AN0500

Ameba-ZII Application Note

Abstract

Ameba-ZII is a high-integrated IC. Its features include 802.11 Wi-Fi, RF, Bluetooth and configurable GPIOs.
This manual introduces users how to develop Ameba-ZII, including SDK compiling and downloading image to Ameba-ZII.

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

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

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.

Revision History

Revision	Release Date	Summary
0.X- 1.X	2018 - 2024	Update for Ameba-ZII
2.0	2024/07/26	Support Ameba-ZII and Ameba-ZIIplus in the same document
2.1	2024/08/23	Set GCC IDE as the default build environment

Table of Contents

COPYRIGHT.....2

DISCLAIMER.....2

TRADEMARKS.....3

USING THIS DOCUMENT.....3

Revision History.....4

List of Figures.....8

List of Table.....9

1 Demo Board User Guide.....10

1.1 PCB Layout Overview.....10

1.2 Pin Mux Alternate Functions.....11

1.2.1 Pin Mux Table.....11

1.2.2 Pin-Out Reference.....12

2 SDK Architecture.....13

2.1 Component.....13

2.2 Doc, Project, Tools.....14

3 SDK Build Environment Setup.....15

3.1 Introduction.....15

3.2 Debugger Settings.....15

3.2.1 J-Link Debugger.....15

3.3 Log UART Settings.....18

3.3.1 EVB v2.0.....18

3.4 GCC Environment on Windows (Using Cygwin).....20

3.4.1 Install Cygwin.....20

3.4.2 Build Non-Trust Zone Project.....20

3.4.3 GCC Memory Configuration.....21

3.4.4 GCC Memory Overflow.....22

3.5 GCC Environment on Ubuntu/Linux.....23

3.5.1 Verify Device Connections.....23

3.5.2 Compile and Generate Binaries.....23

3.5.3 Download and Flash Binaries.....23

4 Image Tool.....25

4.1 Introduction.....25

4.2 Environment Setup 26
4.2.1 Hardware Setup 26
4.2.2 Software Setup 26
4.3 Image Download 27
5 Memory Layout 28
5.1 Memory Type 28
5.2 Flash Memory Layout 29
5.2.1 Partition Table 30
5.2.2 System Data 31
5.2.3 Boot Image 32
5.2.4 Firmware 1/Firmware 2 33
5.3 SRAM Layout 34
6 Boot Process 35
6.1 Boot Flow 35
7 Over-The-Air (OTA) Firmware Update 35
7.1 OTA Operation Flow 37
7.2 OTA Checksum Mechanism 38
7.3 Boot Process Flow 39
7.4 Upgraded Partition 39
7.5 Firmware Image Output 40
7.5.1 OTA Firmware Swap Behavior 40
7.5.2 Configuration for Building OTA Firmware 41
7.6 Implement OTA Over Wi-Fi 43
7.6.1 OTA Using Local Download Server Base on Socket 43
7.6.2 OTA Using Local Download Server Based on HTTP 44
8 Power Save 47
8.1 WLAN Power Management 47
8.1.1 Ameba LPS 47
8.1.2 Ameba IPS 47
8.2 Power Consumption Measurement 48
8.2.1 Hardware preparation 48
8.2.2 Build SDK 49
8.3 Power Consumption Result 49
9 Bluetooth 51

9.1 Features 51
9.2 BT Wi-Fi Coexist 51
9.3 Memory Usage 51
9.3.1 Wi-Fi Only 51
9.3.2 Wi-Fi + BT 51
9.4 Examples 51
9.4.1 ble_peripheral 51
9.4.2 ble_central 52
9.4.3 ble_scatternet 54
9.4.4 bt_beacon 55
9.4.5 bt_config 56
9.4.6 128-bit UUID Configuration 60
9.5 BT Transmit Power 61
9.6 BT Default MAC Address 62
10 Troubleshooting 63
10.1 Hard Fault 63
10.1.1 GCC Environment 63
10.2 Fault Message Redirection 66

List of Figures

Figure 1-1 Top View of Ameba-ZII 2V0 Dev Board 10
Figure 1-2 Ameba-ZII 2V0 Dev Board PCB Layout 10
Figure 1-3 Pin Out Reference for DEV_2V0 12
Figure 2-1 SDK architecture and description part 1 13
Figure 2-2 SDK architecture and description part 2 14
Figure 3-1 Connection between J-Link Adapter and Ameba-ZII SWD connector 15
Figure 3-2 Log UART via FT232 on EVB V2.0 18
Figure 4-1 AmebaZII Image Tool UI 25
Figure 4-2 Ameba-ZII EVB V2.0 Hardware Setup 26
Figure 5-1 Address Allocation of Different Memories on Ameba-ZII 28
Figure 5-2 Flash memory layout 29
Figure 6-1 Overview of boot flow 35
Figure 7-1 Methodology to Update Firmware via OTA 35
Figure 7-2 OTA Process Flow 37
Figure 7-3 Boot Process Flow 39
Figure 7-4 OTA update procedure 39
Figure 7-5 OTA Firmware SWAP Procedure 41
Figure 8-1 timeline of power saving 47
Figure 8-2 LPS state machine 47
Figure 8-3 IPS state machine 48
Figure 8-4 Power consumption measurement 48
Figure 8-5 Measure power consumption from micro usb 48

List of Table

Table 1-1 GPIOA Pin MUX: DEV_2V0 Board 11
Table 5-1 Size of Different Memories on Ameba-ZII 28
Table 5-2 Description of flash layout 29
Table 5-3 The layout of Partition table 30
Table 5-4 Layout of system data 31
Table 5-5 Definition for OTA section in system data 31
Table 5-6 Definition for Flash section in system data 31
Table 5-7 Definition for Log UART section in system data 32
Table 5-8 The layout of boot image 32
Table 5-9 The layout of firmware image 33
Table 5-10 AmebaZII DTCM memory layout 34
Table 5-11 Description of RAM layout 34
Table 8-1 power consumption summary 50
Table 8-2 Wi-Fi power consumption 50
Table 8-3 BT power consumption 50

1 Demo Board User Guide

1.1 PCB Layout Overview

RTL8720C embedded on Ameba-ZII DEV demo board, which consists of various I/O interfaces. For the details of the HDK, please contact us for further reference.

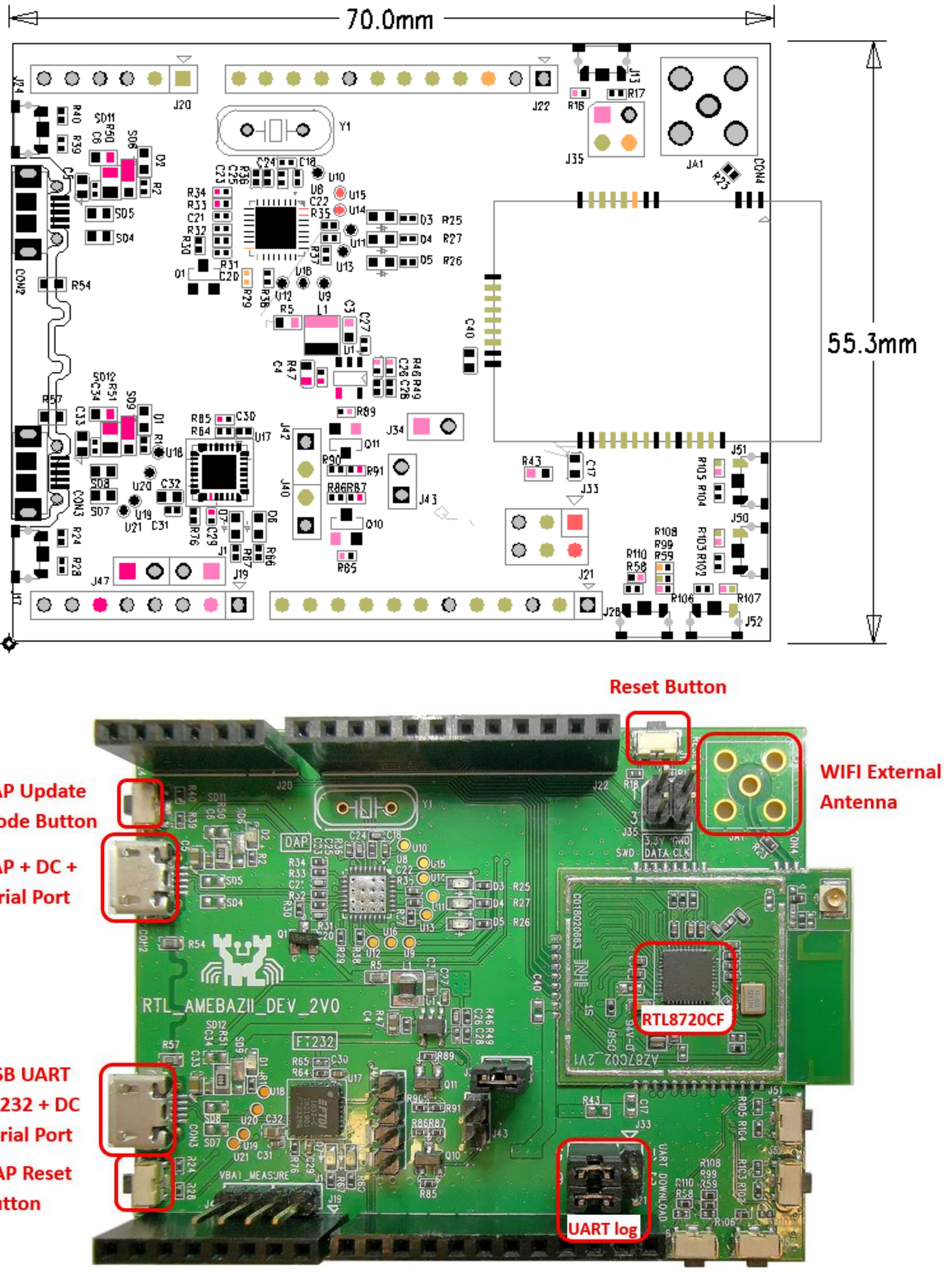


Figure 1-2 Ameba-ZII 2V0 Dev Board PCB Layout

1.2 Pin Mux Alternate Functions

1.2.1 Pin Mux Table

Pin Name	*SPIC-Flash/SDIO*	*JTAG*	*UART*	*SPI/WL_LED/EXT_32K*	*I2C*	*PWM*
GPIOA_0		JTAG_CLK	UART1_IN	EXT_32K		PWM[0]
GPIOA_1		JTAG_TMS	UART1_OUT	BT_LED		PWM[1]
GPIOA_2		JTAG_TDO	UART1_IN	SPI_CSn	I2C_SCL	PWM[2]
GPIOA_3		JTAG_TDI	UART1_OUT	SPI_SCL	I2C_SDA	PWM[3]
GPIOA_4		JTAG_TRST	UART1_CTS	SPI_MOSI		PWM[4]
GPIOA_5			UART1_RTS	SPI_MISO		PWM[5]
GPIOA_6						PWM[6]
GPIOA_7	SPI_M_CS			SPI_CSn		
GPIOA_8	SPI_M_CLK			SPI_SCL		
GPIOA_9	SPI_M_DATA[2]		UART0_RTS	SPI_MOSI		
GPIOA_10	SPI_M_DATA[1]		UART0_CTS	SPI_MISO		
GPIOA_11	SPI_M_DATA[0]		UART0_OUT		I2C_SCL	PWM[0]
GPIOA_12	SPI_M_DATA[3]		UART0_IN		I2C_SDA	PWM[1]
GPIOA_13			UART0_IN			PWM[7]
GPIOA_14	SDIO_INT		UART0_OUT			PWM[2]
GPIOA_15	SD_D[2]		UART2_IN	SPI_CSn	I2C_SCL	PWM[3]
GPIOA_16	SD_D[3]		UART2_OUT	SPI_SCL	I2C_SDA	PWM[4]
GPIOA_17	SD_CMD					PWM[5]
GPIOA_18	SD_CLK					PWM[6]
GPIOA_19	SD_D[0]		UART2_CTS	SPI_MOSI	I2C_SCL	PWM[7]
GPIOA_20	SD_D[1]		UART2_RTS	SPI_MISO	I2C_SDA	PWM[0]
GPIOA_21			UART2_IN		I2C_SCL	PWM[1]
GPIOA_22			UART2_OUT	LED_0	I2C_SDA	PWM[2]
GPIOA_23				LED_0		PWM[7]

Table 1-1 GPIOA Pin MUX: DEV_2V0 Board

Note: This table may not be up-to-date, please check the HDK and datasheet for more details.

1.2.2 Pin-Out Reference

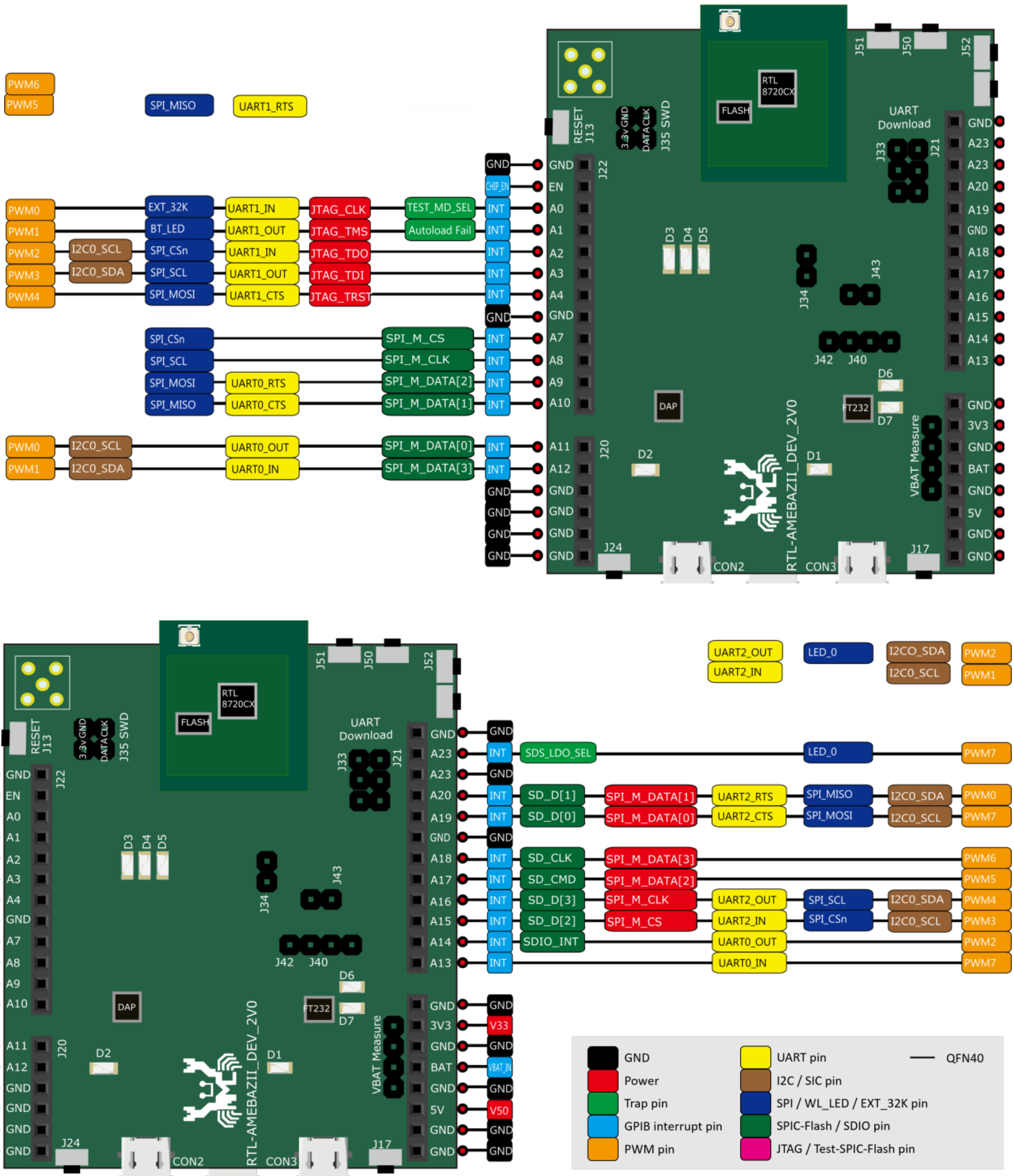
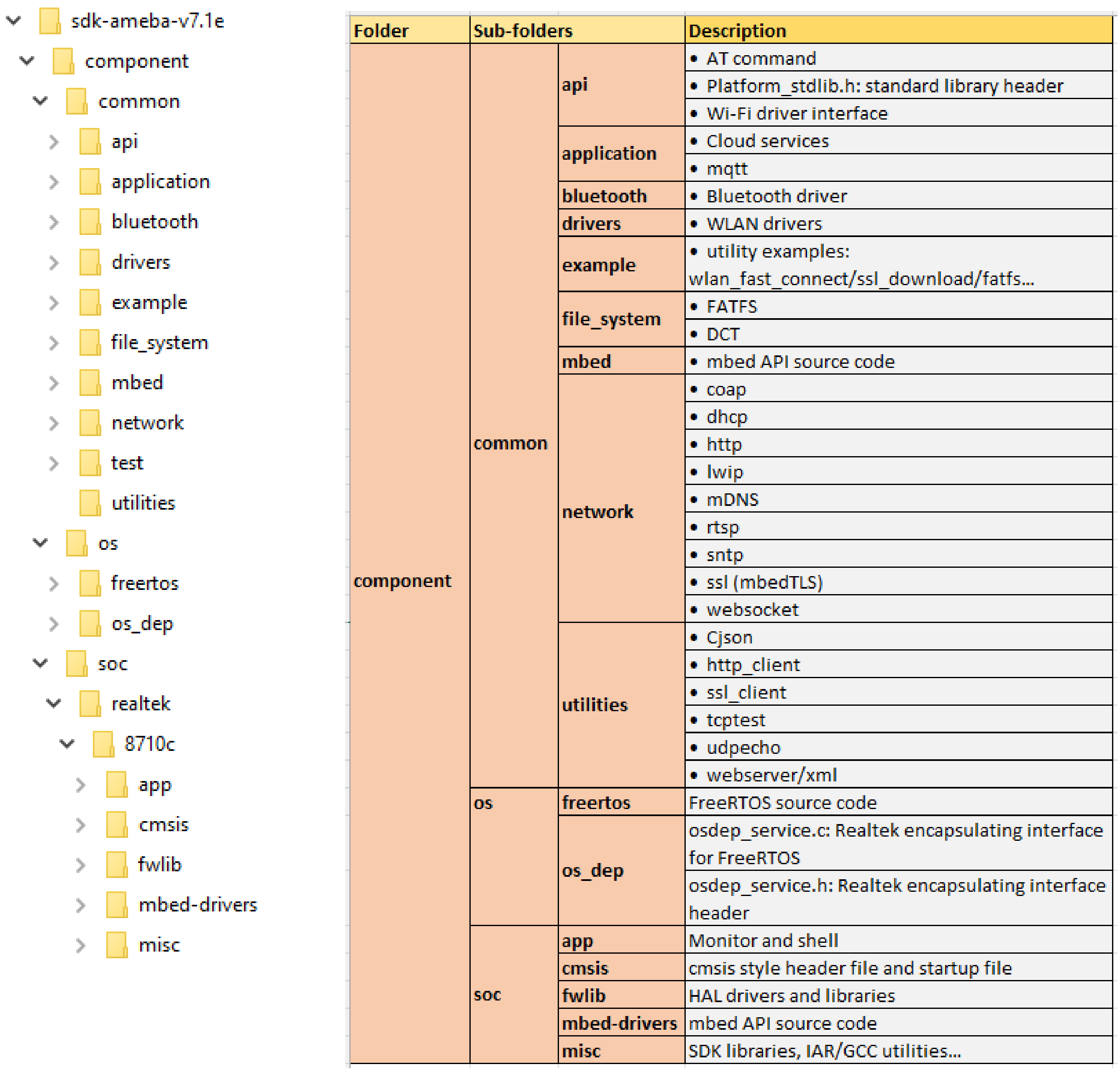


Figure 1-3 Pin Out Reference for DEV_2V0

2 SDK Architecture

In the following chapters in this document, we will take ZII project for example to explain how to use this SDK.
Ameba-ZII SDK includes four folders: ‘**component**’, ‘**doc**’, ‘**project**’ and ‘**tools**’.
The architecture of SDK and descriptions of main folders are shown below.

