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
0.1	2018/11/27	Initial draft
0.2	2019/01/09	Add OTA part
0.3	2019/02/25	Add EVB V2.0 build and image tool environment
0.4	2019/02/28	Add How to enable secure boot & boot time
0.5	2019/03/01	Add DEV_2V0 board user guide
0.6	2019/03/06	Add trust zone project
0.7	2019/03/12	Update OTA implementation method
0.8	2019/03/27	Update TrustZone layout
0.9	2019/04/02	Add How to generate flash image combines both firmware1 and firmware2
1.0	2019/04/22	Add SDK Build Environment Setup, GCC Environment
1.1	2019/04/26	Add Bluetooth section
1.2	2019/05/14	Add Bluetooth 128-bit UUID configuration example
1.3	2019/05/15	Update IAR trust zone project; Add trust zone project instruction under GCC environment; Update trust zone layout; Add troubleshooting chapter.
1.4	2019/05/17	Update OTA FW update behavior
1.5	2019/05/22	Must use FT232 UART adapter to download code
1.6	2019/06/03	Update TrustZone
1.7	2019/06/03	Revise some explanations
1.8	2019/06/04	Update instructions to build and flash in Ubuntu/Linux
1.9	2019/08/02	Add pyOCD/DAPLink
1.10	2019/08/12	Add SDK Architecture description
1.11	2019/08/15	Refine pyOCD/DAPLink
1.12	2019/09/27	Add note of WiFi BT coexistence
1.13	2019/10/02	Improve visuality and explanation of document
1.14	2019/10/11	Add GCC secure boot execution
1.15	2019/11/11	Add Power Save and Efuse Chapters; Improve descriptions and content details; Fix the chapter quotation mismatch
1.16	2020/01/08	Add trust zone project execution of secure boot and OTA
1.17	2020/02/14	Add openOCD for Windows/Cygwin
1.18	2020/02/17	Add OpenOCD for Ubuntu
1.19	2020/03/26	Remove “module features” which user should refer to datasheet, because it may not be update to date in this application note.

Revision	Release Date	Summary
1.20	2020/04/08	Add BT Transmit Power
1.21	2020/04/21	Add description in Force Old OTA section
1.22	2020/05/06	Update write jtag key api description
1.23	2020/05/22	Remove “UserAddr” from “Partition Record” in flash layout, because partition table doesn’t record the address of User Data
1.24	2020/06/05	Add checksum description for OTA routine
1.25	2020/06/09	Correct the description of serial number
1.26	2020/06/10	Make serial number valid from 0x0 to 0xFFFFFFFF with new elf2bin tool
1.27	2020/09/09	Update OpenOCD section
1.28	2020/09/15	Update how to use PSRAM in GCC project
1.29	2020/10/20	Add How to Debug in Secure Boot in IAR IDE
1.30	2020/11/13	Update BT Examples; Modify Memory Usage; Add BT Default MAC Address section
1.31	2020/12/09	Update OpenOCD setup on Windows OS
1.32	2021/03/31	Add Cygwin version requirements for OpenOCD on Windows OS
1.33	2021/04/01	Update OpenOCD section
1.34	2021/10/14	Update system data layout
1.35	2022/06/30	Update pyOCD setup on Linux and OpenOCD setup on Linux sections
1.36	2022/07/06	Update pyOCD setup on Windows OS
1.37	2022/11/11	Update Efuse section
1.38	2023/09/14	Update eFuse section
1.39	2023/12/19	Add fault message redirection section
1.40	2024/03/11	Update power save chapter

Table of Contents

List of Figures

List of Table

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.

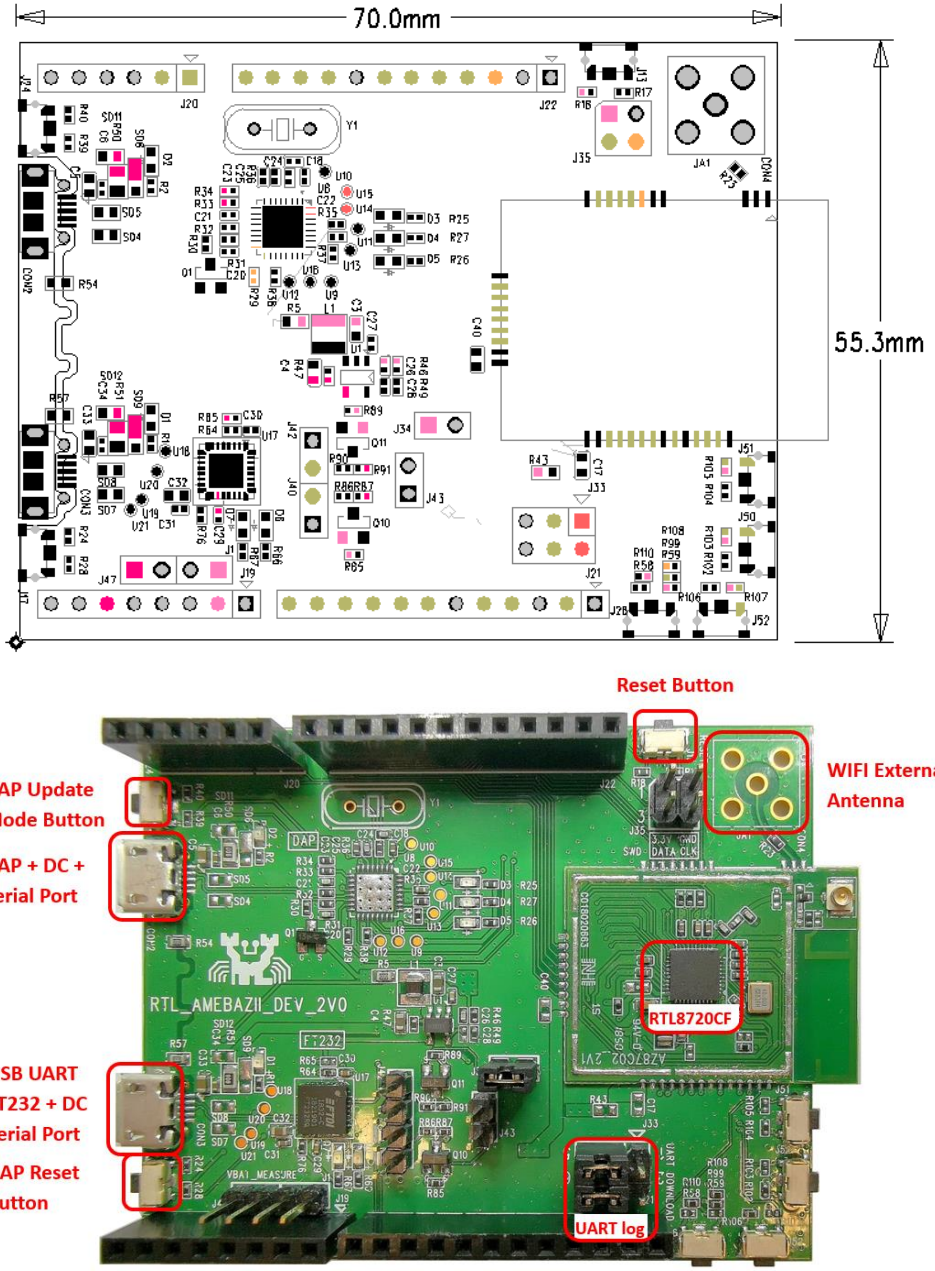


Figure 1-2 Ameba-ZII 2V0 Dev Board PCB Layout

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

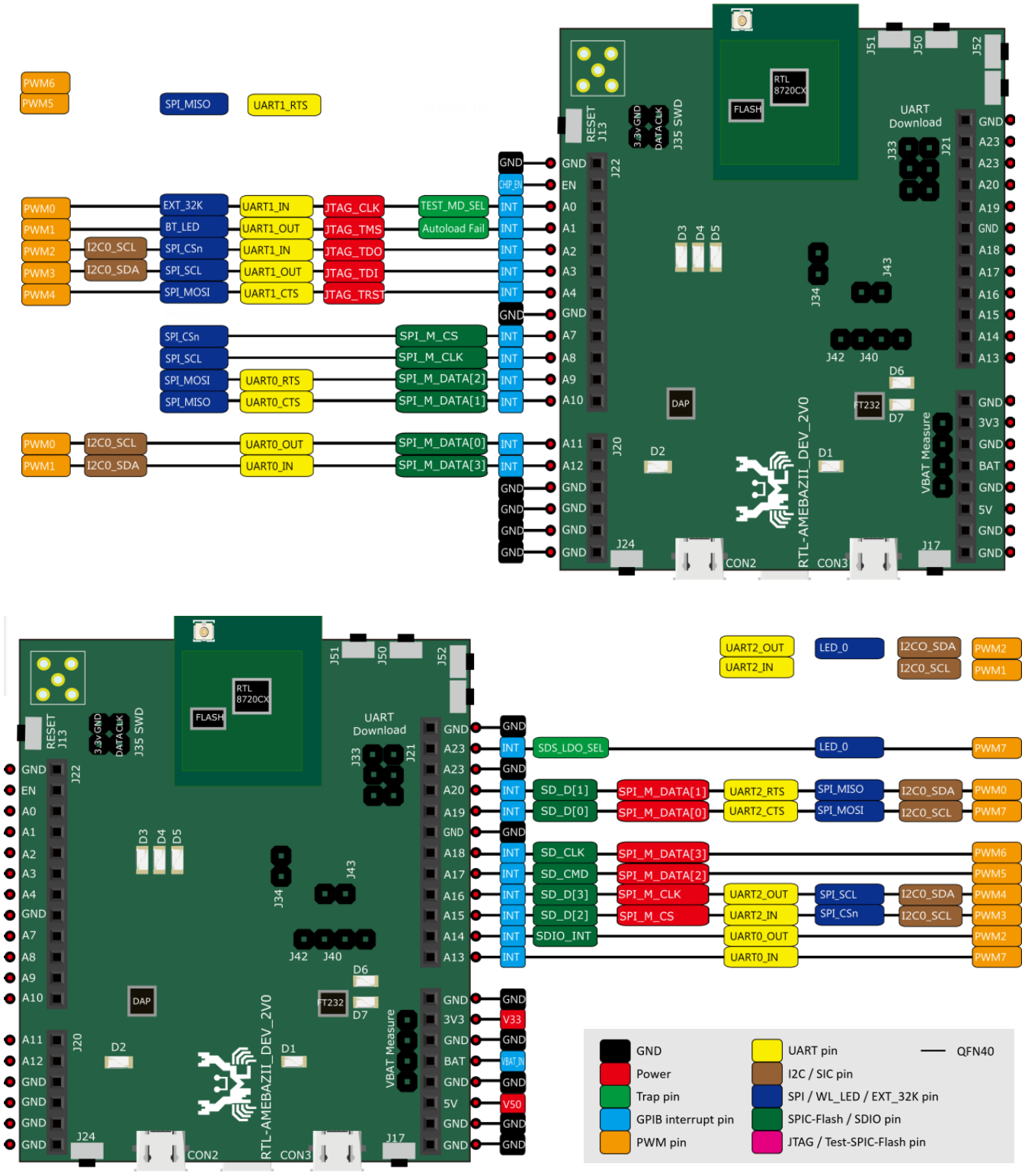
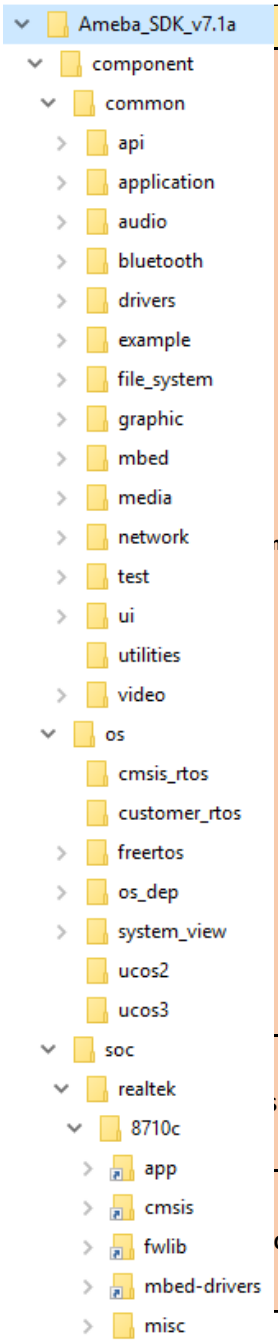


Figure 1-3 Pin Out Reference for DEV_2V0

2 SDK Architecture

Ameba-ZII SDK includes four folders: '**component**', '**doc**', '**project**' and '**tools**'.
The architecture of SDK and descriptions of main folders are shown as below.

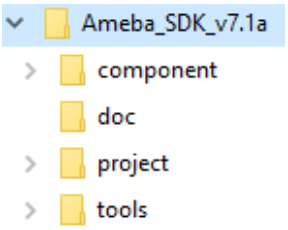
2.1 Component



Sub-folders		Description
non	**Api**	• AT command
		• Platform_stdlib.h: standard library header
		• Wi-Fi driver interface
		• Wi-Fi promisc mode interface
		• Wi-Fi simple configuration
	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
	Media	• multi-media framework
	Network	• coap
		• dhcp
		• http
		• lwip
		• mDNS
		• rtsp
		• sntp
		• ssl (MBEDTLS)
		• tftp
		• websocket
	Utilities	• cJSON
		• http_client
		• ssl_client
		• tcpecho
		• udpecho
		• webserver/xml
	freertos	FreeRTOS source code
	os_dep	osdep_service.c: Realtek encapsulating interface for FreeRTOS
		osdep_service.h: Realtek encapsulating interface header
C	**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



Folder	Sub-folder	Description
DOC	**AN0004**	Realtek low power Wi-Fi MP user guide
	AN0011	Realtek WLAN simple configuration
	AN0012	Realtek secure socket layer (SSL)
	AN0075	Realtek Ameba-all at command v2.0
	AN0500	Realtek Ameba-ZII application note
Project	**realtek_amebaz2_v0_example**	IAR/GCC project entry for AmebaZII
	$/ EWARM-RELEASE	IAR projects
	$/~/Project_is.eww	IAR project for ignore secure (is, non TrustZone) configuration
	$/~/Project_tz.eww	IAR project for TrustZone(tz) configuration
	$/GCC-RELEASE	GCC projects
	$/~/application.is.mk	GCC project for Ignore Secure (is, non TrustZone) configuration
	$/~/application.tz.mk	GCC project for TrustZone (tz) configuration
	$/example_sources	Examples for peripherals
Tools	**Bluetooth**	APP for BT config
	DownloadServer	OTA download TCP server
	DownloadServer (HTTP)	OTA download HTTP server
	pyOCD	GDB server for DAPLink

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 **IAR** IDE will be used for Windows OS and **GCC** IDE will be used for both Windows OS and Linux OS.

Note: For Windows OS, we use **Windows 7 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** and **CMSIS-DAP** 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.

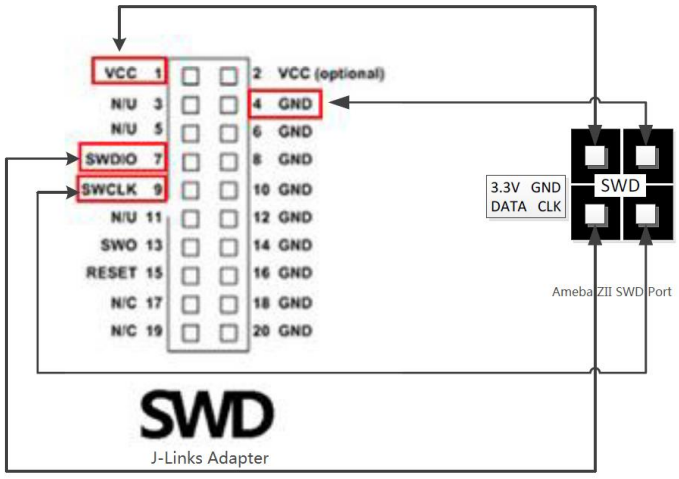


Figure 3-1 Connection between J-Link Adapterand Ameba-ZII SWD connector

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: To support TrustZone feature, it's better to download the **latest version** of J-Link Software. Version 6.40 is used to prepare this document.

The process of 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).

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'.

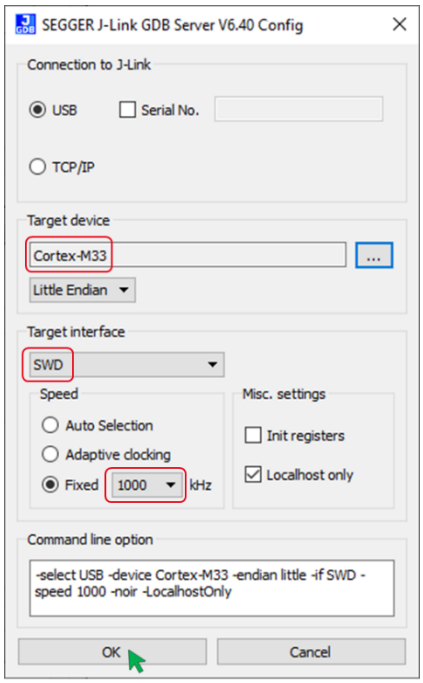


Figure 3-3 J-Link GDB server UI under Windows

4. Check if the below information is shown properly.

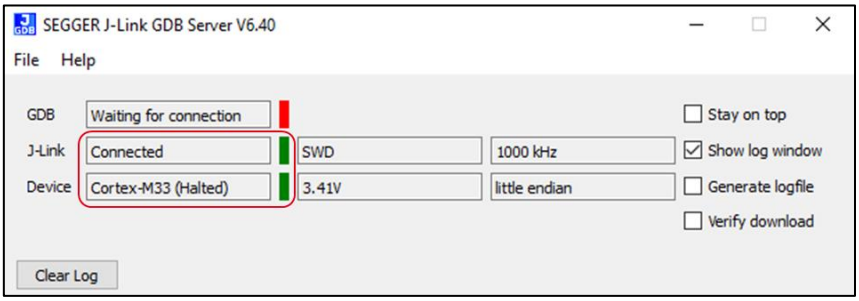


Figure 3-4 J-Link GDB server connect under Windows

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"

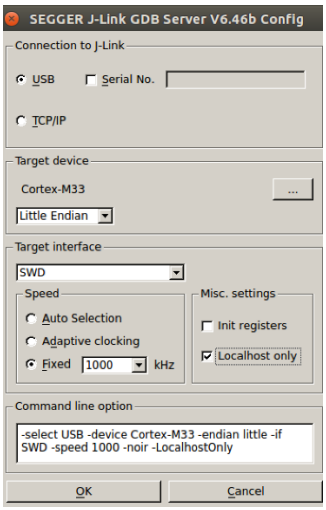
3.2.1.3.2 Steps to Initiate JLinkGDBServer

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.

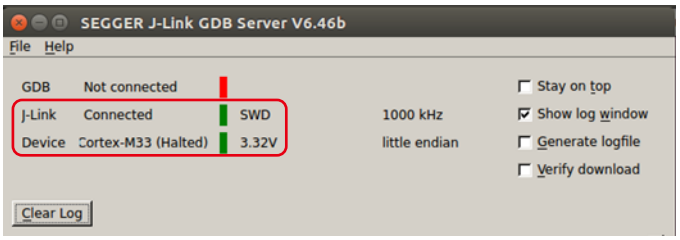
1. Open a new terminal along with the terminal where the SDK is open.

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.



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.



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.

3.2.2 pyOCD/DAPLink

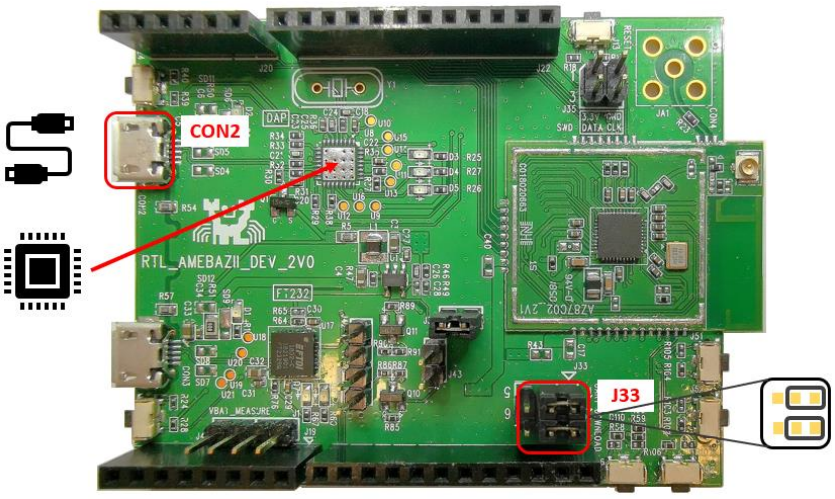
pyOCD (python On-Chip Debugger) provides open source Python library for programming and debugging Arm Cortex-M microcontrollers. pyOCD uses DAPLink as software project which enables programming and debugging application software on running on Arm Cortex CPUs. DAPLink enables developers with drag-and-drop programming and CMSIS-DAP (Cortex Microcontroller Software Interface Standard - Debug Access Port) based debugging. It runs on a secondary MCU that is attached to the JTAG port of the application MCU.

