |
| intr_received    | Called at data out phase of INTR OUT transfer, for application to handle the received data                 |
| isoc_received    | Called at data out phase of ISOC OUT transfer, for application to handle the received data                 |
| bulk_transmitted | Called at data in phase of BULK IN transfer, indicating application that the last transfer is done         |
| intr_transmitted | Called at data in phase of INTR IN transfer, indicating application that the last transfer is done         |
| isoc_transmitted | Called at data in phase of ISOC IN transfer, indicating application that the last transfer is done         |

#### 31.4.3.4.3 Example

An example is provided for user to use Vendor device class. The example turns RTL8721Dx into a vendor-specific device which can communicate with the USB vendor-specific host, e.g. Ameba chip which supports USB host mode and burnt with Vendor host image. Refer to the readme.txt file of the example for details.

This vendor-specific device class and example are designed for loopback test and as a reference for users to develop their own USB device applications.

## 31.5 Design Suggestions

For constant powered USB devices (e.g. battery powered devices), hot plug events shall be properly processed to avoid malfunction or

memory leak.

USB device stack provides the following API to get USB device status for the detection of hot plug events:

```
u8 usbd_get_status(void)
```

And USB device examples (e.g. CDC ACM) provide examples of how to use this API to support hot plug, please refer to the corresponding configuration (e.g. CONFIG\_USDB\_CDC\_ACM\_CHECK\_USB\_STATUS for CDC ACM example) for details.

However, it is not recommended to support hot plug by SW in this way for final products. Instead, it is suggested to check the USB status via hardware VBUS GPIO interrupt along with the usbd\_get\_status API, and the strategy is described in [Table 31-1](#).

Table 31-1 USB status detection strategy

| Event               | usbd_get_status                        | GPIO interrupt   | VBUS status | Detected USB status     |
|---------------------|----------------------------------------|------------------|-------------|-------------------------|
| Reset (detached)    | USB_STATUS_INIT                        | -                | OFF         | Initial detached status |
| Attach to PC        | USB_STATUS_ATTACHED                    | Y (rising edge)  | ON          | Attached to PC          |
| Detach from PC      | USB_STATUS_DETACHED                    | Y (falling edge) | OFF         | Detached                |
| Attach to charger   | USB_STATUS_INIT or USB_STATUS_DETACHED | Y (rising edge)  | ON          | Attached to charger     |
| Detach from charger | USB_STATUS_DETACHED                    | Y (falling edge) | OFF         | Detached                |

By comparing the new detected USB status with the old determined USB status, the exact USB status can be determined. The detailed USB status detect flow is shown in [Figure 31-6](#), only for reference.

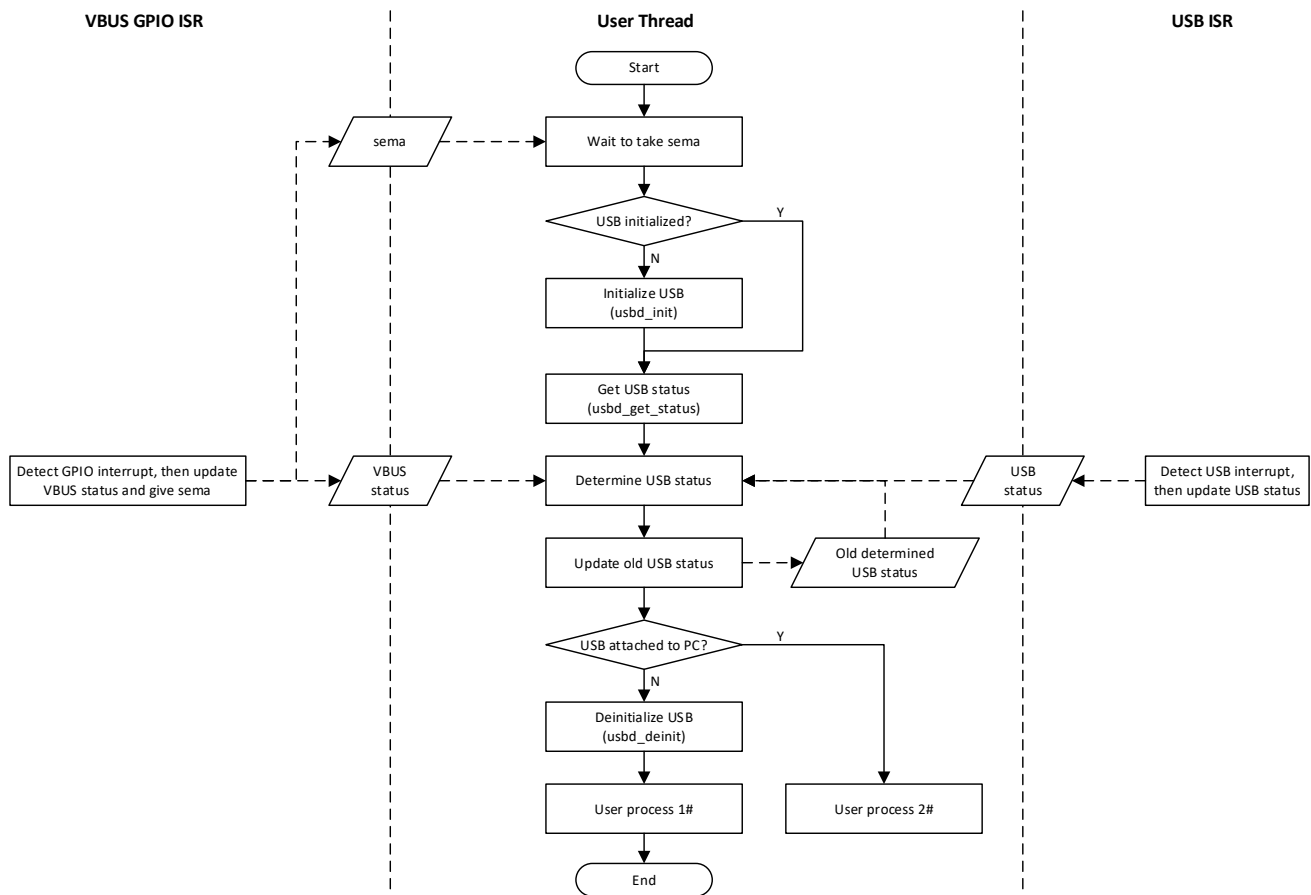


Figure 31-6 USB status detection flow

## 32 True Random Number Generator (TRNG)

### 32.1 Introduction

The TRNG is a true random number generator that provides full entropy outputs to the application as 32-bit samples. It is composed of a live entropy source (digital) and an internal conditioning component.

The TRNG has been tested under NIST-Random Test.

### 32.2 Features

- The TRNG delivers 32-bit true random numbers, produced by a digital entropy source.
- The TRNG is embedded with a health test unit and an error management unit.
- Two independent FIFOs, the one with low priority is for non-secure world, while the other with high priority for secure world
- The throughput of the TRNG is up to about 5Mbps.

### 32.3 Block Diagram

The block diagram of TRNG is shown in [Figure 32-1](#).

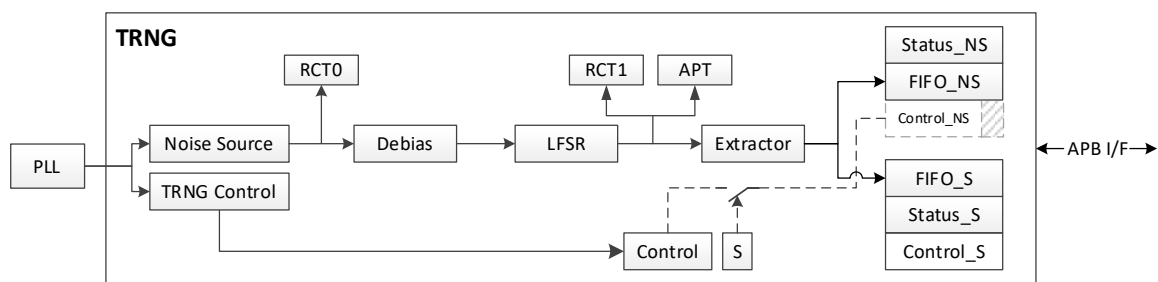


Figure 32-1 TRNG block diagram

The TRNG includes the following sub-modules:

- Clock
  - TRNG bus clock is 40MHz.
- Noise Source
  - The noise source is digital OSC, as a random number source, it is internally composed of ring oscillator.
- TRNG control
  - A bit is added to control whether the control register can be accessed from non-secure world.
  - Ensure that the default setting for OSC can work. ROM will use it only without configuring ROSC.
  - This area is the real control register, and the Control\_S is the access window in the secure world, Control\_NS is the access window in the non-secure world.
- Debias and LFSR and Extractor
  - A serial post-processing circuit
- RCT and APT
  - Two health tests of NIST specification
- Control\_S
  - This area is the access window in the secure world; the real address is "Control".
- Status\_S
  - Indicates the available data in FIFO\_S.
  - Indicates whether an error has happened.
- FIFO\_S
  - Only have one window register instead of all the registers.
  - Read and return all zero when FIFO is empty.
  - FIFO size is 256 bits.
  - When the available data is less than 128 bits, hardware will fill the FIFO\_S to full in a high priority.
- Control\_NS
  - This area is the access window in the non-secure world; the real address is "Control".
  - Only can be accessed when S bit in Control is 0.

- Status\_NS
  - Indicates the available data in FIFO\_NS
  - Indicates whether an error has happened.
- FIFO\_NS
  - Only have one window register instead of all the registers.
  - Read and returns all zero when FIFO is empty.
  - FIFO size is 128 bits.
  - This FIFO has a lower priority than FIFO\_S. If available data is less than 128 bits in FIFO\_S, hardware will not feed any data to this FIFO.

## 32.4 Usage

- If you need to run the system with security attributes, it is suggested to configure TRNG as secure so that the Control Register can only be accessed from secure world.
- When a large amount of random data is required both by secure world and non-secure world simultaneously, request from secure world will be satisfied first for the former has a higher priority. After the request from secure world ends, random data will be generated to satisfy non-secure world.
- It is suggested to call `_rand()` function to get a 32-bit random data.

## 33 Analog-to-Digital Converter (ADC)

This chapter describes the calibration method of the 12-bit analog-to-digital converter (ADC) of RTL8721Dx, and also explains the calculation method of ADC code to analog voltage. The related API and software examples are also given in details.

### 33.1 Introduction

The ADC of RTL8721Dx is a successive-approximation register (SAR) ADC, which includes up to 8 external channels and 3 internal channels. Among the external channels, 1 VBAT channel and 7 normal channels (V33 channel) all support divided mode. [Table 33-1](#) shows the description of the external channels.

Table 33-1 Description of the external channels

| Channel      | ADC channel ID | Pin name  | Input voltage                      |
|--------------|----------------|-----------|------------------------------------|
| V33 channel  | CH0~CH6        | PB13~PB19 | 0~3.3V (Single-ended divided mode) |
| VBAT channel | CH7            | BAT_MEAS  | 0 ~ 5V                             |

### 33.2 ADC Calibration

#### 33.2.1 Calibration Parameters

Normally, there are many factors that affect ADC performance, some of which can be partially eliminate by calibration, such as gain, offset, integral non-linearity (INL) and differential non-linearity (DNL).

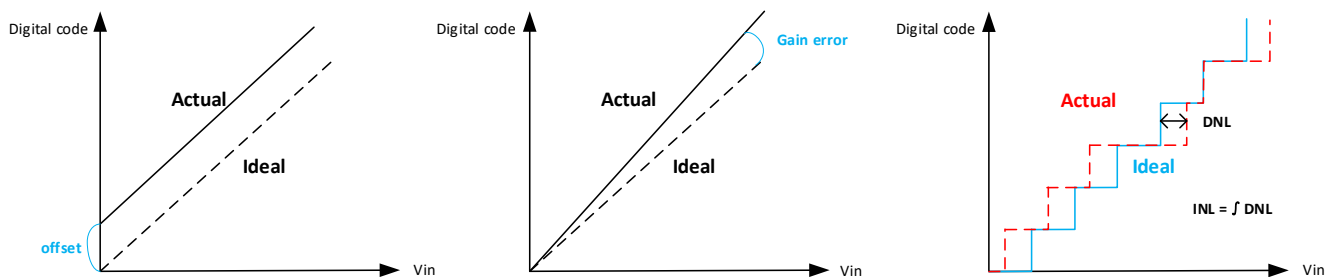


Figure 33-1 Description of Gain, Offset, IND and DNL

In RTL8721Dx, a quadratic curve fitting is used for calibration, which can better eliminate offset error, gain error and non-linear errors. After calibration, each IC can find its own value of  $a$ ,  $b$ ,  $c$  which satisfies the squared vertical distance between each point  $(x_i, y_i)$  and the  $y = a \cdot x^2 + b \cdot x + c$  is minimal. Then converse them to  $A$ ,  $B$  and  $C$  by formula and store them in one time program (OTP). Here  $A = a \cdot 2^{26}$ , where parameter  $a$  is signed data, and  $A$  is stored in OTP with 2's complement.  $B = b \cdot 2^{15}$ , where parameter  $b$  is unsigned data, and  $B$  is stored in OTP with original data.  $C = c \cdot 2^6$ , where parameter  $c$  is signed data, and  $C$  is stored in OTP with 2's complement.

[Table 33-2](#) shows the address of calibration parameters.

Table 33-2 Address of calibration parameters

| Channel        | Mode              | Bit    | Parameter                                                                        |
|----------------|-------------------|--------|----------------------------------------------------------------------------------|
| Normal channel | Single-ended mode | [15:0] | Quadratic fitting: A<br>Bit[15]: sign bit<br>Bit[14:0]: data with 2's complement |
|                |                   | [15:0] | Quadratic fitting: B<br>Bit[15:0]: data without sign bit                         |
|                |                   | [15:0] | Quadratic fitting: C<br>Bit[15]: sign bit<br>Bit[14:0]: data with 2's complement |
|                |                   | [15:0] | Quadratic fitting: B<br>Bit[15:0]: data without sign bit                         |
|                |                   | [15:0] | Quadratic fitting: C<br>Bit[15]: sign bit<br>Bit[14:0]: data with 2's complement |

### 33.2.2 Calculation

In the actual application, customers can read each parameter A, B and C from OTP by software. When acquiring ADC code, use the following formula to get the current voltage with unit as mV, here parameter x is ADC code:

$$y = ax^2 + bx + c = \left(\frac{A}{2^{26}}\right)x^2 + \left(\frac{B}{2^{15}}\right)x + \left(\frac{C}{2^6}\right)$$

For example, if A, B, and C read from OTP are 0xff66, 0x6f91, and 0x53f, and ADC code is 1800. The original A is 0x809a, parameter a = -(0x9A)/2<sup>26</sup>, parameter b = (0x6f91)/2<sup>15</sup>, and parameter c = (0x53f)/2<sup>6</sup>. So, the current voltage y = 1582mV.