2.1 Component



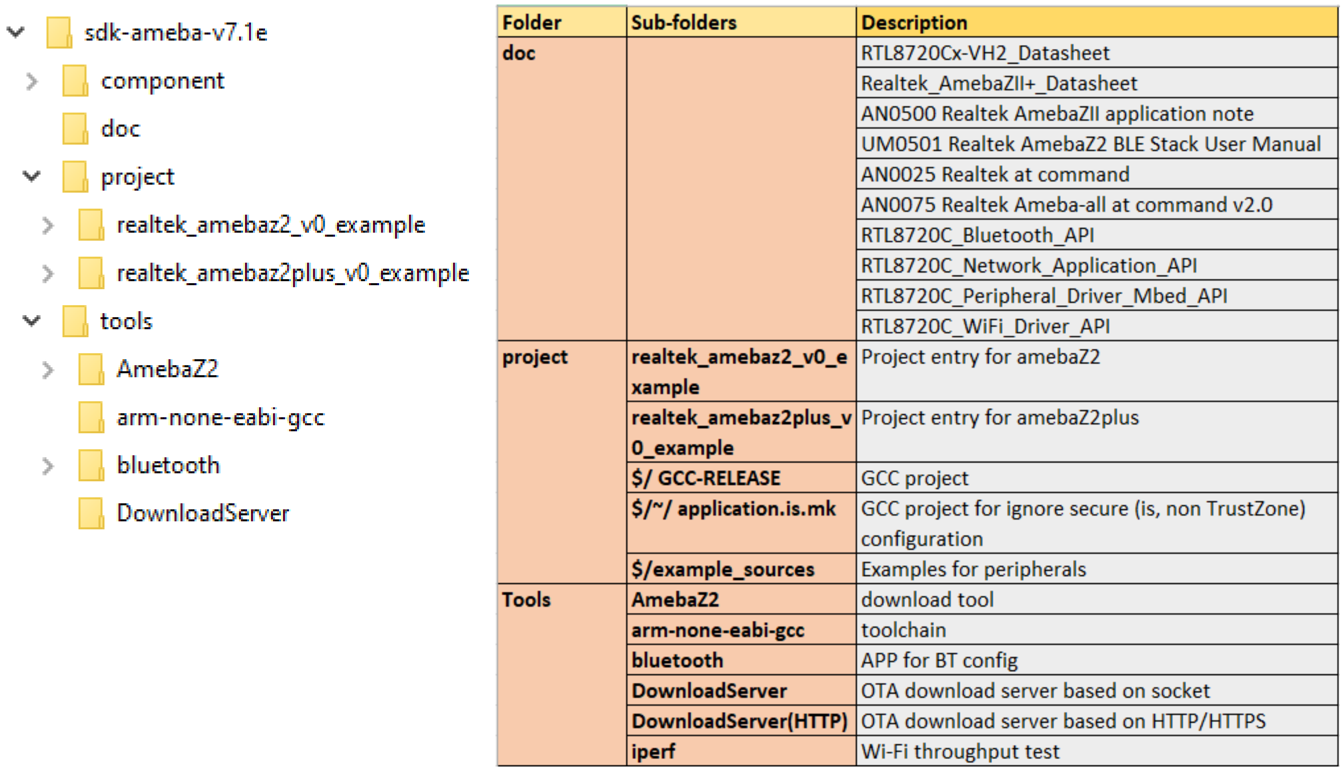
```
sdk-ameba-v7.1e
  component
    common
      api
      application
      bluetooth
      drivers
      example
      file_system
      mbed
      network
      test
      utilities
    os
      freertos
      os_dep
    soc
      realtek
        8710c
          app
          cmsis
          fwlib
          mbed-drivers
          misc
```

Folder	Sub-folders		Description
component	common	api	• AT command
			• Platform_stdlib.h: standard library header
			• Wi-Fi driver interface
		application	• Cloud services
			• mqtt
		bluetooth	• Bluetooth driver
		drivers	• WLAN drivers
		example	• utility examples: wlan_fast_connect/ssl_download/fatfs…
		file_system	• FATFS
			• DCT
		mbed	• mbed API source code
		network	• coap
			• dhcp
			• http
			• lwip
			• mDNS
			• rtsp
			• sntp
			• ssl (mbedTLS)
			• websocket
		utilities	• Cjson
			• http_client
			• ssl_client
			• tcptest
			• udpecho
			• webserver/xml
	os	freertos	FreeRTOS source code
		os_dep	osdep_service.c: Realtek encapsulating interface for FreeRTOS
			osdep_service.h: Realtek encapsulating interface header
	soc	app	Monitor and shell
		cmsis	cmsis style header file and startup file
		fwlib	HAL drivers and libraries
		mbed-drivers	mbed API source code
		misc	SDK libraries, IAR/GCC utilities…

Figure 2-1 SDK architecture and description part 1

2.2 Doc, Project, Tools

Ameba-ZII has 256KB RAM and Ameba-ZIIplus has 384KB RAM, they share the same SDK with different project entrance, please build your code in correct project, the image files are not compatible.



- sdk-ameba-v7.1e
  - component
  - doc
  - project
    - realtek_amebaz2_v0_example
    - realtek_amebaz2plus_v0_example
  - tools
    - AmebaZ2
    - arm-none-eabi-gcc
    - bluetooth
    - DownloadServer

Folder	Sub-folders	Description
doc		RTL8720Cx-VH2_Datasheet
		Realtek_AmebaZII+_Datasheet
		AN0500 Realtek AmebaZII application note
		UM0501 Realtek AmebaZ2 BLE Stack User Manual
		AN0025 Realtek at command
		AN0075 Realtek Ameba-all at command v2.0
		RTL8720C_Bluetooth_API
		RTL8720C_Network_Application_API
		RTL8720C_Peripheral_Driver_Mbed_API
		RTL8720C_WiFi_Driver_API
project	realtek_amebaz2_v0_example	Project entry for amebaZ2
	realtek_amebaz2plus_v0_example	Project entry for amebaZ2plus
	$/ GCC-RELEASE	GCC project
	$/~/ application.is.mk	GCC project for ignore secure (is, non TrustZone) configuration
	$/example_sources	Examples for peripherals
Tools	AmebaZ2	download tool
	arm-none-eabi-gcc	toolchain
	bluetooth	APP for BT config
	DownloadServer	OTA download server based on socket
	DownloadServer(HTTP)	OTA download server based on HTTP/HTTPS
	iperf	Wi-Fi throughput test

Figure 2-2 SDK architecture and description part 2

3 SDK Build Environment Setup

3.1 Introduction

In this chapter, we will illustrate how to build Realtek WiFi SDK. We will start by explaining how to setup debugger on your computer for both **Windows OS** and **Linux OS**. Ameba-ZII uses **J-Link** Debugger and **DAPLink** debugger.

Then, we will illustrate how to connect **logUART** to the debuggers.

Lastly, we will explain how to setup development environment for your computer and how to process the compilation. The **GCC** IDE will be used for both Windows OS and Linux OS.

Note: For Windows OS, we use **Windows 10 64-bit** as our platform.

3.2 Debugger Settings

To download code or debug on Ameba-ZII, user needs to make sure the debugger is setup properly.
Ameba-ZII supports **J-Link** for code download and entering debugger mode. The settings are described below.

3.2.1 J-Link Debugger

3.2.1.1 Connection

Ameba-ZII supports J-Link debugger. you need to connect the **Serial Wire Debug** (SWD) connector of Ameba-ZII to J-Link debugger as shown below and then connect J-Link to PC. You can refer to section 1.2.2 for SWD pin definitions.

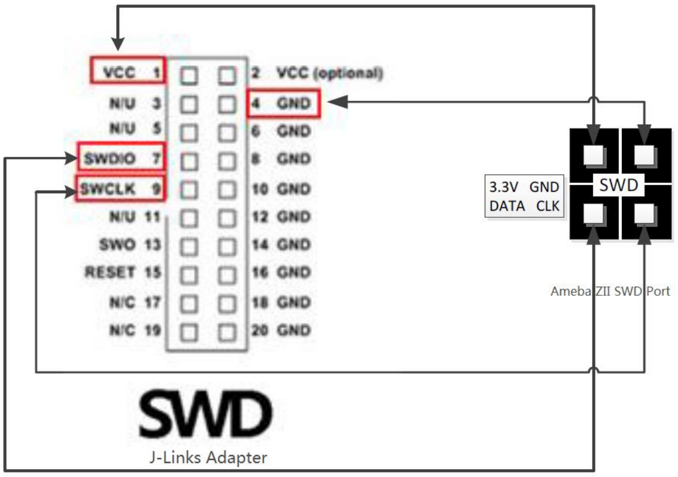


Figure 3-1 Connection between J-Link Adapter and Ameba-ZII SWD connector

Note:

1. *To be able to debugger Ameba-ZII which is powered by Cortex-M33, user needs a J-Link debugger with the latest hardware version (Check https://wiki.segger.com/Software_and_Hardware_Features_Overview for details).*

2. *If you are using **Virtual Machine** as your platform, please make sure the USB connection setting between VM host and client is correct so that the VM client can detect the device.*

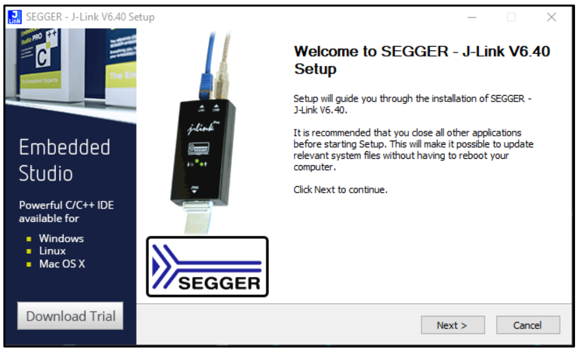
3.2.1.2 Setups on Windows OS

To be able to use J-Link debugger, you need to firstly install J-Link GDB server.
Please check http://www.segger.com and download "**J-Link Software and Documentation Pack**" (https://www.segger.com/downloads/jlink).

Note: It's better to download the **latest version** of J-Link Software. Version 6.40 is used to prepare this document.

The process is as follows:

1. Install J-Link GDB server.
Please check http://www.segger.com and download "J-Link Software and Documentation Pack" (https://www.segger.com/downloads/jlink).



SEGGER - J-Link V6.40 Setup

Welcome to SEGGER - J-Link V6.40 Setup

Setup will guide you through the installation of SEGGER - J-Link V6.40.

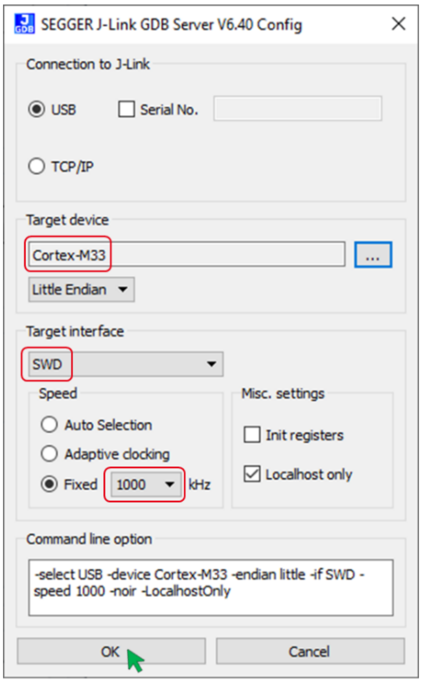
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 3-2 J-Link Setup Interface

2. Open installation location of 'JLink_V640' and run "**JLinkGDBServer.exe**" to check connection.

3. Make sure the configuration is fine and click 'OK'.



SEGGER J-Link GDB Server V6.40 Config

Connection to J-Link
● USB □ Serial No.
○ TCP/IP

Target device
Cortex-M33
Little Endian

Target interface
SWD

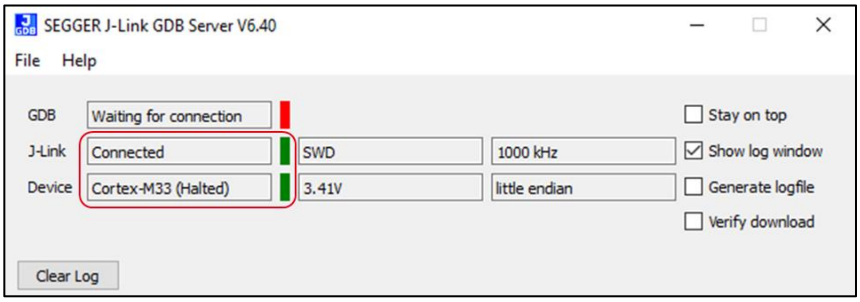
Speed
○ Auto Selection
○ Adaptive clocking
● Fixed 1000 kHz

Misc. settings
□ Init registers
☑ Localhost only

Command line option
-select USB -device Cortex-M33 -endian little -if SWD -speed 1000 -noir -LocalhostOnly

Figure 3-3 J-Link GDB server UI under Windows

4. Check if the below information is shown properly.



GDB: Waiting for connection
J-Link: Connected | SWD | 1000 kHz
Device: Cortex-M33 (Halted) | 3.41V | little endian

☐ Stay on top
☑ Show log window
☐ Generate logfile
☐ Verify download

Figure 3-4 J-Link GDB server connect under Windows

Note: If J-Link GDB Server is unable to detect the device, try re-connecting the wires and re-open 'JLinkGDBServer.exe' may solve the problem.

3.2.1.3 Setups on Linux OS

3.2.1.3.1 Install J-Link Software on Ubuntu/Linux

a. Download the latest JLink software for Ubuntu from the following link:

https://www.segger.com/downloads/jlink/#J-LinkSoftwareAndDocumentationPack

b. Install via the .DEB/ .RPM or TGZ file, the commands below are to install JLink through the .DEB file

   a. "sudo dpkg -i JLink_Linux_V646b_x86_64.deb"

   b. "sudo dpkg --install JLink_Linux_V646b_x86_64.deb"

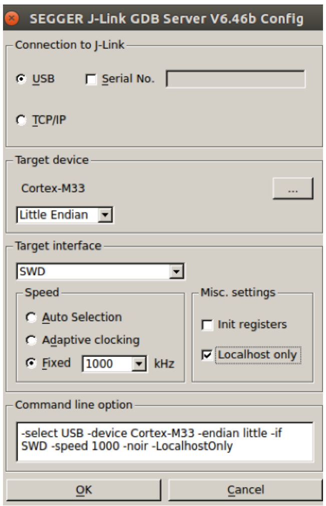
3.2.1.3.2 Steps to Initiate JLinkGDBServer

If all steps mentioned in 3.2.1.2.1 have been followed, the JLlink software as well as the JLink GDB server needs to be installed successfully. Before initiating the flash, follow the steps below to initiate the JLink GDB Server.

1. Open a new terminal along with the terminal where the SDK is open.

2. In the new terminal enter the command "sudo JLinkGDBServerExe".

3. Once this is done, the JLink GDB server config window will pop up as shown below.



Connection to J-Link
● USB ☐ Serial No.
○ TCP/IP

Target device
Cortex-M33
Little Endian

Target interface
SWD

Speed
○ Auto Selection
○ Adaptive clocking
● Fixed 1000 kHz

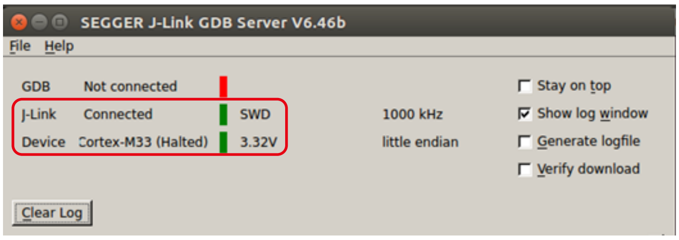
Misc. settings
☐ Init registers
☑ Localhost only

Command line option

```
-select USB -device Cortex-M33 -endian little -if
SWD -speed 1000 -noir -LocalhostOnly
```

4. The configurations to be selected are as shown above, to ensure proper connection please follow the configurations exactly as shown on the image and click "ok".

5. If the connection is successful, the connection window should be as it is shown below.



SEGGER J-Link GDB Server V6.46b

File Help

GDB	Not connected	🟥		
J-Link	Connected	🟩	SWD	1000 kHz
Device	Cortex-M33 (Halted)	🟩	3.32V	little endian

□ Stay on top
☑ Show log window
□ Generate logfile
□ Verify download

Clear Log

6. Once connection is successful, please do not close the terminal from which the GDB server was started as this will result in the termination of the JLink GDB server program and in turn the flash will be a failure.

3.3 Log UART Settings

To be able to start development with the demo board, Log UART must be connected properly.
Different versions of EVBs have different connections.

3.3.1 EVB v2.0

By default, UART2 (GPIOA_15 / GPIOA_16, c heck figure 1-3) is used as system log UART. User needs to connect jumpers to **J33** for **CON3 (FT232)** or **CON2 (DAP)**.

1) Connection to log UART via **FT232 (CON3):**

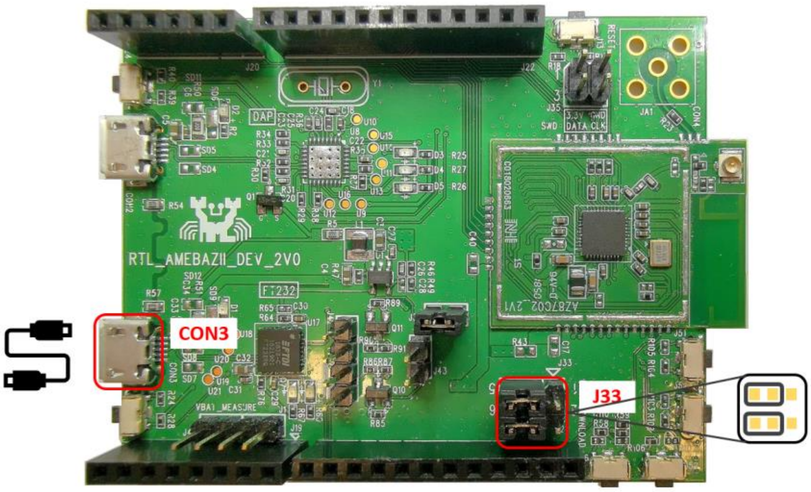


Figure 3-2 Log UART via FT232 on EVB V2.0

2) Connection to log UART via **DAP (CON2):**
Note: You need to check whether the **DAP Chip** is mounted on the board.

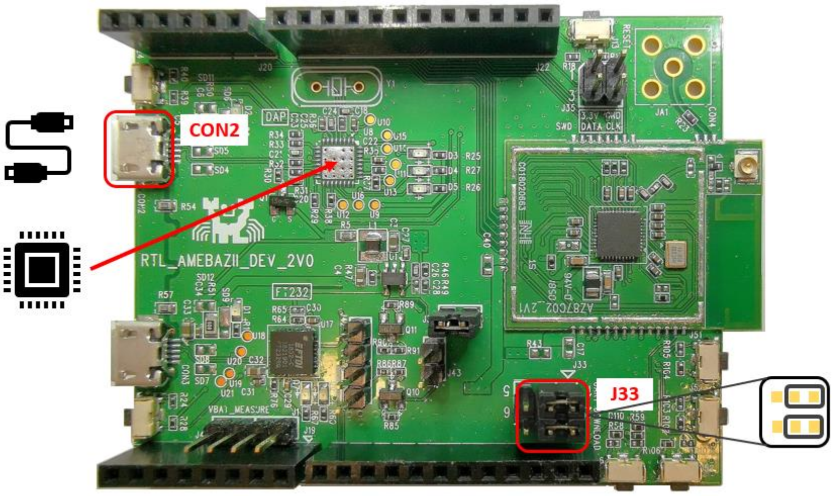


Figure 3-7 Log UART via DAP on EVB V2.0

3.4 GCC Environment on Windows (Using Cygwin)

3.4.1 Install Cygwin

Cygwin is a large collection of GNU and Open Source tools which provide similar functionality as a Linux distribution on Windows. It provides the GCC toolchain for Ameba-ZII to compile projects.
Users can visit the official website of Cygwin and install the software. Please use **Cygwin 32-bit** version.

Note:

- During the Cygwin installation, please install "**math**" "**bc: Arbitrary precision calculator language**"
- During the Cygwin installation, please install "**devel**" "**make: The GNU version of the 'make' utility**"

3.4.2 Build Non-Trust Zone Project

3.4.2.1 Compile Project on Cygwin

1) Open "**Cygwin Terminal**"
2) Direct to compile path. Enter command "**cd** /SDK/project/realtek_amebaz2_v0_example/GCC-RELEASE"
3) Clean up pervious compilation files. Enter command "**make clean**"
4) Build the application. Enter command "**make is**"
5) Make sure there are no errors after compilation.

3.4.2.2 Generate Image Binary

After compilation, the images partition.bin, bootloader.bin, firmware_is.bin and flash_is.bin can be seen in different folders of \GCC-RELEASE.

1) **partition.bin** stores partition table, recording the address of Boot image and firmware image; located at folder \GCC-RELEASE;
2) **bootloader.bin** is bootloader image; located at folder \GCC-RELEASE\bootloader\Debug\bin;
3) **firmware_is.bin** is application image; located at folder \GCC-RELEASE\application_is\Debug\bin;
4) **flash_is.bin** links partition.bin, bootloader.bin and firmware_is.bin. Located at folder \GCC-RELEASE\application_is\Debug\bin.

Note: Users need to choose '**flash_is.bin**' when downloading the image to board by **PG Tool**.

3.4.2.3 Download

After a successfully compilation and '**flash_is.bin**' is generated without error, user can either

1) Directly download the image binary on to demo board from Cygwin (as below)

   Connect **SWD** to board and open "**JLinkGDBServer.exe**". Please refer to Jlink debugger sector for SWD connection.

   Enter command "**make flash**" at Cygwin.
2) Or using the **PG tool** for Ameba-ZII (Will not be shown here, please check chapter Image Tool for details).

3.4.2.4 Debug

After successfully downloading, users can debug with JLINKGDBServer + JLINK by using the following command:
"**make debug**"

Before using this command, it is necessary to select GDB Server by running command:
"**make setup GDB_SERVER=jlink**"

3.4.3 GCC Memory Configuration

The whole memory layout of Ameba-ZII can refer to chapter Memory Layout.
There will be some extra configurations users need to do if they want to put some code to a certain memory region.

3.4.3.1 Configure Memory from makefile

In application.is.mk, users can use below method to move object to DTCM_RAM or PSRAM.

```
SRC_C += ../src/main.c

#SRAM
# --------------------------------------------------------------------
#@SRAM
SRAM_C += ../../../component/common/mbed/targets/hal/rtl8710c/flash_api.c

#PSRAM
# --------------------------------------------------------------------
#@PSRAM
ERAM_C += ../../../component/common/network/ssl/mbedtls-2.4.0/library/aesni.c
```

SRC_C: in XIP by default
SRAM_C: in DTCM_RAM
ERAM_C: in PSRAM

3.4.3.2 Configure Memory from .ld File

Users can also configure memory allocation from .ld file.
The .ld file of Ameba-ZII locates at: “SDK/project/realtek_amebaz2_v0_example/GCC-RELEASE/ **rtl8710c_ram.ld**”
Open “**rtl8710c_ram.ld**” with any text editor. There are some memory regions in it, which are:

- DTCM_RAM
- PSRAM
- XIP_FLASH_C
- XIP_FLASH_P

Note: Users can refer GCC document to understand the format of .ld file.