DAPLink can be downloaded from: https://github.com/ARMmbed/DAPLink

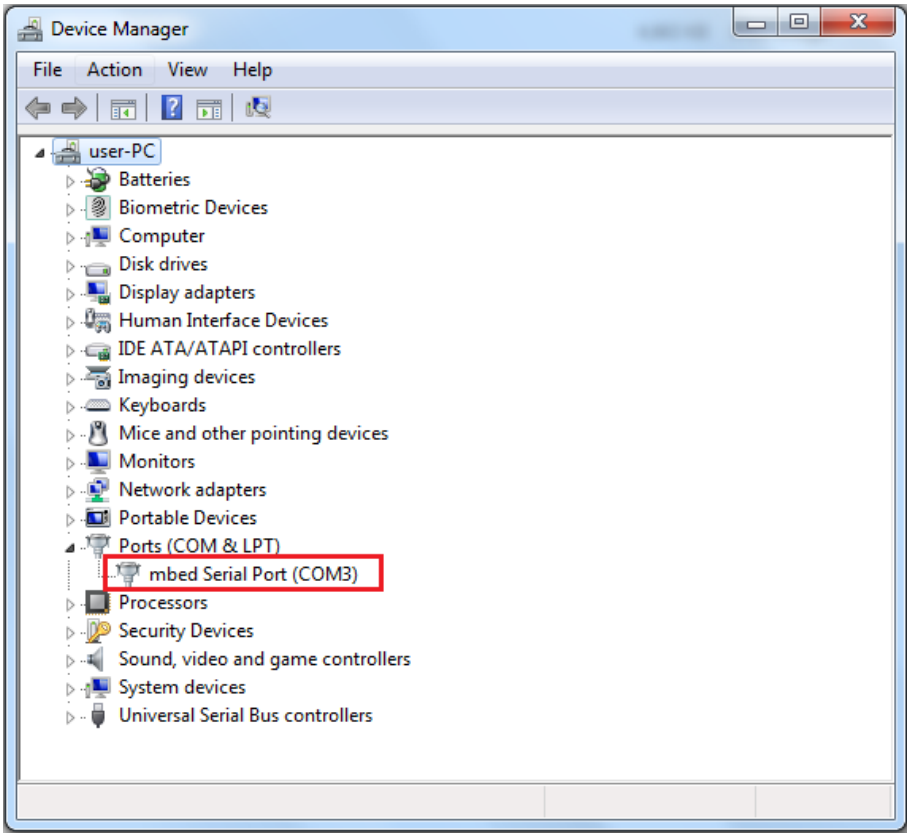
Please install modified version of pyOCD in Ameba-ZII SDK. It can be found in ‘*tools/pyOCD*’ of the SDK. **Official version cannot detect the amebaz2 board.**
Also, official pyOCD source code can be downloaded from: https://github.com/mbedmicro/pyOCD

3.2.2.1 Connection

To use DAPLink you need to connect Ameba-ZII to PC via **CON2** and connect jumpers at **J33** as shown**.**
Note: You need to check whether the **DAP Chip** is mounted on the board.



The DAPLink debug probe also provides a USB serial port which can be bridged through to a TTL UART on the target system. The USB Serial port will appear on a Windows machine as a COM port, or on a Linux machine as a /dev/tty interface and on Mac OS as a /dev/usbmodem. While Linux and Mac OS dont require any drivers, Windows version older than Windows 10 will require a serial port driver which can be found in http://os.mbed.com/media/downloads/drivers/mbedWinSerial_16466.exe.
If Serial to USB driver has been installed and the board is connected to PC, there should be mbed Serial Port shown in Device Manager.



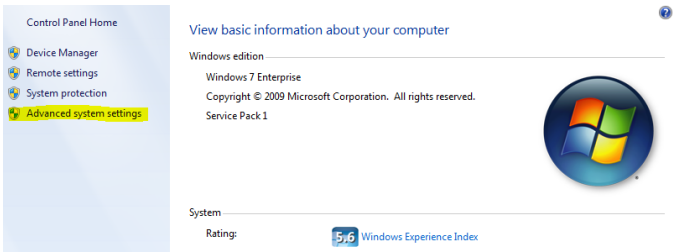
3.2.2.2 Setups on Windows OS

1) Install with Python 2.X

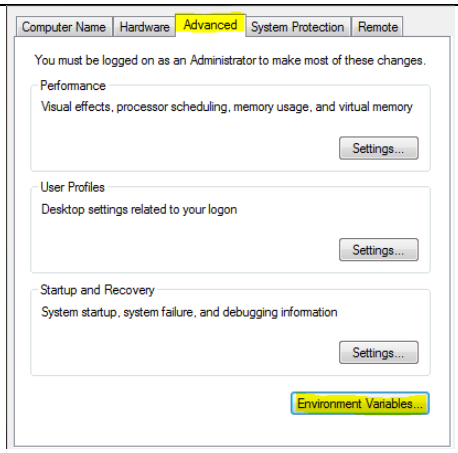
a. If Python 2.7 is not ready on your computer, please download and install **Python 2.7** from website: https://www.python.org/downloads/release/python-2716/

Note: Please choose **32-bit** version “Windows x86 MSI installer”.

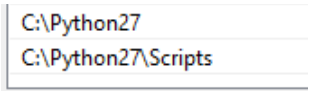
b. Setup **PATH** for Python 2.7:
In ‘Control Panel > System and Security > System’, click **Advanced system settings.**



Choose ‘**Advanced’** tab and click ‘**Environment Variables**’.



Click Edit Path variable in **User variables** or **System variables**, then adding **Python 2.7 install path** and **scripts path** to Path variable.
For example, adding *C:\Python27* and *C:\Python27\Scripts* to Path variable.



c. Upgrade **pip** and install necessary packages in command window by following commands (Note: if there are multiply versions of python installed please specific its' path or add it to system variables when using cmd, e.g. > python2.7 -m …)
> python -m pip install --upgrade pip
> pip install pywinusb enum34 cmsis-pack-manager colorama intelhex intervaltree prettytable pyelftools pyyaml six

d. Install **pyOCD** in SDK folder '*tools/pyOCD/pyocd-0.21.1.dev35+dirty.win32.exe*'

e. Install package to offer easy USB device communication in Python
> pip install pyusb

f. Attached HDK (DAPLink enabled) to computer then using command
> pyocd-gdbserver -p 2331
If successfully detected, the following message will be shown:

```
$ pyocd-gdbserver -p 2331
0000398:WARNING:gdb_server:pyocd-gdbserver is deprecated; please use the new combined pyocd tool.
0000642:WARNING:mbed_board:Board ID 0857 is not recognized, using generic cortex_m target.
0000643:INFO:board:Target type is cortex_m
0000643:WARNING:board:Generic 'cortex_m' target type is selected; is this intentional? You will be able to debug but not program f
lash. To set the target type use the '--target' argument or 'target_override' option. Use 'pyocd list --targets' to see available
targets types.
0000676:INFO:dap:DP IDR = 0x6ba02477 (v2 rev6)
0000694:INFO:ap:AP#0 IDR = 0x84770001 (AHB-AP var0 rev8)
0000707:INFO:ap:AP#1 IDR = 0x54770002 (APB-AP var0 rev5)
0000727:INFO:rom_table:AP#0 ROM table #0 @ 0xe00ff000 (designer=879 part=030)
0000738:INFO:rom_table:[0]<e000e000:SCS-M33 class=9 designer=879 part=033 devtype=00 archid=2a04 devid=0:0:0>
0000746:INFO:rom_table:[1]<e0001000:DWT class=9 designer=879 part=035 devtype=00 archid=1a02 devid=0:0:0>
0000753:INFO:rom_table:[2]<e0002000:BPU class=9 designer=879 part=034 devtype=00 archid=1a03 devid=0:0:0>
0000763:WARNING:rom_table:Invalid coresight component, cidr=0x0
0000763:WARNING:rom_table:Warning: ROM table @ 0x80000000 has unexpected CIDR component class (0x0)
0000767:INFO:cortex_m_v8m:CPU implementer is Realtek.
0000770:INFO:cortex_m_v8m:CPU core #0 is Cortex-M33 r1p0 (security ext present)
0000780:INFO:dwt:1 hardware watchpoints
0000784:INFO:fpb:2 hardware breakpoints, 0 literal comparators
0000796:INFO:server:Semihost server started on port 4444
0000797:INFO:gdbserver:GDB server started on port 2331
```

2) Install with Python 3.X

a. If Python 3.10 is not ready on your computer, please download and install **Python 3.10** from website or Microsoft store:

https://www.python.org/downloads/

Note: Please choose **64-bit** version "Windows installer (64 bit)" as new version of cmsis_pack_manager don't provide 32 bit libs for now.

b. Setup **PATH** for Python 3.10:
In 'Control Panel > System and Security > System', click **Advanced system settings.**
Choose '**Advanced'** tab and click '**Environment Variables**'.

Click Edit Path variable in **User variables** or **System variables**, then adding **Python 3.10 install path** and **scripts path** to Path variable.
For example, adding *C:\Python310* and *C:\Python310\Scripts* to Path variable.

c. Upgrade **pip** and install necessary packages in command window by following commands (Note: if there are multiply versions of python installed please specific its' path or add it to system variables when using cmd, e.g. > path\to\python3.10.exe -m … or > python3.10 -m …)
> python -m pip install --upgrade pip
> pip install pywinusb enum34 cmsis-pack-manager colorama intelhex intervaltree prettytable pyelftools pyyaml six

d. Install **pyOCD** in SDK folder '*tools/pyOCD/pyocd-0.34.X.win-amd64.msi*

e. Install package to offer easy USB device communication in Python
> pip install pyusb

f. Attached HDK (DAPLink enabled) to computer then using command
> python -m pyocd gdbserver -p 2331
or
> pyocd gdbserver -p 2331
If successfully detected, the same message will be shown.

3.2.2.3 Setups on Linux OS

1) Install through python2.X easy_install

a. Similar to windows, **Python 2.7** is required. If there is no Python 2.7 existed in Linux, please install Python 2.7.
If your Linux distribution is **Ubuntu**, please using following command:
> sudo apt install python2.7

b. Then install and upgrade pip and necessary packets
> sudo apt install python-pip
> sudo python -m pip install --upgrade pip
> sudo pip install pyusb enum34 cmsis-pack-manager colorama intelhex intervaltree prettytable pyelftools pyyaml six

c. Install pyOCD in SDK folder *tools/pyOCD/pyocd-0.21.1.dev36-py2.7.egg*
> sudo python -m easy_install tools/pyOCD/pyocd-0.21.1.dev36-py2.7.egg

d. Attached HDK (DAPLink enabled) to computer then using command.
May need using sudo to have the hardware access privilege
> sudo pyocd-gdbserver -p 2331

e. If successfully detected, the following message will be shown:

```
$ pyocd-gdbserver -p 2331
0000398:WARNING:gdb_server:pyocd-gdbserver is deprecated; please use the new combined pyocd tool.
0000642:WARNING:mbed_board:Board ID 0857 is not recognized, using generic cortex_m target.
0000643:INFO:board:Target type is cortex_m
0000643:WARNING:board:Generic 'cortex_m' target type is selected; is this intentional? You will be able to debug but not program f
lash. To set the target type use the '--target' argument or 'target_override' option. Use 'pyocd list --targets' to see available
targets types.
0000676:INFO:dap:DP IDR = 0x6ba02477 (v2 rev6)
0000694:INFO:ap:AP#0 IDR = 0x84770001 (AHB-AP var0 rev8)
0000707:INFO:ap:AP#1 IDR = 0x54770002 (APB-AP var0 rev5)
0000727:INFO:rom_table:AP#0 ROM table #0 @ 0xe00ff000 (designer=879 part=030)
0000738:INFO:rom_table:[0]<e000e000:SCS-M33 class=9 designer=879 part=033 devtype=00 archid=2a04 devid=0:0:0>
0000746:INFO:rom_table:[1]<e0001000:DWT class=9 designer=879 part=035 devtype=00 archid=1a02 devid=0:0:0>
0000753:INFO:rom_table:[2]<e0002000:BPU class=9 designer=879 part=034 devtype=00 archid=1a03 devid=0:0:0>
0000763:WARNING:rom_table:Invalid coresight component, cidr=0x0
0000763:WARNING:rom_table:Warning: ROM table @ 0x80000000 has unexpected CIDR component class (0x0)
0000767:INFO:cortex_m_v8m:CPU implementer is Realtek.
0000770:INFO:cortex_m_v8m:CPU core #0 is Cortex-M33 r1p0 (security ext present)
0000780:INFO:dwt:1 hardware watchpoints
0000784:INFO:fpb:2 hardware breakpoints, 0 literal comparators
0000796:INFO:server:Semihost server started on port 4444
0000797:INFO:gdbserver:GDB server started on port 2331
```

2) Install through python3.X pip

a. As easy_install is now deprecated from higher version of python, there is alternative way. If your Linux distribution is Ubuntu, please using following command:
> sudo apt install python3
Then install and upgrade pip and necessary packets
> sudo apt install python3-pip
> sudo python3 -m pip install --upgrade pip

b. Install pyocd through pip local files:
> python3 -mpip install xxx/pyocd-0.34.X.tar.gz

c. **Update udev rules to change access permission**
please follow the guides from https://github.com/pyocd/pyOCD/tree/main/udev

d. Attached HDK (DAPLink enabled) to computer then using command.
> python3 -mpyocd gdbserver -p 2331

e. If successfully detected, the same message above will be shown.

3.2.3 OpenOCD/DAPLink

The Open On-Chip Debugger (OpenOCD) aims to provide debugging, in-system programming and boundary-scan testing for embedded target devices.

3.2.3.1 Connection

It's the same as 3.2.2.1 .

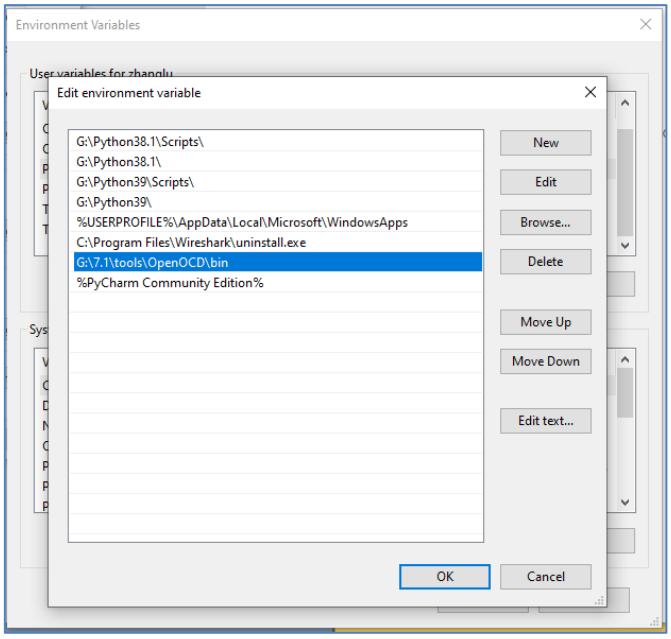
Note: After downloading the image, user must close OpenOCD Debugger, then unplug and plug CON2 to reset the DAPLink before you reset the Ameba-ZII board, otherwise you will see below log when you reset the board.

```
== Rtl8710c IoT Platform ==
Chip VID: 5, Ver: 1
ROM Version: v2.1
Test Mode: boot_cfg1=0x0
Download Image over UART2[tx=16,rx=15] baud=115200
```

3.2.3.2 Setups on Windows OS

For Windows OS, the OpenOCD executive and dependencies are integrated into the SDK, under the *tools/OpenOCD* folder. There is no need to separately download the OpenOCD software.

i. User needs to add the ***tools\OpenOCD\bin*** path in the System Environment Variables Path (Control Panel -> System and Security -> System -> Advanced System Settings -> Advanced tab -> Environment Variables -> Path).

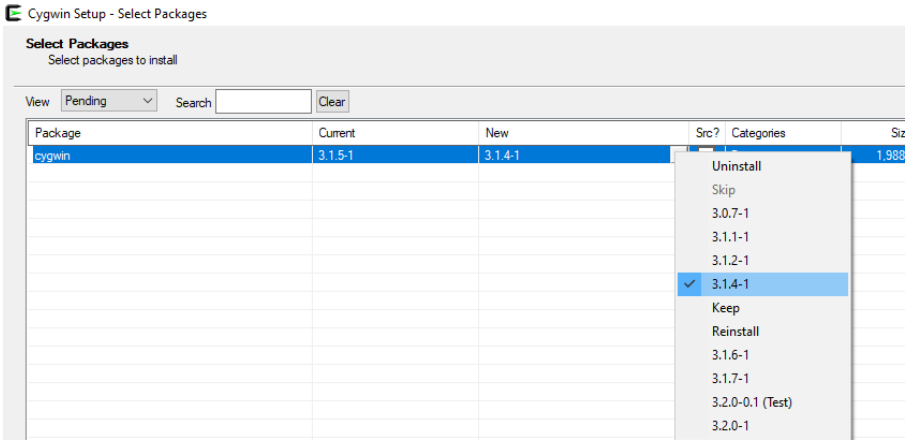


ii. Open Cygwin terminal (32-bit), directed to ***GCC-RELEASE/***

iii. Execute ***./run_openocd.sh*** to open OpenOCD server

```
ample/GCC-RELEASE
$ ./run_openocd.sh
make[1]: Entering directory '/cygdrive/d/7.1forCommit/sdk/project/realtek_amebaz
2_v0_example/GCC-RELEASE'
---------------
Setup openocd
---------------
cp -p ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_debug_openoc
d.txt ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_debug.txt
cp -p ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_flashloader_
openocd.txt ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_flashl
oader.txt
make[1]: Leaving directory '/cygdrive/d/7.1forCommit/sdk/project/realtek_amebaz2
_v0_example/GCC-RELEASE'
Try to search windows process
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "swd". To override use '
transport select <transport>'.
adapter speed: 500 kHz
adapter_nsrst_delay: 200
cortex_m3 reset_config sysresetreq
Info : CMSIS-DAP: SWD  Supported
Info : CMSIS-DAP: Interface Initialised (SWD)
Info : CMSIS-DAP: FW Version = 1.0
Info : SWCLK/TCK = 1 SWDIO/TMS = 1 TDI = 0 TDO = 0 nTRST = 0 nRESET = 1
Info : CMSIS-DAP: Interface ready
Info : clock speed 500 kHz
Info : SWD DPIDR 0x6ba02477
Info : security extensions detected
Info : rtl8710c.cpu: hardware has 2 breakpoints, 1 watchpoints
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00000a00 msp: 0x1003fa00
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* rtl8710c.cpu       cortex_m3 little rtl8710c.cpu       halted
```

Note: Errors such as “Error: Can't find openocd.cfg" may be encountered when running ./run_openocd.sh in Cygwin if the version of the Cygwin package installed is not 3.1.4-1. OpenOCD requires the cygwin1.dll to be from this version in order to run *./run_openocd.sh* properly. If it is not on version 3.1.4-1, run the Cygwin setup file again and install version 3.1.4-1 of the Cygwin package when selecting the packages to install. Alternatively, the error can also be resolved by replacing the cygwin1.dll located in *cygwin/bin* with the version included in the *tools/OpenOCD/bin* folder.