#### NOTE

Use API `ADC_GetVoltage(Chan_data)` to get the corrected ADC voltage directly.

### 33.3 Internal R

For better accuracy, the internal R resistance of each ADC V33 channels (CH0~CH5) in divided mode can be measured and the R resistance offset is stored in OTP. The base R resistance is 400kΩ, so the actual R resistance is equal to the sum of base resistance and offset.

Table 33-3 shows the address of R resistance offset in OTP.

Table 33-3 Address of R impedance

| Function    | Mode                      | Bit   | Parameter                         |
|-------------|---------------------------|-------|-----------------------------------|
| V33 channel | Single-ended divided mode | [7:0] | Internal R resistance offset (kΩ) |

For example, the R resistance offset read from OTP is 0x2D, the actual R resistance = 45 + 400 = 445kΩ.

#### NOTE

Use API `ADC_GetInterR()` to get internal R with V33 channel in divided mode.

In the actual application, due to the impact of internal R, the voltage measured by ADC is little accurate. With combination of peripheral circuit and internal R, ADC digital code becomes lower. However, considering R resistance in the calculation can eliminate this error.

For example, in thermal sensor application, one of CH0~CH5 is usually to measure the voltage of negative temperature coefficient (NTC) thermistor. The simplified block is as illustrated in Figure 33-2.

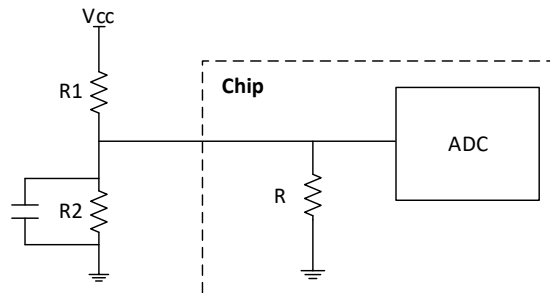


Figure 33-2 Simplified application block

The ideal input voltage to ADC (without internal R) is:

$$V_{ideal} = V_{cc} * R2 / (R1 + R2)$$

But the actual input voltage to ADC (with internal R) is:

$$V_{actual} = V_{cc} * (R2 // R) / (R2 // R + R1) = V_{cc} * R2 / (R2 + R1 * R2 / R + R1)$$

Compare the two formulas, Vactual is smaller than Videal due to the impact of internal R. And the greater the ratio of R1xR2/R is, the greater the error between Vactual and Videal is.

If R2 is NTC thermistor, the actual resistance of R2 is:

$$R2 = R1 * V_{adc} / (V_{cc} - V_{adc} - V_{adc} * R1 / R)$$

Here, Vadc is measured by ADC, R = R offset (read from OTP) + 400kΩ.

By the way, it's optional for customers to choose another V33 channel to measure the voltage of Vcc, which can reduce the impact that Vcc changes while ADC Vref doesn't change.

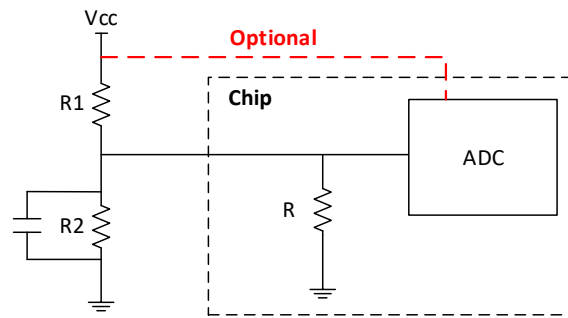


Figure 33-3 Optional application block

## 33.4 Usage

### 33.4.1 Auto Mode

To use ADC auto mode, the following steps are mandatory.

- (1) Configure the ADC pinmux according to the pinmux specification.

For example, call the following functions to use ADC0.

```
Pinmux_Config(_PB_19, PINMUX_FUNCTION_ADC)
PAD_InputCtrl(_PB_19, DISABLE);
PAD_PullCtrl(_PB_19, GPIO_PuPd_NOPULL);
```

- (2) Set the default parameters, and modify ADC mode. After that, channel list and other parameters can be modified in ADC\_InitStruct if needed.

```
ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct);
ADC_InitStruct->ADC_OpMode = ADC_AUTO_MODE;
```

- (3) Initialize ADC and then enable ADC.

```
ADC_Init(ADC_InitTypeDef *ADC_InitStruct);
ADC_Cmd(ENABLE);
```

- (4) Read ADC sample data. You can use polling mode or interrupt mode to acquire ADC sample data.

- Polling mode:

```
ADC_ReceiveBuf(u16 *pBuf, u32 len);
```

- Interrupt mode:

```
ADC_INTConfig(ADC_BIT_IT_FIFO_OVER_EN | ADC_BIT_IT_FIFO_FULL_EN, ENABLE);
InterruptRegister((IRQ_FUN)ADC_IrqRxHandler, AdcIrqNum[CPUID], NULL, 10);
InterruptEn(AdcIrqNum[CPUID], 10);
ADC_AutoCSwCmd(ENABLE);
```

### 33.4.2 Software-trigger Mode

To use ADC software-trigger mode, the following steps are mandatory.

- (1) Configure the ADC pinmux according to the pinmux specification.

For example, call the following functions to use ADC0.

```
Pinmux_Config(_PB_19, PINMUX_FUNCTION_ADC)
PAD_InputCtrl(_PB_19, DISABLE);
PAD_PullCtrl(_PB_19, GPIO_PuPd_NOPULL);
```

- (2) Set the default parameters, and modify ADC mode. After that, channel list and other parameters can be modified in ADC\_InitStruct if needed.

```
ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct);
ADC_InitStruct->ADC_OpMode = ADC_SW_TRI_MODE;
```

- (3) Initialize ADC and then enable ADC.

```
ADC_Init(ADC_InitTypeDef *ADC_InitStruct);
ADC_Cmd(ENABLE);
```

- (4) Read ADC sample data.

```
ADC_SWTrigCmd(ENABLE);
while(ADC_Readable() == 0);
ADC_SWTrigCmd(DISABLE);
ADC_Read();
```

### 33.4.3 Timer-trigger Mode

To use ADC timer-trigger mode, the following steps are mandatory.

- (1) Configure the ADC pinmux according to the pinmux specification.

For example, call the following functions to use ADC0.

```
Pinmux_Config(PB_19, PINMUX_FUNCTION_ADC)
PAD_InputCtrl(PB_19, DISABLE);
PAD_PullCtrl(PB_19, GPIO_PuPd_NOPULL);
```

- (2) Set the default parameters, and modify ADC mode. After that, channel list and other parameters can be modified in ADC\_InitStruct if needed.

```
ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct);
ADC_InitStruct->ADC_OpMode = ADC_TIM_TRI_MODE;
```

- (3) Initialize ADC and then enable ADC.

```
ADC_Init(ADC_InitTypeDef *ADC_InitStruct);
ADC_Cmd(ENABLE);
```

- (4) Read ADC sample data.

```
ADC_INTConfig(ADC_BIT_IT_CV_END_EN | ADC_BIT_IT_CVLIST_END_EN, ENABLE);
InterruptRegister((IRQ_FUN)ADCIrqHandle, AdcIrqNum[CPUID], tim_idx, 10);
InterruptEn(AdcIrqNum[CPUID], 10);
ADC_TimerTrigCmd(tim_idx, period, ENABLE);
```

### 33.4.4 Comparator-assist Mode

To use ADC comparator-assist mode, the following steps are mandatory.

- (1) Configure the ADC pinmux according to the pinmux specification.

For example, call the following functions to use ADC1.

```
Pinmux_Config(PB_18, PINMUX_FUNCTION_ADC)
PAD_InputCtrl(PB_18, DISABLE);
PAD_PullCtrl(PB_18, GPIO_PuPd_NOPULL);
```

- (2) Set the default parameters, and modify ADC mode. After that, channel list and other parameters can be modified in ADC\_InitStruct if needed.

```
ADC_StructInit(ADC_InitTypeDef *ADC_InitStruct);
ADC_InitStruct->ADC_OpMode = ADC_COMP_ASSIST_MODE;
```

- (3) Initialize ADC and then enable ADC.

```
ADC_Init(ADC_InitTypeDef *ADC_InitStruct);
ADC_Cmd(ENABLE);
```

- (4) Set the default parameters. After that, channel list and other parameters can be modified in CMP\_InitStruct if needed.

```
CMP_StructInit(CMP_InitTypeDef *CMP_InitStruct)
```

- (5) Initialize the comparator and then enable the comparator.

```
CMP_Init(CMP_InitTypeDef *CMP_InitStruct);
CMP_Cmd(ENABLE);
```

- (6) Select a comparator operation mode, and then enable it.

- Auto mode:

```
CMP_AutoCSwCmd(ENABLE);
```

- Software-trigger mode:

```
CMP_SwTrigCmd(ENABLE);
```

- Timer-trigger mode:

```
CMP_TimerTrigCmd(tim_idx, period, ENABLE);
```

- (7) Configure the interrupt and register interrupt callback function.

```
InterruptRegister((IRQ_FUN)CMPIrqHandle, CompIrqNum[CPUID], NULL, 10);
InterruptEn(CompIrqNum[CPUID], 10);
```

## 34 Cap-Touch Controller (CTC)

The Cap-Touch Controller (CTC) provides 4 channels for capacitive sensing, which offers a wide range of capacitance detection. The sensitivity and threshold for each channel are configurable. For different applications and surroundings, you should tune parameters to achieve the best performance.

This chapter introduces how to use Cap-Touch and design touch key.

### 34.1 Usage

#### 34.1.1 Cap Test Tool

Cap Test Tool is the official Cap-Touch calibration tool developed by Realtek, which can calibrate the general and channel-specific Cap-Touch configurations.

We suggest doing CTC calibration by the Cap Test Tool directly and automatically. Refer to the application note of Cap Test Tool for more details about usage.

Of course, users can tune the configuration parameters manually to meet special requirements according to section [34.1.2](#) and [34.1.3](#).

#### 34.1.2 CTC Initialization

The CTC initialization is implemented by the following steps:

- (1) Call `CapTouch_StructInit()` to define the configuration parameters of Cap-Touch, such as `SampleCnt`, `ScanInterval`, `DiffThreshold`, etc.
- (2) Set configuration parameters in the following structure, which includes touch threshold, noise threshold, mbias current, enable control for each channel. This method to tune parameters for each channel can be found in Section [34.1.3](#).

```
const CapTouch_CHInitTypeDef ctc_ch_config[4] =
{
 /*DiffThreshold, MbiasCurrent, ETCNNoiseThr, ETCNNoiseThr, CHEnable*/
 {800, 0x0C, 400, 400, ENABLE}, /* Channel 0 */
 {800, 0x0C, 400, 400, DISABLE}, /* Channel 1 */
 {800, 0x0C, 400, 400, DISABLE}, /* Channel 2 */
 {800, 0x0C, 400, 400, DISABLE}, /* Channel 3 */
};
```

#### NOTE

*The configuration parameters can be acquired in Cap Test Tool.*

- (3) Call `CapTouch_Init()` to configure the Cap-Touch.
- (4) Enable Cap-Touch by `CapTouch_Cmd()` and enable the interrupt by `CapTouch_INTConfig()`.

```
void captouch_init(captouch_t *obj)
{
 u32 ch = 0;
 CapTouch_InitTypeDef Touch_InitStruct;
 /* enable ADC function */
 RCC_PeriphClockCmd(APBPeriph_ADC, APBPeriph_ADC_CLOCK, ENABLE);
 /* Load HAL initial data structure default value */
 Touch_InitStruct.CT_DebounceEn = 1;
 Touch_InitStruct.CT_SampleCnt = 6;
 Touch_InitStruct.CT_ScanInterval = 60;
 Touch_InitStruct.CT_ETCStep = 1;
 Touch_InitStruct.CT_ETCFactor = 4;
 Touch_InitStruct.CT_ETCScanInterval = 3;
 for (ch = 0; ch < CT_CHANNEL_NUM; ch++) {
 if (obj->CT_Channel[ch].CT_CHEnable == ENABLE) {
 Touch_InitStruct.CT_Channel[ch].CT_CHEnable=obj->CT_Channel[ch].CT_CHEnable;
 Touch_InitStruct.CT_Channel[ch].CT_DiffThreshold=obj->CT_Channel[ch].CT_
DiffThreshold;
 Touch_InitStruct.CT_Channel[ch].CT_MbiasCurrent=obj->CT_Channel[ch].CT_
MbiasCurrent;
 Touch_InitStruct.CT_Channel[ch].CT_ETCNNoiseThr=obj->CT_Channel[ch].CT_
```

```
ETCNNoiseThr;
 Touch_InitStruct.CT_Channel[ch].CT_ETCPNoiseThr=obj->CT_Channel[ch].CT_
ETCPNoiseThr;
 /* Disable digital path input */
 PAD_InputCtrl(PinMap_CTC[ch].pin, DISABLE);
 pinmap_pinout(PinMap_CTC[ch].pin, PinMap_CTC);
 PAD_SleepPullCtrl(PinMap_CTC[ch].pin, GPIO_PuPd_NOPULL);
}
}
CapTouch_Init(CAPTOUCH_DEV, &Touch_InitStruct);
CapTouch_Cmd(CAPTOUCH_DEV, ENABLE);
CapTouch_INTConfig(CAPTOUCH_DEV, CT_ALL_INT_EN, ENABLE);
/* Register interrupt Callback function */
InterruptRegister((IRQ_FUN) captouch_irq_handler, CTOUCH_IRQ, (u32)obj, 5);
InterruptEn(CTOUCH_IRQ, 5);
}
```

At this time, the Cap-Touch will start to work.