In .ld file, RODATA and (.text) are located in XIP section by default.
To put data or text in DTCM_RAM, add prefix of sram section information in front of the object.
To put data or text in PSRAM, add prefix of psram section information in front of the object.

3.4.3.2.1 Move text section of an object to DTCM_RAM

```
        .ram.code_text :
        {
                ……
                /* Move text section of flash_api.c to DTCM_RAM */
                *flash_api*.o(.text*)
                ……
        } > DTCM_RAM
```

3.4.3.2.2 Move text section of an object to PSRAM

```
        .psram.code_text :
        {
                ……
                /* Move text section of aesni.c to PSRAM */
                * aesni*.o(.text*)
                ……
        } > PSRAM
```

3.4.3.2.3 Move text section of a library to PSRAM

```
        .psram.code_text :
        {
                ……
                /* Move text section of lib_xxx.a to PSRAM, exclude object a/b/c */
                *lib_xxx.a: (EXCLUDE_FILE (*a.o *b.o * c.o) .text*)
                ……
        } > PSRAM
```

3.4.3.3 Configure Memory from C File

There are some section MACROs already pre-defined in component/soc/Realtek/8710c/cmsis/rtl8710c/include/section_config.h. User can use these MACROs to locate a specific function or variable to a specific memory location.

```
PSRAM_RODATA_SECTION
const char user_pattern[12] = {
   'H', 'e', 'l', 'l', 'o', ' ','W', 'o', 'r', 'l', 'd', '!'
};

PSRAM_TEXT_SECTION
void user_func (void)
{
        ……
}
```

3.4.4 GCC Memory Overflow

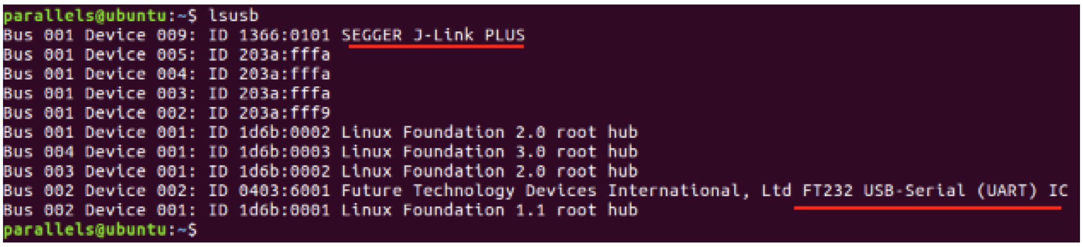
By default, Ameba-ZII places read-only (TEXT and RODATA) section in the XIP area. If XIP does not have enough space, there will be memory overflow error. The **solution** is to either minimize the code or re-allocate the code to other memory region. Same rule applies to SRAM (DTCM_RAM) and PSRAM (if it's available).

3.5 GCC Environment on Ubuntu/Linux

3.5.1 Verify Device Connections

Once the JLink software is installed, the connections to the ubuntu machine of the device need to be verified.

1. Ensure that the JLink debugger is connected to the target board and the USB device is connected to the Ubuntu/Linux machine.
2. Ensure that the micro-usb is connected to **CON3** (FT232) and plugged into the Ubuntu/Linux machine via USB to receive serial logs.
3. To verify if both devices i.e. the JLink device and the device serial port have been detected properly we can use the "**lsusb**" command to see list of devices as shown below:

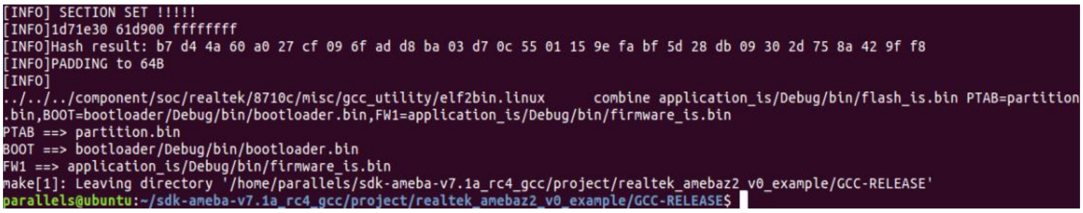


```
parallels@ubuntu:~$ lsusb
Bus 001 Device 009: ID 1366:0101 SEGGER J-Link PLUS
Bus 001 Device 005: ID 203a:fffa
Bus 001 Device 004: ID 203a:fffa
Bus 001 Device 003: ID 203a:fffa
Bus 001 Device 002: ID 203a:fff9
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 0403:6001 Future Technology Devices International, Ltd FT232 USB-Serial (UART) IC
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
parallels@ubuntu:~$
```

4. As you can see above the **SEGGER J-Link** and the **FT232 USB UART** device have been successfully detected.

3.5.2 Compile and Generate Binaries

1. Open the Ubuntu/Linux **terminal**.
2. Direct to compile path. Enter command "**cd** /SDK /project/realtek_amebaz2_v0_example/GCC-RELEASE"
3. Clean up pervious compilation files. Enter command "**make clean**"
4. Build the application. Enter command "**make is**"
5. Once the compilation is successful, you should be able to see the success logs as shown below.



```
[INFO] SECTION SET !!!!!
[INFO]1d71e30 61d900 ffffffff
[INFO]Hash result: b7 d4 4a 60 a0 27 cf 09 6f ad d8 ba 03 d7 0c 55 01 15 9e fa bf 5d 28 db 09 30 2d 75 8a 42 9f f8
[INFO]PADDING to 64B
[INFO]
../../../component/soc/realtek/8710c/misc/gcc_utility/elf2bin.linux      combine application_is/Debug/bin/flash_is.bin PTAB=partition
.bin,BOOT=bootloader/Debug/bin/bootloader.bin,FW1=application_is/Debug/bin/firmware_is.bin
PTAB ==> partition.bin
BOOT ==> bootloader/Debug/bin/bootloader.bin
FW1 ==> application_is/Debug/bin/firmware_is.bin
make[1]: Leaving directory '/home/parallels/sdk-ameba-v7.1a_rc4_gcc/project/realtek_amebaz2_v0_example/GCC-RELEASE'
parallels@ubuntu:~/sdk-ameba-v7.1a_rc4_gcc/project/realtek_amebaz2_v0_example/GCC-RELEASE$
```

3.5.3 Download and Flash Binaries

There are in-built scripts in the makefile that initiate download and flashing of the software via JLink. To flash successfully, the JLinkGDBServer needs to be initiated manually by the user and successful connection needs to be ensured. The JLink GDB server must be active and connected to the target before any type of flash action is taken. To start the JLink GDB server, follow the '**Steps to Initiate JLinkGDBServer'** section.

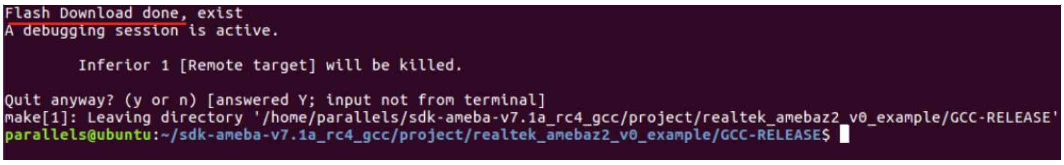
3.5.3.1 Initiate Flash Download

Once the JLink GDB server is set up as per the instructions given before, perform the following steps to initiate the flash download.

1. Proceed back to the previous terminal where the SDK was made, without closing the terminal from which GDB server is running
2. Run the command "**make setup GDB_SERVER=jlink**" to select GDB Server.

3. Run the command “**sudo make flash**”

4. If the flash download is successful, the following log will be printed



```
Flash Download done, exist
A debugging session is active.

        Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) [answered Y; input not from terminal]
make[1]: Leaving directory '/home/parallels/sdk-ameba-v7.1a_rc4_gcc/project/realtek_amebaz2_v0_example/GCC-RELEASE'
parallels@ubuntu:~/sdk-ameba-v7.1a_rc4_gcc/project/realtek_amebaz2_v0_example/GCC-RELEASE$
```

3.5.3.2 Debug

After successfully downloading, users can debug with JLINKGDBServer + JLINK by using the following command:
“**make debug**”

Before using this command, it is necessary to select GDB Server by running command:
“**make setup GDB_SERVER=jlink**”

4 Image Tool

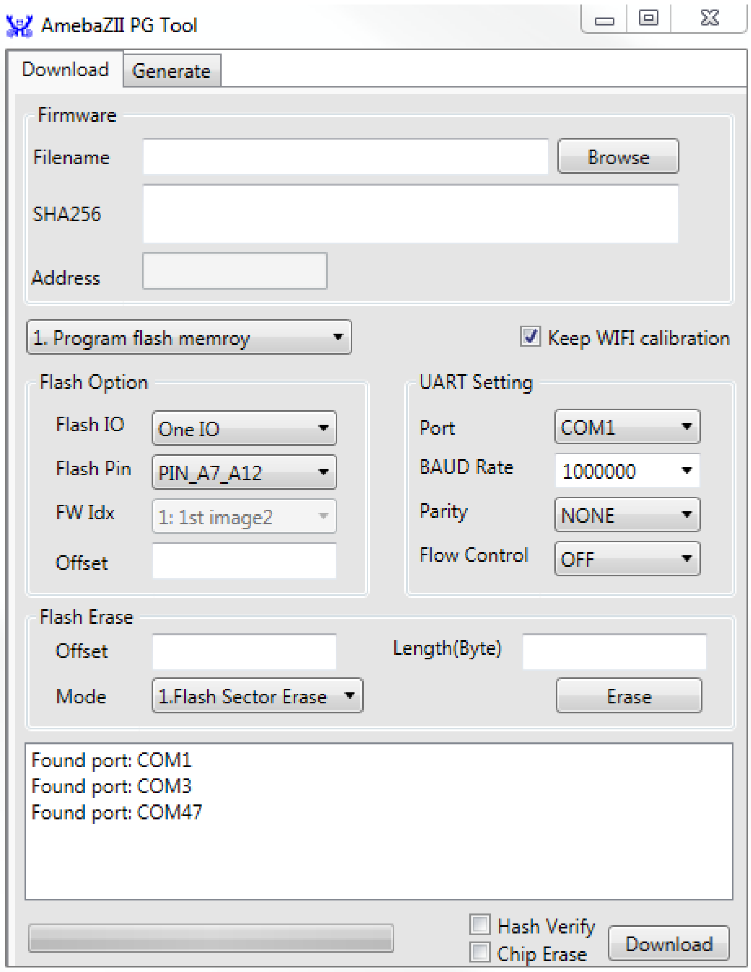
4.1 Introduction

This chapter introduces how to use Image Tool to generate and download images. As show in picture below, Image Tool has two menu pages:

- Download: used as image download server to transmit images to Ameba through UART.
- Generate: contact individual images and generate a composite image.

Please download the ‘**PG Tool Release Package**’ and browse the image tool document ‘**UM0503**’.

Note: If you need to download code via external UART, must use FT232 USB to connect UART dongle.



AmebaZII PG Tool

Download | Generate

Firmware

Filename

SHA256

Address

1. Program flash memroy

☑ Keep WIFI calibration

Flash Option

Flash IO: One IO

Flash Pin: PIN_A7_A12

FW Idx: 1: 1st image2

Offset

UART Setting

Port: COM1

BAUD Rate: 1000000

Parity: NONE

Flow Control: OFF

Flash Erase

Offset

Length(Byte)

Mode: 1.Flash Sector Erase

```
Found port: COM1
Found port: COM3
Found port: COM47
```

☐ Hash Verify

☐ Chip Erase

Figure 4-1 AmebaZII Image Tool UI

4.2 Environment Setup

4.2.1 Hardware Setup

4.2.1.1 EVB V2.0

User needs to connect CON3 to user's PC via a Micro USB cable. Add jumpers for J34 and J33 (J33 is for log UART which has two jumpers) if there is no connection.

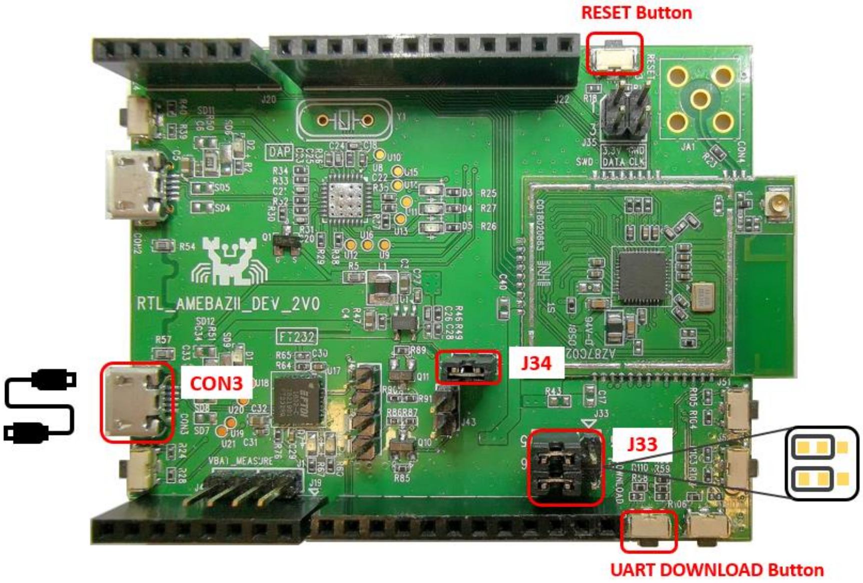


Figure 4-2 Ameba-ZII EVB V2.0 Hardware Setup

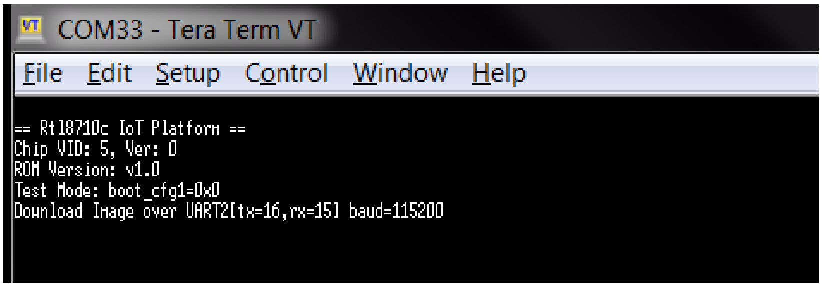
4.2.2 Software Setup

- Environment Requirements: EX. WinXP, Win 7 Above, Microsoft .NET Framework 3.5
- AmebaZII_PGTool.exe

4.3 Image Download

User can download the image to demo board by following steps:

1) Trigger Ameba-ZII chip enter **UART download mode** by:
   a. Press and hold the **UART DOWNLOAD** button then press the **RESET** button and release both buttons. And make sure the log UART is connected properly.
   b. If the chip enters **download mode**, the below log should be shown on log UART console.

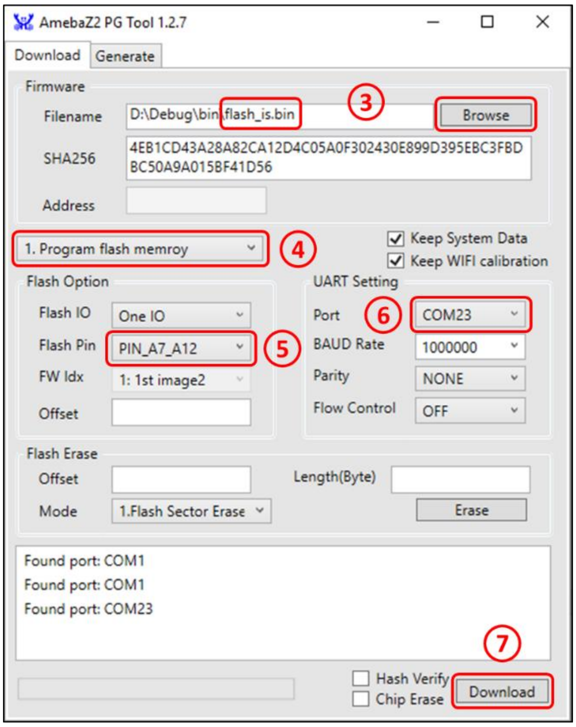


```
== Rtl8710c IoT Platform ==
Chip VID: 5, Ver: 0
ROM Version: v1.0
Test Mode: boot_cfg1=0x0
Download Image over UART2[tx=16,rx=15] baud=115200
```

Figure 4-4 Ameba-ZII UART download mode

   c. After confirming it is in download mode, **remember to disconnect the log UART console before using Image Tool to download**, because the tool will also need to connect to this log UART port.

2) Open **AmebaZ2 PG Tool**



Firmware
Filename: D:\Debug\bin\flash_is.bin
SHA256: 4EB1CD43A28A82CA12D4C05A0F302430E899D395EBC3FBD BC50A9A015BF41D56
Address:

1. Program flash memroy
☑ Keep System Data
☑ Keep WIFI calibration

Flash Option
Flash IO: One IO
Flash Pin: PIN_A7_A12
FW Idx: 1: 1st image2
Offset:

UART Setting
Port: COM23
BAUD Rate: 1000000
Parity: NONE
Flow Control: OFF

Flash Erase
Offset:
Length(Byte):
Mode: 1.Flash Sector Erase

```
Found port: COM1
Found port: COM1
Found port: COM23
```

□ Hash Verify
□ Chip Erase

3) "**Browse**" to choose the image to be downloaded (flash_xx.bin)
4) Choose "**1. Program flash memory**"
5) Choose correct "**Flash Pin**" according to the IC part number

Flash Pin	IC part number
PIN_A7_A12	RTL8710CX/RTL8720CM
PIN_B6_B12	RTL8720CF

6) Choose the correct **UART port** (use **rescan** to update the port list)
7) Click "**Download**" to start downloading image. While downloading, the status will be shown on the left bar.

Note: It's recommended to use the default settings unless user is familiar with them.

5 Memory Layout

This chapter introduces the memory components in Ameba-ZII, including ROM, RAM, SRAM, PSRAM and Flash. Also, this chapter provides a guide for users to place their program to specific memory to fit user's requirement. However, some programs are fixed in specific memory and cannot be moved.

5.1 Memory Type

The size and configuration are as shown below

	Size(bytes)	Description
ROM	384K	Reserved
RAM	256K/ 384K	Internal DTCM
PSRAM	4M	MCM PSRAM, only available on RTL8720CM
XIP	16M	Execute in Place, section TEXT and RODATA, virtual address remapped by SCE (Secure Code Engine). Physical address of Flash starts from 0x98000000 which can refer to the datasheet for more details.

Table 5-1 Size of Different Memories on Ameba-ZII

The graph of configuration on Ameba-ZII is as shown below:

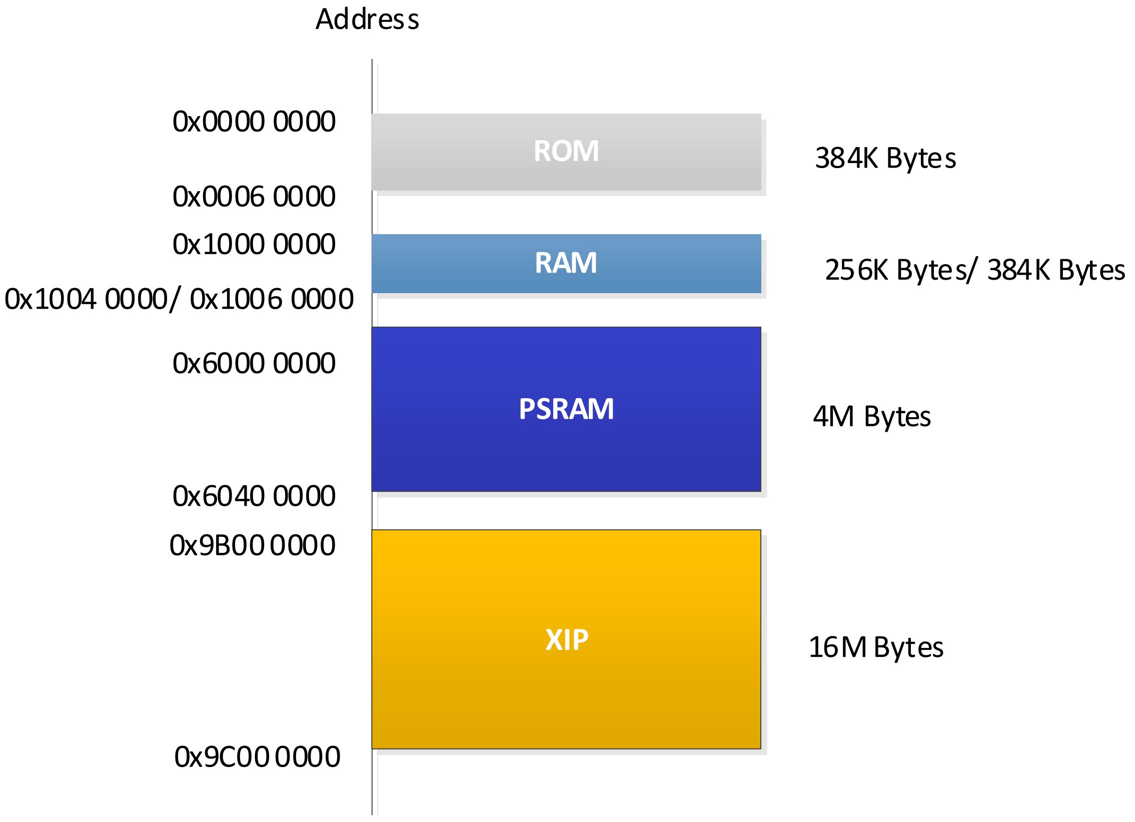


Figure 5-1 Address Allocation of Different Memories on Ameba-ZII

5.2 Flash Memory Layout

The default flash layout used in SDK is shown in the figure below.