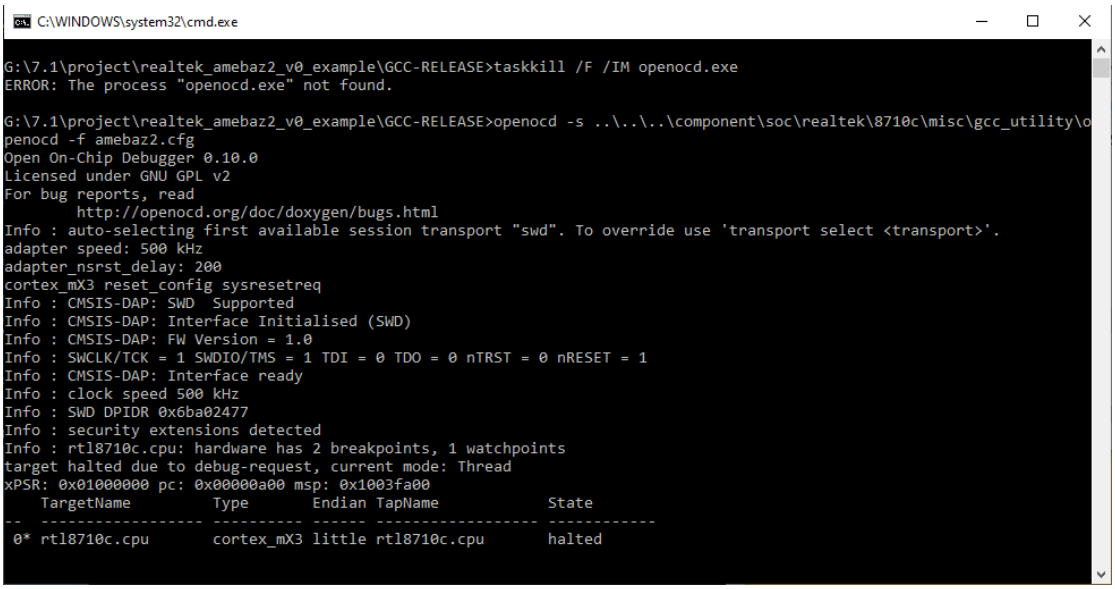
iv. Keep this OpenOCD server unclosed and open another Cygwin terminal for **debugging** or **downloading** via GDB

Alternatively, you can execute **run_openocd.bat** batch file to open OpenOCD server. However, when using it for the first time, you need to manually setup the gdb server to openocd in Cygwin terminal before running **run_openocd.bat**.

> make setup GDB_SERVER=openocd

```
$ make setup GDB_SERVER=openocd
make[1]: Entering directory '/cygdrive/d/Projects/ameba/sdk/v7.1a/project/realtek_amebaz2_v0_example/GCC-RELEASE'
----------------
Setup openocd
----------------
cp -p ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_debug_openocd.txt ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_debug.txt
cp -p ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_flashloader_openocd.txt ../../../component/soc/realtek/8710c/misc/gcc_utility/rtl_gdb_flashloader.txt
make[1]: Leaving directory '/cygdrive/d/Projects/ameba/sdk/v7.1a/project/realtek_amebaz2_v0_example/GCC-RELEASE'
```

> run_openocd.bat



3.2.3.3 Setups on Linux OS

Here is the setup guide for compiling and installing the latest version OpenOCD from github source code. (below steps are verified under 16.04.1-Ubuntu and git://git.code.sf.net/p/openocd/code# 0a11537b3220749107f4ec78c76236ac8c9339d1.)

Firstly, we assume that you have access to root privileges, and you need to install some required packages. The packages include git, gcc build environment, usb-related libraries:

$ sudo apt-get install git build-essential g++ autotools-dev make libtool pkg-config autoconf automake texinfo libudev-dev libusb-1.0-0-dev libfox-1.6-dev

Secondly, we need to install HIDAPI library before OpenOCD. HIDAPI is a library which allows applications to interface with USB devices. You can refer http://www.signal11.us/oss/hidapi/ for more information about it. To install it, we are going to clone the git project and compile it:

$ cd ~/
$ git clone https://github.com/signal11/hidapi.git or https://github.com/libusb/hidapi
$ cd hidapi/
$ ./bootstrap
$ ./configure
$ make
$ sudo make install

Note: Errors such as “configure.ac:16: error: AC_CONFIG_MACRO_DIR can only be used once" caused by hidapi code, use https://github.com/libusb/hidapi to clone the latest version to avoid this issue.

After typing above commands, the HIDAPI should be installed. But we still need to add the location of the hid library into system PATH variable. For Ubuntu, please use an editor to open ~/.profile file:

$ vim ~/.profile

And at the bottom of .profile, please add the following line:

PATH="$HOME/bin:/usr/local/lib:$PATH"

```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

PATH="$HOME/bin:/usr/local/lib:$PATH"
```

To reload the PATH variable, you can use below command:

$ source ~/.profile

And you can use *echo* command to check the updated content of PATH variable:

$ echo $PATH

```
realtek@realtek-VirtualBox:~$ echo $PATH
/home/realtek/bin:/usr/local/lib:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in:/sbin:/bin:/usr/games
```

We also need to update our system shared library cache by following command:

$ sudo ldconfig

Finally, we are going to compile and install OpenOCD library after we installed HIDAPI:

$ cd ~/

$ git clone git://git.code.sf.net/p/openocd/code openocd-code

$ cd openocd-code/

$ ./bootstrap

Since we are using OpenOCD/CMSIS-DAP, we only enable its corresponding configuration:

$ ./configure --enable-cmsis-dap --disable-gccwarnings

$ make

$ sudo make install

At this point, we have installed the newest OpenOCD library and the OpenOCD/CMSIS-DAP connection should be able to work. You can use *-v* command to check its version:

$ openocd –v

```
realtek123@ubuntu:~/sdk-ameba-v7.1d_v2/project/realtek_amebaz2_v0_example/GCC-RELEASE$ openocd -v
Open On-Chip Debugger 0.10.0+dev-01060-g0a11537 (2020-02-17-10:27)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
```

Here is another setup guide installing the latest version OpenOCD from official package management. (below steps are verified under 22.04.1-Ubuntu and openocd0.11.0.)

You can install openocd by:

$ sudo apt-get install openocd

Now if above steps are done successfully, you can run below command to start OpenOCD Debugger

$ sudo ./run_openocd.sh

If everything is fine, you will see below logs

```
realtek123@ubuntu:~/sdk-ameba-v7.1d_v2/project/realtek_amebaz2_v0_example/GCC-RELEASE$ sudo ./run_openocd.sh
Open On-Chip Debugger 0.10.0+dev-01060-g0a11537 (2020-02-17-10:27)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "swd". To override use 'transport select <transport>'.
DEPRECATED! use 'adapter speed' not 'adapter_khz'
DEPRECATED! use 'adapter srst delay' not 'adapter_nsrst_delay'
cortex_m reset_config sysresetreq

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : CMSIS-DAP: SWD  Supported
Info : CMSIS-DAP: FW Version = 1.0
Info : CMSIS-DAP: Interface Initialised (SWD)
Info : SWCLK/TCK = 1 SWDIO/TMS = 1 TDI = 0 TDO = 0 nTRST = 0 nRESET = 1
Info : CMSIS-DAP: Interface ready
Info : clock speed 500 kHz
Info : SWD DPIDR 0x6ba02477
Info : rtl8710c.cpu: hardware has 2 breakpoints, 1 watchpoints
Info : rtl8710c.cpu: external reset detected
Info : Listening on port 3333 for gdb connections
```

But if you get below logs, please check your board connection.

```
realtek123@ubuntu:~/sdk-ameba-v7.1d_v2/project/realtek_amebaz2_v0_example/GCC-RELEASE$ sudo ./run_openocd.sh
[sudo] password for realtek123:
Open On-Chip Debugger 0.10.0+dev-01060-g0a11537 (2020-02-17-10:27)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "swd". To override use 'transport select <transport>'.
DEPRECATED! use 'adapter speed' not 'adapter_khz'
DEPRECATED! use 'adapter srst delay' not 'adapter_nsrst_delay'
cortex_m reset_config sysresetreq

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Error: unable to find CMSIS-DAP device
```

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):**

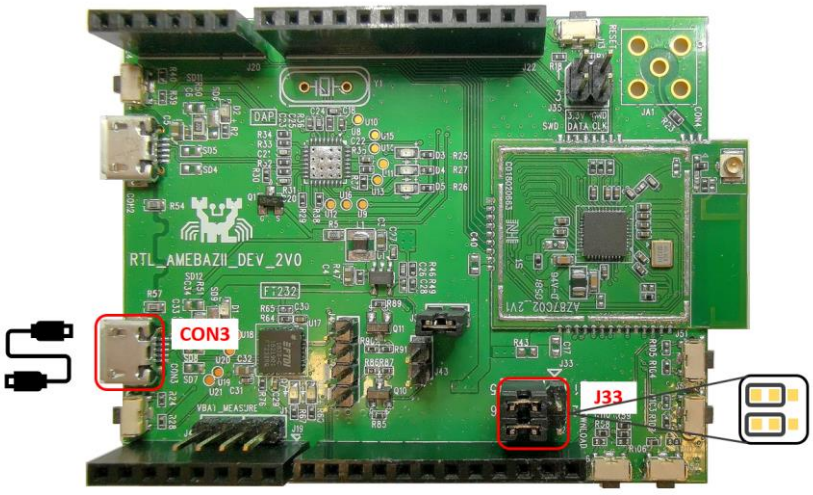


Figure 3-2 Log UART via FT232 on EVB V2.0

2) Connection to log UART via **DAP (CON2):**

Note: You need to check whether the **DAP Chip** is mounted on the board.

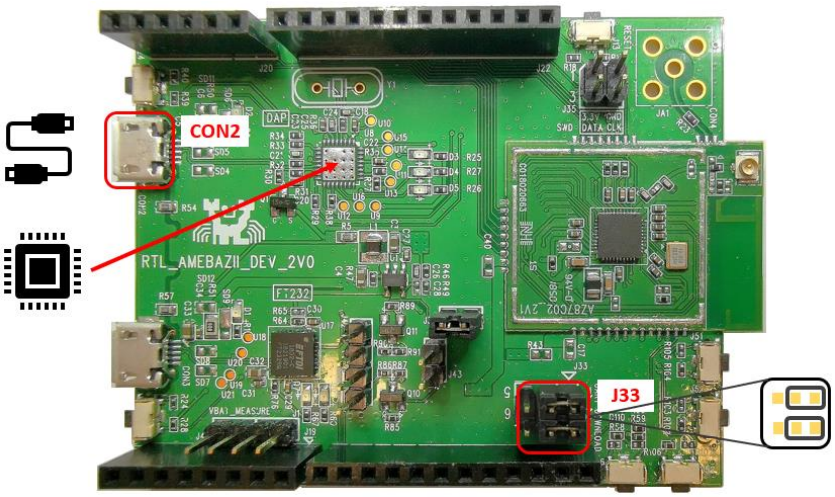


Figure 3-7 Log UART via DAP on EVB V2.0

3.4 IAR IDE Setup on Windows

The IAR IDE (integrated development environment) only supports Windows OS, this section is applicable for **Windows OS only**.

3.4.1 Install IAR IDE

IAR IDE provides the toolchain for Ameba-ZII. It allows users to write programs, compile and upload them to your board. Also, it supports step-by-step debug function.
User can visit the official website of **IAR Embedded Workbench** and install the IDE by following its instructions.

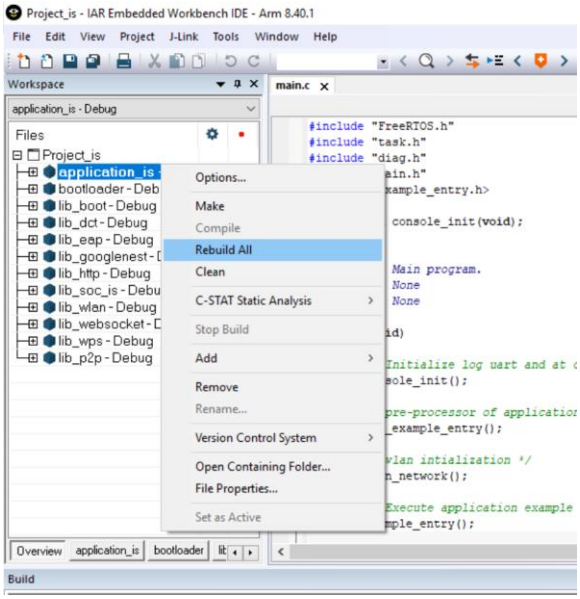
Note: Please use IAR version **8.30** or above.

3.4.2 Build Non-Trust Zone Project

Currently users can use **ignore secure mode**. '**project_is.eww**' ('is' means ignore secure) is the project without **TrustZone** configuration. This project is easier to develop and suit for first-time developer.

3.4.2.1 Compilation

1) Open SDK/project/realtek_amebaz2_v0_example/EWARM-RELEASE/Project_is.eww.
2) Confirm 'application_is' in Work Space, right click 'application_is' and choose "**Rebuild All**" to compile.



3) Make sure there is no error after compile.

3.4.2.2 Generate Image Binary

After compile, the images **partition.bin**, **bootloader.bin**, **firmware_is.bin** and **flash_is.bin** can be seen in the EWARM-RELEASE\Debug\Exe.

1) **partition.bin** stores partition table, recording the address of Boot image and firmware image;
2) **bootloader.bin** is bootloader image;
3) **firmware_is.bin** is application image;
4) **flash_is.bin** links partition.bin, bootloader.bin and firmware_is.bin. 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**' (ignore secure) is generated without error, user can either

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

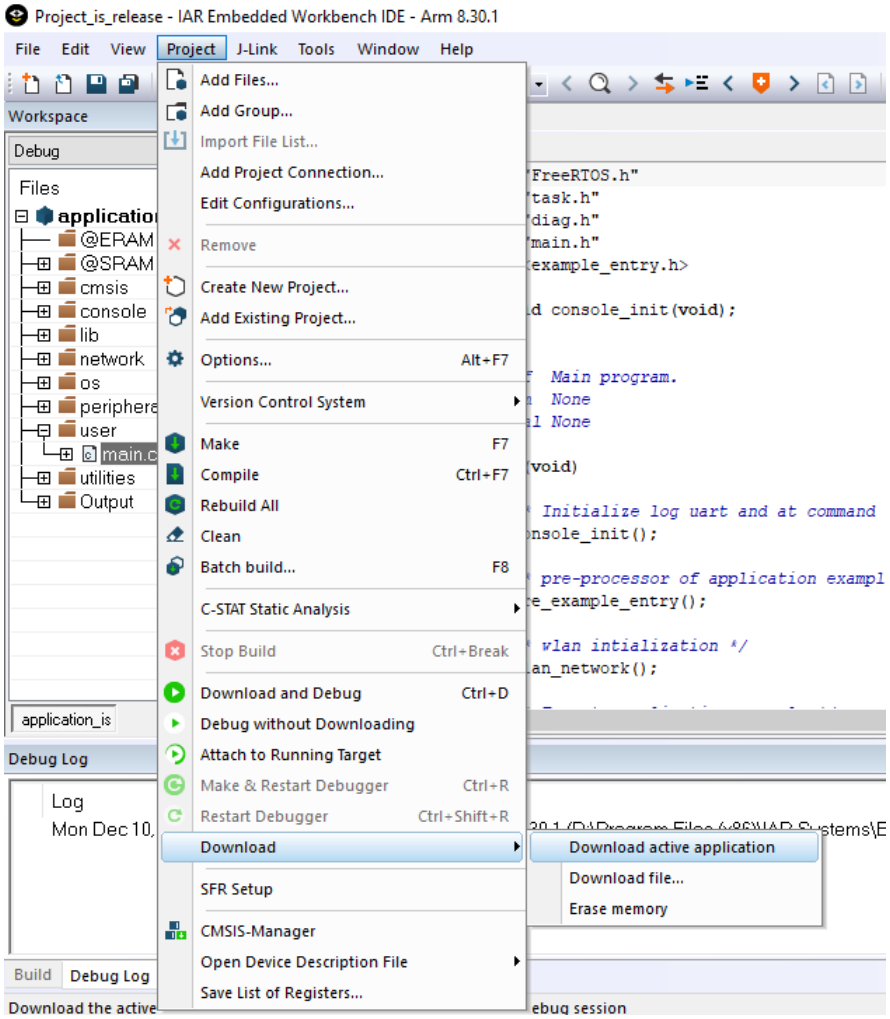


Figure 3-7 IAR download binary on flash

Note: Please '**make'** the project first when some code is modified before download the bin file on the board, otherwise the download will fail and below logs will be shown.

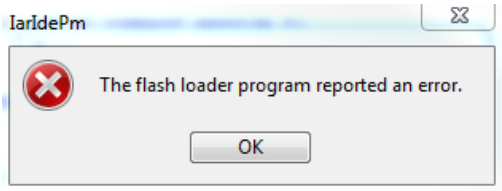


Figure 3-8 IAR download code on flash error message on IDE

Figure 3-9 IAR download code on flash error message on Log UART

2) Or using the PG tool for Ameba-ZII (Will not be shown here, please check chapter Image Tool for details).

3.4.3 Build Trust Zone Project

If the project is building in **trust zone mode**, '***project_tz.eww***' (*tz* means trust zone) is the project with trust zone configuration. The code can be decided as secure or not by putting the code to '***application_s***' or '***application_ns'***. Secure code can be executed by enabling **EXAMPLE_TRUST_ZONE** in **platform_opts.h**.

3.4.3.1 Compilation

1) Open SDK/project/realtek_amebaz2_v0_example/EWARM-RELEASE/Project_tz.eww.
2) Confirm '***application_ns'*** and '**application_s***'* are in Work Space.
3) Right click '***application_s'*** and click "**Rebuild All**" to compile '***application_s'*** first. If '***application_s'*** is compiled successfully, it will generate a file named '***application_s_import_lib.o'*** and the file will be put in "lib" folder of '***application_ns'***, shown in Figure 2-10.

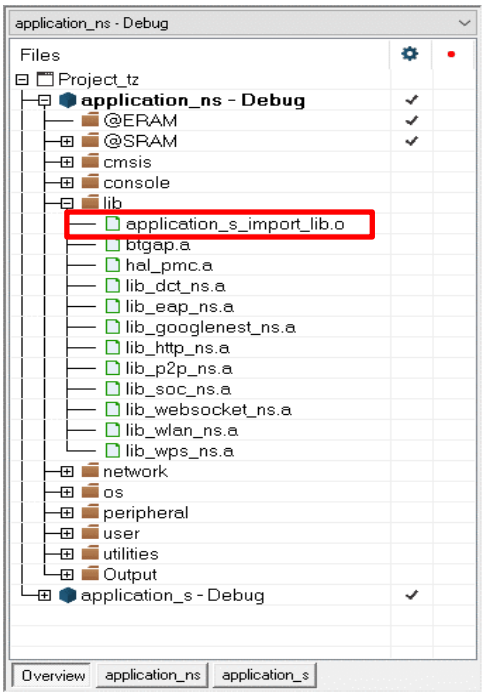


Figure 3-10 application_s compile result

4) Make sure '***application_s'*** is compiled successfully and the file '***application_s_import_lib.o'*** has been put under "lib" in '***application_ns***'.
5) Right click '***application_ns'*** and click "**Rebuild All**" to build '***application_ns***'.
6) Make sure the '***application_ns'*** is compiled successfully.

3.4.3.2 Generate Image Binary

After compile, the images *partition.bin, bootloader.bin, firmware_tz.bin* and *flash_tz.bin* can be seen in the EWARM-RELEASE\Debug\Exe.

1) ***partition.bin*** stores partition table, recording the address of Boot image and firmware image;
2) ***bootloader.bin*** is bootloader image;
3) ***firmware_tz.bin*** is application image;
4) ***flash_tz.bin*** links *partition.bin, bootloader.bin* and *firmware_tz.bin*. Users need to choose *flash_tz.bin* when downloading the image to board by PG Tool.

3.4.3.3 Download

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

1) Directly **download** the image binary on to demo board from IAR IDE (as below)

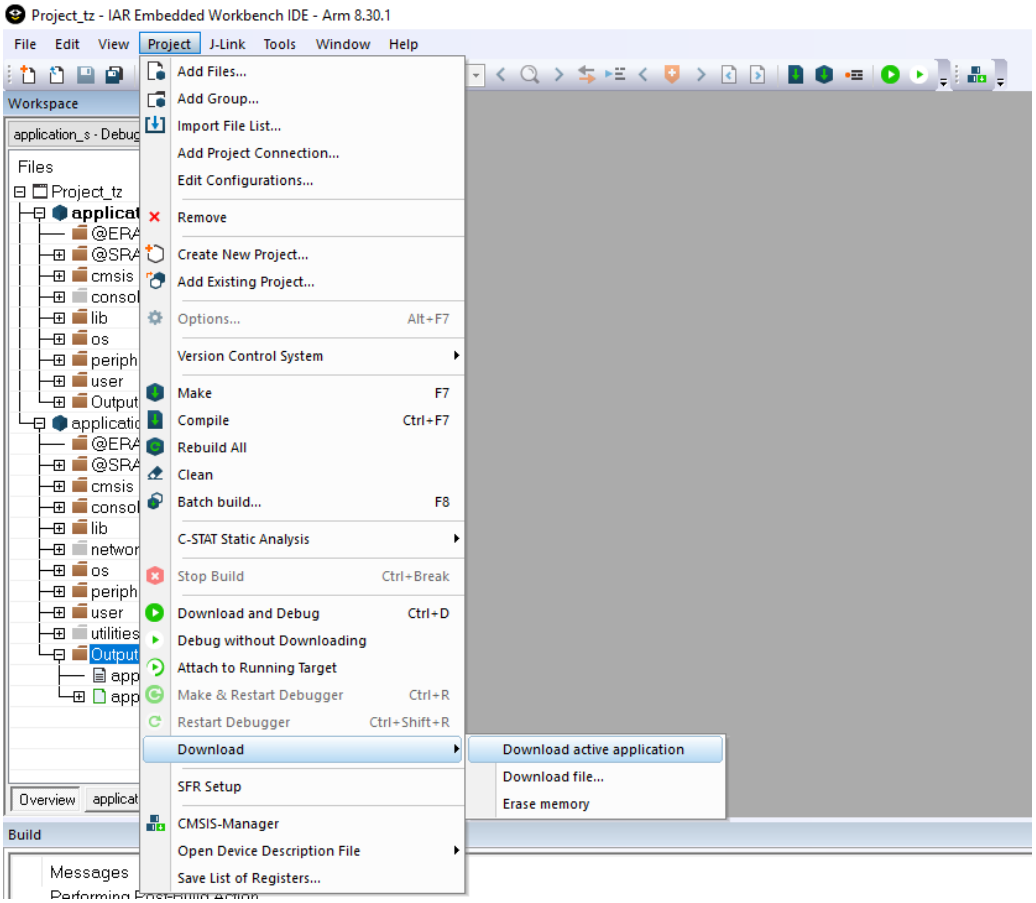


Figure 3-11 IAR download binary on flash

2) Or using the **PG tool** for Ameba-ZII (Will not be shown here, please check chapter Image Tool for details).