### 34.1.3 CTC Calibration

To achieve the best performance (sensitivity, reliability or response time), users need to tune touch threshold, noise threshold and mbias current during development.

The reference value of parameters are listed in [Table 34-1](#).

Table 34-1 Reference value of parameters

| Parameters           | Description                                                                                                 | Reference range        |                     |
|----------------------|-------------------------------------------------------------------------------------------------------------|------------------------|---------------------|
|                      |                                                                                                             | Finger-touch detection | Proximity detection |
| Mbias                | Set to a value which makes the baseline within the reference range.                                         | Baseline: 2500~3500    | Baseline: 3500~3800 |
| Difference threshold | Set to 80% of the touched difference value.                                                                 | 500~600                | 30~80               |
| Noise threshold      | Set to no more than 50% of the difference threshold, and can be adjusted according to the actual situation. | 200~240                | 0~40                |

The parameters for each channel should be tuned individually. For example, follow these steps to calibrate for channel 0.

- (1) Initialize the parameters of CapTouch\_InitStruct with default values using CapTouch\_StructInit to enable channel 0.

```
CapTouch_StructInit(&CapTouch_InitStruct);
CapTouch_InitStruct.CT_Channel[0].CT_CHEnable = ENABLE;
```

- (2) Initialize the Cap-Touch peripheral according to CapTouch\_InitStruct in step (1).

```
CapTouch_Init(CAPTOUCH_DEV, &CapTouch_InitStruct);
```

- (3) Enable Cap-Touch.

```
CapTouch_Cmd(CAPTOUCH_DEV, ENABLE);
```

- (4) Call CapTouch\_GetChAveData() to read the sample data from channel 0 periodically.

```
CapTouch_GetChAveData(CAPTOUCH_DEV, 0);
```

#### NOTE

- Too large mbias will cause the charging voltage to exceed the full range, which may cause damage to the circuit.
- Reference range is based on no-overlay conditions. With overlay, it's better to adjust mbias, difference threshold and noise threshold according to the actual application.
- When using Cap-Touch, to prevent leakage, the Cap-Touch pin should be configured as shutdown or configured as no-pull, at the same time input and output of the pad should be disabled.

### 34.1.4 CTC Wakeup

When using CTC as a wakeup source, configure the CTC and system as follows:

- (1) Initialize CTC and enable its interrupt according to Section [34.1.2](#).
- (2) Set the related wakeup source (WAKE\_SRC\_CTOUCH) in sleep\_wevent\_config[] to WAKEUP\_KM4 or WAKEUP\_KM0 (based on which CPU you want to wake). The interrupt should be registered on the same CPU selected by sleep\_wevent\_config[].
- (3) Switch CTC clock source to CTC IP clock before system enters sleep mode by RCC\_PeriphClockSource\_CTC().
- (4) Enter sleep mode by releasing the wakelock in KM4 (PMU\_OS needs to be released since it is acquired by default when boot).
- (5) Clear the CTC interrupt when wakeup and switching CTC clock source to LBUS clock by RCC\_PeriphClockSource\_CTC().

## 35 Key-Scan

### 35.1 Introduction

As a keypad scan device, the Key-Scan supports up to 8\*8 keypad array, and its rows and columns are configurable. The Key-Scan also supports multi-key detection, and it can work in low power mode.

#### 35.1.1 Keypad

The columns and rows of Key-Scan are configurable, and the maximum columns or rows can be as 8.

At the beginning, all columns output low, and all rows are set as input with internal pull up. When there is a key or multiple keys pressed (short between the Column and the Row), it triggers an internal state machine to start a Key-Scan cycle to determine the column and row of the key that is pressed. The state machine sets first column as an output low and all other columns are in Hi-Z state. The state machine then scans all rows to determine which keys are being pressed, and then second column is output low until the last column.

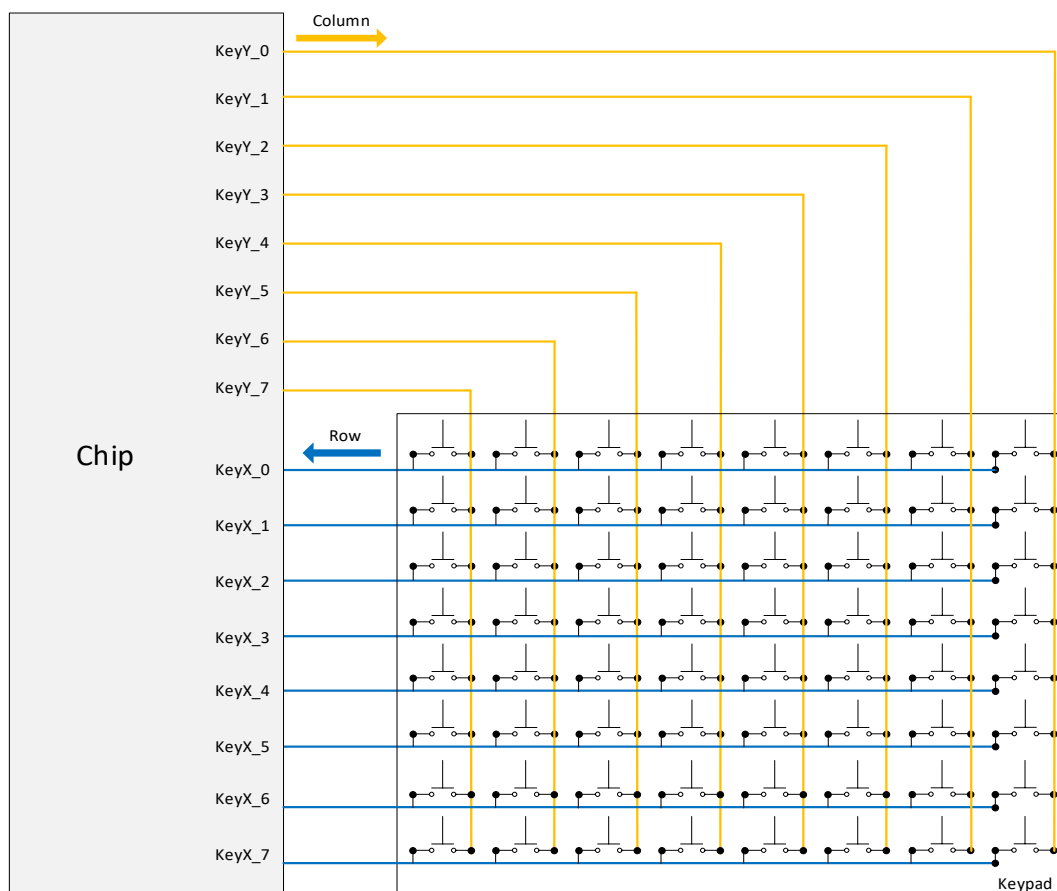


Figure 35-1 Keypad array

#### 35.1.2 Work Mode

Key-Scan contains two work modes: Event Trigger Mode and Regular Scan Mode. [Figure 35-2](#) shows the difference of FIFO items between Event Trigger Mode and Regular Scan Mode during a key press and release.

- In Event Trigger Mode, key press or release event is stored in the key event FIFO only once in each key press and release operation.
- In Regular Scan Mode, at each full scan, any key press event is stored in the key event FIFO until it is released (in this condition, only key press event is stored in the key event FIFO).

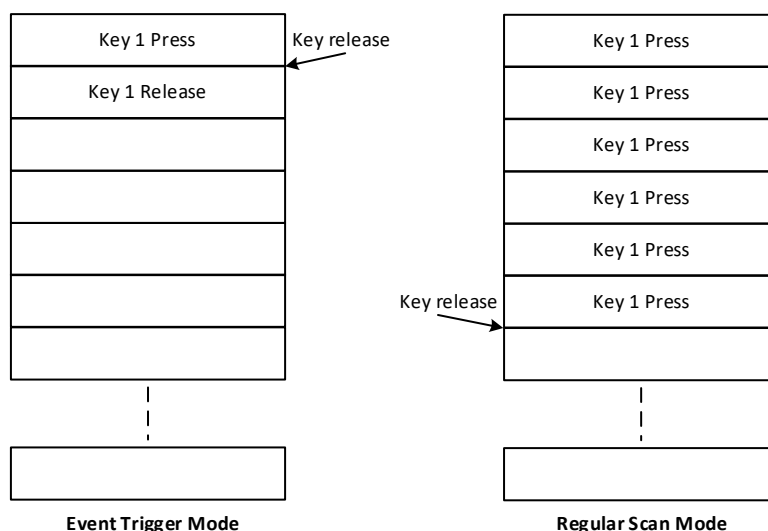


Figure 35-2 Difference of FIFO items between two work modes

### 35.1.3 Stuck Key

Key stuck may occur in some applications. In this case, if there is no solution in reply to key stuck, CPU will keep active to perform continuous keypad scans, thus CPU cannot enter low power mode, which will result in high power consumption.

To solve this problem, the Key-Scan supports auto check key stuck function. When enabling this function, if a key stuck time is over than stuck time threshold, this key will be regarded as a stuck key. Read key stuck status registers to get the stuck key, and set the default status of the stuck key row to let CPU enter low power mode.

To lower the power consumption, stuck row detection function may be enabled to set row pins "no pull" detection time interval.

## 35.2 Usage

### 35.2.1 Event Trigger Mode

To use Key-Scan event trigger mode, perform the following procedures:

- (1) Pinmux configuration for Key-Scan keypads.
  - a) Set column pads to no-pull, and set column pinmux function.

```

PAD_PullCtrl(pad_colx, GPIO_PuPd_NOPULL);
Pinmux_Config(pad_colx, PINMUX_FUNCTION_KEY_COLx);

```
  - b) Set row pads to pull-up, and set key row pinmux function.

```

PAD_PullCtrl(pad_rowx, GPIO_PuPd_UP);
Pinmux_Config(pad_rowx, PINMUX_FUNCTION_KEY_ROWx);

```
- (2) Initialize Key-Scan parameters.

Select key row number and column number (one bit one row or column) according to keypads, and set work mode to event trigger mode, etc.

```

KeyScan_StructInit(&KeyScan_InitStruct);
KeyScan_InitStruct.KS_ColSel = 0xFF; //8 columns
KeyScan_InitStruct.KS_RowSel = 0xFF; //8 rows
KeyScan_InitStruct.KS_WorkMode = KS_EVENT_TRIGGER_MODE;
KeyScan_Init(KeyScan, &KeyScan_InitStruct);

```
- (3) Enable Key-Scan interrupt and register Key-Scan interrupt handle.
- (4) Enable Key-Scan.

```

KeyScan_Cmd(KeyScan, ENABLE);

```
- (5) Wait Key-Scan interrupt, and handle Key-Scan interrupts.

### 35.2.2 Regular Scan Mode

To use Key-Scan regular scan mode, perform the following procedures:

- (1) Pinmux configuration for Key-Scan keypads.
  - a) Set column pads to no-pull, and set column pinmux function.  

```
PAD_PullCtrl(pad_colx, GPIO_PuPd_NOPULL);
Pinmux_Config(pad_colx, PINMUX_FUNCTION_KEY_COLx);
```
  - b) Set row pads to pull-up, and set key row pinmux function.  

```
PAD_PullCtrl(pad_rowx, GPIO_PuPd_UP);
Pinmux_Config(pad_rowx, PINMUX_FUNCTION_KEY_ROWx);
```
- (2) Initialize Key-Scan parameters.  
 Select key row number and column number (one bit one row or column) according to keypads, set work mode to regular scan mode, etc.  

```
KeyScan_StructInit(&KeyScan_InitStruct);
KeyScan_InitStruct.KS_ColSel = 0xFF; //8 columns
KeyScan_InitStruct.KS_RowSel = 0xFF; //8 rows
KeyScan_Init(KeyScan, &KeyScan_InitStruct);
```
- (3) Enable Key-Scan interrupt and register Key-Scan interrupt handle.
- (4) Enable Key-Scan.  

```
KeyScan_Cmd(KeyScan, ENABLE);
```
- (5) Wait Key-Scan interrupt, and handle Key-Scan interrupts.

### 35.2.3 Key Stuck

When existing key stuck, perform the following procedures:

- (1) Pinmux configuration for Key-Scan keypads.
  - a) Set column pads to no-pull, and set column pinmux function.  

```
PAD_PullCtrl(pad_colx, GPIO_PuPd_NOPULL);
Pinmux_Config(pad_colx, PINMUX_FUNCTION_KEY_COLx);
```
  - b) Set row pads to pull-up, and set key row pinmux function.  

```
PAD_PullCtrl(pad_rowx, GPIO_PuPd_UP);
Pinmux_Config(pad_rowx, PINMUX_FUNCTION_KEY_ROWx);
```
- (2) Initialize Key-Scan parameters.  
 Select key row number and column number (one bit one row or column) according to keypads, set work mode to regular scan mode, etc.  

```
KeyScan_StructInit(&KeyScan_InitStruct);
KeyScan_InitStruct.KS_ColSel = 0xFF; //8 columns
KeyScan_InitStruct.KS_RowSel = 0xFF; //8 rows
KeyScan_Init(KeyScan, &KeyScan_InitStruct);
```
- (3) Enable Key-Scan stuck auto check function.  

```
KeyScan_StuckAutoCmd(KeyScan, ENABLE);
```
- (4) Set stuck time threshold, and configure stuckrow detect time and interval time.  

```
KeyScan_SetStuckThreshold(KeyScan, 10); //stuck time threshold: 10ms
KeyScan_StuckPeriodicalPull(KeyScan, 2000, 4000); //pull time: 2000us, no pull time: 4000us
```
- (5) Enable Key-Scan stuck and all default interrupt and register Key-Scan interrupt handle.
- (6) Enable Key-Scan.  

```
KeyScan_Cmd(KeyScan, ENABLE);
```
- (7) Wait Key-Scan interrupt, and handle Key-Scan interrupts.
  - In stuck event interrupt, get row status, and set row default status to indicate stuck row, then mask all the keys of the stuck row.  

```
row_status = KeyScan_GetStuckRow(KeyScan);
KeyScan_SetStuckRow(KeyScan, row_status);
```
  - In all default interrupt, reset row default status to initial value, disable stuck event interrupt and all default interrupt, enable scan event interrupt and all release interrupt, then other keys except the stuck key can work normally.  

```
KeyScan_INTConfig(KeyScan, KS_BIT_ALL_DEFAULT_INT_MASK | KS_BIT_STUCK_EVENT_INT_MASK,
DISABLE);
KeyScan_SetStuckRow(KeyScan, 0);
KeyScan_INTConfig(KeyScan, KS_BIT_ALL_RELEASE_INT_MASK | KS_BIT_SCAN_EVENT_INT_MASK,
ENABLE);
```
- (8) Key or keys except the stuck press or release will generate the corresponding interrupt normally.