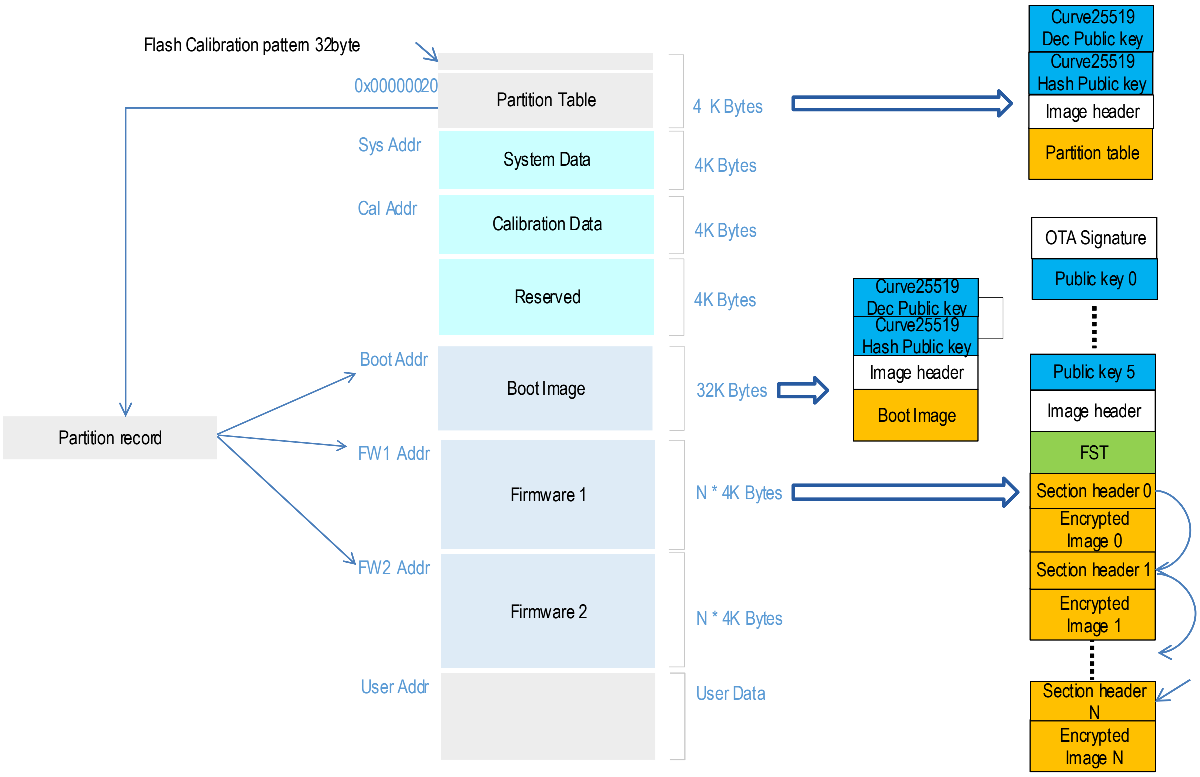


Figure 5-2 Flash memory layout

And the description of each block is listed in the table below.

Items	Start Offset Address	Limit Offset Address	Address adjustable	Size	Description	Mandatory
Partition table	0x00000000	0x00001000-1	N	4KB	The first 32 bytes is flash calibration pattern. The actual partition table starts from 0x20	Y
System data	0x00001000	0x00002000-1	N	4KB	To store some system settings	Y
Calibration data	0x00002000	0x00003000-1	N	4KB	RESERVED, user don't need to configure this portion	Y
Reserved	0x00003000	0x00004000-1	N	4KB	RESERVED, user don't need to configure this portion	Y
Boot image	0x00004000	0x0000C000-1	Y	32KB	Bootloader code/data	Y
Firmware 1	0x0000C000	0x0000C000 + N*4KB-1	Y	N*4KB	Application image; the address can be adjusted in partition.json	Y
Firmware 2	0x0000C000 + N*4KB	0x0000C000 + 2*N*4KB-1	Y	N*4KB	Application image; the address can be adjusted in partition.json	N
User Data	0x0000C000 + 2*N*4KB	0x00200000	Y	unfixed	User Data; the address can be defined in platform_opts.h	N

Table 5-2 Description of flash layout

5.2.1 Partition Table

5.2.1.1 The Layout of Partition Table

The partition table stores following information:

Curve25519 Dec public key (32byte)															
Curve25519 Hash public key (32byte)															
Header (96byte)															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
RESV				NUM	FW1 IDX	FW2 IDX	RESV								Key_Exp_Op
User Data Len		RESV													
Boot Image Partition record (64byte)															
Partition record 0 (64byte)															
...															
Partition record N (64byte)															
User Data															
Hash (32byte)															

Table 5-3 The layout of Partition table

- **Curve25519 Dec public key**: the public key used to generate AES key to decrypt image
- **Curve25519 Hash public key**: the public key used to generate Hash key to validate the hash value
- **Header**: partition table image header
- Partition table image info
    - **NUM**: The record number in the partition table, not including "Boot Image Partition record"
    - **FW1/FW2 IDX**: FW1/FW2 Partition record index
    - **Key_Exp_Op [2:0]**
        - 1: Export AES keys of the latest FW
        - 2: Export AES keys for both FW1 & FW2
        - Other: Don't export any AES to RAM code
- **User Data Len**: the length (in bytes) of user data
- Partition record x (includes boot image partition record)

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Start address				Length				TYPE	DBG_SKIP	RESV					
HKeyValid	RESV [15]														
Hash key															
Hash Key															

    - **Start address**: the offset address on Flash for the image
    - **Length**: the length of the image, align to 4K
    - **TYPE**: type of the image (Pt=0/boot/sys/cal/user/fw1/fw2/resv)
    - **DBG_SKIP**: skip download to ram from flash when debug mode is enabled
    - **HKeyValid**: indicates the Hash Key is valid (bit [0]! = 0) or not(bit [0] = 0)
    - **Hash key**: to do all firmware validation (from first byte to end)
- **User Data**: user secret data
- **Hash**: from the first byte of partition table to the end of user data (two public keys + Header + partition info + partition records + user data), calculated before encryption if the encryption is on

5.2.2 System Data

System data section is the one which stores some system settings, including OTA section, Flash section, Log UART section etc… The size of system data section is 4KB.

Offset	0x00	0x04	0x08	0x0C
0x00	RSVD	RSVD	Force old OTA	RSVD
0x10	RSVD	RSVD	RSVD	RSVD
0x20	WORD1: RSVD WORD0: SPI Mode	WORD1: Flash Size WORD0: Flash ID	RSVD	RSVD
0x30	ULOG Baudrate	RSVD	RSVD	RSVD
0x40 ~ 0x70	RSVD (SPIC calibration setting)			
…	RSVD			
0xFF0	BT Parameter Data			

Table 5-4 Layout of system data

5.2.2.1 OTA Section

Offset	Bit	Function	Description
0x00	[31:0]	RSVD	RSVD
0x04	[31:0]	RSVD	RSVD
0x08	[31:8]	RSVD	RSVD
	[7:0]	Force old OTA	Select GPIO to force booting from old OTA image. Available GPIO pins may vary from different Chip part number. (GPIOA2~6, GPIOA13) BIT[7]: active_state, 0 or 1 BIT[6]: RSVD BIT[5]: port BIT[4:0]: pin number
0x0C	[31:0]	RSVD	RSVD

Table 5-5 Definition for OTA section in system data

5.2.2.1.1 Force Old OTA

The platform provides a "Force old OTA" option to let user roll back to the previous OTA image by using a GPIO pin as trigger. Please note that if secure boot enabled, "key_exp_op" from *partition.json* need set as 2 to export both FW1 & FW2 AES key, to prevent boot fail when user try to roll back to previous image.

5.2.2.2 Flash Section

Offset	Bit	Function	Description
0x20	[31:16]	RSVD	RSVD
	[15:0]	SPI IO Mode	0xFFFF: Quad IO mode 0x7FFF: Quad Output mode 0x3FFF: Dual IO mode 0x1FFF: Dual Output mode 0x0FFF: One IO mode
0x24	[31:16]	Flash Size	0xFFFF: 2MB 0x7FFF: 32MB 0x3FFF: 16MB 0x1FFF: 8MB 0x0FFF: 4MB 0x07FF: 2MB 0x03FF: 1MB
	[15:0]	Flash ID	Use it only if the flash ID cannot get by flash ID cmd
0x28	[31:0]	RSVD	RSVD
0x2C	[31:0]	RSVD	RSVD

Table 5-6 Definition for Flash section in system data

5.2.2.3 Log UART Section

Offset	Bit	Function	Description
0x30	[31:0]	Baudrate	0xFFFFFFFF: 115200 110~40000000
0x34	[31:0]	RSVD	RSVD
0x38	[31:0]	RSVD	RSVD
0x3C	[31:0]	RSVD	RSVD

Table 5-7 Definition for Log UART section in system data

5.2.3 Boot Image

5.2.3.1 The Layout of Boot Image

The format of the boot image is as below:

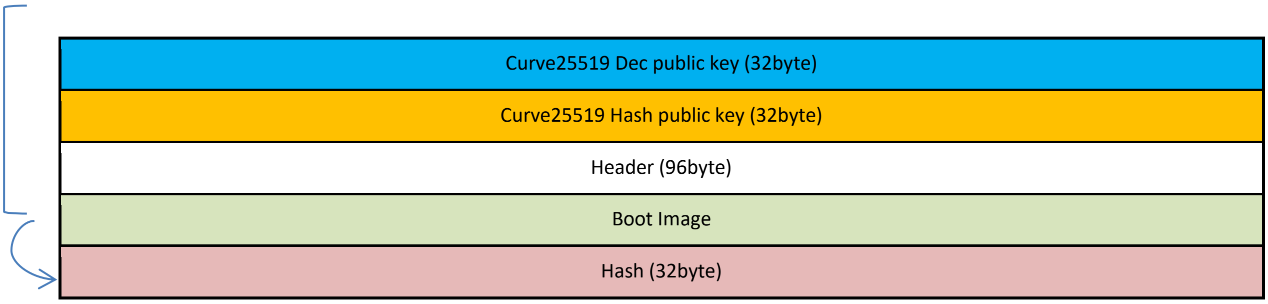

Curve25519 Dec public key (32byte)
Curve25519 Hash public key (32byte)
Header (96byte)
Boot Image
Hash (32byte)

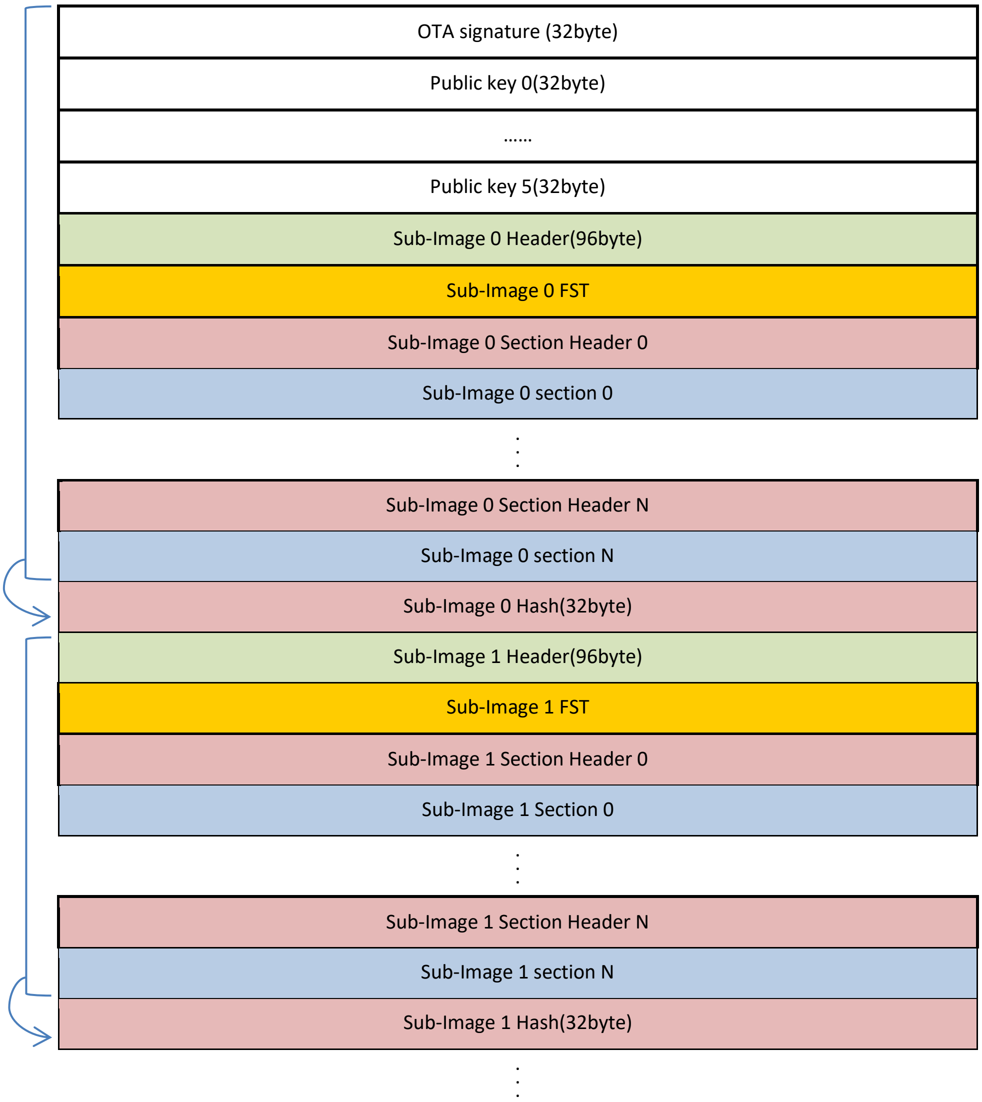
Table 5-8 The layout of boot image

- Curve25519 Dec public key: the public key used to generate AES key to decrypt image
- Curve25519 Hash public key: the public key used to generate Hash key to validate the hash value
- Header: boot image header
- Boot Image: boot image body (TEXT+DATA), will be padded with 0 to make its size is multiple of 32 bytes.
- Hash: two public keys + Header + Boot Image, calculated before encryption if image encryption is on

5.2.4 Firmware 1/Firmware 2

5.2.4.1 The Layout of Firmware Image

The format of the Firmware image is as below:



OTA signature (32byte)

Public key 0(32byte)

……

Public key 5(32byte)

Sub-Image 0 Header(96byte)

Sub-Image 0 FST

Sub-Image 0 Section Header 0

Sub-Image 0 section 0

Sub-Image 0 Section Header N

Sub-Image 0 section N

Sub-Image 0 Hash(32byte)

Sub-Image 1 Header(96byte)

Sub-Image 1 FST

Sub-Image 1 Section Header 0

Sub-Image 1 Section 0

Sub-Image 1 Section Header N

Sub-Image 1 section N

Sub-Image 1 Hash(32byte)

Table 5-9 The layout of firmware image

- **OTA signature:** The hash result of the 1st Image header "Sub FW Image 0 Header"
- **Public key 0 ~ 5:** Encryption key
  - key 0 is dedicated to enc/dec all "OTA signature/Header/FST"
  - key 1~5 are reserved
- **Sub-image x Header:** image header of FW sub-image x
- **Sub-image x FST:** Firmware Security Table of FW sub-image x
  - Each sub-image has image sections which have a section header and a section image body
- **Hash:** calculated with Encrypted FW image if image encryption is on
  - The 1st sub-image
    - From OTA Signature to the last image section, including all padding bytes
  - Other sub-image
    - From the sub image header to the last image section, including all padding bytes

5.3 SRAM Layout

The range of DTCM is from 0x10000000 to 0x10040000(256KB) or 0x10000000 to 0x10060000(384KB). The layout of this memory region is illustrated below.

Note: the layout may be changed according to the actual application, please refer to the linker file for exact layout details.

0x10000000	Vector Table
0x100000A0	Reserved for ROM
0x10000480	RAM Entry Table
0x100004F0	RAM Image Signature
0x10000500	Image2 RAM
0x10030000	RAM Bootloader
0x1003EA00	MSP
0x1003FA00	Reserved for ROM
0x1003FFFF	Configured as Heap (only for 384KB package)
0x1005FFFF	

Table 5-10 AmebaZII DTCM memory layout

Items	Start Offset Address	Limit Offset Address	Address adjustable	Size	Description	Mandatory
Vector Table	0x10000000	0x100000A0 -1	N	160B	The vector table	Y
Reserved for ROM	0x100000A0	0x10000480-1	N	992B	Reserved for ROM code	Y
RAM Entry Table	0x10000480	0x100004F0-1	N	112B	Entry function table of Image 2	Y
RAM Image Signature	0x100004F0	0x10000500-1	N	16B	RTK pattern for RAM Image	Y
Image2 RAM	0x10000500	0x10030000-1	Y	~190KB	User application image (TEXT+DATA+BSS+HEAP)	Y
RAM Bootloader	0x10030000	0x1003EA00-1	N	~58KB	RAM boot image, will be recycled by Image2 for BSS and HEAP	Y
MSP	0x1003EA00	0x1003FA00-1	Y	4KB	CPU main stack	Y
Reserved for ROM	0x1003FA00	0x1003FFFF	N	1535B	Reserved for ROM(NS) code	Y
Configured as Heap	0x10040000	0x1005FFFF	Y	128KB	Configured as Heap in SDK (only for 384KB package)	N

Table 5-11 Description of RAM layout

6 Boot Process

This chapter describes the boot process of AmebaZII platform.

6.1 Boot Flow

While booting, the system will firstly load the **partition table** which has all image information, such as the image address, keys, user data etc… Then from the partition table, **boot image** will be loaded, and **firmware image** will be loaded at the end of the boot process.