3.4.4 IAR Memory Configuration

The whole memory layout of Ameba-ZII can refer to chapter Memory Layout.
And there will be some extra configurations user needs to do if they want to put some code to certain memory region.

3.4.4.1 Configure Memory from IAR IDE

In IAR Workspace, there are “@ERAM” and “@SRAM” group. “@ERAM” represents **external RAM**, which can be known as **PSRAM**. “@SRAM” represents **SRAM**. Except “@ERAM” and “@SRAM” group, the rest of read-only section (TEXT and RODATA) will be placed on **XIP**.

If users want to link a particular file into PSRAM (need to make sure that PSRAM is available), users can drag-and-drop the files into “@ERAM”. For example, “test.c” will be linked to PSRAM as the graph below. If users want to place specific source file into SRAM, users can drag-and-drop the files into “@SRAM”. For example, “flash_api.c”, “flash_fatfs.c”, “hal_flash.c” is placed in SRAM as the graph below.

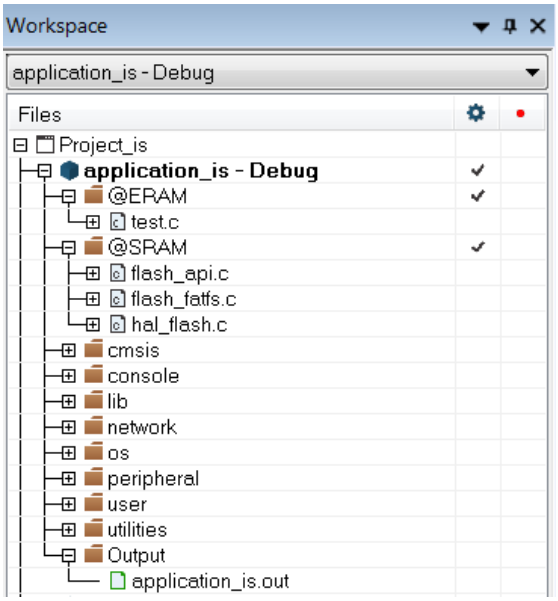


Figure 3-12 Overview of IAR Workspace

Note: There are some files which must NOT be linked to XIP section. Please keep the default settings of the IAR project if one doesn’t know about this.

3.4.4.2 Configure Memory from ICF File

IAR uses ICF (IAR Configuration File) to configure memory allocation so users can configure memory allocation by ICF file.
ICF file of Ameba-ZII locates: “SDK/project/realtek_amebaz2_v0_example/EWARM-RELEASE/ application_is.icf”
Open “application_is.icf” with any text editor. There are some memory regions in it, which is:

- DTCM_RAM_region
- ERAM_region
- XIP_FLASH_region

Users can reference IAR document if they don’t know the format of ICF.

3.4.5 IAR Memory Overflow

In default, Ameba-ZII place read-only (TEXT and RODATA) section in XIP area. If XIP does not have enough space, it will show the errors as below while linking.

```
Error[Lp011]: section placement failed
unable to allocate space for sections/blocks with a total estimated minimum size of 0x110'3e5e bytes
(max align 0x8) in <[0x9b00'0140-0x9bff'ffff]> (total uncommitted space 0xff'fec0).
```

The solution is to either minimize the code or move the code to other memory region. Same rule applies to SRAM and PSRAM if it's available.

3.5 GCC IDE Setup on Windows (Using Cygwin)

3.5.1 Install Cygwin

Cygwin is a large collection of GNU and Open Source tools which provide the similar functionality as a Linux distribution on Windows. It provides the GCC toolchain for Ameba-ZII to compile projects.
User 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**"

3.5.2 Build Non-Trust Zone Project

3.5.2.1 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 all libraries and application. Enter command "**make all**"
5) Make sure there is no error after compile.

3.5.2.2 Generate Image Binary

After compile, 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**.

3.5.2.3 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).

3.5.3 Build Trust Zone Project

3.5.3.1 Compile

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 all libraries and application. Enter command "**make tz**".

5) Make sure there is no error after compile.

3.5.3.2 Generate Image Binary

After compile, the images partition.bin, bootloader.bin, firmware_tz.bin and flash_tz.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_tz.bin** is application image; located at folder \GCC-RELEASE\application_tz;

4) **flash_tz.bin** links partition.bin, bootloader.bin and firmware_tz.bin. Located at folder \GCC-RELEASE\application_tz.

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

3.5.3.3 Download

After a successfully compilation and '**flash_tz.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 section' for SWD connection.

   Enter command "**make setup GDB_SERVER=jlink or pyocd**" to select GDB Server.

   Enter command "**make flash_tz**" at Cygwin.

2) Or using the **PG tool** for Ameba-ZII (Will not be shown here, please check chapter Image Tool for details).

3.5.3.4 Debug

After a successfully downloading, user can debug with pyOCD + DAPLink enabled HDK or using JLINKGDBServer + JLINK by following command:

"**make debug_tz**"

Note: Before using "make debug_tz", "make setup GDB_SERVER=jlink or pyocd" to select GDB Server is necessary.

3.5.4 GCC Memory Configuration

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

3.5.4.1 Configure Memory from makefile

In application.is.mk, user 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

3.5.4.2 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.

3.5.4.2.1 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
```

3.5.4.2.2 Move text section of an object to PSRAM

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

3.5.4.2.3 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
```

3.5.4.3 Configure Memory from C File

There are some section MACROs are 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)
{
        ……
}
```

3.5.5 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).

3.6 GCC Environment on Ubuntu/Linux

3.6.1 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 in order 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.

3.6.2 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 all libraries and application. Enter command "**make all**"
5. Once the build 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$
```

3.6.3 Download and Flash Binaries

There are in-built scripts in the makefile that initiate download and flashing of the software via JLink. In order 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. In order to start the JLink GDB server, follow the '**Steps to Initiate JLinkGDBServer'** section.

3.6.3.1 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 or pyocd" 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$
```

3.6.3.2 Debug

After a successfully downloading, user can debug with pyOCD + DAPLink enabled HDK or using JLINKGDBServer + JLINK by following command

“make debug_tz”

Before using “make debug_tz”, “make setup GDB_SERVER=jlink or pyocd” to select GDB Server is necessary.

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.

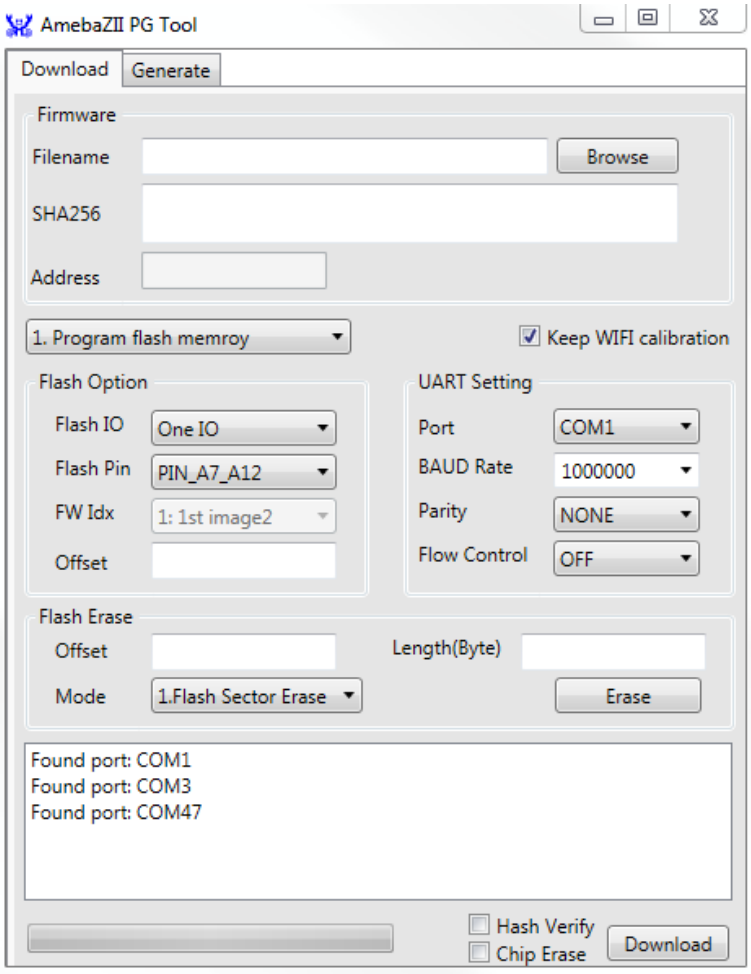


Figure 4-1 AmebaZII Image Tool UI

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.

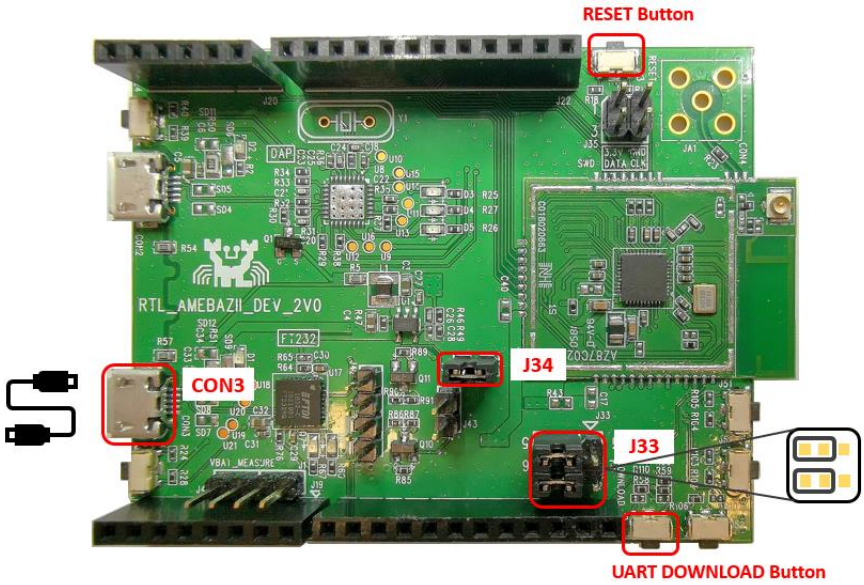


Figure 4-2 Ameba-ZII EVB V2.0 Hardware Setup

4.2.2 Software Setup

- Environment Requirements: EX. WinXP, Win 7 Above, Microsoft .NET Framework 3.5
- AmebaZII_PGTool_v1.0.1.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.

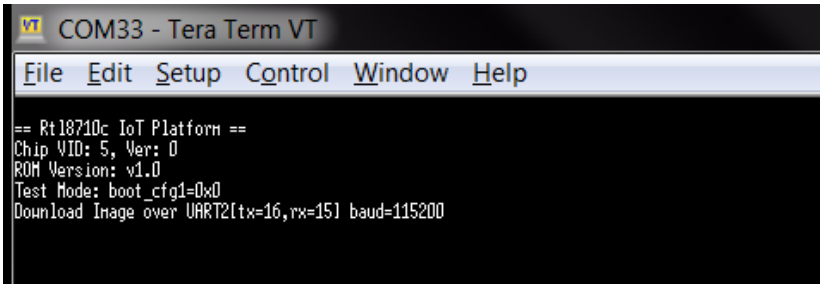
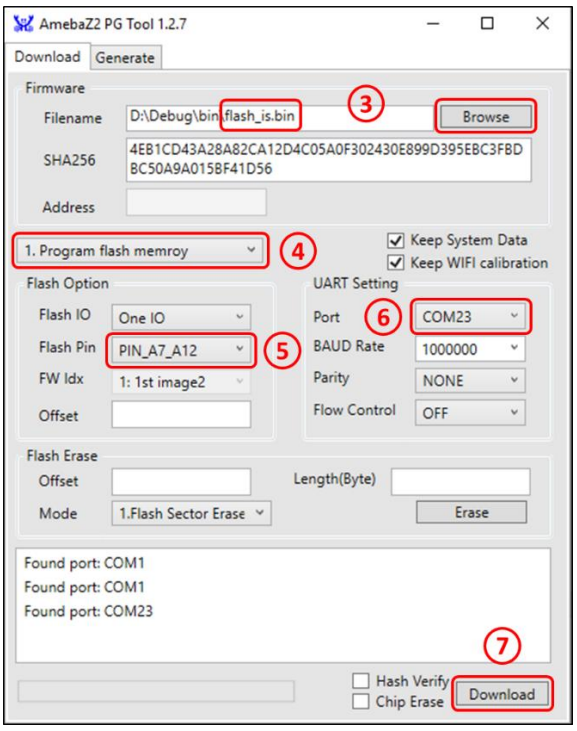


Figure 4-4 Ameba-ZII UART download mode

   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**



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	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:

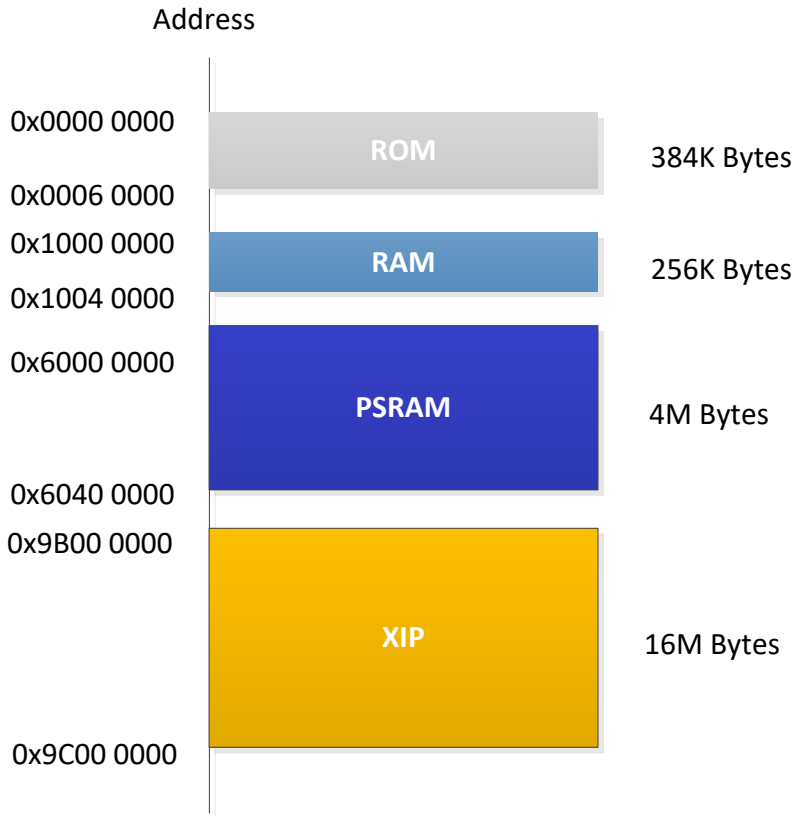


Figure 5-1 Address Allocation of Different Memories on Ameba-ZII

5.2 Flash Memory Layout

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

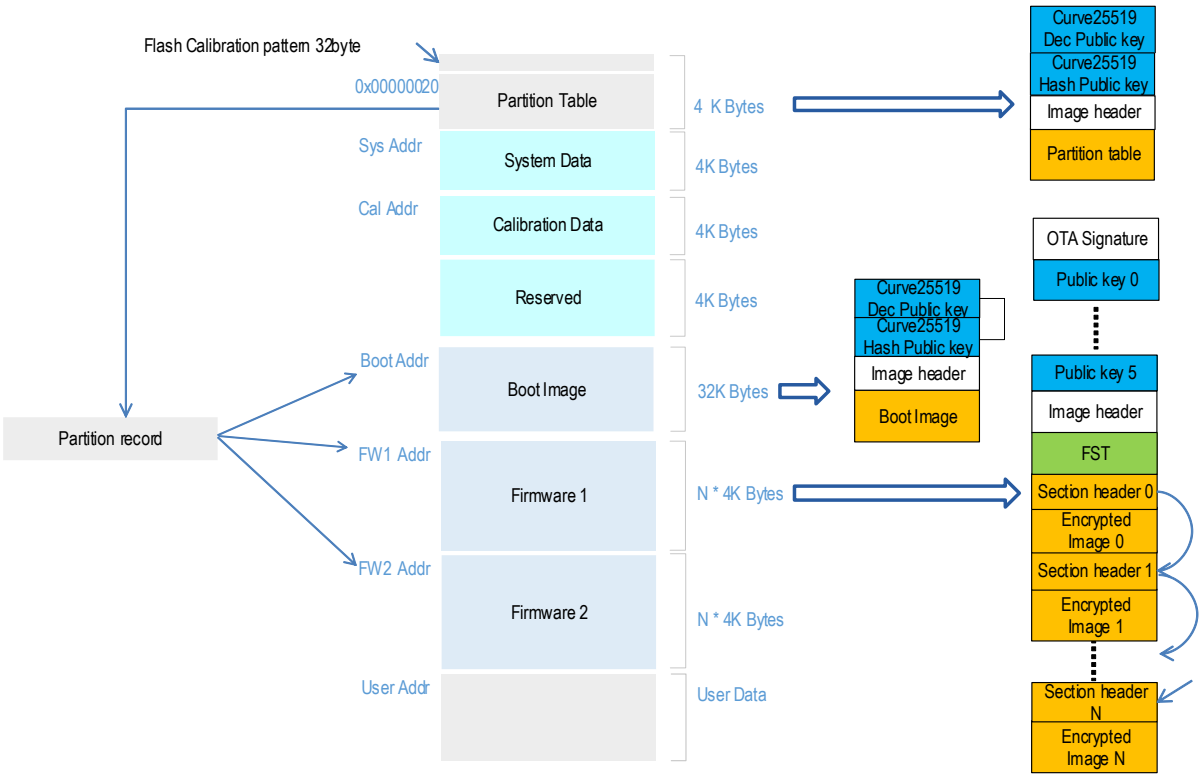


Figure 5-2 Flash memory layout

And the description of each block is listed in 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	Y

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:

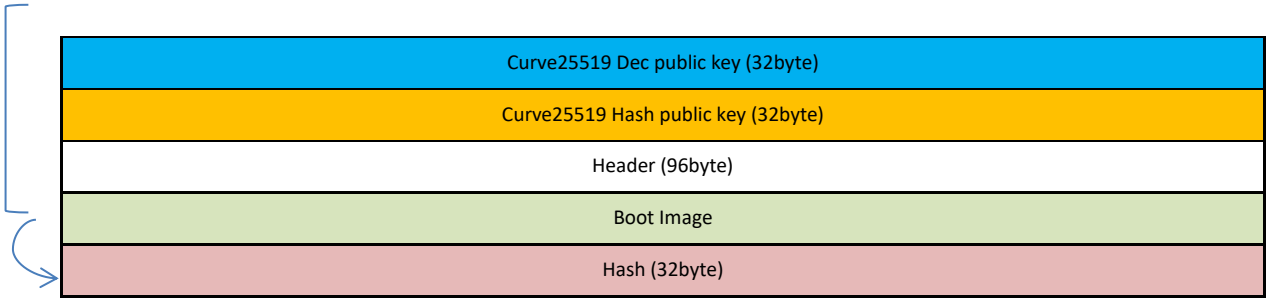


Table 5-8 The layout of boot image

- 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:

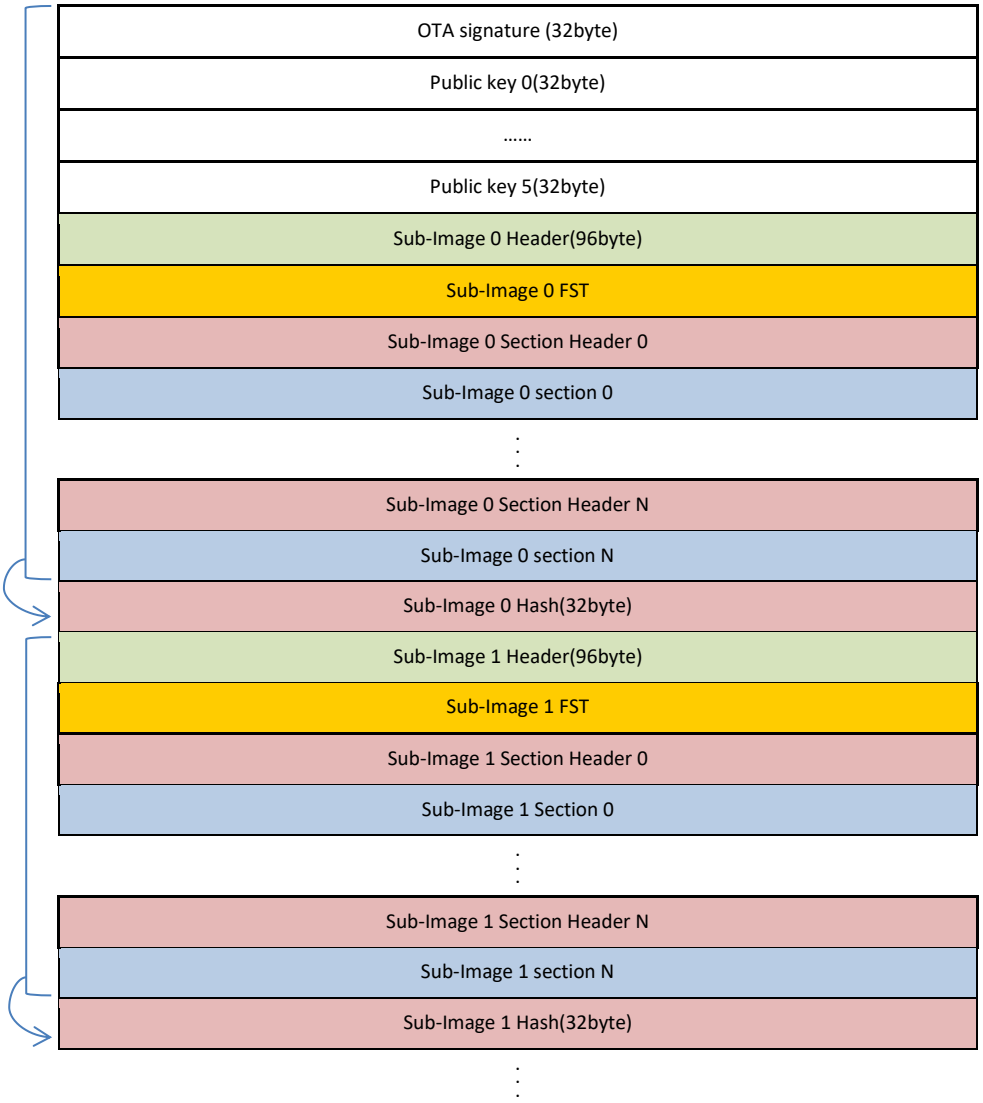


Table 5-9 The layout of firmware image

- **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

5.2.4.2 How to Generate Flash Image Combines Both Firmware1 and Firmware2

There may be requirements need to generate flash image combines both firmware1 and firmware2. The default set of AmebaZ2 flash image contains partition table, boot image, and firmware1 image. In order to add firmware2 image to flash image, extra configurations need to be done.

- Prepare firmware2 image.
  Please not that there is serial number needs to be customized for firmware2 image. Refer to "*7.4.2 Configuration for building OTA firmware*" for details of the serial number setting.

  Note: that default serial number is "100", any number larger than "100" will set firmware2 image as default for flash image otherwise firmware1 image is default.

  Add firmware2 image at "project\realtek_amebaz2_v0_example\EWARM-RELEASE\Debug\Exe"

- Update "postbuild_is.bat" for combines firmware1 and firmware2
  Add "FW2=Debug\Exe\firmware2_is.bin" after "FW1=Debug\Exe\firmware_is.bin" at "component\soc\realtek\8710c\misc\iar_utility\postbuild_is.bat" refers the following table