# 36 Audio

## 36.1 Introduction

The audio framework is a whole audio architecture aiming to provide different layers of audio interfaces for applications. The interfaces involve Audio HAL and Audio Framework.

- Audio HAL: defines unified audio hardware interfaces. It interacts with audio driver to do audio streaming or audio settings.
- Audio Framework: provides interfaces for audio streaming, volume, and other settings.

The audio framework provides two architectures to meet different needs of audio. Different architectures have different implementations, but the interfaces are the same.

- Audio mixer architecture: supports audio recording, and audio playback. For audio playback, this architecture provides audio mixing functions, and format, rate, channel will be converted to a unified format for mixing.
- Audio passthrough architecture: supports audio recording, and audio playback. This architecture has no audio mixing, only one audio playback is allowed at the same moment.

### 36.1.1 Mixer Overview

The audio interfaces and the entire implementation are shown in [Figure 36-1](#).

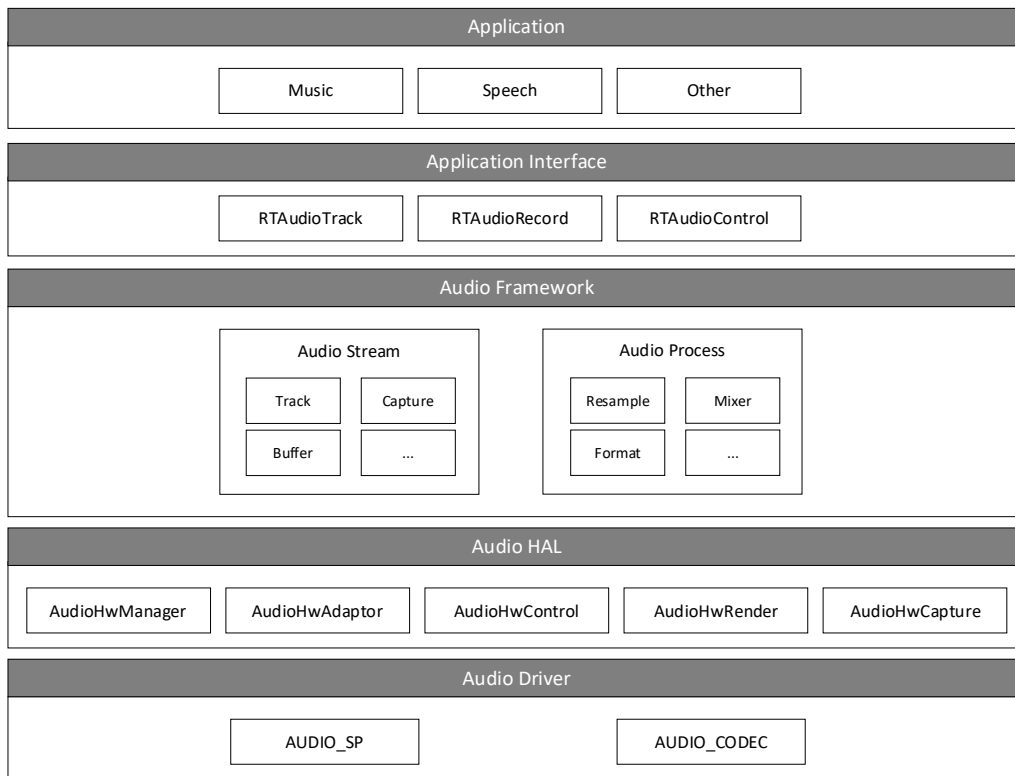


Figure 36-1 Audio mixer overview

The whole audio mixer architecture includes the following sub-modules:

- Application Interface
  - RTAudioTrack provides interfaces to play sound.
  - RTAudioRecord provides interfaces to capture sound.
  - RTAudioControl provides interfaces to control sound volumes and so on.
- Audio Framework
  - The audio framework provides audio reformat, resample, volume, and mixer functions for audio stream playback.
- Audio HAL
  - AudioHwManager provides interfaces that manage audio adapters through a specific adapter driver program loaded based on the given audio adapter descriptor.
  - AudioHwAdaptor provides interfaces that manage audio adapter capabilities, including initializing ports, creating rendering

- and capturing tasks, and obtaining the port capability set.
- `AudioHwControl` provides interfaces for `RTAudioControl`, and set commands to audio driver.
- `AudioHwRender` provides interfaces that get data from the upper layer and render data to audio driver user interfaces.
- `AudioHwCapture` provides interfaces to capture data from audio driver user interfaces and deliver the data to the upper layer.
- Audio Driver
  - `AUDIO_SP` provides interfaces to configure audio sports.
  - `AUDIO_CODEC` provides interfaces to configure audio codec.

### 36.1.2 Passthrough Overview

The audio interfaces and the entire implementation are shown in [Figure 36-2](#).

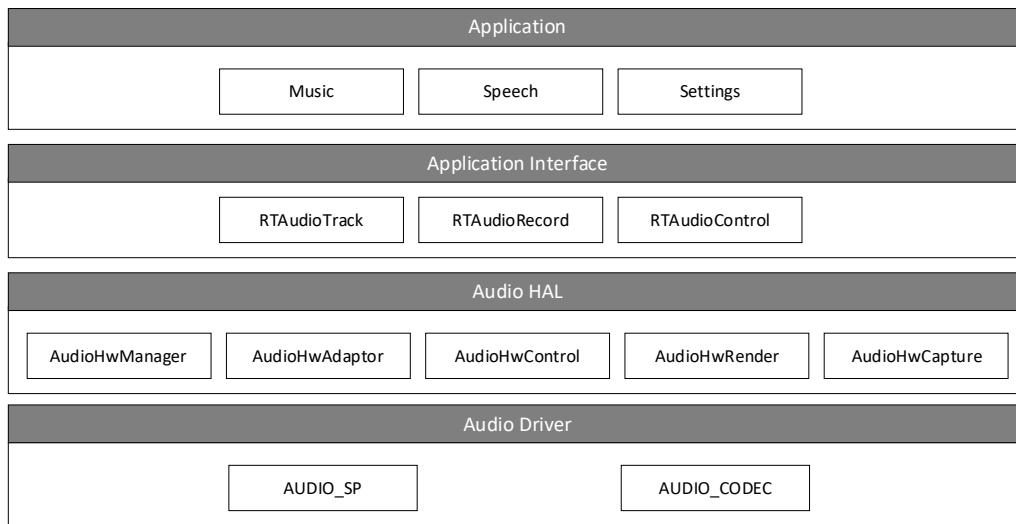


Figure 36-2 Audio passthrough overview

The audio passthrough architecture has no Audio Framework layer compared to audio mixer architecture, other layers are nearly the same.

### 36.1.3 How to Choose Architecture

The above sections describe two architectures of audio: mixer and passthrough. Users can choose the suitable architecture according to the project's requirements. Here is a comparison table of the two architectures:

| Architecture | Mem occupancy | Code size | Playback basic function | Playback mixing function | Playback latency | Capture function |
|--------------|---------------|-----------|-------------------------|--------------------------|------------------|------------------|
| Mixer        | More          | More      | Nearly the same         | Support mixing           | More             | Same             |
| Passthrough  | Less          | Less      | Nearly the same         | Not support mixing       | Less             | Same             |

- For record implementation, the two architectures are the same. Both architectures can meet user's record function needs.
- For playback implements, the two architectures are different. If the user has the requirements to play two sounds together at the same time, choose mixer architecture, because only the mixer architecture can do the mixing.
- The mixer architecture takes more memory and has a bigger code size. If the user wants to save memory and code size and has no requirements of audio mixing, choose the passthrough architecture.

## 36.2 Terminology

The meanings of some widely-used audio terms in this chapter are listed below.

| Terms   | Introduction                                                                                                                                                                                                                   |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PCM     | PCM (Pulse Code Modulation), audio data is a raw stream of uncompressed audio sample data, which is standard digital audio data converted from analog signals through sampling, quantization, and encoding.                    |
| channel | A channel sound is an independent audio signal captured or played in different spatial positions during recording or playing, so the number of channels is the number of sound sources during sound recording or the number of |

|                 |                                                                                                                                                                                                                                                                                  |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 | speakers during playback.                                                                                                                                                                                                                                                        |
| mono            | Mono means only one single-channel sound.                                                                                                                                                                                                                                        |
| stereo          | Stereo means two channels.                                                                                                                                                                                                                                                       |
| bit depth       | Bit depth (sample depth) represents how finely the sound intensity is recorded in the sample. Sampling depth can be understood as the resolution of processing the sound. The larger the value, the higher the resolution, and the more realistic the sound recorded and played. |
| sample          | A number representing the audio value of a single channel at a point in time.                                                                                                                                                                                                    |
| sample rate     | The audio sampling rate refers to the number of frames that the signal is sampled per unit time. The higher the sampling frequency, the more realistic and natural the waveform will be.                                                                                         |
| frame           | A frame is a sound unit whose length is the product of the sample length (number of samples) multiplies the number of channels.                                                                                                                                                  |
| gain            | Audio signal gain control to adjust the signal level.                                                                                                                                                                                                                            |
| interleaved     | It is a recording method of audio data. In the interleaved mode, the data is stored in a continuous manner, that is, the left channel sample and the right channel sample of frame 1 are first stored, and then the storage of frame 2 is started.                               |
| latency         | Time delay when a signal passes through the whole system.                                                                                                                                                                                                                        |
| overflow        | The buffer is too full to let buffer producer write more data.                                                                                                                                                                                                                   |
| underrun        | The buffer producer is too slow to write data to the buffer so that the buffer is empty when the consumer wants to consume data.                                                                                                                                                 |
| xrun            | Overflow or underrun.                                                                                                                                                                                                                                                            |
| volume          | Volume, also known as sound intensity and loudness, refers to the human ear's subjective perception of the intensity of the sound heard, and its objective evaluation scale is the amplitude of the sound.                                                                       |
| hardware volume | The volume of audio codec.                                                                                                                                                                                                                                                       |
| software volume | The volume set in software algorithm.                                                                                                                                                                                                                                            |
| resample        | Convert the sample rate.                                                                                                                                                                                                                                                         |
| reformat        | Convert the bit depth of the sample.                                                                                                                                                                                                                                             |
| mix             | Mix several audio streaming together. Users can hear several audio streaming playing together.                                                                                                                                                                                   |
| Audio codec     | The DAC and ADC controller inside the chip.                                                                                                                                                                                                                                      |

## 36.3 Data Format

This section describes the data format that audio framework and audio HAL supports. The common part of audio framework and audio HAL is described here. And the different parts will be described in their own sections.

Both audio framework and audio HAL support interleaved streaming data.

The two-channel interleaved data is illustrated in [Figure 36-3](#), and the four-channel interleaved data is illustrated in [Figure 36-4](#).



Figure 36-3 Two-channel interleaved

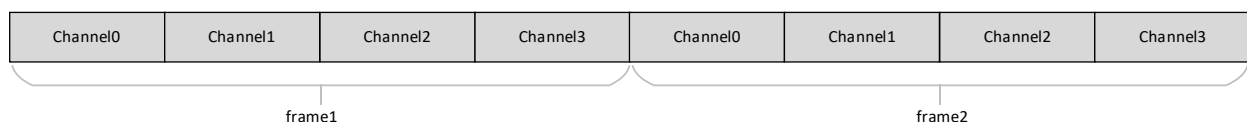


Figure 36-4 Four-channel interleaved

### 36.3.1 Framework Format

This section describes the format that Audio Framework supports. Before playback, or capture, make sure your sound format is supported.