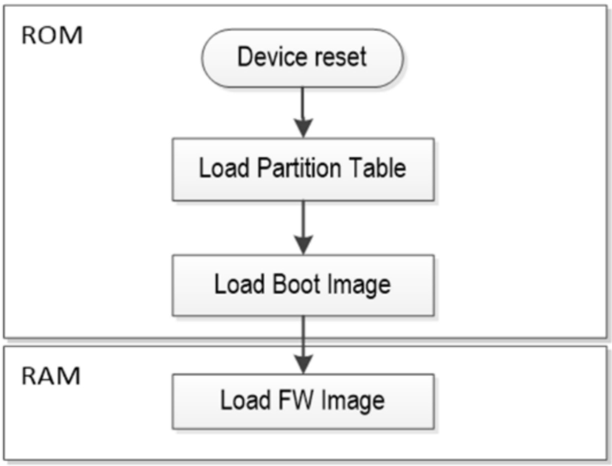


Figure 6-1 Overview of boot flow

7 Over-The-Air (OTA) Firmware Update

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

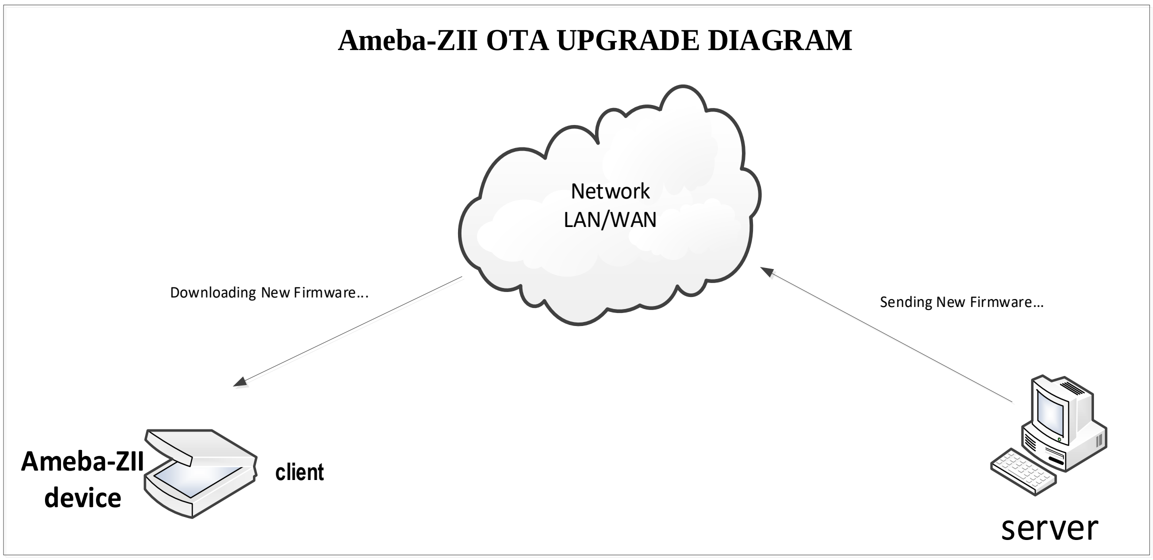


Figure 7-1 Methodology to Update Firmware via OTA

7.1 OTA Operation Flow

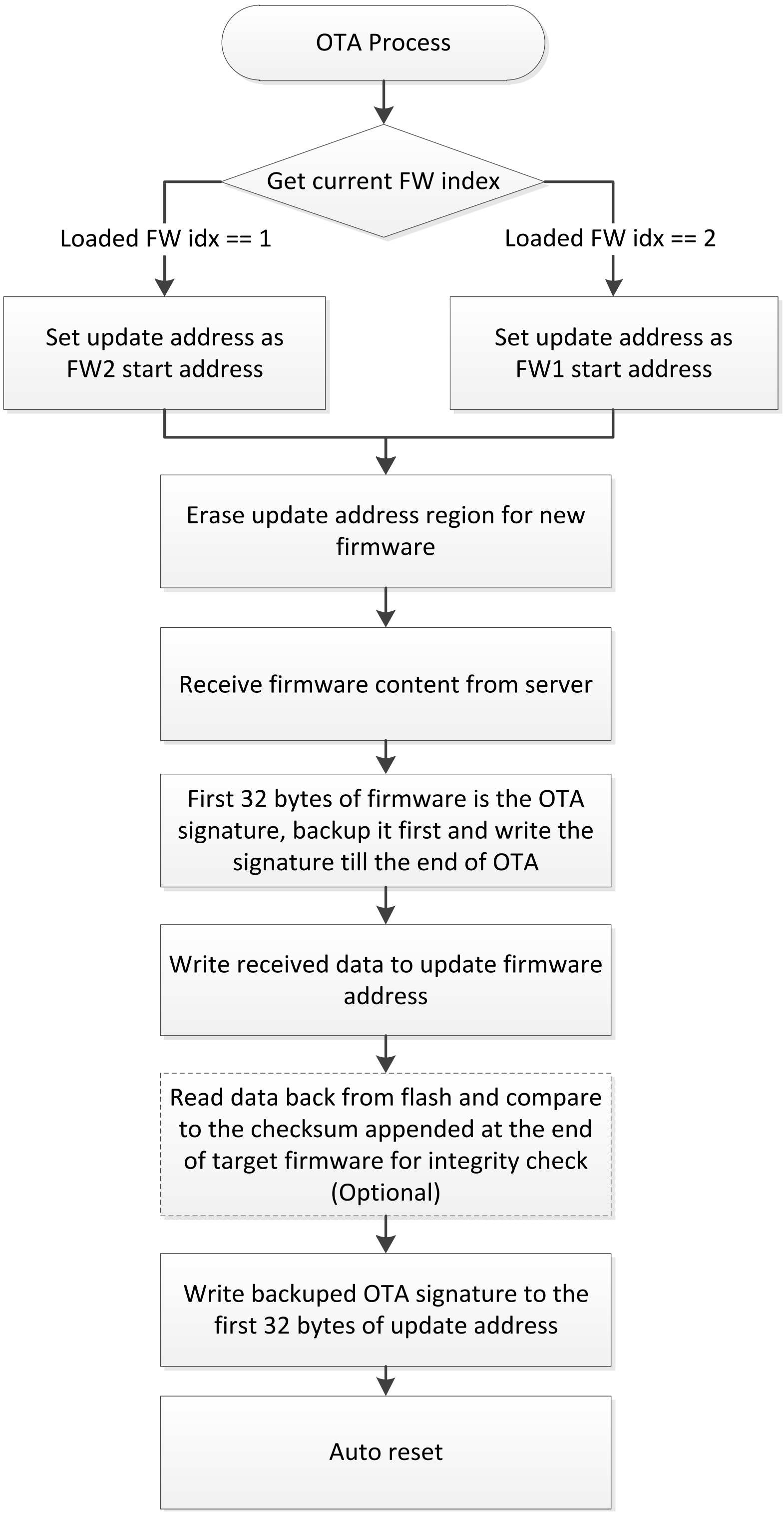

OTA Process
Get current FW index
Loaded FW idx == 1
Loaded FW idx == 2
Set update address as FW2 start address
Set update address as FW1 start address
Erase update address region for new firmware
Receive firmware content from server
First 32 bytes of firmware is the OTA signature, backup it first and write the signature till the end of OTA
Write received data to update firmware address
Read data back from flash and compare to the checksum appended at the end of target firmware for integrity check (Optional)
Write backuped OTA signature to the first 32 bytes of update address
Auto reset

Figure 7-2 OTA Process Flow

As Table 5-9 described, the first 32 bytes of firmware image would be OTA signature, which is the hash result of the image header. During the step of "Write received data to update firmware address", the 32 bytes OTA signature need set to 0xff, which is invalid signature. The correct OTA signature needs to be appended at the end of OTA process to prevent device booting from incomplete firmware.

7.2 OTA Checksum Mechanism

The 32-bytes OTA signature is used to notify the bootloader that the overall OTA process is done without any network disconnection or re-boot during the OTA process. However, this 32-bytes OTA signature cannot guarantee the correctness of firmware image content.

Users can design a mechanism to calculate the hash of target OTA firmware for integrity check during the OTA update process. For the default OTA example in SDK, there is USE_CHECKSUM option for this integrity check purpose. During image postbuild, SDK would append 4 bytes checksum at the end of firmware image. And when performing OTA routine, right after the firmware is downloaded and programmed into flash, it would read back all the programed data from flash and compare with the checksum value from target firmware if USE_CHECKSUM is enabled. In such way, it can ensure the downloaded firmware is transferred completely and correctly. For the detailed implementation, please refer to OTA example *ota_8710c.c* in SDK:

```
#define USE_CHECKSUM 1
```

Please note that this checksum mechanism in OTA example is added afterward. The original SDK might not have the logic for appending 4 bytes checksum value during postbuild process. Before user enable USE_CHECKSUM in *ota_8710c.c*, please make sure the target OTA firmware did have this 4-bytes checksum appended. Or the OTA procedure would always end in failure due to unmatched checksum value.

To check whether the postbuild generates CHECKSUM, please refer to below files.
GCC: (project\realtek_amebaz2_v0_example\GCC-RELEASE\application.is.mk)

```
.PHONY: manipulate_images
manipulate_images: | application_is
	@echo ===========================================================
	@echo Image manipulating
	@echo ===========================================================
	cp $(AMEBAZ2_BOOTLOADERDIR)/bootloader.axf $(BOOT_BIN_DIR)/bootloader.axf
ifeq ($(findstring Linux, $(OS)), Linux)
	chmod 0774 $(ELF2BIN) $(CHKSUM)
endif
	$(ELF2BIN) keygen keycfg.json
	$(ELF2BIN) convert amebaz2_bootloader.json BOOTLOADER secure_bit=0
	$(ELF2BIN) convert amebaz2_bootloader.json PARTITIONTABLE secure_bit=0
	$(ELF2BIN) convert amebaz2_firmware_is.json FIRMWARE secure_bit=0
	$(CHKSUM) $(BIN_DIR)/firmware_is.bin
	$(ELF2BIN) combine $(BIN_DIR)/flash_is.bin
PTAB=partition.bin,BOOT=$(BOOT_BIN_DIR)/bootloader.bin,FW1=$(BIN_DIR)/firmware_is.bin
```

If postbuild does not generate CHECKSUM and ota_8710.c does not check CHESKSUM: No influence
If postbuild does not generate CHECKSUM and ota_8710.c check CHECKSUM: OTA fail
If postbuild generate CECHKSUM, and ota_8710.c does not check CHECKSUM: No influence
If postbuild generate CHECKSUM, and ota_8710.c check CHECSUM: OTA with integrity check

7.3 Boot Process Flow

Boot loader will select latest (based on serial number) updated firmware and load it.

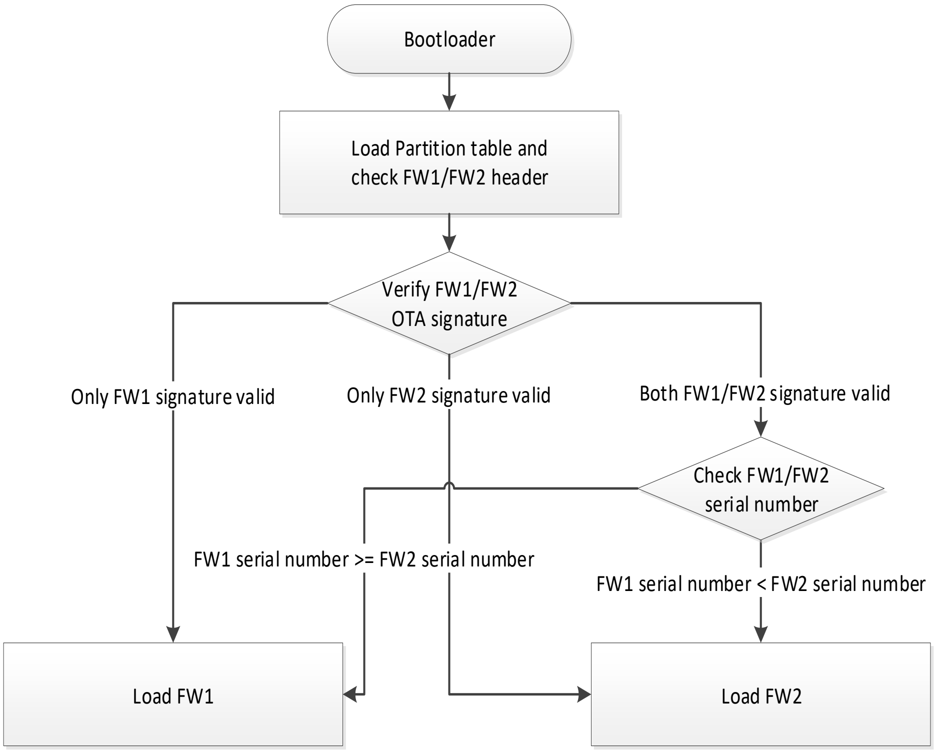

Bootloader
Load Partition table and check FW1/FW2 header
Verify FW1/FW2 OTA signature
Only FW1 signature valid
Only FW2 signature valid
Both FW1/FW2 signature valid
Check FW1/FW2 serial number
FW1 serial number >= FW2 serial number
FW1 serial number < FW2 serial number
Load FW1
Load FW2

Figure 7-3 Boot Process Flow

7.4 Upgraded Partition

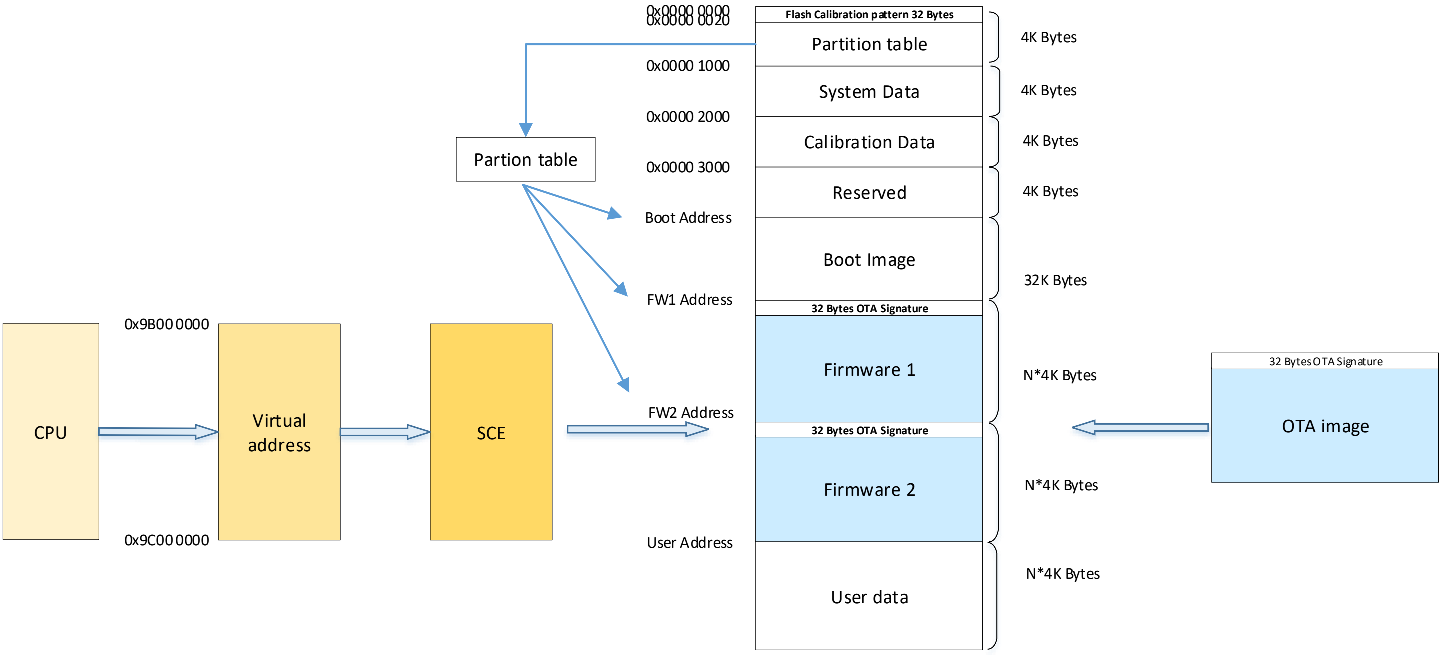


Figure 7-4 OTA update procedure

In Ameba-ZII OTA update procedure, **Firmware 1** and **Firmware 2** are swapped to each other.
The Firmware 1/Firmware 2 addresses are stored in partition records, defined in ‘***partition.json***’ under ‘*project\realtek_amebaz2_v0_example\GCC-RELEASE\*’. Please adjust it according to your firmware size.

```
"fw1":{
        "start_addr" : "0x10000",
        "length" : "0x80000",
        "type": "FW1",
        "dbg_skip": false,

"hash_key":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F"
     },
"fw2":{
        "start_addr" : "0x90000",
        "length" : "0x80000",
        "type": "FW2",
        "dbg_skip": false,

"hash_key":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F"
     }
```

7.5 Firmware Image Output

After building project source files in SDK, it would generate firmware as ‘**firmware_is.bin**’, which is the OTA Firmware as mentioned earlier.

7.5.1 OTA Firmware Swap Behavior

When device executes OTA procedure, it would update the other OTA block, rather than the current running OTA block. The OTA firmware swap behavior should be looked like as below figure if the updated firmware keeps using newer serial number value.

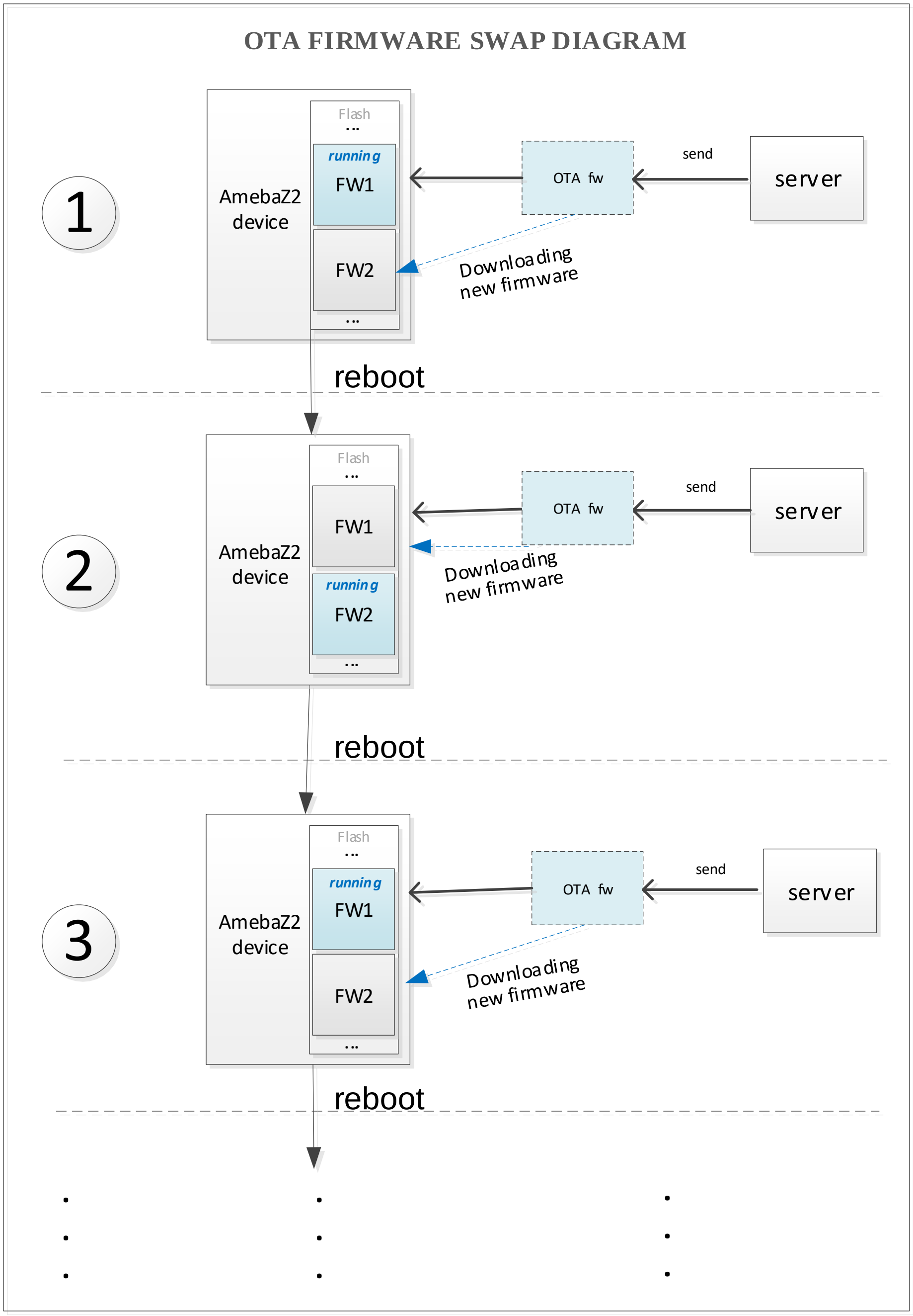


Figure 7-5 OTA Firmware SWAP Procedure

7.5.2 Configuration for Building OTA Firmware

Before building the project, the bootloader would check the serial number of OTA firmware to determine the boot sequence, the serial number of the OTA firmware needs to be configured correctly before project build.

7.5.2.1 Serial Number

Ameba-ZII OTA use serial number to decide the boot sequence if the signature of firmware is valid. Hence before building the project, please make sure the serial number is correctly configured.

For **ignore secure project**, to set the serial number of a firmware, please follow below steps:

Step 1: The serial number setting of a firmware is the same as the serial number of its first image. You can check the images sequence in *project\realtek_amebaz2_v0_example\GCC-RELEASE\amebaz2_firmware_is.json*.

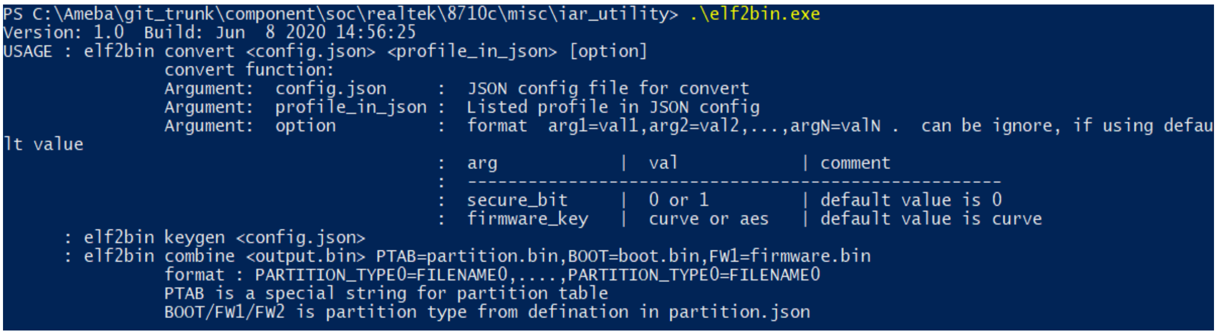
```
"FIRMWARE":{
                    "images":[
                              {"img": "FWHS", "offset":"0x00"},
                              {"img": "XIP_FLASH_C", "offset":"0x00"},
                              {"img": "XIP_FLASH_P", "offset":"0x00"}
                    ]
          },
```

For this example, the FWHS is located at the top sequence. Hence it is the first image of this firmware.