```
::generate flash image, including partition + bootloader + firmware
%tooldir%\elf2bin.exe combine Debug/Exe/flash_is.bin
PTAB=Debug\Exe\partition.bin,BOOT=Debug\Exe\bootloader.bin,FW1=Debug\Exe\firmware_is.bin,FW2=Debug\Exe\firmware2_is.bin >> postbuild_is_log.txt
if not exist Debug\Exe\flash_is.bin (
        echo flash_is.bin isn't generated, check postbuild_is_log.txt
        echo flash_is.bin isn't generated > postbuild_is_error.txt
        goto error_exit
)
```

  Note: that firmware2 image must have the same name as the code added in "postbuild_is.bat".

- Generate and download flash image
  Please refer to *"3.4.2 Build Non-Trust Zone Project"* for generate flash image. The firmware1 image is auto generated when generating flash image. The generated image ("flash_is.bin") is the flash image combines both fimware1 and firmware2. Please note that the flash image is only downloadable by Image Tool. Refer to *"4.3 Image Download"* for details of using Image Tool downloading.

- Switch between firmware1 image and firmware2 image
  To switch firmware1 image and firmware2 image please refer to ATCMD "ATSC" and "ATSR" which is detailed in "AN0075 Realtek Ameba-all at command v2.0.docx".

5.3 SRAM Layout

The range of DTCM is from 0x10000000 to 0x10040000. The layout of this memory region is illustrated below.
Note: the layout may be changed according to 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 0x1003FFFF	Reserved for ROM

Table 5-10 AmebaZII DTCM (256KB) 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

Table 5-11 Description of RAM layout

5.4 TrustZone Memory Layout

By default, the memory is marked as secure, unless the address matches a region defined in SAU (Security Attribution Unit) which is used to define several memory regions as non-secure or NSC (Non-Secure Callable).

AmebaZII support 4 SAUs, and the regions are configured for ROM, RAM, PSRAM and XIP, which covers all memory units. The total region and boundary for RAM, PSRAM and XIP are configurable. Below is the default configuration of each SAU.

- **SAU 0:** 0x00019000 ~ 0x0005ffff as Non-Secure
    - ROM. The region is fixed and must not be modified.
- **SAU 1:** 0x10008000 ~ 0x4fffffff as Non-Secure
    - RAM.
- **SAU 2:** 0x60100000 ~ 0x9bfeffff as Non-Secure
    - PSRAM and XIP share SAU2. The start address of SAU2 must be located in PSRAM and the end address of SAU2 must be located in XIP
- **SAU 3:** 0x9bfc0000 ~ 0x9bffffff as Secure (NSC)
    - XIP. NSC is suggested to be located at the end (end is 0x9c00 0000)

Thus, if the TrustZone is enabled, the system memory layout will become as below.

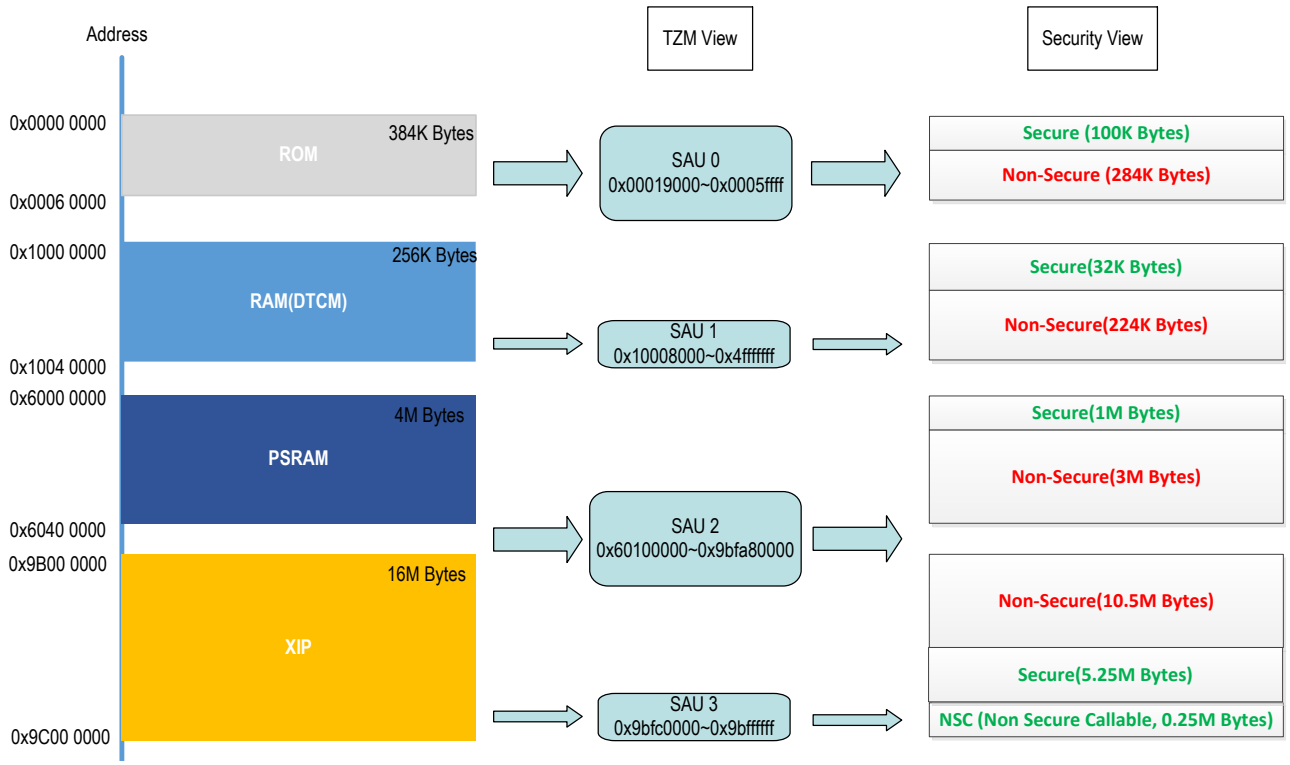


Figure 5-3 TrustZone Memory Layout

6 Boot Process

This chapter descripts 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.

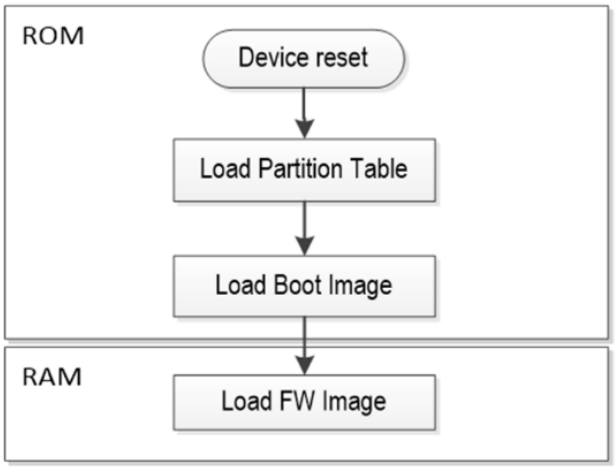


Figure 6-1 Overview of boot flow

6.2 Secure Boot

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

If the image is valid, then the authentication will be successful, which means the firmware is safe. And the subsequent process can be continued. Otherwise, the SoC will go into infinite loop.

6.2.1 Secure Boot Flow

While booting, the system will use the **encrypted private key** which locates in **super secure efuse**, and the **public key** which locates in **flash**, to generate AES keys and Hash keys, then use them to decrypt and verify each image.

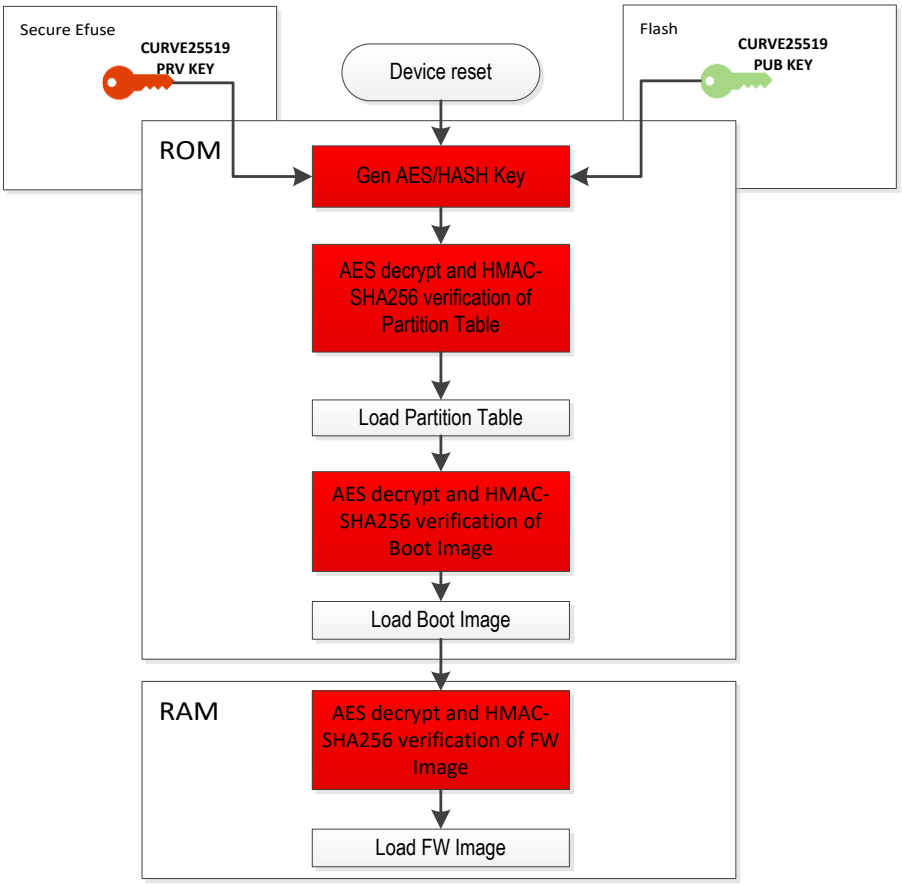


Figure 6-2 Secure boot flow

6.2.2 Partition Table and Boot Image Decryption Flow

Figure 6-3 illustrates the decryption flow of partition table and boot image in secure boot process.

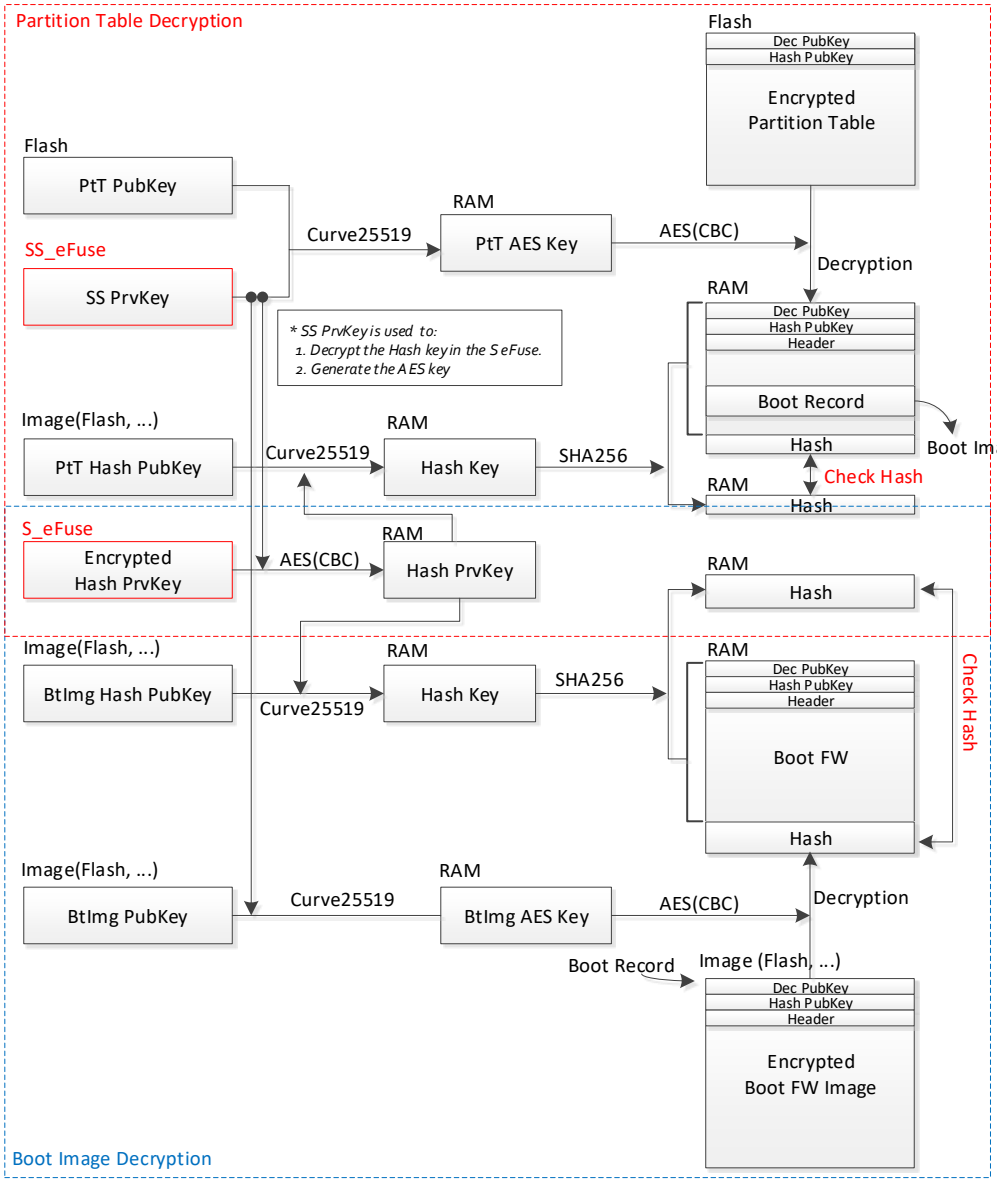


Figure 6-3 Partition table and boot image decryption flow

6.2.3 Secure Boot Use Scenario

Figure 6-4 illustrates the use scenario of secure boot in a real product development. Firstly, software developer needs to generate key pairs, and encrypt and hash the firmware properly. And when it comes to mass production, manufacturing facility will program the encrypted firmware along with a reference hash value on device, and program the private key in super secure efuse zone. While device boots up, it will use the super secure private key and public key to verify the hash value and decrypt each image. If case of any error or failure in the image validation process, the device won't boot.

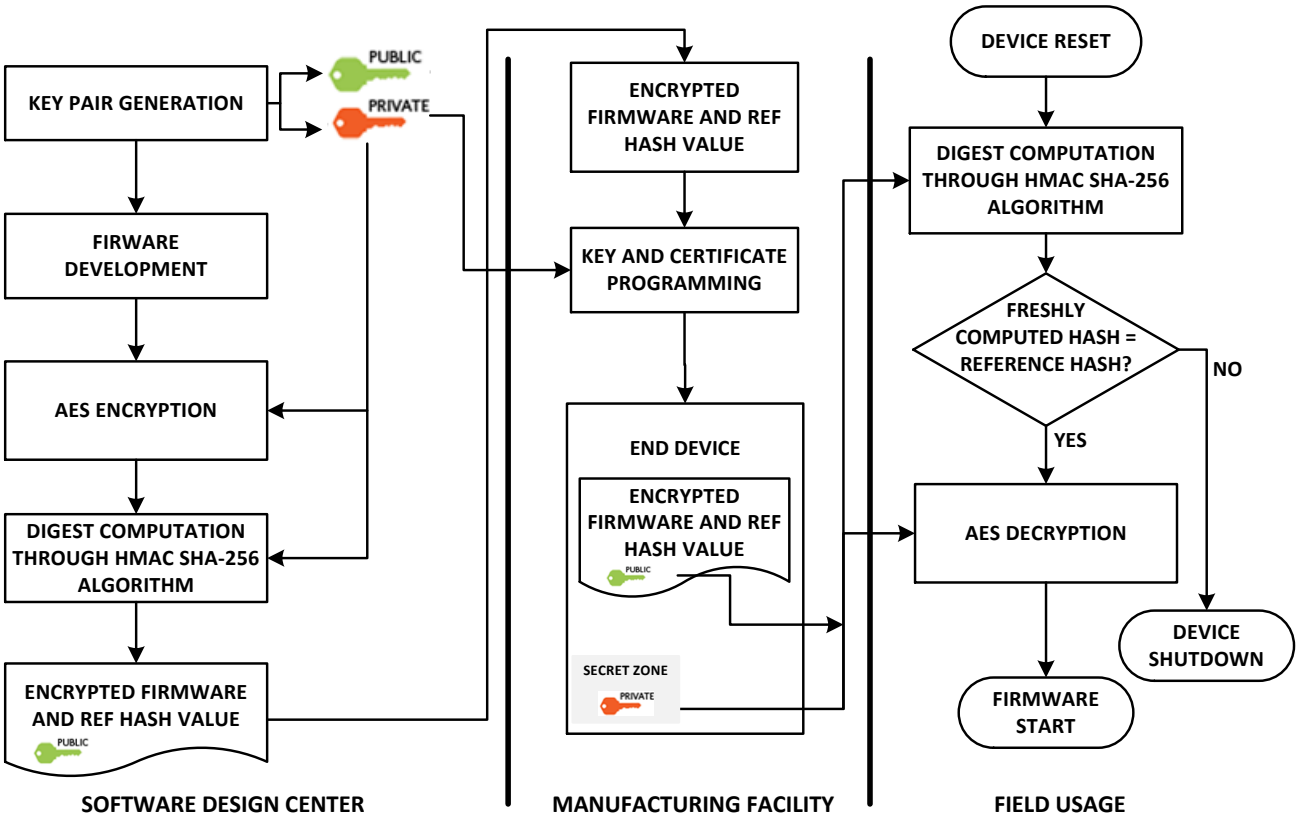


Figure 6-4 secure boot use scenario

6.2.4 How to Enable Secure Boot

This section shows how to enable the secure boot.

6.2.4.1 Keys Configuration

As mentioned above, while booting, the system will use the encryption private key which locates in super secure efuse, and the public key which locates in flash, to generate AES keys and Hash keys, then use them to decrypt and verify each image.
The super secure private key (named as “privkey_enc”) and the encrypted hash private key (named as “privkey_hash”) are configured in *keycfg.json* under *project\realtek_amebaz2_v0_example\EWARM-RELEASE*.

What shows following is the default value. You can configure the keys by yourself in *keycfg.json.*

```
{
   "__comment_0":"configuration for private key, use auto to generate random key",
   "__comment_1":"private key maybe different from your desired input, program will modify first byte and
last byte of key",
   "__comment_2":"to get actual private key, please open key.json after key generated.",
   "__comment_3":"hash private key in key.json will be encrypted",
   "privkey_enc":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F",
   "privkey_enc1":"auto",
   "privkey_hash":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F",
   "privkey_hash1":"auto"
}
```

6.2.4.2 Keys Programming in Efuse

After configuring the keys in *keycfg.json,* rebuild the whole project in IAR.

The *key.json* file under ‘*project\realtek_amebaz2_v0_example\EWARM-RELEASE\’* will be updated after compiling. The “privkey_enc” is the same with the “privkey_enc” in *keycfg.json*, but the “privkey_hash” has been changed after *Hash* algorithm.