Audio Framework has the following types of bit depth:

- RTAUDIO\_FORMAT\_INVALID - invalid bit depth of audio stream
- RTAUDIO\_FORMAT\_PCM\_8\_BIT - audio stream has 8-bit depth
- RTAUDIO\_FORMAT\_PCM\_16\_BIT - audio stream has 16-bit depth
- RTAUDIO\_FORMAT\_PCM\_32\_BIT - audio stream has 32-bit depth

- RTAUDIO\_FORMAT\_PCM\_FLOAT - audio stream has 32-bit float format
- RTAUDIO\_FORMAT\_PCM\_24\_BIT\_PACKED - audio stream has 24-bit depth
- RTAUDIO\_FORMAT\_PCM\_8\_24\_BIT - audio stream has 24-bit + 8-bit depth

The following table describes the supported formats for playback and recording. 'Y' means the format is supported; 'N' means the format is not supported.

| Bit depth                        | Playback-Mixer | Playback-Passthrough | Capture (Mixer/Passthrough) |
|----------------------------------|----------------|----------------------|-----------------------------|
| RTAUDIO_FORMAT_PCM_8_BIT         | Y              | Y                    | Y                           |
| RTAUDIO_FORMAT_PCM_16_BIT        | Y              | Y                    | Y                           |
| RTAUDIO_FORMAT_PCM_32_BIT        | Y              | N                    | N                           |
| RTAUDIO_FORMAT_PCM_FLOAT         | Y              | N                    | N                           |
| RTAUDIO_FORMAT_PCM_24_BIT_PACKED | Y              | N                    | N                           |
| RTAUDIO_FORMAT_PCM_8_24_BIT      | N              | Y                    | Y                           |

The sample rate is another important format of audio streaming. For playback and recording, audio framework supports the following sample rates. 'Y' means the sample rate is supported; 'N' means the sample rate is not supported.

| Sample rate | Playback (Mixer/Passthrough) | Capture (Mixer/Passthrough) |
|-------------|------------------------------|-----------------------------|
| 8000        | Y                            | Y                           |
| 11025       | Y                            | Y                           |
| 16000       | Y                            | Y                           |
| 22050       | Y                            | Y                           |
| 32000       | Y                            | Y                           |
| 44100       | Y                            | Y                           |
| 48000       | Y                            | Y                           |
| 88200       | Y                            | Y                           |
| 96000       | Y                            | Y                           |
| 192000      | N                            | N                           |

To do audio streaming, the channel count parameter setting is necessary, too. For playback and recording, audio framework supports the following channel counts. 'Y' means the channel count is supported; 'N' means the channel count is not supported.

| Channel count | Playback (Mixer/Passthrough) | Capture (Mixer/Passthrough) |
|---------------|------------------------------|-----------------------------|
| 1 (mono)      | Y (mono)                     | Y (mono)                    |
| 2 (stereo)    | Y (left, right)              | Y (left, right)             |
| 4 (quad)      | N                            | Y                           |
| 6             | N                            | Y                           |
| 8             | N                            | Y                           |

## 36.3.2 HAL Format

Mixer and passthrough architecture have the same audio HAL. Audio Hal has the following types of bit depth:

- AUDIO\_HW\_FORMAT\_INVALID - invalid bit depth of audio stream
- AUDIO\_HW\_FORMAT\_PCM\_8\_BIT - audio stream has 8-bit depth
- AUDIO\_HW\_FORMAT\_PCM\_16\_BIT - audio stream has 16-bit depth
- AUDIO\_HW\_FORMAT\_PCM\_32\_BIT - audio stream has 32-bit depth
- AUDIO\_HW\_FORMAT\_PCM\_FLOAT - audio stream has 32-bit float format
- AUDIO\_HW\_FORMAT\_PCM\_24\_BIT\_PACKED - audio stream has 24-bit depth
- AUDIO\_HW\_FORMAT\_PCM\_8\_24\_BIT - audio stream has 24-bit + 8-bit depth

If using the Audio HAL interface, please check the bit depth HAL supported for Playback and Capture. 'Y' means the format is supported; 'N' means the format is not supported.

| Bit depth                  | Playback | Capture |
|----------------------------|----------|---------|
| AUDIO_HW_FORMAT_PCM_8_BIT  | Y        | Y       |
| AUDIO_HW_FORMAT_PCM_16_BIT | Y        | Y       |
| AUDIO_HW_FORMAT_PCM_32_BIT | N        | N       |
| AUDIO_HW_FORMAT_PCM_FLOAT  | N        | N       |

|                                   |   |   |
|-----------------------------------|---|---|
| AUDIO_HW_FORMAT_PCM_24_BIT_PACKED | N | N |
| AUDIO_HW_FORMAT_PCM_8_24_BIT      | Y | Y |

The sample rate is another important format of HAL audio streaming. For playback and recording, audio HAL supports the following sample rates. 'Y' means the sample rate is supported; 'N' means the sample rate is not supported.

| Sample rate | Playback | Capture |
|-------------|----------|---------|
| 8000        | Y        | Y       |
| 11025       | Y        | Y       |
| 16000       | Y        | Y       |
| 22050       | Y        | Y       |
| 32000       | Y        | Y       |
| 44100       | Y        | Y       |
| 48000       | Y        | Y       |
| 88200       | Y        | Y       |
| 96000       | Y        | Y       |
| 192000      | N        | N       |

To do audio streaming, the channel count parameter setting is necessary, too. For playback and recording, audio HAL supports the following channel counts. 'Y' means the channel count is supported; 'N' means the channel count is not supported.

| Channel count | Playback        | Capture         |
|---------------|-----------------|-----------------|
| 1 (mono)      | Y               | Y               |
| 2 (stereo)    | Y (left, right) | Y (left, right) |
| 4 (quad)      | N               | Y               |
| 6             | N               | Y               |
| 8             | N               | Y               |

## 36.4 Architecture

### 36.4.1 Playback Architecture

The block diagram of audio mixer playback architecture is shown in [Figure 36-5](#).

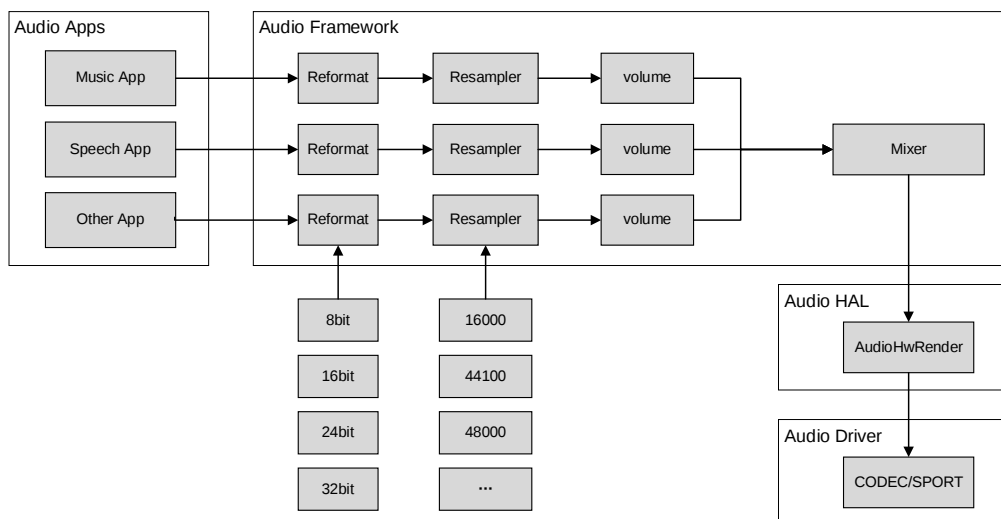


Figure 36-5 Playback mixer architecture

The block diagram of audio passthrough playback architecture is shown in [Figure 36-6](#).

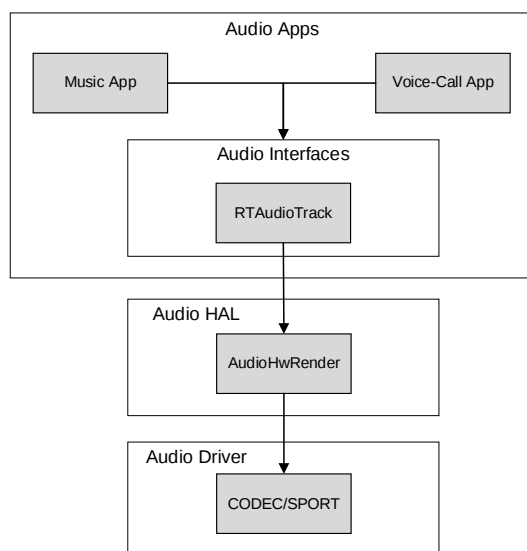


Figure 36-6 Playback passthrough architecture

The audio playback architecture includes the following sub-modules:

- Audio Framework (only mixer architecture)
  - Audio framework is responsible for audio playback sound mixing.
  - Audio framework supports at most 32 sound playing together.
  - Before mixing, all sound will be converted to one unified audio format, which is 16-bit, 44100Hz, 2-channel currently. Sub-modules reformat, resampler are responsible to do the conversion. There is also volume sub-module in audio framework to adjust volumes for different audio types, for example, music, speech may have different volumes.
  - Audio framework supports sound rate from 8k-96k, channel mono/stereo, format 8-bit, 16-bit, 24-bit, and 32-bit float.
- Audio HAL
  - Audio HAL gets playback data from audio framework, and sends the data to audio driver.
- Audio Driver
  - Audio Driver gets playback data from audio HAL and sends data to audio hardware.

## 36.4.2 Record Architecture

Audio mixer and passthrough have the same record architecture. The block diagram of audio record is shown in [Figure 36-7](#).

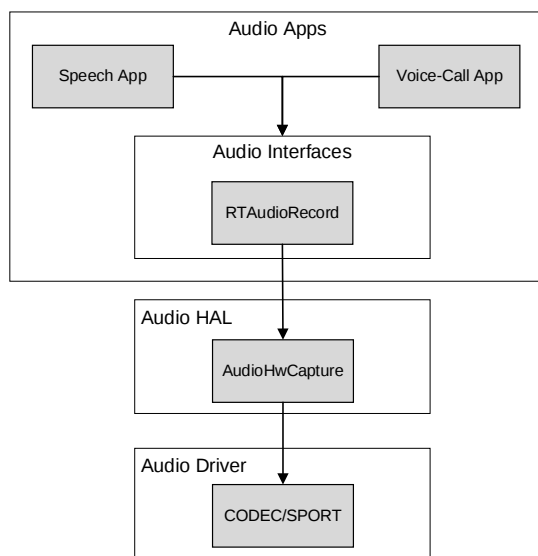


Figure 36-7 Record architecture

The audio record architecture includes the following sub-modules:

- RTAudioRecord
  - RTAudioRecord captures data from Audio HAL, and provides data to audio applications, which want to record data.

- Audio HAL
  - Audio HAL gets record data from Audio driver, and sends the data to RTAudioRecord.
- Audio Driver
  - Audio Driver gets record data from Audio hardware, and sends data to audio HAL.

### 36.4.3 Control Architecture

Audio mixer and passthrough have the same control architecture. The block diagram of audio control is shown in [Figure 36-8](#).

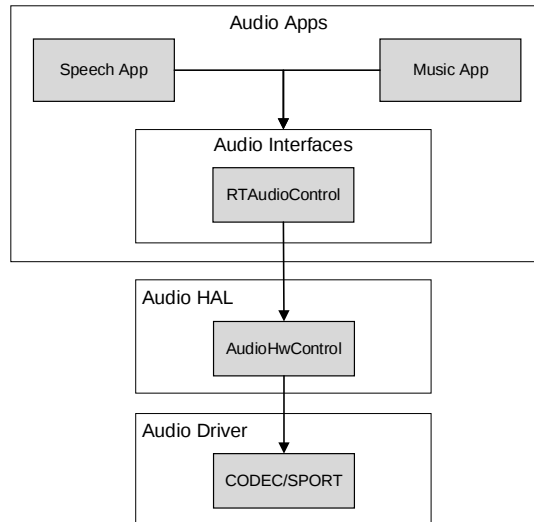


Figure 36-8 Control architecture

The audio control architecture includes the following sub-modules:

- RTAudioControl
  - Called by Apps, and interacts with HAL to do audio control settings.
- HAL
  - Audio HAL to do audio control settings by calling Driver APIs.
- Driver
  - Audio Driver to control audio codec hardware.

#### 36.4.3.1 Hardware Volume

RTAudioControl provides interfaces to set and get hardware volume.

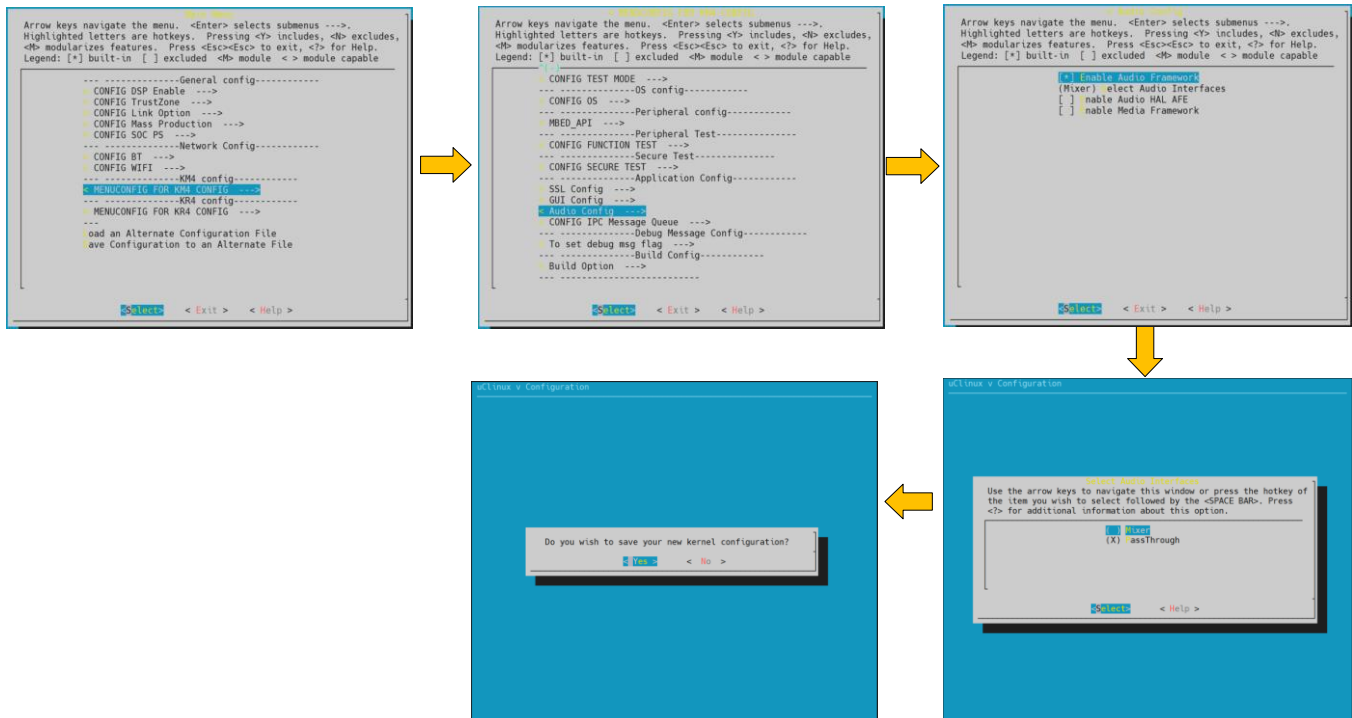
```
#include "audio/audio_control.h"
int32_t RTAudioControl_SetHardwareVolume(float left_volume, float right_volume)
```

Users set the left and right channel volume to 0.0-1.0, linear maps to -65.625-MAXdb.

## 36.5 Configurations

### 36.5.1 MenuConfig

If users want to use audio interfaces, select the following audio configurations, and choose the suitable audio architecture according to section [36.1.3](#).



## 36.5.2 Framework Configuration

Audio Framework configurations lie in `\component\soc\amebadplus\usrcfg\ameba_audio_mixer_usrcfg.cpp`.

If users want to change the audio HAL period buffer size, or audio mixer's buffer policy, please change `kPrimaryAudioConfig` according to `\component\soc\amebadplus\usrcfg\include\ameba_audio_mixer_usrcfg.h`'s description.