Step 2: Modify the serial number setting of the first image. Take above figure for example, we need to modify the serial number of “**FWHS**”:

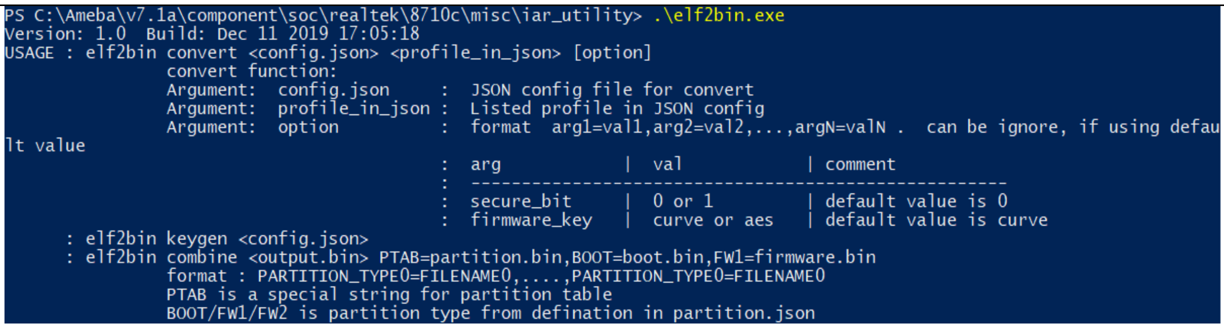
```
"FWHS": {
                    "source":"application_is/Debug/bin/application_is.axf",
                    "header":{
                              "next":null,
                              "__comment_type":"Support
Type:PARTAB,BOOT,FWHS_S,FWHS_NS,FWLS,ISP,VOE,WLN,DTCM,ITCM,SRAM,ERAM,XIP,M0,CPFW",
                              "type":"FWHS_S",
                              "enc":false,
        "user_key2":"BB0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F",
                              "__comment_pkey_idx":"assign by program, no need to configurate",
                              "serial": 100
                    },
```

In new version of elf2bin tool, the Serial number is stored as 32-bit unsigned int in firmware header, valid range is from 0(0x0) ~ 4294967295(0xFFFFFFFF) and larger number means newer version. Please modify it according to your firmware version. If negative value is given to “serial”, it will become 0. If value larger than 0xFFFFFFFF is given, it will become 0xFFFFFFFF. Please make sure the elf2bin tool is below version or newer.



```
PS C:\Ameba\git_trunk\component\soc\realtek\8710c\misc\iar_utility> .\elf2bin.exe
Version: 1.0  Build: Jun  8 2020 14:56:25
USAGE : elf2bin convert <config.json> <profile_in_json> [option]
                convert function:
                Argument:  config.json     :  JSON config file for convert
                Argument:  profile_in_json :  Listed profile in JSON config
                Argument:  option          :  format  arg1=val1,arg2=val2,...,argN=valN .  can be ignore, if using defau
lt value
                                           :  arg             |  val             | comment
                                           :  -----------------------------------------------------
                                           :  secure_bit      |  0 or 1          | default value is 0
                                           :  firmware_key    |  curve or aes    | default value is curve
      : elf2bin keygen <config.json>
      : elf2bin combine <output.bin> PTAB=partition.bin,BOOT=boot.bin,FW1=firmware.bin
                format : PARTITION_TYPE0=FILENAME0,.....,PARTITION_TYPE0=FILENAME0
                PTAB is a special string for partition table
                BOOT/FW1/FW2 is partition type from defination in partition.json
```

Important: There was a limitation in the old version of elf2bin tool. The valid number is from 0(0x0) to 2147483647(0x7FFFFFFF, INT_MAX), if negative value is given to “serial”, -2147483648 ~ -1 will be 2147483648(0x80000000) ~ 4294967295(0xFFFFFFFF) which will be even larger. Any version or build time which is older than below one has this limitation.



```
PS C:\Ameba\v7.1a\component\soc\realtek\8710c\misc\iar_utility> .\elf2bin.exe
Version: 1.0  Build: Dec 11 2019 17:05:18
USAGE : elf2bin convert <config.json> <profile_in_json> [option]
                convert function:
                Argument:  config.json     :  JSON config file for convert
                Argument:  profile_in_json :  Listed profile in JSON config
                Argument:  option          :  format  arg1=val1,arg2=val2,...,argN=valN .  can be ignore, if using defau
lt value
                                           :  arg            |  val            | comment
                                           :  -----------------------------------------------------
                                           :  secure_bit     |  0 or 1         | default value is 0
                                           :  firmware_key   |  curve or aes   | default value is curve
      : elf2bin keygen <config.json>
      : elf2bin combine <output.bin> PTAB=partition.bin,BOOT=boot.bin,FW1=firmware.bin
                format : PARTITION_TYPE0=FILENAME0,.....,PARTITION_TYPE0=FILENAME0
                PTAB is a special string for partition table
                BOOT/FW1/FW2 is partition type from defination in partition.json
```

Step 3: After building project source files in SDK, it should automatically generate *SDK_folder/project/project_name/GCC-RELEASE/application_is/Debug/bin/firmware_is.bin*, which is the application of OTA Firmware. The serial information would also be included in this firmware.

7.6 Implement OTA Over Wi-Fi

7.6.1 OTA Using Local Download Server Base on Socket

The example shows how device updates image from a local download server. The local download server sends image to device based on network socket.
Make sure both device and PC are connecting to the same local network.

7.6.1.1 Build OTA Application Image

Turn on OTA command

The flag defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts.h*.

```
//on/off relative commands in log service
#define CONFIG_OTA_UPDATE       1
```

Download the firmware to Ameba-ZII board to execute OTA.

7.6.1.2 Setup Local Download Server

Step 1: Build **new** firmware firmware_is.bin and place it in tools\DownloadServer folder.

Step 2: Edit start.bat file: Port = 8082, file = firmware_is.bin

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

Step 3: Execute ‘start.bat’.

```
c():checksum 0x202f57d
Listening on port (8082) to send firmware_is.bin (318592 bytes)

Waiting for client ...
```

7.6.1.3 Execute OTA Procedure

After the device connects to AP, enter command: ATWO=IP[PORT]. Please note that the device and your PC need under the same AP. The IP in ATWO command is the IP of your PC.

```
# ATWO=192.168.0.103[8082]
[ATWO]: _AT_WLAN_OTA_UPDATE_
```

```
[MEM] After do cmd, available heap 92768


#
[update_ota_local_task] Update task start
[update_ota_prepare_addr] fw1 sn is 100, fw2 sn is 0
[update_ota_prepare_addr] NewFWAddr 00090000

[update_ota_local_task] Read info first
[update_ota_local_task] info 12 bytes
[update_ota_local_task] tx file size 0x4dc80
[update_ota_local_task] Current firmware index is 1

[update_ota_erase_upg_region] NewFWLen 318592
[update_ota_erase_upg_region] NewFWBlkSize 78  0x4e
[update_ota_local_task] Start to read data 318592 bytes
.
[update_ota_local_task] sig_backup for 32 bytes from index 0
......................................................................
........
Read data finished

[update_ota_signature] Append OTA signature
[update_ota_signature] signature:
 64 89 F2 09 0A 2A EC 7B 82 3F 1A 15 3C 92 00 66
 98 6E 45 94 1E 1D 71 9C E0 E3 15 7A 7F 76 B1 89
[update_ota_local_task] Update task exit
[update_ota_local_task] ReaÄy to reboot
== Rtl8710c IoT Platform ==
```

Local download server success message:

```
c():checksum 0x202f57d
Listening on port (8082) to send firmware_is.bin (318592 bytes)

Waiting for client ...
Accept client connection from 192.168.0.108
Send checksum and file size first
Send checksum byte 12
Sending file...
......................................................................
......................................................................
......................................................................
......................................................................
....................................
Total send 318592 bytes
Client Disconnected.
Waiting for client ...
```

After finishing downloading image, device will be auto-rebooted, and the bootloader will boot by the firmware with larger serial number.

7.6.2 OTA Using Local Download Server Based on HTTP

This example shows how device updates image from a local http download server. The local http download server will send the http response which data part is ‘***firmware_is.bin***’ after receiving the http request.

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

7.6.2.1 Build OTA Application Image

Turn on OTA command

The flags defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts.h* and *\component\soc\realtek\8710c\misc\platform\ota_8710c.h*.

```
/* platform_opts.h */
//on/off relative commands in log service
#define CONFIG_OTA_UPDATE       1

#define CONFIG_EXAMPLE_OTA_HTTP       1
```

```
/* ota_8710c.h */
#define HTTP_OTA_UPDATE
```

Define Server IP and PORT in example_ota_http.c file

(In *\component\common\example\ota_http\example_ota_http.c*)

```
#define PORT                    8082
#define IP                      "192.168.0.103"
#define RESOURCE      "firmware_is.bin"
```

Download the firmware to Ameba-ZII board to execute OTA.

Communication with Local HTTP download server

1. In *http_update_ota_task()*, after connecting with server, Ameba will send a HTTP request to server : "GET /RESOURCE HTTP/1.1\r\nHost: host\r\n\r\n".

2. The local HTTP download server will send the HTTP response after receiving the request. The response header contains the "Content-Length" which is the length of the *firmware_is.bin*. The response data part is just *firmware_is.bin.*

3. After Ameba receives the HTTP response, it will parse the http response header to get the content length to judge if the receiving *firmware_is.bin* is completed.

7.6.2.2 Setup Local Http Download Server

Step 1: Build **new** firmware firmware_is.bin and place to tools\DownloadServer(HTTP) folder.

Step 2: Edit http_server.py file:

```
is_https = 0
server_ip = 0.0.0.0
server_port = 8082
```

Step 3: Execute http_server.py.
Command: **python http_server.py**

7.6.2.3 Execute OTA Procedure

Reboot the device and connect to AP, it should start the OTA update through HTTP protocol after 1 minute.

```
#
#
[update_ota_prepare_addr] fw1 sn is 100, fw2 sn is 0
[update_ota_prepare_addr] NewFWAddr 00090000
```

```
[http_update_ota] Download new firmware begin, total size : 320256

[http_update_ota] Current firmware index is 1

[http_update_ota] fw size 320256, NewFWAddr 00090000

[update_ota_erase_upg_region] NewFWLen 320256
[update_ota_erase_upg_region] NewFWBlkSize 79  0x4f.
[http_update_ota] sig_backup for 32 bytes from 0 index
.....................................................................
.........
[http_update_ota] Download new firmware 320256 bytes completed

[update_ota_signature] Append OTA signature
[update_ota_signature] signature:
 DD E9 FE 19 3B 15 79 99 8A 3C 84 FE 28 FB A2 13
 53 0F DE 71 3B 7E 46 48 9F 9D 03 2C DB EB D3 B7
[http_update_ota_task] Update task exit
[http_update_ota_task] ReaÄy to reboot
== Rtl8710c IoT Platform ==
```

Local download server success message:

```
<Local HTTP Download Server>
Listening on port (8082) to send firmware_is.bin (320256 bytes)

Waiting for client ...
Accept client connection from 192.168.0.108
Waiting for client's request...
Receiving GET request, start sending file...
.....................................................................
.....................................................................
.....................................................................
.....................................................................
......................................
Total send 320299 bytes
Client Disconnected.
Waiting for client ...
```

After finishing downloading image, device will be auto-rebooted, and the bootloader will load new firmware if it exists.

8 Power Save

8.1 WLAN Power Management

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

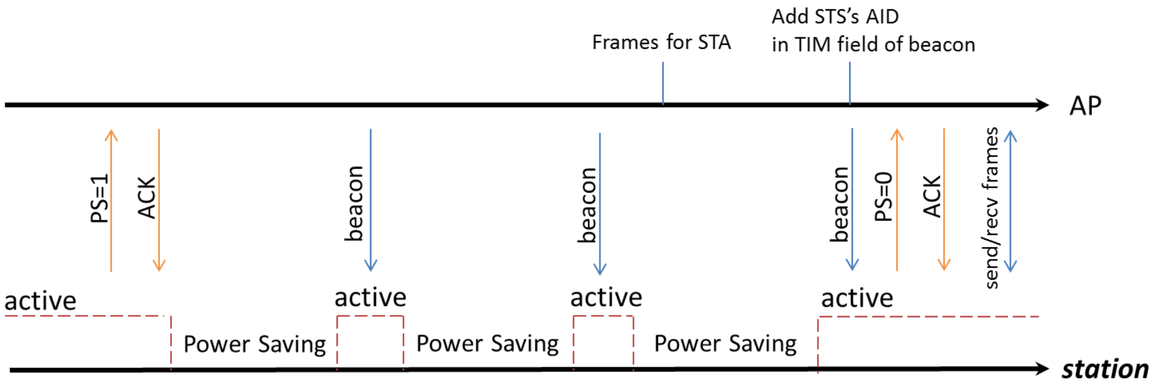


Figure 8-1 timeline of power saving

8.1.1 Ameba LPS

This feature is implemented in wlan driver. wlan driver enters LPS automatically without user application involved.

Ameba LPS (Leisure Power Save) implements IEEE 802.11 power management. Wlan driver enters LPS if flowing criteria meets:

(i) TX + RX packets count <= 8 in 2 seconds

(ii) RX packets count <= 2 in 2 seconds

It is checked in traffic status watch dog. The criteria are to keep high performance while traffic is busy. After entering LPS, there is PMU (Power Management Unit) control state machines.

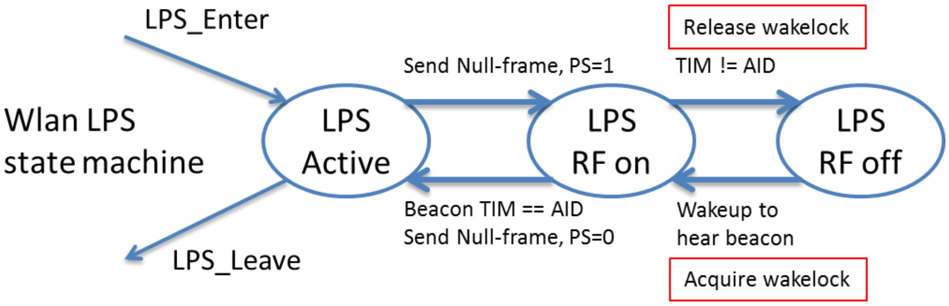


Figure 8-2 LPS state machine

8.1.2 Ameba IPS

This feature is implemented in wlan driver. Wlan driver enters IPS automatically without user application involved.

Ameba LPS is for the situation that Ameba is associate to an AP. If Ameba is not associated to an AP, the driver automatically turns off RF and other module to save power. Wlan Driver also releases wlan's wakelock at this time. When wlan driver needs to use RF related function, it automatically turns RF on and acquire wlan's wakelock. This scenario is called IPS (Inactive Power Save).

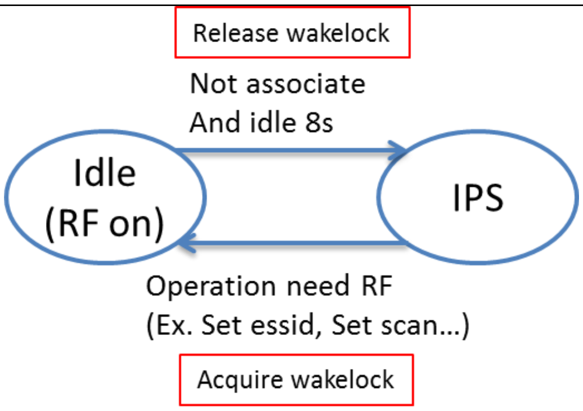


Figure 8-3 IPS state machine

8.2 Power Consumption Measurement

8.2.1 Hardware preparation

In Ameba-ZII reference board, there are other components that consume power. For example, there are cortex-M0 for DAP usage, LEDs, FT232, and capacitances. To measure power consumptions only for Ameba-ZII, you need to wire the connector J34 and measure power consumption between these two pins.

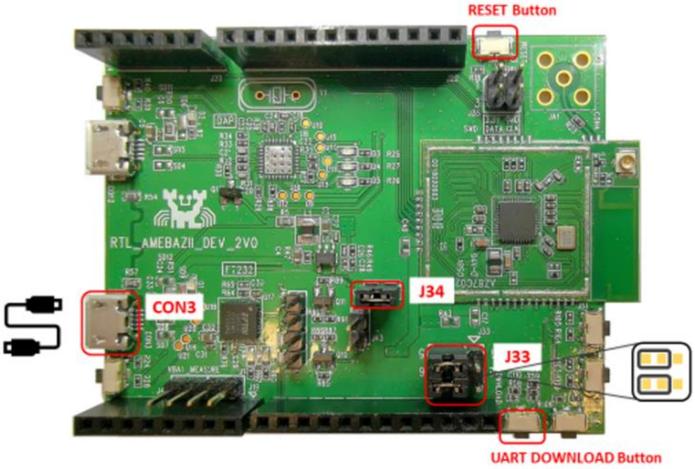


Figure 8-4 Power consumption measurement

You can use micro USB to power the board, and link current meter use J34 like following Figure:

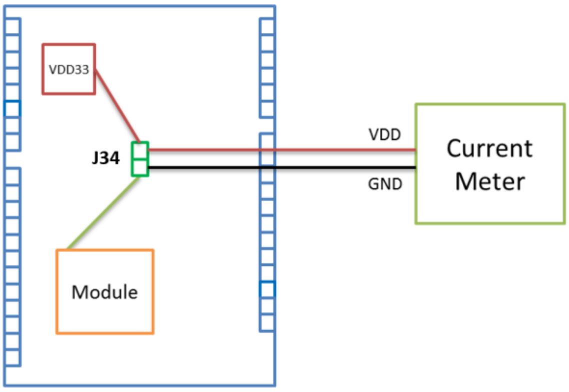


Figure 8-5 Measure power consumption from micro usb

8.2.2 Build SDK

Below are the suggested operations to measure power consumption:

1. Commands to measure No.1 (Wlan beacon only mode):

```
ATW0=<ap_ssid>
ATW1=<ap_password>
ATWC
```

Wait until wlan associate success.

```
ATXP=lps,0
```

2. Commands to measure No.2/3/4 (Wlan LPS):

```
ATW0=<ap_ssid>
ATW1=<ap_password>
ATWC
```

Wait until wlan associate success.

```
ATXP=lps,0
ATXP=dtim,<1/3/10>
ATXP=lps,1
```

Wait until Wlan becomes idle, it will enter LPS mode

3. Commands to measure No.5 (Wlan IPS)

```
ATW0=<ap_ssid>
ATW1=<ap_password>
ATWC
```

Wait until wlan associate success.

```
ATWD
```

Wait ~2s after Wlan is disconnected, it will enter IPS mode

8.3 Power Consumption Result

The following table lists the power consumption of Ameba-ZII under 3.3V power supply.

Board Information:

- Board number: AMEBAZII_DEV_2V0
- Module number: AZ87CC1_2V1
- Chip number: RTL8720_CX
- FLASH is external, GPIO_A7 to GPIO_A12 is occupied for FLASH
- JTAG is enabled, GPIO_A1 and GPIO_A0 is occupied for JTAG
- log UART is GPIO_A15 and GPIO_A16

SVN version

- v7.1c

Wakeup Source	Clock (Hz)					
	250k	4M	250k	4M	250k	4M
	DeepSleep (uA)		Standby (uA)		Sleep (uA)	
Stimer	25.9	25.8	181	197	407	408
GPIO_A2	301	302	458	471	684	702
GPIO_A3	294	301	457	470	686	692
GPIO_A4	303	301	458	470	684	686
GPIO_A13	315	303	456	472	687	686
GPIO_A14	301	300	454	469	679	680
GPIO_A17	298	300	457	469	685	678
GPIO_A18	301	301	456	468	682	683
GPIO_A19	299	300	459	471	680	684

GPIO_A20	302	300	457	473	681	680
GPIO_A23	304	299	462	470	678	678
UART_0	NA	NA	753	770	1036	1033
Gtimer_0	NA	NA	677	689	942	948
Gtimer_1	NA	NA	672	692	953	945
Gtimer_2	NA	NA	678	692	945	944
Gtimer_3	NA	NA	681	685	952	947
Gtimer_4	NA	NA	679	692	947	948
Gtimer_5	NA	NA	678	687	947	950
Gtimer_6	NA	NA	680	688	950	945
PWM_0 PA_20	NA	NA	703	714	970	966
PWM_2 PA_2	NA	NA	689	715	967	965
PWM_3 PA_3	NA	NA	699	712	968	967
PWM_4 PA_4	NA	NA	695	710	971	970
PWM_5 PA_17	NA	NA	706	713	974	968
PWM_6 PA_18	NA	NA	702	717	970	969
PWM_7 PA_13	NA	NA	694	709	972	971

Table 8-1 power consumption summary

No.	Mode	MCU State	Description	Power Supply 3.3V	
				8720CN RF PIN8 LDO RF PIN12 SWR	8720CN RF PIN8 LDO RF PIN12 LDO
1	Wlan beacon only mode	Active		***52 mA***	***60 mA***
2	Wlan asoc Idle (2.4G), RF ON (LPS)	Active	DTIM = 1	***16.813 mA***	***19.588 mA***
3	Wlan asoc Idle (2.4G), RF ON (LPS)	Active	DTIM = 3	***16.149 mA***	***18.619 mA***
4	Wlan asoc Idle (2.4G), RF ON (LPS)	Active	DTIM = 10	***16.823 mA***	***17.284 mA***
5	Wlan un-asoc, RF OFF (IPS)	Active		***11.6mA***	***11.6mA***
NOTICE: Result in this table was tested in shielding room; It may be different under different environment.					

Table 8-2 Wi-Fi power consumption

No.	Item	Power Supply	MCU state	BT mode	Current(mA)
1	BT Tx	3.3v	Active	BT MP	127 (2.5dBm) 131 (4.5dBm) 145 (6.5dBm)
2	BT Rx			BT Central Mode	60
3	BT ADV			BT Peripheral Mode	61
4	BT Connection			BT Central Mode	61

Table 8-3 BT power consumption

9 Bluetooth

9.1 Features

Please refer to user manual '***UM0501 Realtek AmebaZ2 BLE Stack User Manual EN*.pdf**'.

9.2 BT Wi-Fi Coexist

Since Wi-Fi and BT share the same RF block, ensure that Wi-Fi power save is not enabled when BT is active. When BT is enabled, the wifi_disable_powersave() API will be called, so avoid calling the wifi_enable_powersave() API while BT is on.

9.3 Memory Usage

Since Wi-Fi and BT share the same RF block, enabling BT requires Wi-Fi to be enabled as well. Below is the memory usage for Wi-Fi only and Wi-Fi + BT.

9.3.1 Wi-Fi Only

- XIP code size: 436 KB
- SRAM used: 72 KB
- Available Heap Size: 122 KB

9.3.2 Wi-Fi + BT

BT examples	Code size (XIP, Kbyte)	RAM size (Kbyte) (Ccompare with Wi-Fi only)		
		SRAM	Heap Used	Total (SRAM+Heap)
Example ble_peripheral	569	+ 3	+ 23	+ 26
Example bt_central	584	+ 3	+ 25	+ 28
Example ble_scatternet	598	+ 4	+ 26	+ 30
Example bt_beacon	561	+ 3	+ 22	+ 25
Example bt_config	570	+ 3	+ 34	+ 37

9.4 Examples

9.4.1 ble_peripheral

This example shows how to create and run GATT service on GATT server.