```
{
        "__comment_EFUSE":"should keep these two key in safe place, or secure firmware protection is
useless",
        "EFUSE":{
                "__comment_privkey_enc":"this key in EFUSE SUPER SECURE ZONE",

        "privkey_enc":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F",
                "__comment_privkey_hash":"this key in EFUSE SECURE ZONE, encrypted by upper key",

        "privkey_hash":"64A7433FCF027D19DDA4D446EEF8E78A22A8C33CB2C337C07366C040612EE0F2"
        },
        "TOOL":{
                "__comment_pubkey_enc":"public key for encryption",

        "pubkey_enc":"8F40C5ADB68F25624AE5B214EA767A6EC94D829D3D7B5E1AD1BA6F3E2138285F",
                "__comment_pubkey_hash":"public key for hash, only for partition table and bootloader",

        "pubkey_hash":"8F40C5ADB68F25624AE5B214EA767A6EC94D829D3D7B5E1AD1BA6F3E2138285F"
        }
}
```

"example_secure_boot.c" is an example provided for enabling secure boot. To write the keys to efuse, firstly, change CONFIG_EXAMPLE_SECURE_BOOT to 1 in platform_opts.h.

```
/*For secure boot example */
#define CONFIG_EXAMPLE_SECURE_BOOT 1
```

Secondly, modify the super secure key "susec_key[]" and the secure key "sec_key[]" to correspond with "privkey_enc" and "privkey_hash" in *key.json* file under *project\realtek_amebaz2_v0_example\EWARM-RELEASE* in "example_secure_boot.c". In another word, "susec_key[]" should be the same with "privkey_enc" in *key.json*, and "sec_key[]" should be the same with "privkey_hash" in *key.json*.

```
//these two keys are the same default keys used in SDK
const uint8_t susec_key[PRIV_KEY_LEN] = {
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
    0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x5F
};
const uint8_t sec_key[PRIV_KEY_LEN] = {
    0x64, 0xA7, 0x43, 0x3F, 0xCF, 0x02, 0x7D, 0x19,
    0xDD, 0xA4, 0xD4, 0x46, 0xEE, 0xF8, 0xE7, 0x8A,
    0x22, 0xA8, 0xC3, 0x3C, 0xB2, 0xC3, 0x37, 0xC0,
    0x73, 0x66, 0xC0, 0x40, 0x61, 0x2E, 0xE0, 0xF2
};
```

Thirdly, modify 0 to 1 to enable write super secure key function and write secure key function to write keys to efuse. What needs to be noted is, the efuse is one-time writable, please make sure the key is correct before programming in efuse.

```
// write SS key
   memset(write_buf, 0xFF, PRIV_KEY_LEN);
   if(1){ // fill your data
      for(i=0; i<PRIV_KEY_LEN; i++)
         write_buf[i] = susec_key[i];
   }
   if(1){ // write
      device_mutex_lock(RT_DEV_LOCK_EFUSE);
      ret = efuse_susec_key_write(write_buf);
      device_mutex_unlock(RT_DEV_LOCK_EFUSE);
      if(ret < 0){
         dbg_printf("efuse SS key: write address and length error\r\n");
         goto exit;
      }
      dbg_printf("\r\nWrite Done.\r\n");
   }else{
      dbg_printf("\r\nPlease make sure the key is correct before programming in efuse.\r\n");
   }
       dbg_printf("\r\n");
// write S key
   memset(write_buf, 0xFF, PRIV_KEY_LEN);
   if(1){ // fill your data
      for(i=0; i<PRIV_KEY_LEN; i++)
         write_buf[i] = sec_key[i];
   }
   if(1){ // write
      device_mutex_lock(RT_DEV_LOCK_EFUSE);
```

```
    ret = efuse_sec_key_write(write_buf, 0);
    device_mutex_unlock(RT_DEV_LOCK_EFUSE);
    if(ret < 0){
      dbg_printf("efuse S key: write address and length error\r\n");
      goto exit;
    }
    dbg_printf("\r\nWrite Done.\r\n");
  }else{
    dbg_printf("\r\nPlease make sure the key is correct before programming in efuse.\r\n");
  }
  dbg_printf("\r\n");
```

The SS key locker can be enabled by changing 0 to 1 marked as yellow here. If the locker is enabled, the SS key turns to be unreadable forever. So, this configuration is irreversible, please do if only you are certain about SS key.

```
/*
Step 3: lock and protect the SS key from being read by CPU
  */
  // lock SS key, make SS key unreadable forever.
  // this configure is irreversible, so please do this only if you are certain about SS key
  if(1){
    device_mutex_lock(RT_DEV_LOCK_EFUSE);
    ret = efuse_lock_susec_key();
    device_mutex_unlock(RT_DEV_LOCK_EFUSE);
    if(ret < 0){
      dbg_printf("efuse SS key lock error\r\n");
      goto exit;
    }
  }
```

Enable secure boot by setting the flag to 1 in step 4. After enabling the secure boot, the device will only boot with encrypted image. The configure is also irreversible, so please do this if you are certain that the firmware image is encrypted and hashed with the correct SS key and S key.

```
/*
  Step 4: enable the secure boot so that device will only boot with encrypted image
  */
  // enable secure boot, make device boot only with correctly encrypted image
  // this configure is irreversible, so please do this only if you are certain that the fw image is encrypted and
hashed with the correct SS key and S key
    if(1){
    device_mutex_lock(RT_DEV_LOCK_EFUSE);
    ret = efuse_fw_verify_enable();
    device_mutex_unlock(RT_DEV_LOCK_EFUSE);
    if(ret < 0){
      dbg_printf("efuse secure boot enable error\r\n");
      goto exit;
    }
    device_mutex_lock(RT_DEV_LOCK_EFUSE);
    ret = efuse_fw_verify_check();
    device_mutex_unlock(RT_DEV_LOCK_EFUSE);
    if(ret)
      dbg_printf("secure boot is enabled!");
    }
```

What needs to highlight is, the two sections above aim to write keys to efuse and enable secure boot. Before enabling secure boot, the Ameba-ZII is in non-secure boot mode, that means encrypted image cannot boot. So, till now, the application is built to enable secure boot. After enabling, encrypt the image and then the Ameba-ZII can boot with encrypted image.

6.2.4.3 Encrypt the Image

The 'boot/firmware 1/firmware 2' addresses are stored in partition records, defined in '*partition.json*' under '*project**realtek_amebaz2_v0_example**EWARM-RELEASE*\'. The addresses can be modified if needed.

```
"boot":{
        "start_addr" : "0x4000",
        "length" : "0x8000",
        "type": "BOOT",
        "dbg_skip": false,
        "hash_key":"FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF"
    },
    "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"
    }
```

Ameba-ZII is defaulted as non-secure boot mode, secure boot asks encrypted firmware. The keys to enable firmware encryption are defined in '*amebaz2_bootloader.json/amebaz2_firmware_is.json/amebaz2_firmware_tz.json*' under '*project**realtek_amebaz2_v0_example**EWARM-RELEASE*\'. "*enc*" can be changed from *false* to *true* in following code to enable corresponding module encryption.

In *amebaz2_bootloader.json:*

```
"PARTAB": {
                    "source":null,
                    "header":{
                            "next":null,
                            "__comment_type":"Support
Type:PARTAB,BOOT,FWHS,FWLS,ISP,VOE,WLN,DTCM,ITCM,SRAM,ERAM,XIP,M0,CPFW",
                            "type":"PARTAB",
                            "enc":true,
                            "serial": 0
                    },
                    "list" : ["partab"],
                    "partab": {
                            "__comment_ptable":"move to partition.json",

                            "__comment_file":"TODO: use binary file directly",
                            "file": null
                    }
        },
"BOOT": {
                    "source":"Debug/Exe/bootloader.out",
                    "header":{
```

```
                "next":null,
                "__comment_type":"Support
Type:PARTAB,BOOT,FWHS,FWLS,ISP,VOE,WLN,DTCM,ITCM,SRAM,ERAM,XIP,M0,CPFW",
                "type":"BOOT",
                "enc":true,
        "user_key1":"AA010203040506070809 0A0B0C0D0E0F101112131415161718191A1B1C1D1E1F",
                "serial": 0
            },
            "list" : ["sram"],
            "sram": {
                "__comment_option":"TODO: not ready",
                "option": null,

                "__comment_entry":"startup function table symbol",
                "entry":"gRamStartFun",

                "start": "RAM_FUNTAB$$Base",
                "end": "RAM_RODATA$$Limit",

                "__comment_file":"TODO: use binary file directly",
                "file": null
            }
        }
```

For **ignore secure project**, need to modify *amebaz2_firmware_is.json*, "*enc*" also is set to ***true*** to encrypt the image. But what needs to be noted is, the "*enc*" in "*XIP_FLASH_P*" could not be set to ***true*** because this section is reserved for plain data.

```
"FWHS": {
            "source":"Debug/Exe/application_is.out",
            "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":true,

          "user_key2":"BB010203040506070809 0A0B0C0D0E0F101112131415161718191A1B1C1D1E1F",
                "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 107
            },
      "FST":{
                "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                "__comment_FST1": "validpat is used for section header validation",
                "__comment_FST2": "hash_en/enc_en?",
        "enc_algorithm":"cbc",
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":true,
                "cipherkey":null,
                "cipheriv":null
      },
     }

"XIP_FLASH_C": {
```

```
                "source":"Debug/Exe/application_is.out",
                "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":"XIP",
                        "enc":true,
                        "__comment_pkey_idx":"assign by program, no need to configurate",
                        "serial": 0
                },
    "FST":{
                        "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                        "__comment_FST1": "validpat is used for section header validation",
                        "__comment_FST2": "hash_en/enc_en?",
      "enc_algorithm":"cbc",
                        "hash_algorithm":"sha256",
                        "part_size":"4096",

                        "__comment_validpat": "use auto or dedicated value",
                        "validpat":"0001020304050607",
                        "hash_en":true,
                        "enc_en":true,
                        "cipherkey":null,
                        "cipheriv":null
    },
}

"XIP_FLASH_P": {
                "source":"Debug/Exe/application_is.out",
                "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":"XIP",
                        "enc":false,
                        "__comment_pkey_idx":"assign by program, no need to configurate",
                        "serial": 0
                },
    "FST":{
                        "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                        "__comment_FST1": "validpat is used for section header validation",
                        "__comment_FST2": "hash_en/enc_en?",
      "enc_algorithm":"cbc",
                        "hash_algorithm":"sha256",
                        "part_size":"4096",

                        "__comment_validpat": "use auto or dedicated value",
                        "validpat":"0001020304050607",
                        "hash_en":true,
                        "enc_en":false,
                        "cipherkey":null,
                        "cipheriv":null
    },
}
```

For **trust zone project**, need to modify *amebaz2_firmware_tz.json*, "*enc*" also is set to ***true*** to encrypt the image. But what needs to be noted is, the "*enc*" in "*XIP_S_P*" and "*XIP_NS_P*" could not be set to ***true*** because these sections are reserved for plain data.

```
{
   "msg_level": 3,

      "__comment":"example key",
      "000priv":"A0D6DAE7E062CA94CBB294BF896B9F68CF8438774256AC7403CA4FD9A1C9564F",
      "000pub":"68513EF83E396B12BA059A900F36B6D31D11FE1C5D25EB8AA7C550307F9C2405",
      "001priv":"882AA16C8C44A7760AA8C9AB22E3568C6FA16C2AFA4F0CEA29A10ABCDF60E44F",
      "001pub":"48AD23DDBDAC9E65719DB7D394D44D62820D19E50D68376774237E98D2305E6A",
      "002priv":"58A3D915706835212260C22D628B336D13190B539714E3DB249D823CA5774453",
      "002pub":"FD8D3F3E516D96186E10F07A64B24C7DE736826A24FAFE367E79F1FBB2F1C832",
      "003priv":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F",
      "003pub":"8F40C5ADB68F25624AE5B214EA767A6EC94D829D3D7B5E1AD1BA6F3E2138285F",

      "PROFILE":["FIRMWARE"],
      "FIRMWARE":{
      "rand_pad": false,
      "__comment_xip_pg_size":"XIP remapping page size/alignment setting: 0/1/2: 16K/32K/64K",
         "xip_pg_size": 0,

      "__comment_mode":"mode 0: bootloader and partition table, mode 1: firmware",
               "mode": 1,
               "file":"Debug/Exe/firmware_tz.bin",
               "__comment_too_privkey":"if user want to fix key, can set priviate key here, if not, will use
random key",

      "privkey_enc":"A0D6DAE7E062CA94CBB294BF896B9F68CF8438774256AC7403CA4FD9A1C9564F",

               "__comment_hash_key_src":"hash key from partition table FW1/FW2 (must match type in
partition item)",
               "hash_key_src" : "FW1",

      "__comment_images":"offset = null => cascade ( align to 64 ), should be zero if valid",
               "images":[
                        {"img": "FWHS_S", "offset":"0x00"},
                        {"img": "FWHS_NSC", "offset":"0x00"},
                        {"img": "FWHS_NS", "offset":"0x00"},
                        {"img": "XIP_S_C", "offset":"0x00"},
                        {"img": "XIP_S_P", "offset":"0x00"},
                        {"img": "XIP_NS_C", "offset":"0x00"},
                        {"img": "XIP_NS_P", "offset":"0x00"}
               ]
      },
      "FWHS_S": {
               "source":"Debug/Exe/application_s.out",
               "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":true,
        "user_key2":"BB0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F",
                        "__comment_pkey_idx":"assign by program, no need to configurate",
                        "serial": 100
               },
      "FST":{
                        "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                        "__comment_FST1": "validpat is used for section header validation",
                        "__comment_FST2": "hash_en/enc_en?",
        "enc_algorithm":"cbc",
```

```
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":true,
                "cipherkey":null,
                "cipheriv":null
        },
        "list" : ["sram","psram"],
        "sram": {
                "secthdr":{
                        "type": "SRAM"
                },
                "__comment_option":"TODO: not ready",
                "option": null,

                "__comment_entry":"startup function table symbol",
                "entry":"gRamStartFun",

                "sections":     ["FIRMWARE_FUNTAB*", "FIRMWARE_SIGN*",
"FIRMWARE_SRAM_RO*", "FIRMWARE_SRAM_RW*"],
                "__comment_file":"TODO: use binary file directly",
                "file": null
        },
        "psram": {
                "secthdr":{
                        "type": "PSRAM"
                },
                "__comment_option":"TODO: not ready",
                "option": null,

                "sections":     ["FIRMWARE_ERAM_RO*", "FIRMWARE_ERAM_RW*"],
                "__comment_file":"TODO: use binary file directly",
                "file": null
        }
    },
    "FWHS_NS": {
        "source":"Debug/Exe/application_ns.out",
        "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_NS",
                "enc":true,
                "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 0
        },
    "FST":{
                "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                "__comment_FST1": "validpat is used for section header validation",
                "__comment_FST2": "hash_en/enc_en?",
      "enc_algorithm":"cbc",
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
```

```
                "hash_en":true,
                "enc_en":true,
                "cipherkey":null,
                "cipheriv":null
        },
                "list" : ["sram","vector","psram"],
                "sram": {
                        "secthdr":{
                                "type": "SRAM"
                        },
                        "__comment_option":"TODO: not ready",
                        "option": null,

                        "__comment_entry":"startup function table symbol",

                        "sections":     ["FIRMWARE_FUNTAB*", "FIRMWARE_SIGN*",
"FIRMWARE_SRAM_RO*", "FIRMWARE_SRAM_RW*"],
                        "__comment_file":"TODO: use binary file directly",
                        "file": null
                },
                "vector": {
                        "secthdr":{
                                "type": "SRAM"
                        },
                        "__comment_option":"TODO: not ready",
                        "option": null,

                        "sections":     ["FIRMWARE_VECTOR*"],
                        "__comment_file":"TODO: use binary file directly",
                        "file": null
                },
                "psram": {
                        "secthdr":{
                                "type": "PSRAM"
                        },
                        "__comment_option":"TODO: not ready",
                        "option": null,

                        "sections":     ["FIRMWARE_ERAM_RO*", "FIRMWARE_ERAM_RW*"],
                        "__comment_file":"TODO: use binary file directly",
                        "file": null
                }
        },
        "FWHS_NSC": {
                "source":"Debug/Exe/application_s.out",
                "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":"XIP",
                        "enc":true,
                        "__comment_pkey_idx":"assign by program, no need to configurate",
                        "serial": 0
                },
        "FST":{
                        "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                        "__comment_FST1": "validpat is used for section header validation",
                        "__comment_FST2": "hash_en/enc_en?",
          "enc_algorithm":"cbc",
```

```
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":true,
                "cipherkey":null,
                "cipheriv":null
        },
                "list" : ["nsc"],
        "nsc": {
                "secthdr":{
                        "type": "XIP",
                        "xip_key": "A0D6DAE7E062CA94CBB294BF896B9F68",
                        "xip_iv": "94879487948794879487948794879487"
                },
                "__comment_option":"TODO: not ready",
                "option": null,

                "__comment_entry":"XIP text, RO_data",

                "sections":         ["FIRMWARE_NSC*"],
                "__comment_file":"TODO: use binary file directly",
                "file": null
                }
        },
        "XIP_S_C": {
                "source":"Debug/Exe/application_s.out",
                "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":"XIP",
                "enc":true,
                "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 0
                },
        "FST":{
                "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                "__comment_FST1": "validpat is used for section header validation",
                "__comment_FST2": "hash_en/enc_en?",
          "enc_algorithm":"cbc",
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":true,
                "cipherkey":null,
                "cipheriv":null
        },
                "list" : ["xip"],
                "xip": {
                "secthdr":{
                        "type": "XIP",
                        "xip_key": "A0D6DAE7E062CA94CBB294BF896B9F68",
                        "xip_iv": "94879487948794879487948794879487"
```