The config `out_min_frames_stage` in `kPrimaryAudioConfig` only supports `RTAUDIO_OUT_MIN_FRAMES_STAGE1` and `RTAUDIO_OUT_MIN_FRAMES_STAGE2`.

- `RTAUDIO_OUT_MIN_FRAMES_STAGE1` means more data output to audio HAL one time
  - `RTAUDIO_OUT_MIN_FRAMES_STAGE2` means less data output to audio HAL one time.
- `RTAUDIO_OUT_MIN_FRAMES_STAGE2` may reduce the framework's latency, but may cause noise. It's the user's choice to set it.

## 36.5.3 HAL Configuration

Audio hardware configurations lie in `\component\soc\amebadplus\usrcfg\include\ameba_audio_hw_usrcfg.h`.

Different boards have different configurations. For example, some boards need to use an amplifier, while others do not. Different boards may use different pins to enable the amplifier; the start-up time is different for different amplifiers. In addition, the pins used by each board's DMICs may be different, and the stable time of DMICs may be different. All the information needs to be configured in the configuration file.

The `ameba_audio_hw_usrcfg.h` file has the description for each configuration, please set them according to the description.

## 36.6 Interfaces

The audio component provides two layers of interfaces.

| Interface layers           | Introduction                                                                           |
|----------------------------|----------------------------------------------------------------------------------------|
| Audio Driver Interfaces    | Audio Hardware Interfaces.                                                             |
| Audio HAL Interfaces       | Audio Hardware Abstraction Layer Interfaces.                                           |
| Audio Framework Interfaces | High-level Interfaces for applications to render/capture stream, set volume and so on. |

The interfaces layer is shown in [Figure 36-9](#).

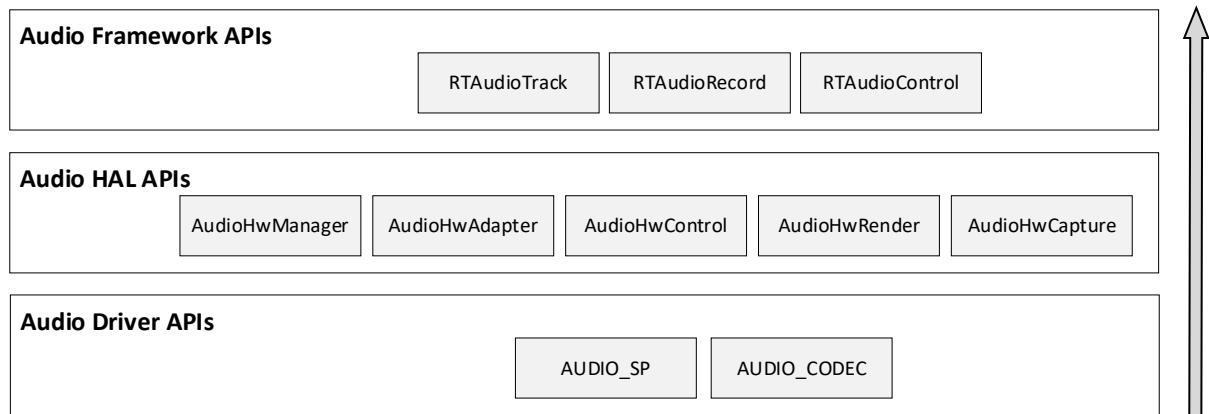


Figure 36-9 Audio interfaces

## 36.6.1 Driver Interfaces

### 36.6.1.1 Audio Clock and Function APIs

| API                                           | Introduction                                            |
|-----------------------------------------------|---------------------------------------------------------|
| <code>RCC_PeriphClockCmd</code>               | Enable or disable the APB peripheral clock and function |
| <code>RCC_PeriphClockSource_SPORT</code>      | Configure SPORT clock                                   |
| <code>RCC_PeriphClockSource_AUDIOCODEC</code> | Select audio codec clock                                |

### 36.6.1.2 SPORT APIs

| API                                  | Introduction                                                                                             |
|--------------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>AUDIO_SP_StructInit</code>     | Fill each <code>SP_StructInit</code> member with its default value                                       |
| <code>AUDIO_SP_Register</code>       | Register audio SPORT with its index, direction, and <code>SP_StructInit</code> members.                  |
| <code>AUDIO_SP_Unregister</code>     | Unregister audio SPORT with its index                                                                    |
| <code>AUDIO_SP_Reset</code>          | Reset SPORT                                                                                              |
| <code>AUDIO_SP_GetTXChnLen</code>    | Get the audio SPORT Tx channel length                                                                    |
| <code>AUDIO_SP_GetRXChnLen</code>    | Get the audio SPORT Rx channel length                                                                    |
| <code>AUDIO_SP_SetTXClkDiv</code>    | Set the audio SPORT Tx BCLK and LRCK                                                                     |
| <code>AUDIO_SP_SetRXClkDiv</code>    | Set the audio SPORT Rx BCLK and LRCK                                                                     |
| <code>AUDIO_SP_SetMclk</code>        | Set audio SPORT MCLK enable or disable                                                                   |
| <code>AUDIO_SP_SetMclkDiv</code>     | Set the audio SPORT MCLK                                                                                 |
| <code>AUDIO_SP_SetFixBclk</code>     | Set audio SPORT fixed BCLK mode                                                                          |
| <code>AUDIO_SP_SelFixBclk</code>     | Select audio SPORT BCLK divider factor                                                                   |
| <code>AUDIO_SP_TXCHNSrcSel</code>    | Select audio SPORT Tx channel source                                                                     |
| <code>AUDIO_SP_RXFIFOSrcSel</code>   | Select audio SPORT Rx FIFO source                                                                        |
| <code>AUDIO_SP_TXSetFifo</code>      | Set audio SPORT TX FIFO enable or disable                                                                |
| <code>AUDIO_SP_RXSetFifo</code>      | Set audio SPORT RX FIFO enable or disable                                                                |
| <code>AUDIO_SP_Init</code>           | Initialize the audio SPORT registers according to the specified parameters in <code>SP_InitStruct</code> |
| <code>AUDIO_SP_TXStart</code>        | Start or stop SPORT Tx path                                                                              |
| <code>AUDIO_SP_RXStart</code>        | Start or stop SPORT Rx path                                                                              |
| <code>AUDIO_SP_DmaCmd</code>         | Enable or disable SPORT DMA handshake                                                                    |
| <code>AUDIO_SP_SetSelfLPBK</code>    | Set SPORT self-loopback mode                                                                             |
| <code>AUDIO_SP_SetTXWordLen</code>   | Set the audio SPORT Tx word length                                                                       |
| <code>AUDIO_SP_SetRXWordLen</code>   | Set the audio SPORT Rx word length                                                                       |
| <code>AUDIO_SP_GetTXWordLen</code>   | Get the audio SPORT Tx word length                                                                       |
| <code>AUDIO_SP_GetRXWordLen</code>   | Get the audio SPORT Rx word length                                                                       |
| <code>AUDIO_SP_SetMonoStereo</code>  | Set the audio SPORT channel number                                                                       |
| <code>AUDIO_SP_SetMasterSlave</code> | Set the audio SPORT master or slave mode                                                                 |
| <code>AUDIO_SP_TXGDMA_Init</code>    | Initialize GDMA peripheral for Tx data                                                                   |
| <code>AUDIO_SP_RXGDMA_Init</code>    | Initialize GDMA peripheral for Rx data                                                                   |

|                              |                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------|
| AUDIO_SP_TXGDMA_Restart      | Restart GDMA peripheral for Tx data                                                   |
| AUDIO_SP_RXGDMA_Restart      | Restart GDMA peripheral for Rx data                                                   |
| AUDIO_SP_LLPTXGDMA_Init      | Initialize Link-list mode GDMA peripheral for Tx data                                 |
| AUDIO_SP_LLPRXGDMA_Init      | Initialize Link-list mode GDMA peripheral for Rx data                                 |
| AUDIO_SP_SetTXCounter        | Set SPORT Tx counter                                                                  |
| AUDIO_SP_SetTXCounterCompVal | Set SPORT Tx counter compare value                                                    |
| AUDIO_SP_ClearTXCounterIrq   | Clear SPORT Tx counter IRQ                                                            |
| AUDIO_SP_SetPhaseLatch       | Set SPORT phase latch                                                                 |
| AUDIO_SP_GetTXCounterVal     | Get SPORT Tx counter value                                                            |
| AUDIO_SP_GetTXPhaseVal       | Get SPORT Tx phase value when channel length is 32bit, none of the other bits will do |
| AUDIO_SP_SetRXCounter        | Set SPORT Rx counter                                                                  |
| AUDIO_SP_SetRXCounterCompVal | Set SPORT Rx counter compare value                                                    |
| AUDIO_SP_ClearRXCounterIrq   | Clear SPORT Rx counter IRQ                                                            |
| AUDIO_SP_GetRXCounterVal     | Get SPORT Rx counter value                                                            |
| AUDIO_SP_GetRXPhaseVal       | Get SPORT RX phase when channel length is 32bit, none of the other bits will do       |
| AUDIO_SP_SetDirectOutMode    | Set SPORT directout mode                                                              |
| AUDIO_SP_Deinit              | De-initialize the audio SPORT index and direction                                     |
| AUDIO_SP_SetPinMux           | Set the AUDIO SPORT pinmux function mux: dout or din                                  |

### 36.6.1.3 Codec APIs

| API                           | Introduction                                                            |
|-------------------------------|-------------------------------------------------------------------------|
| AUDIO_CODEC_SetAudioIP        | Enable or disable audio codec IP                                        |
| AUDIO_CODEC_SetI2SIP          | Enable or disable I2S IP                                                |
| AUDIO_CODEC_SetI2SSRC         | Select I2S master source.                                               |
| AUDIO_CODEC_SetI2SRXTDM       | Select I2S Rx and I2S Rx TDM mode                                       |
| AUDIO_CODEC_I2S_StructInit    | Default I2S initialization parameter                                    |
| AUDIO_CODEC_SetI2SParameters  | Set I2S initialization parameter                                        |
| AUDIO_CODEC_SetADCSRsrc       | Select ADC sample rate and source, all ADC share source and sample rate |
| AUDIO_CODEC_EnableADC         | Enable or disable per AD and AD FIFO channel clock                      |
| AUDIO_CODEC_SetADCVolume      | Set the gain of ADC digital volume                                      |
| AUDIO_CODEC_SetADCHPF         | Set per ADC channel HPF mode and select HPF FC                          |
| AUDIO_CODEC_SetADCASRC        | Set ADC ASRC mode                                                       |
| AUDIO_CODEC_SetADCMute        | Mute ADC path                                                           |
| AUDIO_CODEC_SetADCMixMute     | Set per ADC mix mute type and state                                     |
| AUDIO_CODEC_SetADCDmicFilter  | Enable per DIMC channel filter                                          |
| AUDIO_CODEC_SetDmicClk        | Enable and select DMIC clock                                            |
| AUDIO_CODEC_SetDmicSrc        | Set DMIC channel source                                                 |
| AUDIO_CODEC_SetADCEQClk       | Enable or disable adc channel EQ clk                                    |
| AUDIO_CODEC_SetADCEQFilter    | set EQ band as filter for DMIC rx                                       |
| AUDIO_CODEC_SetADCEQBand      | Enable or disable adc channel EQ band                                   |
| AUDIO_CODEC_SetADCZDET        | Set ADC path zero detection function                                    |
| AUDIO_CODEC_SetADCZDETTimeOut | Set ADC path zero detection function                                    |
| AUDIO_CODEC_Record            | Audio codec record flow for test                                        |

### 36.6.1.4 I2S PLL APIs

| API                      | Introduction                                                                                       |
|--------------------------|----------------------------------------------------------------------------------------------------|
| PLL_I2S0_CLK             | I2S0 CPU PLL CLOCK choose when system clock is an integer multiple of 98.304M or 45.1584M          |
| PLL_I2S1_CLK             | I2S1 CPU PLL CLOCK choose when system clock is an integer multiple of 98.304M or 45.1584M          |
| PLL_I2S0_CLK_Auto        | I2S0 CPU PLL CLOCK auto choose when system clock is not an integer multiple of 98.304M or 45.1584M |
| PLL_I2S1_CLK_Auto        | I2S1 CPU PLL CLOCK auto choose when system clock is not an integer multiple of 98.304M or 45.1584M |
| PLL_I2S_CLKGet           | Get 98.304M or 45.1584M                                                                            |
| PLL_I2S_98P304M_ClkTune  | I2S PLL output adjust by ppm when system clock is an integer multiple of 98.304M                   |
| PLL_I2S_45P1584M_ClkTune | I2S PLL output adjust by ppm when system clock is an integer multiple of 45.1584M                  |

There are two ways to generate I2S PLL, one is that the system clock is an integer multiple of 98.304M or 45.1584M, we add the system clock

in SocClk\_Info array, so you can modify the index of SocClk\_Info array in bootloader\_km4.c. When you need high-quality audio applications, you can use this method. And the other is that the system clock is not an integer multiple of 98.304M or 45.1584M, in this case, we automatically get the 98.304M or 45.1584M. The details are shown in [Figure 36-10](#).

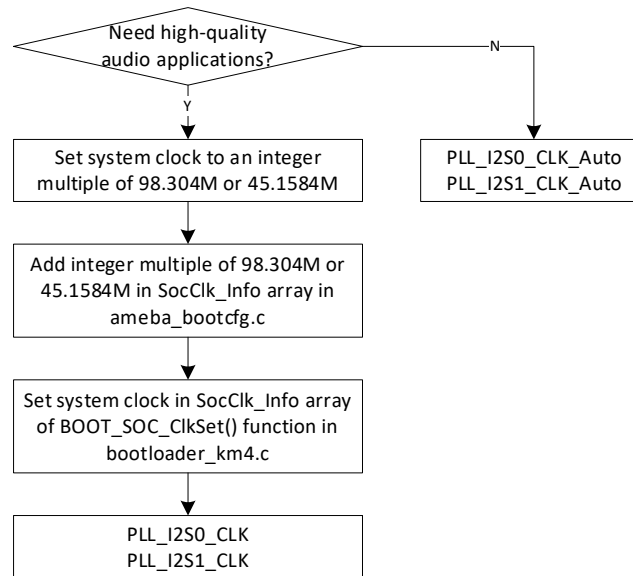


Figure 36-10 How to use I2S PLL interfaces

## 36.6.2 HAL Interfaces

Audio HAL provides AudioHwRender/AudioHwCapture/AudioHwControl interfaces to interact with audio hardware. The interfaces lie in /component/audio/interfaces/hardware/audio. The interfaces have specific descriptions in them, read them before use.