9.4.1.1 Image Generation

(1) To run ble_peripheral example, turn on the following flags defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts_bt.h*

```
#define CONFIG_BT                              1
#if CONFIG_BT
```

```
#define CONFIG_FTL_ENABLED
#define CONFIG_BT_CONFIG                0
#define CONFIG_BT_PERIPHERAL            1
#define CONFIG_BT_CENTRAL               0
#define CONFIG_BT_SCATTERNET            0
#define CONFIG_BT_BEACON                0
#define CONFIG_BT_MESH_PROVISIONER      0
#define CONFIG_BT_MESH_DEVICE           0
```

(2) Build image and download image to your board.

9.4.1.2 Test Procedure

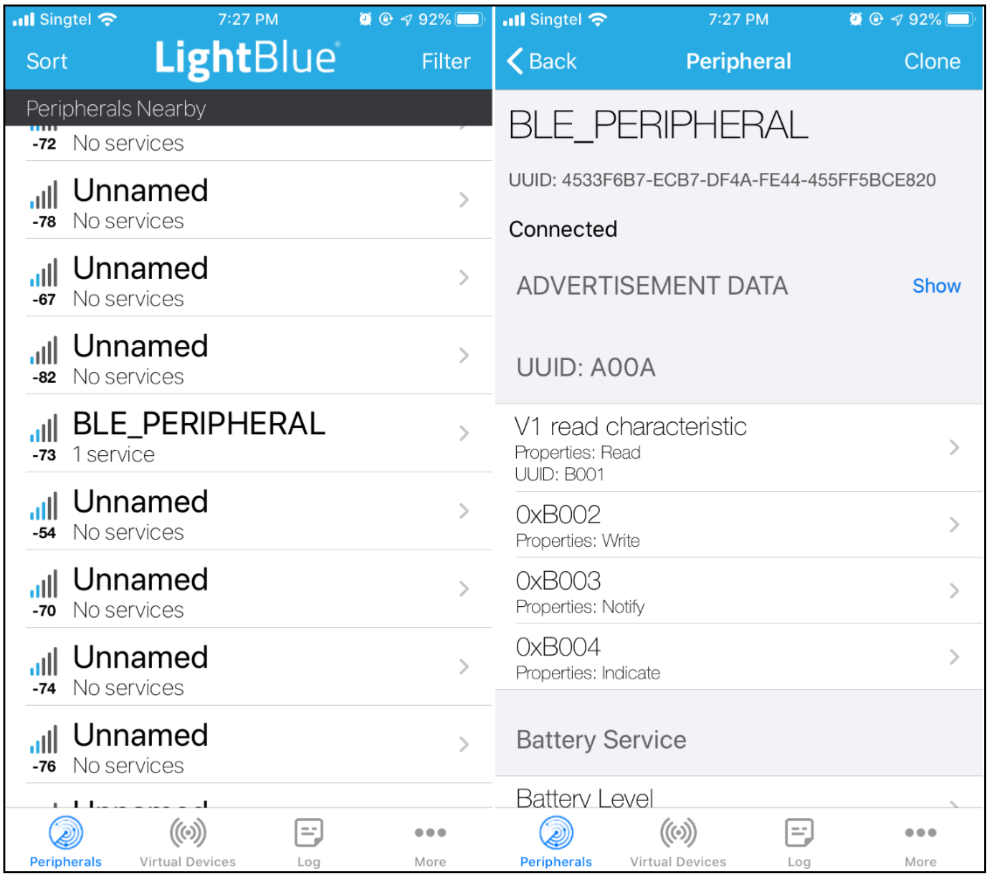
(1) After downloading image to your Ameba-ZII board, reset it. The default device name is BLE_PERIPHERAL.

(2) Download apps such as “LightBlue” or “nRF Connect” and use as GATT Client to connect it.

(3) ATBp is an AT command for BT Peripheral. Using “ATBp=1” to initialize BT Peripheral stack, which can send advertising package out and scannable by other devices.

```
[BLE peripheral] GAP stack ready

GAP adv start

[MEM] After do cmd, available heap 88064
```

(4) Search for BLE_PERIPHERAL device and connect to it.



9.4.2 ble_central

This example shows how to discover service on GATT server.

9.4.2.1 Image Generation

(1) To run ble_central example, turn on the following flags defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts_bt.h*

```
#define CONFIG_BT                        1

#if CONFIG_BT
#define CONFIG_FTL_ENABLED
#define CONFIG_BT_CONFIG                 0
#define CONFIG_BT_PERIPHERAL             0
#define CONFIG_BT_CENTRAL                1
#define CONFIG_BT_SCATTERNET             0
#define CONFIG_BT_BEACON                 0
#define CONFIG_BT_MESH_PROVISIONER       0
#define CONFIG_BT_MESH_DEVICE            0
```

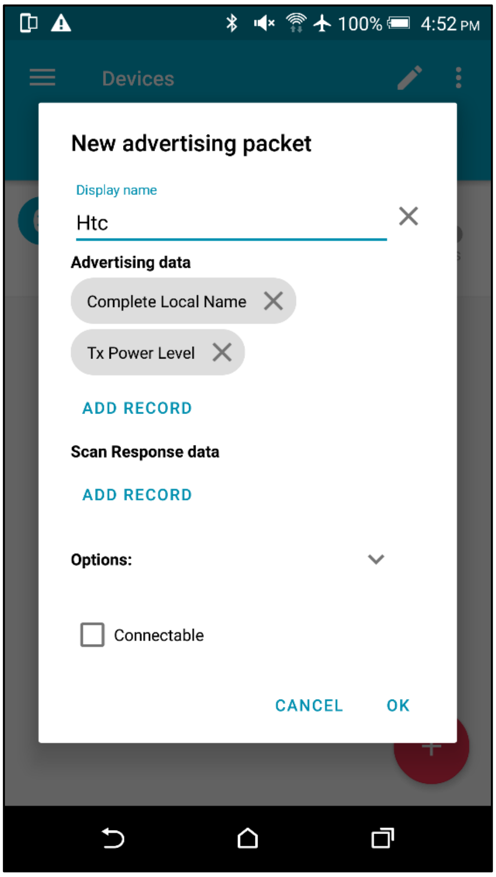
(2) Build image and download image to your board.

9.4.2.2 Test Procedure

(1) After downloading image to your Ameba-ZII board, reset it.

(2) Download app “nRF Connect” and use as GATT Server to be connected.

(3) Add a new advertising packet and set its additional data.



New advertising packet

Display name: Htc

Advertising data

Complete Local Name

Tx Power Level

Scan Response data

Options:

☐ Connectable

(4) ATBc is an AT command for BT Central. Using “ATBc=1” to turn BT Central stack ON.

(5) Using “ATBS=1” to scan available BT devices nearby.

(6) Using “ATBC=P/R, BLE_BD_ADDR” to connect to the device.

BT Central scan and connect log:

```
#ATBS=1
Start scan, scan_filter_policy = 0, scan_filter_duplicate = 1

[MEM] After do cmd, available heap 90360


# GAP scan start
ADVType                        | AddrType       |BT_Addr
|rssi
NON_CONNECTABLE                random   5e:3b:de:4e:96:38       -82
```

```
GAP_ADTYPE_FLAGS: 0x0
GAP_ADTYPE_LOCAL_NAME_XXX: HTC_E9pw
GAP_ADTYPE_POWER_LEVEL: 0x2
ADVType                  | AddrType      |BT_Addr
|rssi
NON_CONNECTABLE          random  5f:ee:5f:ce:06:1f       -100
GAP_ADTYPE_MANUFACTURER_SPECIFIC: company_id 0x6, len 27
ADVType                  | AddrType      |BT_Addr
|rssi
NON_CONNECTABLE          random  03:af:5e:a9:3f:70       -92
GAP_ADTYPE_MANUFACTURER_SPECIFIC: company_id 0x6, len 27
#ATBS=0
Stop scan

[MEM] After do cmd, available heap 90360


# GAP scan stop
# ATBC=R,5e3bde4e9638

[MEM] After do cmd, available heap 86696


# cmd_con, DestAddr: 0x5E:0x3B:0xDE:0x4E:0x96:0x38
```

For more AT commands used for BT Central, please refer to user manual '***UM0201 Ameba Common BT Application User Manual EN.pdf*** '.

9.4.3 ble_scatternet

BLE Scatternet is the coexistence of BLE Central mode and BLE Peripheral mode. Once BLE Scatternet stack initialized, AT command of BLE Central and BLE Peripheral are available. This example shows how to turn BLE Scatternet on.

9.4.3.1 Image Generation

(1) To run ble_central example, turn on the following flags defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts_bt.h*

```
#define CONFIG_BT                       1

#if CONFIG_BT
#define CONFIG_FTL_ENABLED
#define CONFIG_BT_CONFIG                0
#define CONFIG_BT_PERIPHERAL            0
#define CONFIG_BT_CENTRAL               0
#define CONFIG_BT_SCATTERNET            1
#define CONFIG_BT_BEACON                0
#define CONFIG_BT_MESH_PROVISIONER      0
#define CONFIG_BT_MESH_DEVICE           0
```

(2) Build image and download image to your board.

9.4.3.2 Test Procedure

(1) After downloading image to your Ameba-ZII board, reset it.

(2) Using “ATBf=1” to turn BT Scatternet stack ON.

(3) Once see the following message, you can continue input other AT command of BT Scatternet mode as well as BT Central mode and BT Peripheral mode.

```
START ADV!!
GAP adv start
```

For other AT commands used for BT Scatternet, please refer to '***UM0201 Ameba Common BT Application User Manual EN.pdf*** '.

9.4.4 bt_beacon

This example shows how to send BLE Beacons. Ameba-ZII provides two types of Beacon: Apple iBeacon and Radius Networks AltBeacons.

9.4.4.1 Image Generation

(1) To run bt_beacon example, turn on the following flags defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts_bt.h.*

```
#define CONFIG_BT                       1

#if CONFIG_BT
#define CONFIG_FTL_ENABLED
#define CONFIG_BT_CONFIG                0
#define CONFIG_BT_PERIPHERAL            0
#define CONFIG_BT_CENTRAL               0
#define CONFIG_BT_SCATTERNET            0
#define CONFIG_BT_BEACON                1
#define CONFIG_BT_MESH_PROVISIONER      0
#define CONFIG_BT_MESH_DEVICE           0
```

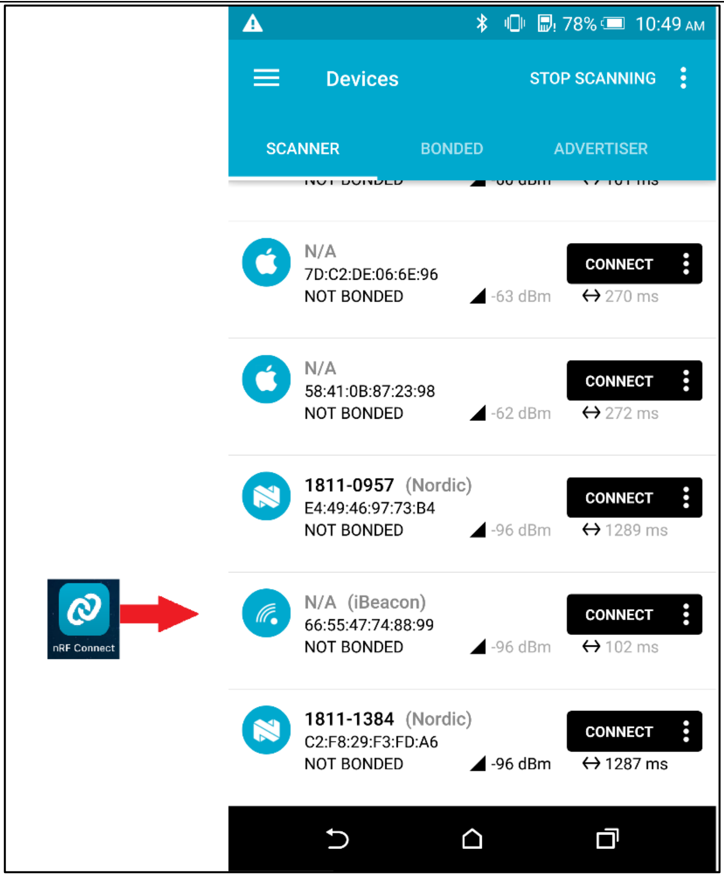
(2) Build image and download image to your board.

9.4.4.2 Test Procedure

(1) Choose beacon type by using “ATBJ=1,1” or “ATBJ=1,2” command.

```
# ATBJ
[ATBJ] Start BT I_Beacon: ATBJ=1,1
[ATBJ] Start BT Alt_Beacon: ATBJ=1,2
[ATBJ] Stop  BT Beacon: ATBJ=0
```

(2) You can use apps such as “Locate” on iOS, “LightBlue” or “nRF Connect” to observe beacons. “Locate” observe beacon by it adv UUID. Below screenshot is taken using Android “nRF Connect”.



N/A
7D:C2:DE:06:6E:96
NOT BONDED -63 dBm ↔ 270 ms
CONNECT

N/A
58:41:0B:87:23:98
NOT BONDED -62 dBm ↔ 272 ms
CONNECT

1811-0957 (Nordic)
E4:49:46:97:73:B4
NOT BONDED -96 dBm ↔ 1289 ms
CONNECT

N/A (iBeacon)
66:55:47:74:88:99
NOT BONDED -96 dBm ↔ 102 ms
CONNECT

1811-1384 (Nordic)
C2:F8:29:F3:FD:A6
NOT BONDED -96 dBm ↔ 1287 ms
CONNECT

9.4.5 bt_config

BT Config provides a simple way for Wi-Fi device to associate to AP easily.

9.4.5.1 Image Generation

(1) To run bt_config example, turn on the following flags defined in *\project\realtek_amebaz2_v0_example\inc\platform_opts_bt.h.*

```
#define CONFIG_BT                          1

#if CONFIG_BT
#define CONFIG_FTL_ENABLED
#define CONFIG_BT_CONFIG                   1
#define CONFIG_BT_PERIPHERAL               0
#define CONFIG_BT_CENTRAL                  0
#define CONFIG_BT_SCATTERNET               0
#define CONFIG_BT_BEACON                   0
#define CONFIG_BT_MESH_PROVISIONER         0
#define CONFIG_BT_MESH_DEVICE              0
```

(2) Build image and download image to your board.

9.4.5.2 APP Installation

(1) The installation package is located at *\tools\bluetooth\BT Config* in SDK. You can install Android or iOS as your phone OS.

9.4.5.3 Test Procedure

(2) ATBB is an AT command for BT Config. Using “ATBB=1” to enter BT Config mode, which allows BT Config APP to discover and connect to AmebaZII. Reset your AmebaZII board, and input command “ATBB=1”.

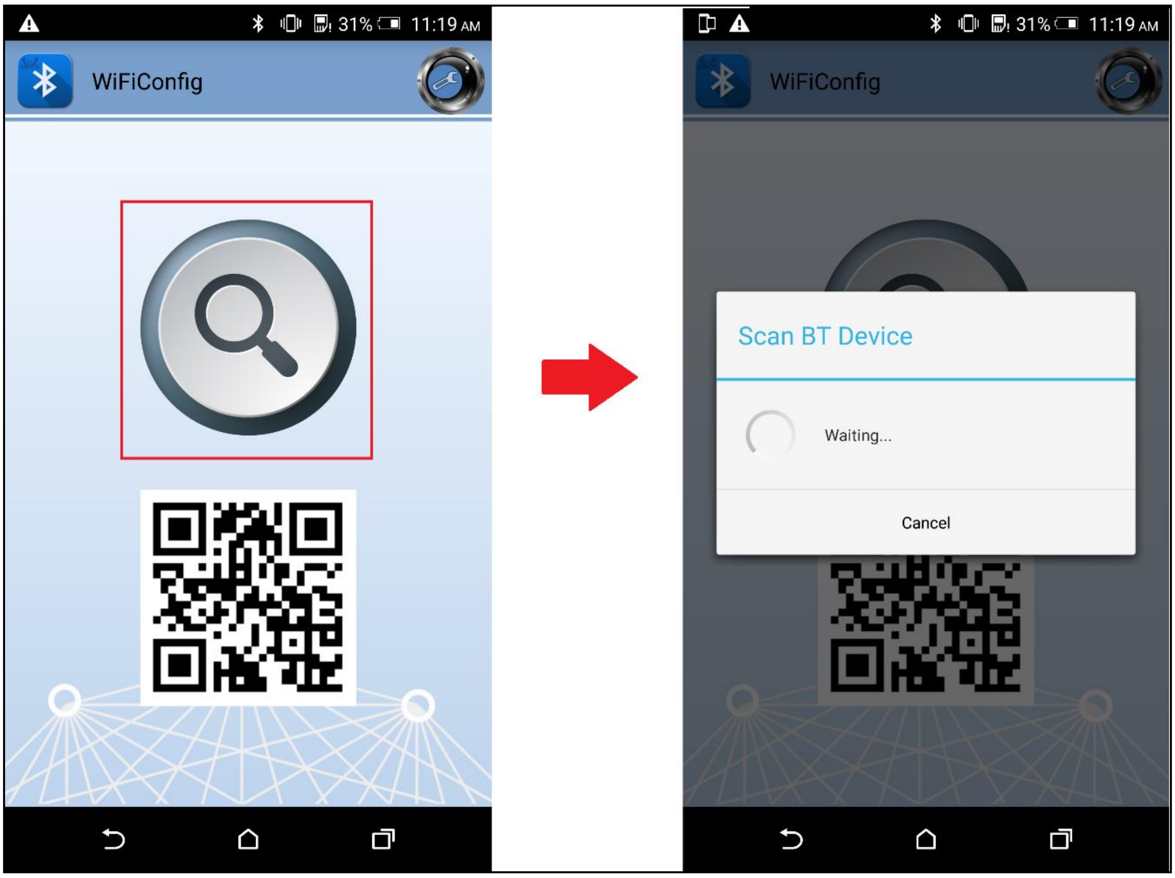
(3) Once see the following message, you can open BT Config APP to associate AP.

BT Initialize and start adv log:

```
[BT Config Wifi] BT Config Wifi ready

[BT Config Wifi] ADV started
```

(4) Click the BT config icon to launch it. Scan and connect with AmebaZII BT using BT Config app.

Display on BT config app:

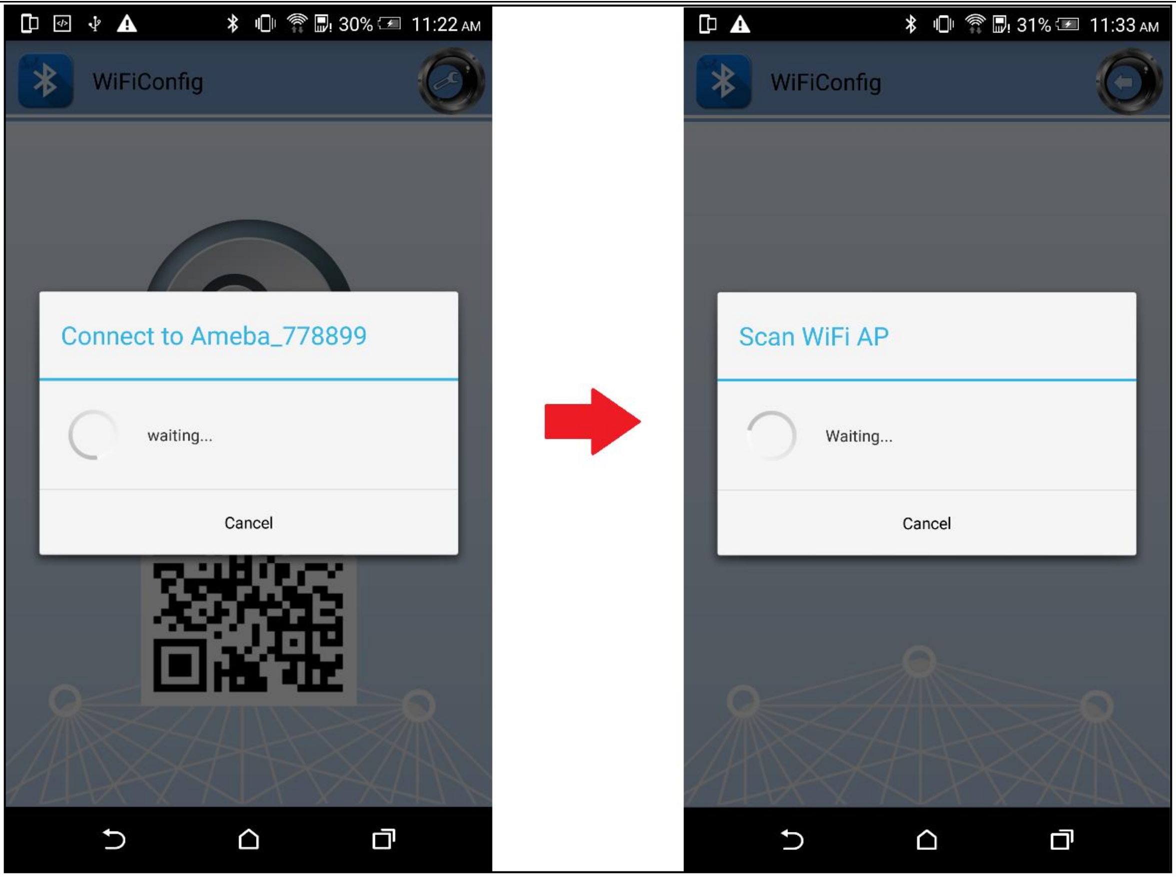


(5) Once BT Config APP is connected to AmebaZII, below log will be show. When connection is established AmebaZII will start searching for AP.

BT Connection log:

```
[BT Config Wifi] Bluetooth Connection Established

[BT Config Wifi] Band Request
[BT Config Wifi] Scan Request
[BT Config Wifi] Scan 2.4G AP
```

Display on BT config app:



30% 11:22 AM

WiFiConfig

Connect to Ameba_778899

waiting...