```
                },
                "__comment_option":"TODO: not ready",
                "option": null,

                "__comment_entry":"XIP text, RO_data",

                "sections":         ["FIRMWARE_XIP_S_C*"],
                "__comment_file":"TODO: use binary file directly",
                "file": null
        }
    },
    "XIP_S_P": {
        "source":"Debug/Exe/application_s.out",
        "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":"XIP",
                "enc":false,
                "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 0
        },
    "FST":{
                "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                "__comment_FST1": "validpat is used for section header validation",
                "__comment_FST2": "hash_en/enc_en?",
      "enc_algorithm":"cbc",
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":false,
                "cipherkey":null,
                "cipheriv":null
    },
        "list" : ["xip"],
        "xip": {
                "secthdr":{
                        "type": "XIP",
                        "xip_key": "A0D6DAE7E062CA94CBB294BF896B9F68",
                        "xip_iv": "94879487948794879487948794879487"
                },
                "__comment_option":"TODO: not ready",
                "option": null,

                "__comment_entry":"XIP text, RO_data",

                "sections":         ["FIRMWARE_XIP_S_P*"],
                "__comment_file":"TODO: use binary file directly",
                "file": null
        }
    },
    "XIP_NS_C": {
        "source":"Debug/Exe/application_ns.out",
        "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":"XIP",
                "enc":true,
                "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 0
            },
        "FST":{
                "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                "__comment_FST1": "validpat is used for section header validation",
                "__comment_FST2": "hash_en/enc_en?",
          "enc_algorithm":"cbc",
                "hash_algorithm":"sha256",
                "part_size":"4096",

                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":true,
                "cipherkey":null,
                "cipheriv":null
        },
            "list" : ["xip"],
            "xip": {
                "secthdr":{
                        "type": "XIP",
                        "xip_key": "A0D6DAE7E062CA94CBB294BF896B9F68",
                        "xip_iv": "94879487948794879487948794879487"
                },
                "__comment_option":"TODO: not ready",
                "option": null,

                "__comment_entry":"XIP text, RO_data",

                "sections":         ["FIRMWARE_XIP_C*"
                ],
                "__comment_file":"TODO: use binary file directly",
                "file": null
            }
        },
        "XIP_NS_P": {
            "source":"Debug/Exe/application_ns.out",
            "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":"XIP",
                "enc":false,
                "__comment_pkey_idx":"assign by program, no need to configurate",
                "serial": 0
            },
        "FST":{
                "__comment_FST0": "enc_algorithm: cbc/ecb with cipher key",
                "__comment_FST1": "validpat is used for section header validation",
                "__comment_FST2": "hash_en/enc_en?",
          "enc_algorithm":"cbc",
                "hash_algorithm":"sha256",
                "part_size":"4096",
```

```
                "__comment_validpat": "use auto or dedicated value",
                "validpat":"0001020304050607",
                "hash_en":true,
                "enc_en":false,
                "cipherkey":null,
                "cipheriv":null
        },
                "list" : ["xip"],
                "xip": {
                        "secthdr":{
                                "type": "XIP",
                                "xip_key": "A0D6DAE7E062CA94CBB294BF896B9F68",
                                "xip_iv": "94879487948794879487948794879487"
                        },
                        "__comment_option":"TODO: not ready",
                        "option": null,

                        "__comment_entry":"XIP text, RO_data",

                        "sections":         ["FIRMWARE_XIP_P*"
                        ],
                        "__comment_file":"TODO: use binary file directly",
                        "file": null
                }
        }
    }
```

6.2.4.4 Open “secure_bit”

6.2.4.4.1 IAR

For **ignore secure project**, “*secure_bit*” in *postbuild_is.bat* under *component**soc**realtek**8710c**misc**iar_utility* needs to be set to 1.

```
%tooldir%\elf2bin.exe convert amebaz2_bootloader.json BOOTLOADER secure_bit=1 >> postbuild_is_log.txt
if not exist Debug\Exe\bootloader.bin (
        echo bootloader.bin isn't generated, check postbuild_is_log.txt
        echo bootloader.bin isn't generated > postbuild_is_error.txt
        pause
        exit 2 /b
)

::generate partition table
%tooldir%\elf2bin.exe convert amebaz2_bootloader.json PARTITIONTABLE secure_bit=1 >>
postbuild_is_log.txt
if not exist Debug\Exe\partition.bin (
        echo partition.bin isn't generated, check postbuild_is_log.txt
        echo partition.bin isn't generated > postbuild_is_error.txt
        pause
        exit 2 /b
)

::generate firmware image
if not exist amebaz2_firmware_is.json (
        echo amebaz2_firmware_is.json is missing
        echo amebaz2_firmware_is.json is missing > postbuild_is_error.txt
        pause
```

```
        exit 2 /b
)
%tooldir%\elf2bin.exe convert amebaz2_firmware_is.json FIRMWARE secure_bit=1 >> postbuild_is_log.txt
if not exist Debug\Exe\firmware_is.bin (
        echo firmware_is.bin isn't generated, check postbuild_is_log.txt
        echo firmware_is.bin isn't generated > postbuild_is_error.txt
        pause
        exit 2 /b
)
```

For **trust zone project**, *"secure_bit"* in *postbuild_tz.bat* under *component**soc**realtek**8710c**misc**iar_utility* needs to be set to 1.

```
    %tooldir%\elf2bin.exe convert amebaz2_bootloader.json BOOTLOADER secure_bit=1 >>
    postbuild_tz_log.txt
    if not exist Debug\Exe\bootloader.bin (
        echo bootloader.bin isn't generated, check postbuild_tz_log.txt
        echo bootloader.bin isn't generated > postbuild_tz_error.txt
        goto error_exit
    )

    ::generate partition table
    %tooldir%\elf2bin.exe convert amebaz2_bootloader.json PARTITIONTABLE secure_bit=1 >>
    postbuild_tz_log.txt
    if not exist Debug\Exe\partition.bin (
        echo partition.bin isn't generated, check postbuild_tz_log.txt
        echo partition.bin isn't generated > postbuild_tz_error.txt
        goto error_exit
    )

    ::generate firmware image
    if not exist amebaz2_firmware_tz.json (
        echo amebaz2_firmware_tz.json is missing
        echo amebaz2_firmware_tz.json is missing > postbuild_tz_error.txt
        goto error_exit
    )
    %tooldir%\elf2bin.exe convert amebaz2_firmware_tz.json FIRMWARE secure_bit=1 >>
    postbuild_tz_log.txt
    if not exist Debug\Exe\firmware_tz.bin (
        echo firmware_tz.bin isn't generated, check postbuild_tz_log.txt
        echo firmware_tz.bin isn't generated > postbuild_tz_error.txt
        goto error_exit
    )
```

6.2.4.4.2 GCC

For GCC compilation, "secure_bit" needs to be set to 1 in application.is.mk under *project**realtek_amebaz2_v0_example**GCC-RELEASE* for **ignore secure project**.

```
.PHONY: manipulate_images
manipulate_images:
        @echo ============================================================
        @echo Image manipulating
        @echo ============================================================
        cp $(AMEBAZ2_BOOTLOADERDIR)/bootloader.axf $(BOOT_BIN_DIR)/bootloader.axf
ifeq ($(findstring Linux, $(OS)), Linux)
        chmod 0774 $(ELF2BIN)
endif
```

```
	$(ELF2BIN) keygen keycfg.json
	$(ELF2BIN) convert amebaz2_bootloader.json BOOTLOADER secure_bit=1
	$(ELF2BIN) convert amebaz2_bootloader.json PARTITIONTABLE secure_bit=1
	$(ELF2BIN) convert amebaz2_firmware_is.json FIRMWARE secure_bit=1
	$(ELF2BIN) combine $(BIN_DIR)/flash_is.bin
PTAB=partition.bin,BOOT=$(BOOT_BIN_DIR)/bootloader.bin,FW1=$(BIN_DIR)/firmware_is.bin
```

For **trust zone project**, *"secure_bit"* needs to be set to 1 in application.tz.mk under *\project\realtek_amebaz2_v0_example\GCC-RELEASE*.

```
.PHONY: manipulate_images
manipulate_images: | application_tz
	@echo ============================================================
	@echo Image manipulating
	@echo ============================================================
	cp $(AMEBAZ2_BOOTLOADERDIR)/bootloader.axf $(BOOT_BIN_DIR)/bootloader.axf
ifeq ($(findstring Linux, $(OS)), Linux)
	chmod 0774 $(ELF2BIN)
endif
	$(ELF2BIN) keygen keycfg.json
	$(ELF2BIN) convert amebaz2_bootloader.json BOOTLOADER secure_bit=1
	$(ELF2BIN) convert amebaz2_bootloader.json PARTITIONTABLE secure_bit=1
	$(ELF2BIN) convert amebaz2_firmware_tz.json FIRMWARE secure_bit=1
	$(ELF2BIN) combine $(TARGET)/flash_tz.bin
PTAB=partition.bin,BOOT=$(BOOT_BIN_DIR)/bootloader.bin,FW1=$(TARGET)/firmware_tz.bin
```

6.2.4.5 Secure Boot Execution

Set the "privkey_enc" and "privkey_hash" in *keycfg.json* as shown below, in this example, just change the last character of default value from *F* to *0*.

```
{
   "__comment_0":"configuration for private key, use auto to generate random key",
   "__comment_1":"private key maybe different from your desired input, program will modify first byte and
last byte of key",
   "__comment_2":"to get actual private key, please open key.json after key generated.",
   "__comment_3":"hash private key in key.json will be encrypted",
   "privkey_enc":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E50",
   "privkey_enc1":"auto",
   "privkey_hash":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E50",
   "privkey_hash1":"auto"
}
```

After rebuilding the project, the keys are updated in *key.json*.

```
{
	"__comment_EFUSE":"should keep these two key in safe place, or secure firmware protection is
useless",
	"EFUSE":{
		"__comment_privkey_enc":"this key in EFUSE SUPER SECURE ZONE",

	"privkey_enc":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E50",
		"__comment_privkey_hash":"this key in EFUSE SECURE ZONE, encrypted by upper key",

	"privkey_hash":"1C219D029C32C0744FBDAAE9BC306D0CC57F02049217C3A94D8E105D5AA264A0"
	},
	"TOOL":{
```

```
        "__comment_pubkey_enc":"public key for encryption",

    "pubkey_enc":"B496A6CF209834D1F22C7FEA41172F5888F9540B069874F4700B411E77576E03",
        "__comment_pubkey_hash":"public key for hash, only for partition table and bootloader",

    "pubkey_hash":"B496A6CF209834D1F22C7FEA41172F5888F9540B069874F4700B411E77576E03"
    }
}
```

Set the flag CONFIG_EXAMPLE_SECURE_BOOT to 1 in 'platform_opts.h'.

```
/*For secure boot example */
#define CONFIG_EXAMPLE_SECURE_BOOT 1
```

Change the keys in 'example_secure_boot.c', make susec_key[] equal to "privkey_enc" and sec_key[] equal to "privkey_hash" in *key.json*.

```
//these two keys are the same default keys used in SDK
const uint8_t susec_key[PRIV_KEY_LEN] = {
   0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
   0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
   0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
   0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x50
};
const uint8_t sec_key[PRIV_KEY_LEN] = {
   0x1C, 0x21, 0x9D, 0x02, 0x9C, 0x32, 0xC0, 0x74,
   0x4F, 0xBD, 0xAA, 0xE9, 0xBC, 0x30, 0x6D, 0x0C,
   0xC5, 0x7F, 0x02, 0x04, 0x92, 0x17, 0xC3, 0xA9,
   0x4D, 0x8E, 0x10, 0x5D, 0x5A, 0xA2, 0x64, 0xA0
};
```

Enable write SS key function, write S key function, lock SS key function and secure boot function by setting if condition as 1 (details are shown in section 6.2.4.2). Build the application and download it to Ameba-ZII to enable secure boot.
If secure boot is enabled successfully, the message will be:

```
#
efuse secure boot: Test Start
[0]     FF FF FF FF  FF FF FF FF
[8]     FF FF FF FF  FF FF FF FF
[16]    FF FF FF FF  FF FF FF FF
[24]    FF FF FF FF  FF FF FF FF

Write Done.

[0]     00 01 02 03  04 05 06 07
[8]     08 09 0A 0B  0C 0D 0E 0F
[16]    10 11 12 13  14 15 16 17
[24]    18 19 1A 1B  1C 1D 1E 50
[0]     FF FF FF FF  FF FF FF FF
[8]     FF FF FF FF  FF FF FF FF
[16]    FF FF FF FF  FF FF FF FF
[24]    FF FF FF FF  FF FF FF FF

Write Done.

[0]     1C 21 9D 02  9C 32 C0 74
[8]     4F BD AA E9  BC 30 6D 0C
[16]    C5 7F 02 04  92 17 C3 A9
[24]    4D 8E 10 5D  5A A2 64 A0
efuse secure boot keys: Test Done
eFuse Key Locked!!, Super-Secure Key Reading is Inhibited!!
secure boot is enabled!
```

Enabling secure boot successfully means only encrypted image can boot on this Ameba-ZII board.

To run your own application, you need to encrypt image by setting the "*enc*" as ***true*** in '*amebaz2_bootloader.json/amebaz2_firmware_is.json* under *project\realtek_amebaz2_v0_example\EWARM-RELEASE*'

Also, set "**secure_bit=1**" in 'postbuild_is.bat' under '*component\soc\realtek\8710c\misc\iar_utility*' (details are shown in section 6.2.4.4).

Finally, the Ameba-ZII board can run with encrypted image.

6.2.5 How to Debug in Secure Boot in IAR IDE

Once secure boot is successfully enabled, a super sec key is obtained. When trying to debug the program in IAR IDE, issue arises because of the default setup in IAR IDE. During debug, IAR IDE performs a CPU reset which clears the super sec key. When operation resume, the super sec key will be read again, resulting in boot failure because the super sec key can be read only once unless there is a power reset, or watchdog reset. This issue happens in both "Download and Debug" and "Debug without Downloading". Thus, it is not recommendable to debug the program using these two methods.

Note: GCC SDK doesn't have this issue because it can be fully controlled by our GDB script.

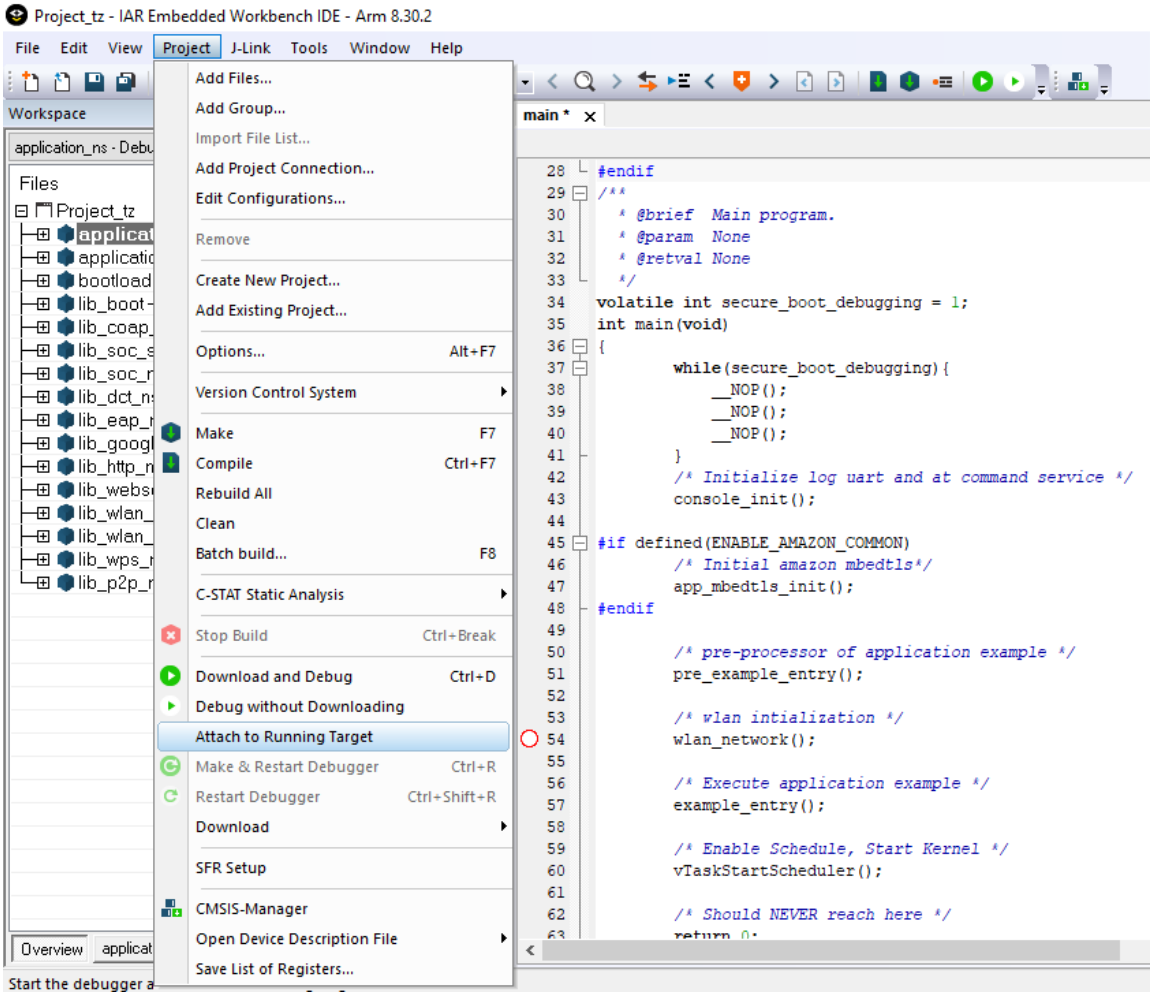
However, one alternative to debug the program is as follow:

Firstly, add this piece of code into main.c and make the project.