- AudioHwRender: receives PCM data from the upper layer, writes data via audio driver to send PCM data to hardware, and provides information about audio output hardware driver.
- AudioHwCapture: receives PCM data via audio driver and sends to the upper layer.
- AudioHwControl: receives control calling from the upper layer, and sets control information to the driver.

The AudioHwRender/AudioHwCapture is managed by AudioHwAdapter interface. It is responsible for creating/destroying AudioHwRender/AudioHwCapture instance. AudioHwAdapter is a physical or virtual hardware to process audio stream. It contains a set of ports and pins as shown in [Figure 36-11](#).

- Port – the stream input of the audio adapter is called “port”.
- Pin – The output stream of audio adapter is called “pin”.

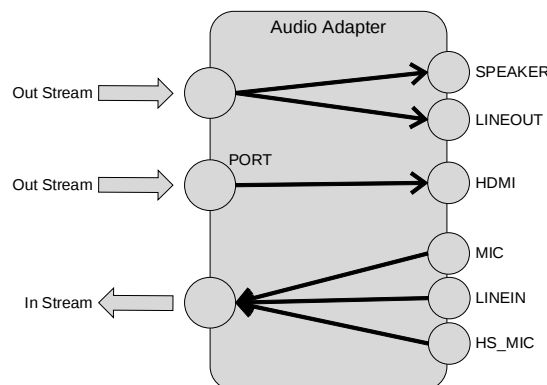


Figure 36-11 AudioHwAdapter example

The AudioHwManager manages system's all AudioHwAdapters and loads a specific adapter driver based on the given audio adapter descriptor.

### 36.6.2.1 Using AudioHwRender

Users can check the example of AudioHwRender in \component\example\audio\audio\_hal\_render.

Here is the description showing how to use audio HAL interfaces to play audio raw data (PCM format):

- (1) Use `GetAudioManagerFuncs` to get `AudioHwManager` instance:

```
struct AudioHwManager *audio_manager = GetAudioManagerFuncs();
```

- (2) Use `GetAllAdapters` to get all audio adapter descriptors:

```
int32_t size = -1;
struct AudioHwAdapterDescriptor *descs = nullptr;
audio_manager->GetAllAdapters(audio_manager, &descs, &size);
```

- (3) Choose a specific adapter to play (currently audio manager only support primary audio adapter):

```
for (int index = 0; index < size; index++){
 struct AudioHwAdapterDescriptor *desc = &descs[index];
 for (int port = 0; (desc != nullptr && port < static_cast<int>(desc->portNum)); port++){
 if (desc->ports[port].dir == PORT_OUT &&
 (audio_manager->LoadAdapter(audio_manager, desc, &audio_adapter)) == 0) {
 render_port = desc->ports[port];
 break;
 }
 }
}
```

- (4) Create `AudioHwSampleAttributes` according to the sample rate, channel, format, and `AudioHwDeviceDescriptor`, then use `CreateRender` to create an `AudioHwRender` based on the specific audio adapter:

```
struct AudioHwSampleAttributes param;
param.sample_rate = sample_rate;
param.channel_count = channel_count;
param.format = AUDIO_FORMAT_PCM_16_BIT;
param.interleaved = false;
```

```
struct AudioHwDeviceDescriptor device_desc;
device_desc.port_id = render_port.port_id;
device_desc.pins = AUDIO_HW_PIN_OUT_SPEAKER;
device_desc.desc = NULL;
int32_t ret = audio_adapter->CreateRender(audio_adapter, &device_desc, ¶m,
&audio_render);
```

- (5) Use `GetBufferSize` to get `AudioHwRender` buffer size:

```
ssize_t size = ((struct AudioHalStream *)audio_render)->GetBufferSize((struct
AudioHalStream *)audio_render);
```

- (6) Write PCM data to `AudioHwRender` repeatedly. This size can be defined by users, instead of the size got in the last step. Users need to make sure the size/frame\_size is integer.

```
int32_t bytes = audio_render->Write(audio_render, buffer, size);
```

- (7) Use `DestroyRender` to close `AudioHwRender` when finishing playing:

```
audio_adapter->DestroyRender(audio_adapter, audio_render);
```

- (8) Use `UnloadAdapter` to unload the `AudioHwAdapter` and finally call `DestoryAudioManager` to release `AudioHwManager` instance:

```
audio_manager->UnloadAdapter(audio_manager, desc, &audio_adapter);
DestoryAudioManager(&audio_manager);
```

### 36.6.2.2 Using AudioHwCapture

Users can check the example of `AudioHwRender` in `\component\example\audio\audio_hal_capture`.

Here is the description showing how to use audio HAL interfaces to capture audio raw data:

- (1) Use `GetAudioManagerFuncs` to get `AudioHwManager` instance:

```
struct AudioHwManager *audio_manager = GetAudioManagerFuncs();
```

- (2) Use `GetAllAdapters` to get all audio adapter descriptors:

```
int32_t size = -1;
struct AudioHwAdapterDescriptor *descs = nullptr;
audio_manager->GetAllAdapters(audio_manager, &descs, &size);
```

- (3) Choose a specific adapter to capture (currently audio manager only support primary audio adapter):

```
for (int index = 0; index < size; index++){
 struct AudioHwAdapterDescriptor *desc = &descs[index];
 for (int port = 0; (desc != nullptr && port < static_cast<int>(desc->portNum)); port++){
 if (desc->ports[port].dir == PORT_IN &&
 (audio_manager->LoadAdapter(audio_manager, desc, &audio_adapter)) == 0) {
```

```

mCapturePort = desc->ports[port];
 break;
}
}
}

```

- (4) Construct `AudioHwSampleAttributes` according to the sample rate, channel, format, and `AudioHwDeviceDescriptor`, then use `CreateCapture` to create an `AudioHwCapture` based on the specific audio adapter:

```

struct AudioHwSampleAttributes param;
param.sample_rate = sample_rate;
param.channel_count = channel_count;
param.format = AUDIO_FORMAT_PCM_16_BIT;
param.interleaved = false;
struct AudioHwDeviceDescriptor device_desc;
device_desc.port_id = render_port.port_id;
device_desc.pins = AUDIO_HW_PIN_IN_MIC;
device_desc.desc = NULL;
int32_t ret = audio_adapter->CreateCapture(audio_adapter, &device_desc, ¶m,
&audio_capture);

```

- (5) Use `GetBufferSize` to get `AudioHwCapture` buffer size:
- ```

ssize_t size = ((struct AudioHalStream *)audio_capture)->GetBufferSize((struct
AudioHalStream *)audio_capture);

```

- (6) Read PCM data from `AudioHwCapture` repeatedly. This size can be defined by users, instead of the size got in the last step. Users need to make sure the `size/frame_size` is integer.

```

int32_t bytes = audio_capture->Read(audio_capture, buffer, size);

```

- (7) Use `DestroyCapture` to close `AudioHwCapture` when finishing recording:

```

audio_adapter->DestroyCapture(audio_adapter, audio_capture);

```

- (8) Use `UnloadAdapter` to unload the `AudioHwAdapter`, and finally call `DestoryAudioManager` to release `AudioHwManager` instance.

```

audio_manager->UnloadAdapter(audio_manager, desc, &audio_adapter);
DestoryAudioManager(&audio_manager);

```

36.6.2.3 Using AudioHwControl

Here is an example showing how to use audio HAL interfaces to control audio codec:

`AudioHwCotrol` is always thread-safe, and the calling is convenient. To use `AudioHwCotrol`, the first parameter of the function call should always be `GetAudioHwControl()`. Take the hardware volume setting for example:

```

GetAudioHwControl()->SetHardwareVolume(GetAudioHwControl(), left_volume, right_volume);

```

36.6.3 Framework Interfaces

36.6.3.1 Streaming Interfaces

Audio Streaming Interfaces include `RTAudioTrack` and `RTAudioRecord` interfaces. The interfaces lie in `\component\audio\interfaces\audio`. The interfaces have specific descriptions in them, please read them before using.

- `RTAudioTrack`: initializes the format of playback data streaming in the framework, receives PCM data from the application, and writes data to Audio Framework (mixer) or Audio HAL (passthrough).
- `RTAudioRecord`: initializes the format of record data streaming in the framework, receives PCM data from Audio HAL, and sends data to applications.

36.6.3.1.1 Using RTAudioTrack

`RTAudioTrack` includes support for playing variety of common audio raw format types so that audio can be easily integrated into applications. At most 32 `RTAudioTracks` can play together.

Audio Framework has the following audio playback stream types. Applications can use the types to initialize `RTAudioTrack`. Framework gets the streaming type and does the volume mixing according to the types.

- `RTAUDIO_CATEGORY_MEDIA` - if the application wants to play music, then its type is `RTAUDIO_CATEGORY_MEDIA`, it can use this type to init `RTAudioTrack`. Then audio framework will know its type, and mix it with media's volume.
- `RTAUDIO_CATEGORY_COMMUNICATION` - if the application wants to start a phone call, it can output the phone call's sound, the sound's type should be `RTAUDIO_CATEGORY_COMMUNICATION`.
- `RTAUDIO_CATEGORY_SPEECH` - if the application wants to do voice recognition, and output the speech sound.

- RTAUDIO_CATEGORY_BEEP - if the sound is key tone, or other beep sound, then its type is RTAUDIO_CATEGORY_BEEP.

The test demo of RTAudioTrack lies in \component\example\audio\audio_track.

Here is an example showing how to play audio raw data:

- (1) Before using RTAudioTrack, RTAudioService should be initialized:

```
RTAudioService_Init();
```

- (2) To use RTAudioTrack to play a sound, create it:

```
struct RTAudioTrack* audio_track = RTAudioTrack_Create();
```

Apps can use the Audio Configs API to provide detailed audio information about a specific audio playback source, including stream type (type of playback source), format, number of channels, sample rate, and RTAudioTrack ringbuffer size. The syntax is as follows:

```
typedef struct {
    uint32_t category_type;
    uint32_t sample_rate;
    uint32_t channel_count;
    uint32_t format;
    uint32_t buffer_bytes;
} RTAudioTrackConfig;
```

Where

- category_type - define the stream type of the playback data source.
- sample_rate - playback source raw data's rate.
- channel_count - playback source raw data's channel number.
- format - playback source raw data's bit depth.
- buffer_bytes - ringbuffer size for RTAudioTrack to avoid xrun.

NOTE

The buffer_bytes in RTAudioTrackConfig is very important. The buffer size should always be more than the minimum buffer size Audio framework calculated. Otherwise overrun will occur.

Use the interface to get minimum RTAudioTrack buffer bytes, and use it as a reference to define RTAudioTrack buffer size, for example, you can use minimum buffer size *4 as buffer size. The bigger size you use, the smoother playing you will get, yet it may cause more latency. It's your choice to define the size.

```
int track_buf_size = RTAudioTrack_GetMinBufferBytes(audio_track, type, rate, format, channels) * 4;
```

- (3) Use this buffer size and other audio parameters to create RTAudioTrackConfig object, here's an example:

```
RTAudioTrackConfig track_config;
track_config.category_type = RTAUDIO_CATEGORY_MEDIA;
track_config.sample_rate = rate;
track_config.format = format;
track_config.buffer_bytes = track_buf_size;
track_config.channel_count = channel_count;
```

With RTAudioTrackConfig object, we can initialize RTAudioTrack. In this step, a ringbuffer will be created according to the buffer bytes.

```
RTAudioTrack_Init(audio_track, &track_config);
```

- (4) When all the preparations are completed, start RTAudioTrack and check if starts success.

```
if(RTAudioTrack_Start(audio_track) != 0){
    //track start fail
    return;
}
```

The default volume of RTAudioTrack is maximum 1.0, you can change the volume with the following API calling. Users can adjust the volume of track by using

```
RTAudioTrack_SetVolume(audio_track, 1.0, 1.0);
```

NOTE

- ◆ In the mixer architecture, this API sets the software volume of the current audio_track.
- ◆ In the passthrough architecture, this API is not supported.

- (5) Write audio data to the framework. The write_size can be defined by users. Users need to make sure the write_size/frame_size is integer.

```
RTAudioTrack_Write(audio_track, buffer, write_size, true);
```

- If users want to pause, stop writing data, and then call the following APIs to tell the framework to do pause:

```
RTAudioTrack_Pause(audio_track);
RTAudioTrack_Flush(audio_track);
```

- If users want to stop playing audio, stop writing data, and then call RTAudioTrack_Stop() API:

```
RTAudioTrack_Stop(audio_track);
```

- (6) Delete audio_track pointer when it's no use.

```
RTAudioTrack_Destroy(audio_track);
```

36.6.3.1.2 Using RTAudioRecord

RTAudioRecord supports variety of common audio raw format types, so that you can easily integrate record into applications.

RTAudioRecord supports the following audio input sources:

- RTPIN_IN_MIC - if the application wants to capture data from microphone, then choose this input source.
- RTPIN_IN_HS_MIC - if the application wants to capture data from headset microphone, then choose this input source.
- RTPIN_IN_HS_MIC - if the application wants to capture data from LINE-IN, then choose this input source.

The test demo of RTAudioRecord lies in \component\example\audio\audio_record.

Here is an example showing how to record audio raw data:

- (1) Create RTAudioRecord first:

```
struct RTAudioRecord *audio_record = RTAudioRecord_Create();
```

- (2) Apps can use the Audio Configs API to provide detailed audio information about a specific audio record source, including record device source, format, number of channels, and sample rate. The syntax is as follows:

```
typedef struct {
    uint32_t sample_rate;
    uint32_t channel_count;
    uint32_t format;
    uint32_t device;
} RTAudioRecordConfig;
```

Where

- sample_rate - record source raw data's rate.
- channel_count - record source raw data's channel number.
- format - record source raw data's bit depth.
- device - audio input device source for data record.