Cancel

31% 11:33 AM

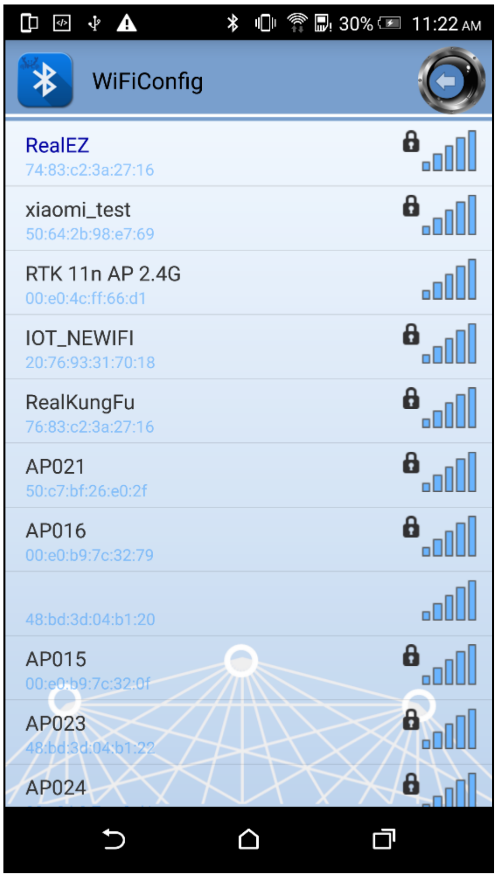
WiFiConfig

Scan WiFi AP

Waiting...

Cancel

Scanned and reachable APs will be show on BT config app:



30% 11:22 AM

WiFiConfig

RealEZ
74:83:c2:3a:27:16

xiaomi_test
50:64:2b:98:e7:69

RTK 11n AP 2.4G
00:e0:4c:ff:66:d1

IOT_NEWIFI
20:76:93:31:70:18

RealKungFu
76:83:c2:3a:27:16

AP021
50:c7:bf:26:e0:2f

AP016
00:e0:b9:7c:32:79

48:bd:3d:04:b1:20

AP015
00:e0:b9:7c:32:0f

AP023
48:bd:3d:04:b1:22

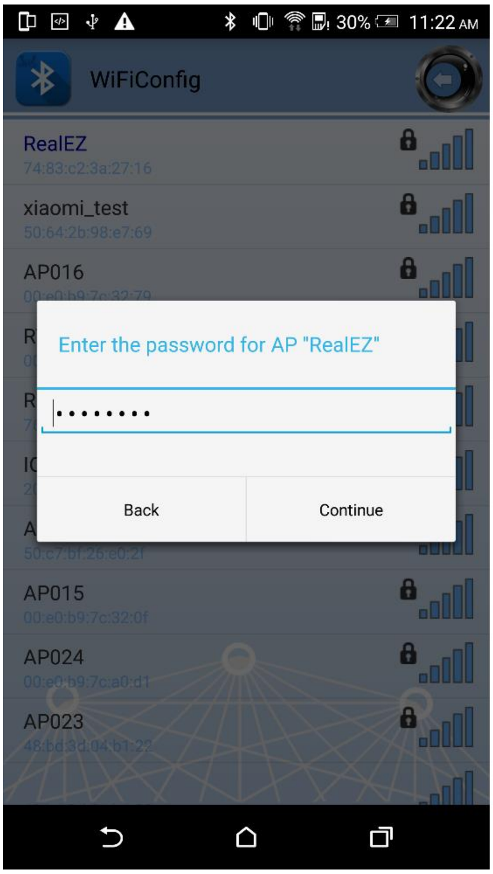
AP024

(6) Select an AP to connect to and input password (if any).

AP Connection log:

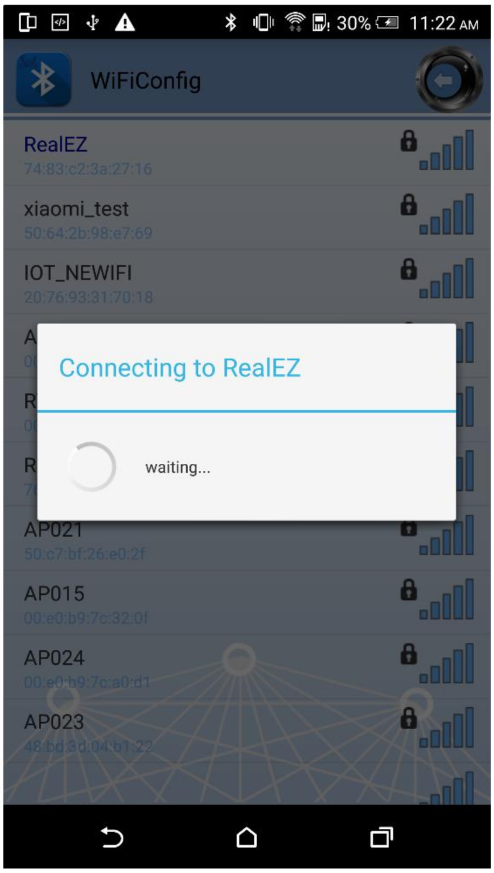
```
[BT Config Wifi] Connect Request
[Driver]: set BSSID: 90:94:e4:c5:d3:f0

[Driver]: set ssid [Test_ap]

[Driver]: start auth to 90:94:e4:c5:d3:f0

[Driver]: auth success, start assoc

[Driver]: association success(res=7)

[Driver]: set pairwise key to hw: alg:4(WEP40-1 WEP104-5 TKIP-2 AES-
4)

[Driver]: set group key to hw: alg:2(WEP40-1 WEP104-5 TKIP-2 AES-4)
keyid:1

[BT Config Wifi] Connected after 3458ms.

Interface 0 IP address : 192.168.0.102
[BT Config Wifi] Got IP after 3500ms.
```

Display on BT config app:



Enter the password for AP "RealEZ"

••••••••



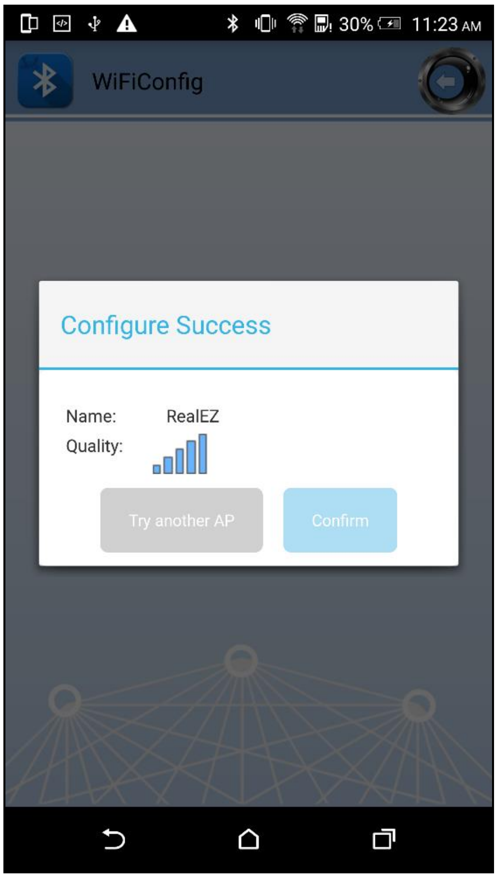
Connecting to RealEZ

waiting...

(7) When AmebaZII is connected to an AP, users can confirm connection or select another AP. Click “Confirm” to confirm AP connection. Click “Try another AP” to go back to Wi-Fi scan list page and choose another AP to connect to.
After confirming BT config result, Bluetooth connection is disconnected, AmebaZ2 becomes undiscoverable to BT Config APP.
BT Disconnect log:

```
[BT Config Wifi] Bluetooth Connection Disconnected

[BT Config Wifi] ADV started

[BT Config Wifi] [BC_status_monitor] wifi connected, delete
BC_cmd_task and BC_status_monitor

[BT Config Wifi] ADV stopped
```

Display on BT config app:



Configure Success

Name: RealEZ
Quality:

(8) You can use “ATBB=1” to restart BT Config mode again.

Command	**Usage**
ATBB=1	Start BT Config
ATBB=0	Stop BT Config

Note: Enter BT Config mode will disconnect existing Wi-Fi connection.
Please refer to BT Config APP User Guide in *tools**bluetooth**BT Config* for more details.

9.4.6 128-bit UUID Configuration

This example shows how to configure BLE service with 128-bit UUID.
Modify service table to configure BLE service with 128-bit UUID.

```
const uint8_t   GATT_UUID128_CUSTOMIZED_PRIMARY_SERVICE [16] =
{0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x00};
#define GATT_UUID128_CUSTOMIZED_CHAR   0x01, 0x23, 0x45, 0x67, 0x89, 0x0A, 0xBC, 0xDE, 0xFF,
0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
static const T_ATTRIB_APPL customized_UUID128_service_tbl[] =
{
   {
      (ATTRIB_FLAG_VOID | ATTRIB_FLAG_LE),
      {
         LO_WORD(GATT_UUID_PRIMARY_SERVICE),
         HI_WORD(GATT_UUID_PRIMARY_SERVICE),
      },
      UUID_128BIT_SIZE,
      (void *) GATT_UUID128_CUSTOMIZED_PRIMARY_SERVICE,
      GATT_PERM_READ
   },
   {
      ATTRIB_FLAG_VALUE_INCL,
      {
         LO_WORD(GATT_UUID_CHARACTERISTIC),
         HI_WORD(GATT_UUID_CHARACTERISTIC),
         GATT_CHAR_PROP_READ | GATT_CHAR_PROP_WRITE,
      },
      1,
      NULL,
      GATT_PERM_READ
   },
   {
      ATTRIB_FLAG_VALUE_APPL | ATTRIB_FLAG_UUID_128BIT,
      {
         GATT_UUID128_CUSTOMIZED_CHAR
      },
      0,
      NULL,
      GATT_PERM_READ | GATT_PERM_WRITE
   },

};
```

9.5 BT Transmit Power

BT advertising transmit power can be changed during runtime.

The following is the API and the description for changing the advertising transmit power.

```
/**
  * @brief  Set the advertising tx power for the device, or reset advertising tx power to default value.
Default power: 4.5dBm
  *
  *       NOTE: This function can be called after @ref vendor_cmd_init is invoked.
  *
  * @param[in] option  Set to 0.
  * @param[in] tx_gain index for power level. NOTE: The following tx gain table may be changed in
future version.
                  tx_gain   Power
                  0x0D     -10 dBm
                  0x21      0   dBm
                  0x2A      4.5 dBm
  * @retval GAP_CAUSE_SUCCESS Operation success.
  * @retval GAP_CAUSE_SEND_REQ_FAILED Operation failure.
  *
  */
#if BT_VENDOR_CMD_ADV_TX_POWER_SUPPORT
T_GAP_CAUSE le_adv_set_tx_power(uint8_t option, uint8_t tx_gain);
```

```
#endif
```

Note: for this API to work, please call the API after advertising is enabled.
Please refer to *component\common\bluetooth\realtek\sdk\board\amebaz2\src\vendor_cmd\ vendor_cmd_bt.h* for the complete description and example of usage case.

As stated in the description, the API only accepts three input variable values, 0x0D, 0x21 and 0x2A, which represent -10dBm, 0dBm and 4.5dBm respectively. These variables are not absolute values, they are offset values based on calibrated transmit powers during MP.
For example, in the default case BT transmit power is calibrated to 4.5dBm during MP. When the API is called with a pass in variable value of 0x21, the advertising transmit power at normal mode will be 0dBm. However, if BT transmit power is calibrated to 5.5dBm during MP and the API is called with a pass in variable value of 0x21, the actual advertising transmit power at normal mode will be 1dBm.
If board has not been calibrated, BT transmit power will follow the default case.

9.6 BT Default MAC Address

Bluetooth MAC address is stored in eFuse. If Bluetooth MAC address in eFuse is empty, default Bluetooth MAC address will be use. Default Bluetooth MAC is 0x99, 0x88, 0x77, 0x44, 0x55, 0x66.

Modify below array to change Bluetooth default MAC address, which defined in *component\common\bluetooth\realtek\sdk\board\amebaz2\src\hci.c*

```
unsigned char rtlbt_init_config[] =
{
    0x55, 0xab, 0x23, 0x87,
    0x10, 0x00,
    0x30, 0x00, 0x06, 0x99, 0x88, 0x77, 0x44, 0x55, 0x66, /* BT MAC address */
    //0x0c, 0x00, 0x04, 0x1d, 0x70, 0x00, 0x00, /* Baudrate 115200 */
     0x0c, 0x00, 0x04, 0x04, 0x50, 0xF7, 0x05,//*/ /* Baudrate 921600 */

    0x18, 0x00, 0x01,0x5c, /* flow control */
    /*efuse about*/
    0x94, 0x01, 0x06, 0x08, 0x00, 0x00, 0x00,0x2e, 0x07,

    0x9f, 0x01, 0x05, 0x2a, 0x2a, 0x2a, 0x2a,0x50,

    0xA4, 0x01, 0x04, 0xfe, 0xfe, 0xfe, 0xfe,
};
```

Note: only the highlighted variables can be modified.

10 Troubleshooting

There may be issues while developing user applications. Hence, there are some troubleshooting methods that can be referred to.

10.1 Hard Fault

AmebaZ2 platform provides detailed back trace information when a hard fault exception happens. Please refer to the following steps to view the full backtrace, which will help a lot in debugging.

10.1.1 GCC Environment

10.1.1.1 Install Cygwin

Please refer to section *“3.4.1 Install Cygwin”*.

10.1.1.2 Unzip Toolchain

1) Open “**Cygwin Terminal**”.

2) Direct to unzip path. Enter command “**cd** /SDK /project/realtek_amebaz2_v0_example/GCC-RELEASE”.

3) Enter command “**make toolchain**” to unzip toolchain.

10.1.1.3 Trace Hard Fault

Please refer to the following example of tracing the hard fault.

```
S-Domain Fault Handler: msp=0x1003f998 psp=0x1002bf70 lr=0xfffffff1
fault_id=2

Bus Fault:
SCB Configurable Fault Status Reg = 0x00000400

Bus Fault Status:
BusFault Address Reg is invalid(Asyn. BusFault)
Imprecise data bus error:
a data bus error has occurred, but the return address in the stack frame
is not related to the instruction that caused the error.

S-domain exception from Handler mode, Standard Stack frame on S-MSP
Registers Saved to stack

Stacked:
R0  = 0x10018f60
R1  = 0x9b01b7d1
R2  = 0x00000000
R3  = 0x1001dee4
R4  = 0x10017860
R5  = 0x1002c02b
R6  = 0x0002ea5d
R7  = 0x0002f424
R8  = 0x00000000
R9  = 0x1002c02b
R10 = 0x9b801c5b
R11 = 0x1002c068
R12 = 0x00000000
LR  = 0x9b0465e1
PC  = 0x9b01b7d0
PSR = 0xa100001c

Current:
```

```
LR   = 0xfffffff1
MSP  = 0x1003f9b8
PSP  = 0x1002bf70
xPSR = 0xa0000005
CFSR  = 0x00000400
HFSR  = 0x00000000
DFSR  = 0x00000000
MMFAR = 0x00000000
BFAR  = 0x00000000
AFSR  = 0x00000000
PriMask = 0x00000000
SVC priority: 0x00
PendSVC priority: 0xe0
Systick priority: 0xe0

MSP Data:
1003F9B8:    10018F60    9B01B7D1    00000000    1001DEE4
1003F9C8:    00000000    9B0465E1    9B01B7D0    A100001C
1003F9D8:    00000065    FFFFFFFD    00000000    100007C4
1003F9E8:    0000002D    0001869F    10008044    9B005959
1003F9F8:    9B0468B8    61000000    77CF8CC5    8B024015
1003FA08:    26384558    942D314C    0CEF815D    2AA0505C
1003FA18:    CBB9C6F0    1847AA69    BE94F781    37E00DAD
1003FA28:    CFE4C7DC    849BE050    2FFA91C4    89421B95
1003FA38:    FABAC7E8    356CADA8    8DF7F0D3    B10E0054
1003FA48:    D9F23435    E4AA8154    F6AE6C73    300910C2
1003FA58:    C1E4AFA1    49208098    3F0E59BE    B1B32F18
1003FA68:    3D179AF4    DC5894C0    8E33CDBC    E0323486
1003FA78:    A0FD56A3    AD4C2ACE    B6571FF4    E94209D0
1003FA88:    1FF5FD14    B8960ACF    373E09F4    17819289
1003FA98:    EF31AB8D    27F1EC18    529B29C4    E26100D0
1003FAA8:    7F3908FE    768860C0    9F7568AD    65D81576

PSP Data:
1002BF70:    1000E0B8    00000065    40040400    00000010
1002BF80:    00000000    0002EA69    000060D4    21000000
1002BF90:    0000000B    0002ECD3    0005F650    9B802D9C
1002BFA0:    1002BFE0    FFFFFFFF    1000DA78    00000001
1002BFB0:    00000000    00000000    1002C02A    00000000
1002BFC0:    00000000    00000000    00000000    00000000
1002BFD0:    00000001    FFFFFFFF    FFFFFFFF    0000001A
1002BFE0:    00000300    00000000    00000000    00000000
1002BFF0:    00000000    00000000    00000000    00000000
1002C000:    00000000    00000000    00000000    00000000
1002C010:    00000000    00000000    00000000    00000000
1002C020:    00000000    00000000    0A310000    00000020
1002C030:    00000000    0002EB15    00000001    00000001
1002C040:    00000001    9B801C40    9B801C64    10007FB4
1002C050:    00000200    9B005F2F    1002C048    00000004
1002C060:    9B00AD61    00000001    00000200    1002C048

 == NS Dump ==
CFSR_NS  = 0x00000000
HFSR_NS  = 0x00000000
DFSR_NS  = 0x00000000
MMFAR_NS = 0x00000000
BFAR_NS  = 0x00000000
AFSR_NS  = 0x00000000
MSP_NS   = 0x00000000
PSP_NS   = 0x00000000
NS HardFault Status Reg = 0x00000000
SCB Configurable Fault Status Reg = 0x00000000
```

```
== Back Trace ==

msp=0x1003f9b8 psp=0x1002bf70
Main stack back trace:
top=0x1003fa00 lim=0x1003ea00
9b01b7d0 @ sp = 00000000
9b0465dd @ sp = 00000000
0001869b @ sp = 1003f9ec
9b005955 @ sp = 1003f9f4

Backtrace information may not correct! Use this command to get C source
level information:
arm-none-eabi-addr2line -e ELF_file -a -f 9b01b7d0 9b0465dd 0001869b
9b005955
```

1) Use '**cd**' command to direct to the path of '**application_is.axf**' which under /project/realtek_amebaz2_v0_example/GCC-RELEASE/application_is/Debug/bin.

2) Use installed '**arm-none-eabi-addr2line.exe**' to get the back trace.
$ **/tools/arm-none-eabi-gcc/asdk/cygwin/newlib/bin/arm-none-eabi-addr2line.exe -e application_is.axf -a -f 9b01b7d0 9b0465dd 0001869b 9b005955**

The result will be:

```
/cygdrive/d/v7.1a/project/realtek_amebaz2_v0_example/GCC-
RELEASE/application_is/Debug/bin
$ /cygdrive/d/v7.1a/tools/arm-none-eabi-gcc/asdk/cygwin/newlib/bin/arm-
none-eabi-addr2line.exe -e application_is.axf -a -f 9b01b7d0 9b0465dd
0001869b 9b005955
0x9b01b7d0
_freertos_up_sema_from_isr
D:\v7.1a\component\os\freertos/freertos_service.c:139
0x9b0465dd
axi_bus_dma_Interrupt
D:\v7.1a\component\common\drivers\wlan\realtek\src\hci\axi/axi_intf.c:20
5
0x0001869b
??
??:0
0x9b005955
xPortStartScheduler
D:\v7.1a\component\os\freertos\freertos_v10.0.1\Source\portable\GCC\ARM_
RTL8710C/port.c:319
```

According to the result, user can trace the hard fault from ***xPortStartScheduler** -> **axi_bus_dma_Interrupt** -> **_freertos_up_sema_from_isr***. The hard fault comes from ***_freertos_up_sema_from_isr()*** that located at D:\v7.1a\component\os\freertos/freertos_service.c:139.

10.2 Fault Message Redirection

The default configuration of the fault log was set to output exclusively through the UART port, and this brings limitations as it does not allow saving logs to alternative storage mediums.
To enable more flexible storage, fault message redirection was introduced. Users can save fault logs to any storage medium.

The Fault Message Redirection functions are as follow:

- **fault_handler_override()** is used to redeclare fault handler to utilize RAM code for fault triggering. It requires two callback functions, fault_log() and bt_log() to handle the logs obtained.
- **fault_log()** is the callback functions to handle fault event logs, which includes register and stack memory dump logs.
- **bt_log()** is the callback functions to handle stack backtrace logs.

The example code under **“/project/realtek_amebaz2_v0_example/inc/main_faultlog.c”** demonstrates how fault logs are saved into flash memory.

```
int main(void)
{
        /* Read last fault data and redeclare fault handler to use ram code */
        read_last_fault_log();
        fault_handler_override(fault_log, bt_log);

        /* Initialize log uart and at command service */
        console_init();

        ...
}
```

When a fault occurs, the log is saved to flash memory via fault_log() and bt_log().
After the system restarts, the function **read_last_fault_log()** is called to retrieve the latest fault log from flash memory.

For detailed implementation guidance, users should refer to the main_faultlog.c file, which serves as a practical example of how to integrate these logging capabilities into their projects.