Here's an example showing how to create RTAudioRecordConfig object of RTAudioRecord:

```
RTAudioRecordConfig record_config;
record_config.sample_rate = rate;
record_config.format = RTAUDIO_FORMAT_PCM_16_BIT;
record_config.channel_count = channels;
record_config.device = RTPIN_IN_MIC;
```

With RTAudioRecordConfig object created, you can initialize RTAudioRecord, in this step, Audio HAL's AudioHwAdapter will be loaded, according to the audio input device source:

```
RTAudioRecord_Init(audio_record, &record_config);
```

- (3) When all the preparations are completed, start audio_record:

```
RTAudioRecord_Start(audio_record);
```

- (4) Read audio microphone data. The read size can be defined by users. Users need to make sure size/frame_size is integer.

```
RTAudioRecord_Read(audio_record, buffer, size, true);
```

- (5) When the record ends, stop the record:

```
RTAudioRecord_Stop(audio_record);
```

- (6) When audio_record no use, destroy it to avoid memory leak:

```
RTAudioRecord_Destroy(audio_record);
```

36.6.3.2 Control Interfaces

Audio Control Interfaces include RTAudioControl interfaces to interact with audio control HAL. RTAudioControl provides interfaces to set and get hardware volume, set output device, and so on. The interfaces lie in \component\audio\interfaces\audio\audio_control.h. The interfaces have specific descriptions, read them before use.

36.6.3.2.1 Using RTAudioControl

Here is an example of how to use RTAudioControl:

- (1) Call RTAudioControl to set audio hardware volume of DAC:

```
RTAudioControl_SetHardwareVolume(0.5, 0.5);
```

For playback and record case, most RTAudioControl APIs can be called at any time, any place, they can work directly. Only RTAudioControl_SetPlaybackDevice() can only be called before RTAudioService_Init() in mixer architecture, and before RTAudioTrack_Start() in passthrough architecture.

36.7 Audio Hardware Application

36.7.1 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.

36.7.1.1 DMIC-in Mono Mode

Tie the L/R of a digital microphone to ground or VDD if only one digital microphone is placed.

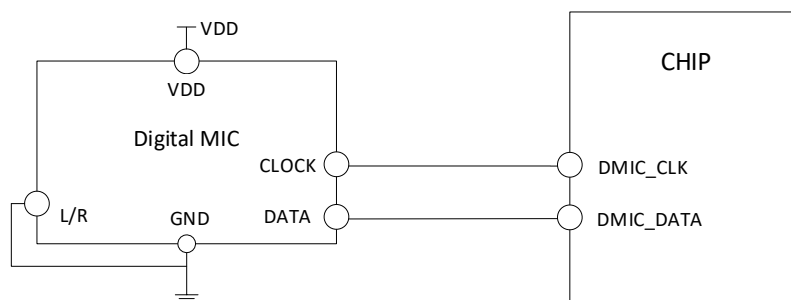


Figure 36-12 DMIC-in mono mode connection

36.7.1.2 DMIC-in Stereo Mode

Tie the L/R of two digital microphones to ground and VDD respectively if a stereo microphone is needed. The two microphones share the DMIC_DATA according to the rising/falling edge.

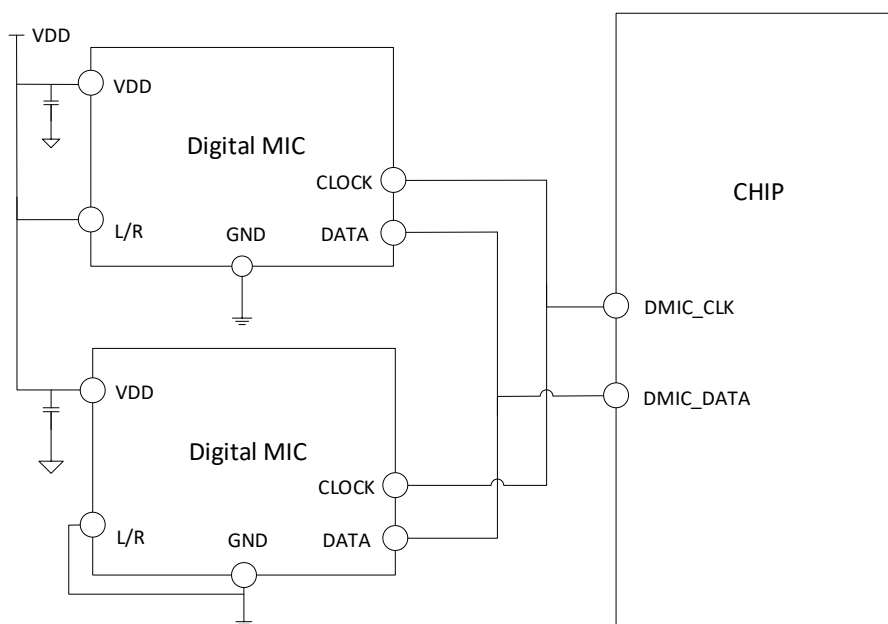


Figure 36-13 DMIC-in stereo mode connection

36.7.2 I2S Data Pin

The data paths of SPORT0 and SPORT1 are shown in [Figure 36-14](#) and [Figure 36-15](#).

- For I2S TDM mode: Only SD_O_0 and SD_I_0 can be used.
- For I2S Multi-IO mode: SD_O_0/1/2/3 and SD_I_0/1/2/3 all can be used. By default, use SD_O_0/1/2/3 and SD_I_0/1/2/3 in order according to the number of channels.

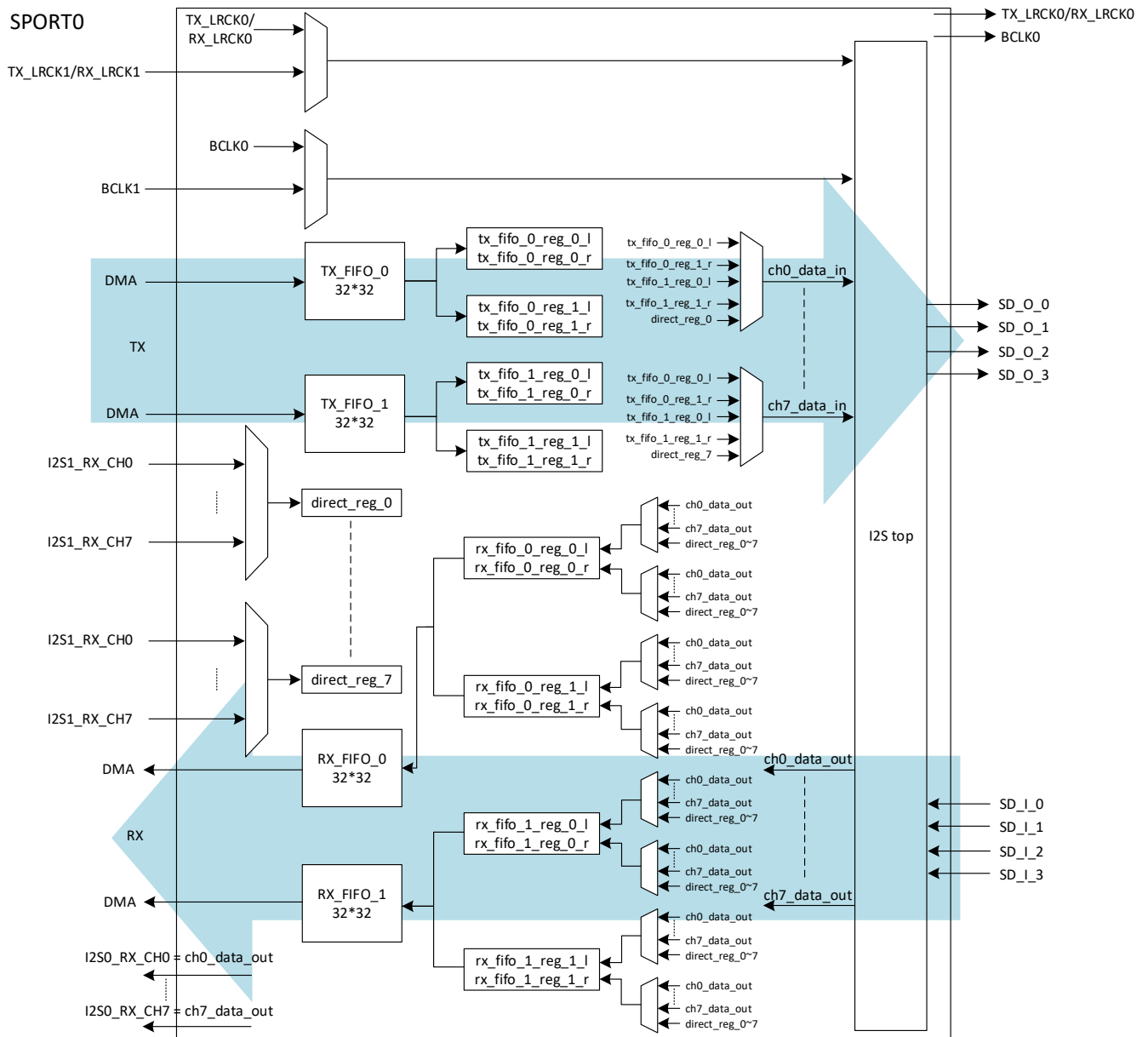


Figure 36-14 SPORT0 data path



- SD_O: AUDIO_SP_TXCHNSrcSel(u32 index, u32 fifo_num, u32 NewState)
- SD_I: AUDIO_SP_RXFIFOSrcSel(u32 index, u32 fifo_num, u32 NewState)

36.7.3 SPORT0 Rx with External I2S0

Application Note

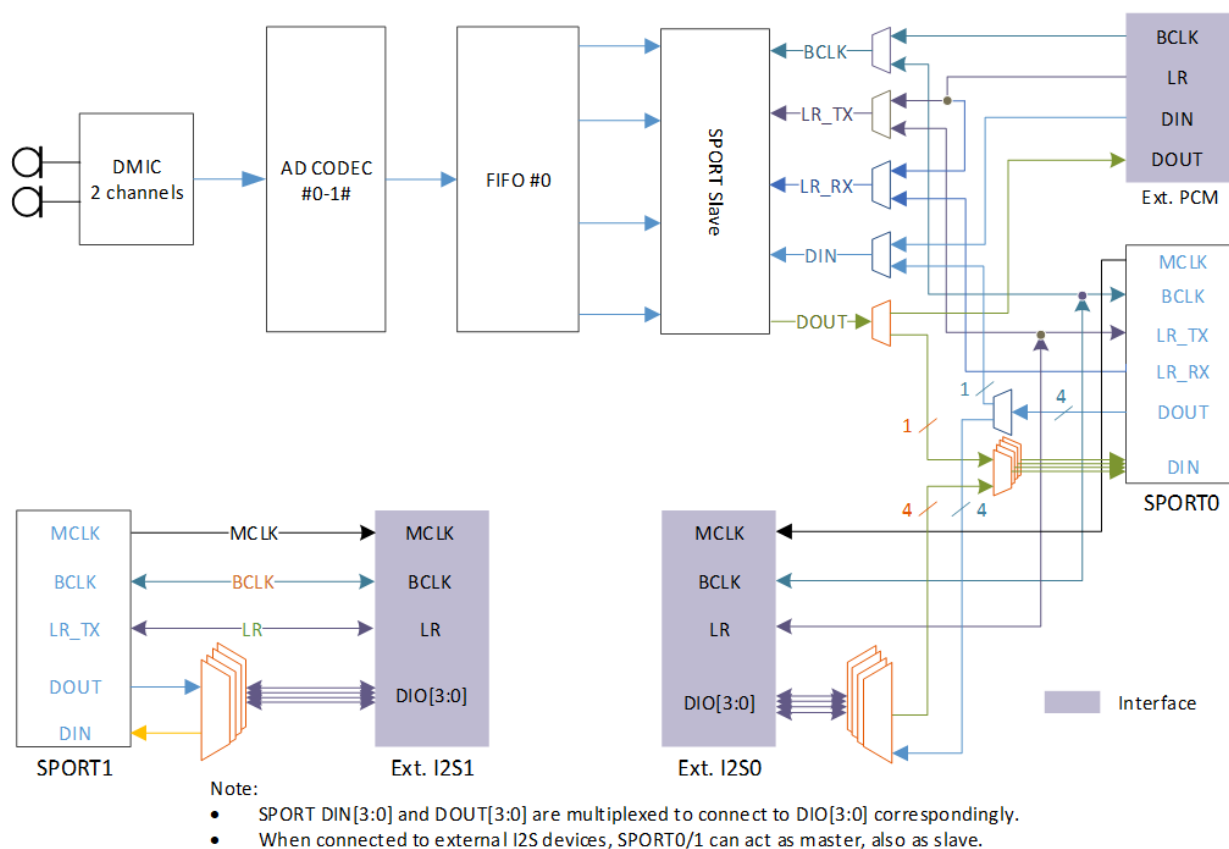


Figure 36-16 Audio block diagram

When SPORT0 RX connected with EXT.I2S0, the internal connection of SPORT Slave and SPORT0 must be disconnected by the interface:

```
AUDIO_CODEC_SetI2SSRC(I2S0, EXTERNAL_I2S)
```

NOTE

Calling this API can access register of audio codec, so you need to enable the function and clock of audio codec at first:

```
RCC_PeriphClockCmd(APBPeriph_AC, APBPeriph_AC_CLOCK, ENABLE)
```

Revision History

Date	Revision	Release Notes
2024-03-21	2.0	● Added the following chapters: ■ One-time Programmable Controller (OTPC) ■ CHIP Enable ■ Inter-Processor Communication (IPC) ■ Direct Memory Access Controller (DMAC) ■ Pseudo-Static Random Access Memory (PSRAM) ■ General Purpose Input/Output (GPIO) ■ Pin Controller (Pinctrl) ■ Pin Multiplexing Instructions ■ RTC_IO ■ Infrared Radiation (IR) ■ Light Emitting Diode Controller (LEDC) ■ Universal Serial Bus On-The-Go (USB OTG) ■ True Random Number Generator (TRNG) ■ Analog-to-Digital Converter (ADC) ■ Cap-Touch Controller (CTC) ■ Key-Scan ■ Audio ● Updated the SDK path
2023-06-30	1.0	Initial release