```
/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
volatile int secure_boot_debugging = 1;
int main(void)
{
        while(secure_boot_debugging){
          __NOP();
          __NOP();
          __NOP();
        }
        /* Initialize log uart and at command service */
        console_init();
```

Secondly, download the image binary on to the demo board from IAR IDE as shown in chapter 3.4.2.1.3. Afterwards, reset the demo board and you will observe that the program is halt before executing the main program in main.c.

Thirdly, debug the program using “Attach to Running Target” as shown below.



When everything mentioned above is completed, please follow the instructions below.

Stop the program with “Break” in the debug tab, and please add a breakpoint at "consol_init()”in main.c. The breakpoint is used to ensure that the program will break right after the while loop, and to prevent the program to run to the end. Then, change the value of “secure_boot_debugging” to 0 in the Watch window to exit the while loop. Now, you can continue to debug the program.

Note: In situation where you want to reset the debugging process, please do not use the reset button under the debug tab. Instead, stop the debugging process, reset the demo board and debug using “Attach to Running Target” again.

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.

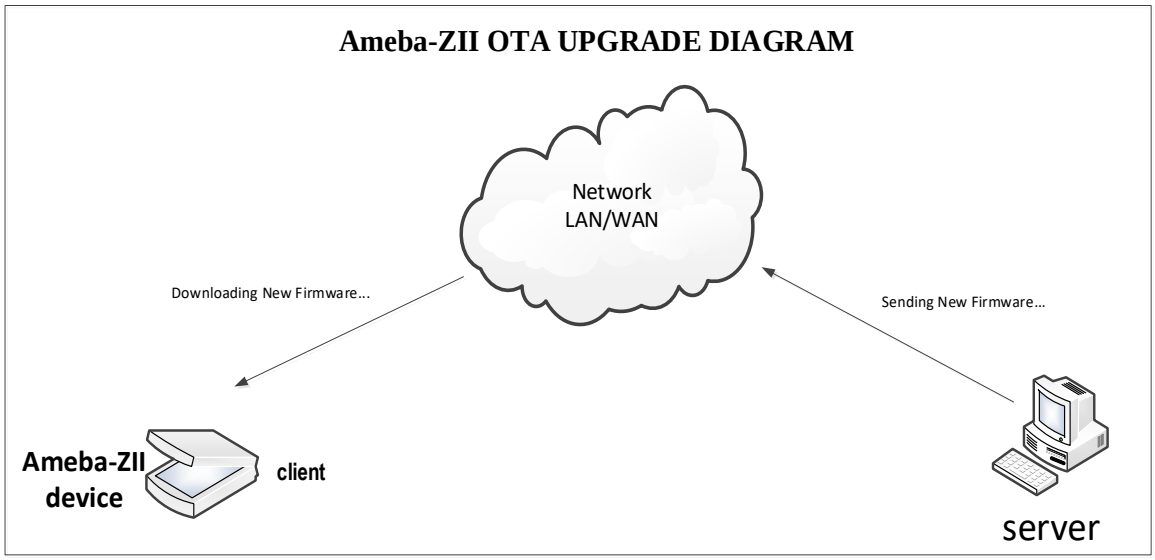


Figure 7-1 Methodology to Update Firmware via OTA

7.1 OTA Operation Flow

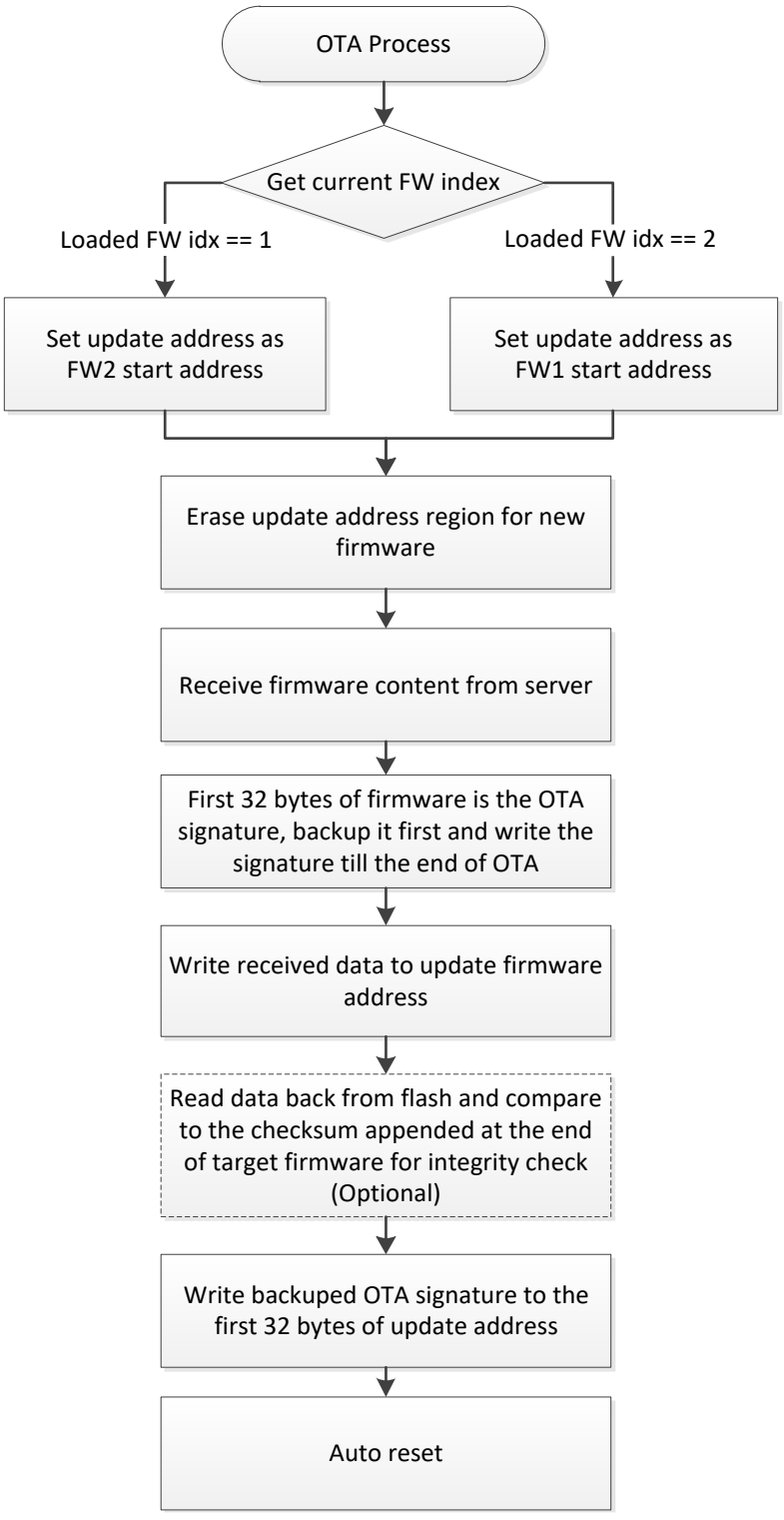


Figure 7-2 OTA Process Flow

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 notice the bootloader the overall OTA process is done without any network disconnection or re-boot during the OTA. However, this 32 bytes OTA signature cannot guarantee the content of firmware image is correct.

User 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 enabled. In such way, it can ensure the downloaded firmware is transferred completely and correct. For the detail 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 generate CHECKSUM, please refer to below files.

IAR: (component\soc\realtek\8710c\misc\iar_utility\postbuild_is.bat or postbuild_tz.bat)

```
%tooldir%\elf2bin.exe convert amebaz2_firmware_is.json FIRMWARE secure_bit=0 >>
postbuild_is_log.txt
if not exist Debug\Exe\firmware_is.bin (
    echo firmware_is.bin isn't generated, check postbuild_is_log.txt
    echo firmware_is.bin isn't generated > postbuild_is_error.txt
    goto error_exit
)

:: generate firmware ota image
%tooldir%\checksum.exe Debug\Exe\firmware_is.bin

::generate flash image, including partition + bootloader + firmware
%tooldir%\elf2bin.exe combine Debug/Exe/flash_is.bin
PTAB=Debug\Exe\partition.bin,BOOT=Debug\Exe\bootloader.bin,FW1=Debug\Exe\firmware_is.bin >>
postbuild_is_log.txt
if not exist Debug\Exe\flash_is.bin (
    echo flash_is.bin isn't generated, check postbuild_is_log.txt
    echo flash_is.bin isn't generated > postbuild_is_error.txt
    goto error_exit
)
```

GCC: (project\realtek_amebaz2_v0_example\GCC-RELEASE\application.is.mk or application.tz.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.

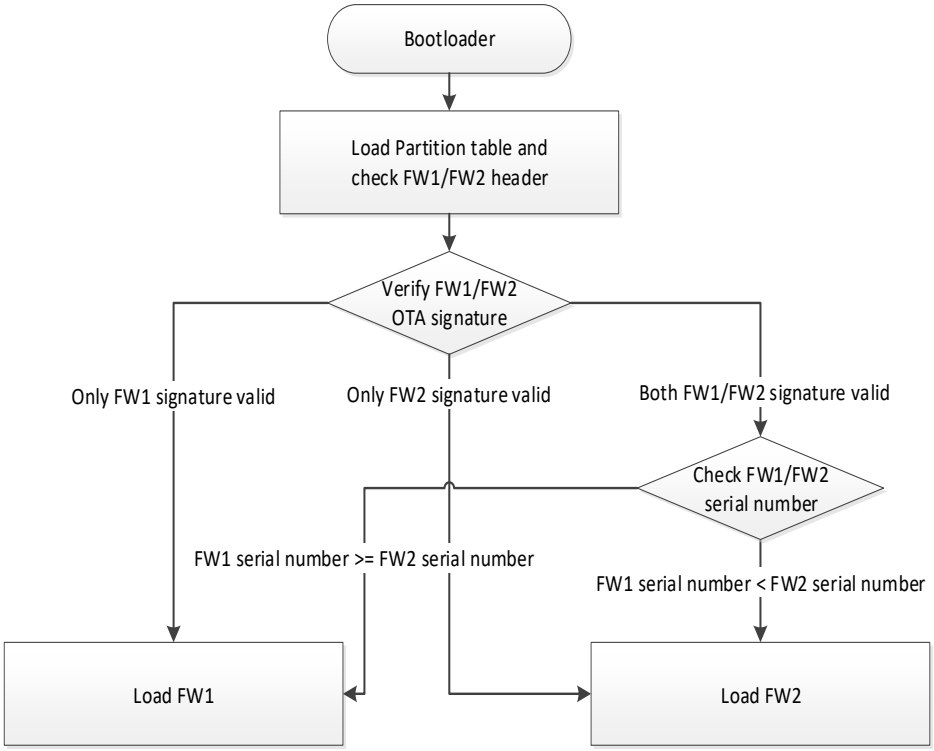


Figure 7-3 Boot Process Flow

7.4 Upgraded Partition

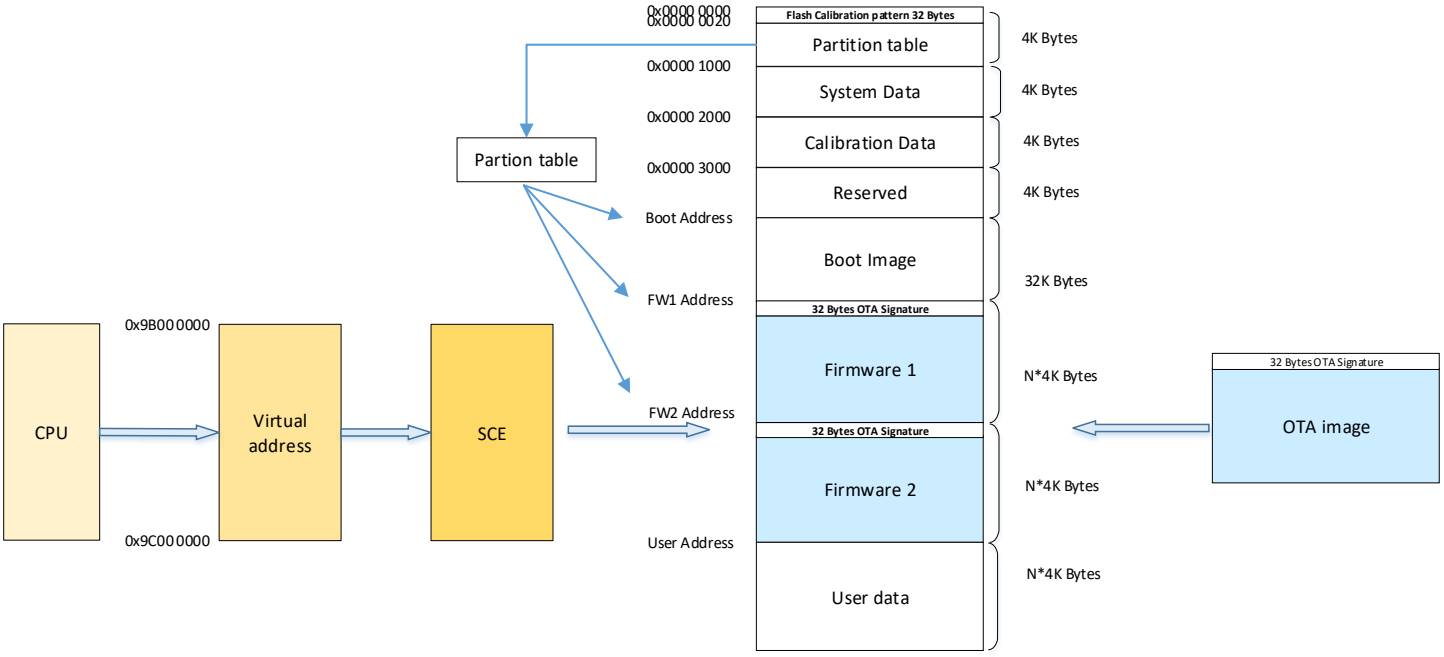


Figure 7-4 OTA update procedure

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\EWARM-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.

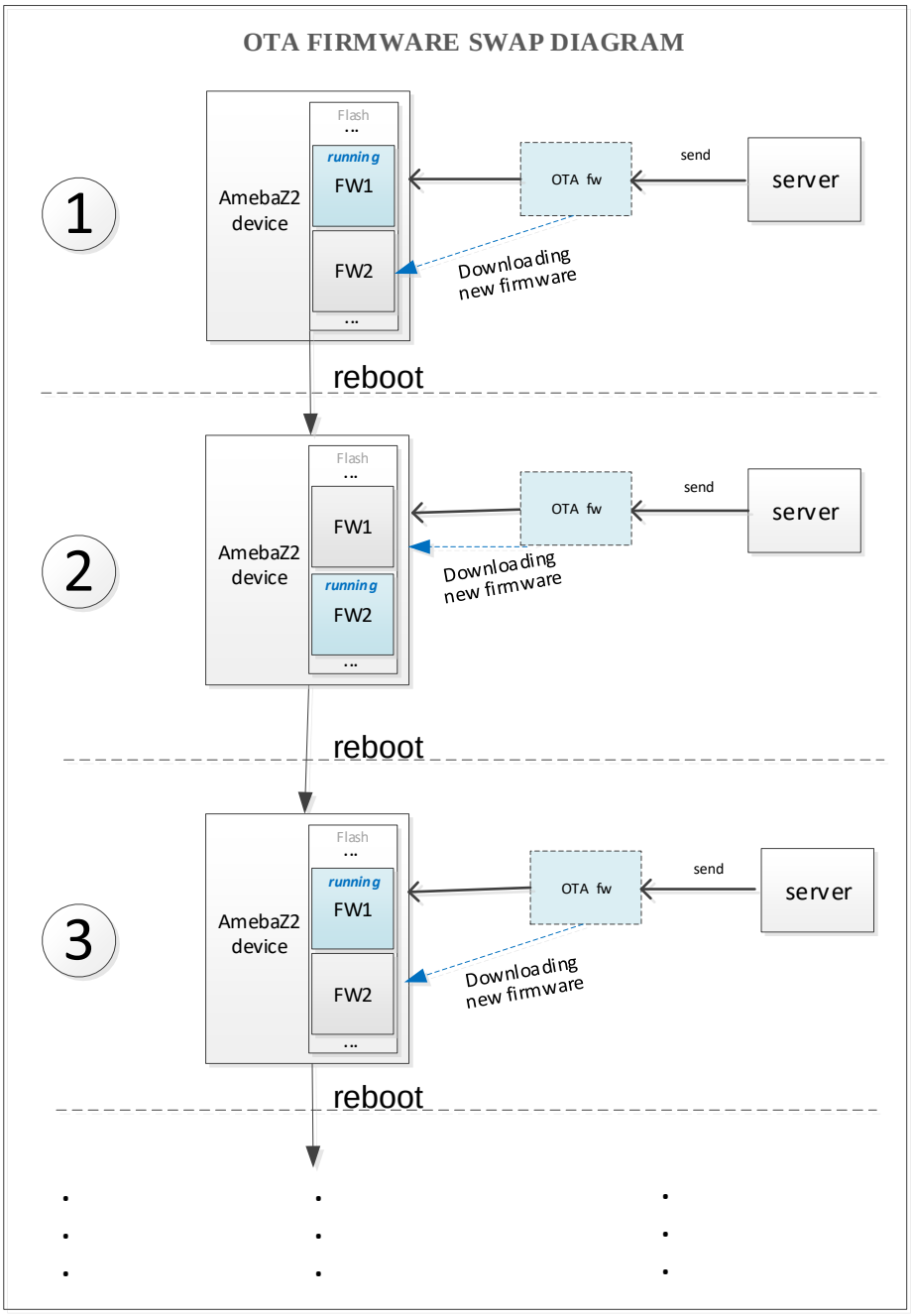


Figure 7-5 OTA Firmware SWAP Procedure

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 need 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 as same as the serial number of its first image. You can check the images sequence in p*roject\realtek_amebaz2_v0_example\EWARM-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":"Debug/Exe/application_is.out",
                "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 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/EWARM-RELEASE/Debug/Exe/firmware_is.bin*, which is the application of OTA Firmware. The serial information would also be included in this firmware.

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

Step 1: The serial number setting of a firmware is as same as the serial number of its first image. You can check the images sequence in p*roject\realtek_amebaz2_v0_example\EWARM-RELEASE\amebaz2_firmware_tz.json*.

```
"FIRMWARE":{
                "images":[
                        {"img": "FWHS_S", "offset":"0x00"},
                        {"img": "FWHS_NSC", "offset":"0x00"},
                        {"img": "FWHS_NS", "offset":"0x00"},
                        {"img": "XIP_S_C", "offset":"0x00"},
                        {"img": "XIP_S_P", "offset":"0x00"},
                        {"img": "XIP_NS_C", "offset":"0x00"},
                        {"img": "XIP_NS_P", "offset":"0x00"}
                ]
        },
```

For this example, the FWHS_S 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_S**”:

```
"FWHS_S": {
                "source":"Debug/Exe/application_s.out",
```

```
                "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
                },
```

Step 3: After building project source files in SDK, it should automatically generate */project/project_name/EWARM-RELEASE/Debug/Exe/firmware_tz.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 send 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 to 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 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 receiving 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 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.

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

Waiting for client ...
```

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.

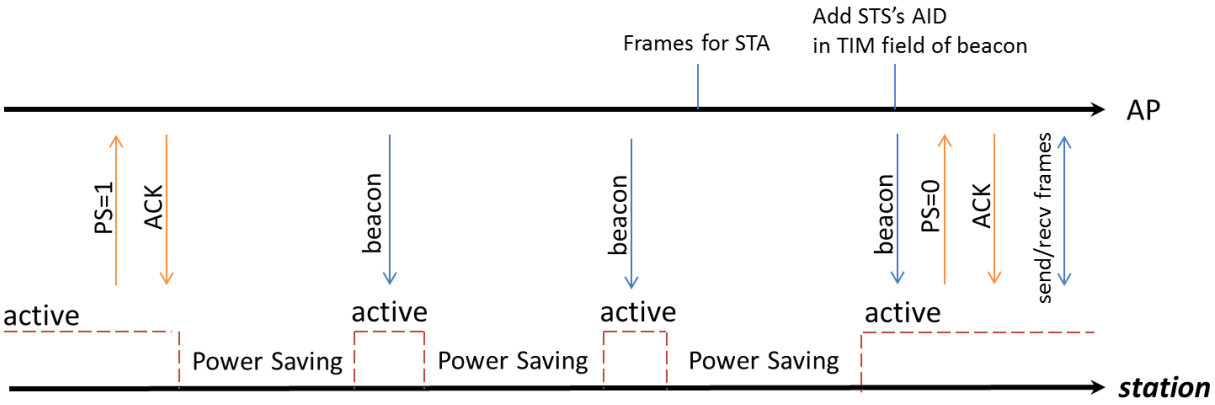


Figure 8-1 timeline of power saving

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.

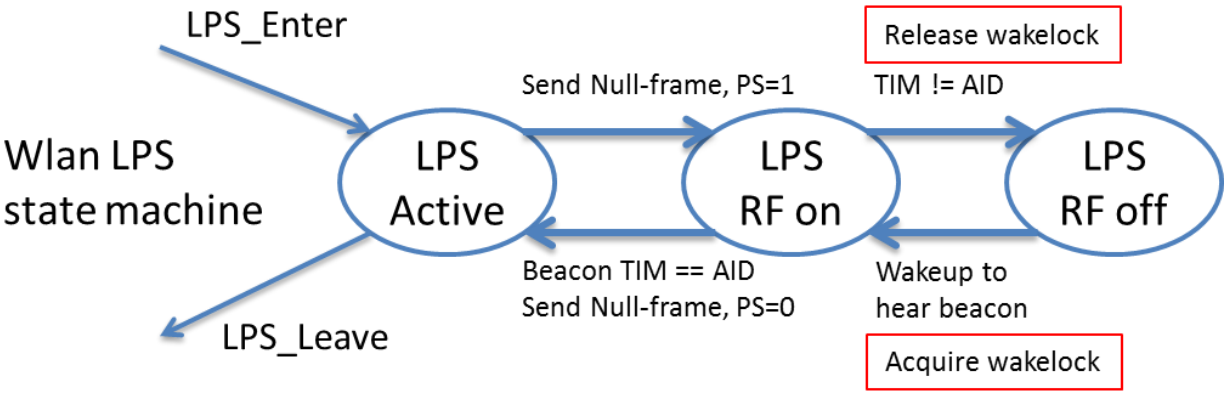


Figure 8-2 LPS state machine

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 situation that Ameba is associate to an AP. If Ameba is not associated to an AP, 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).

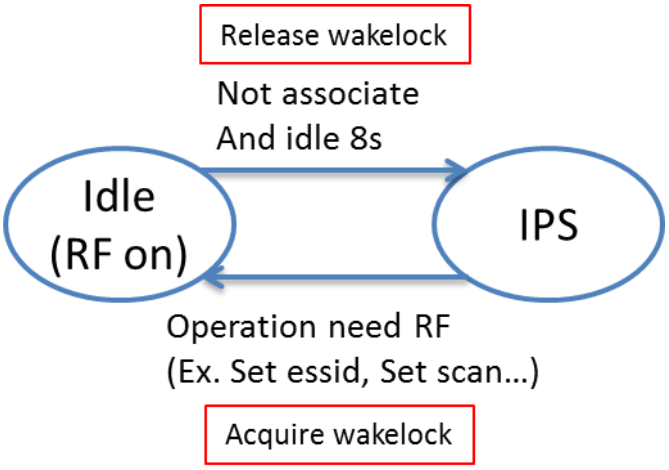


Figure 8-3 IPS state machine

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.

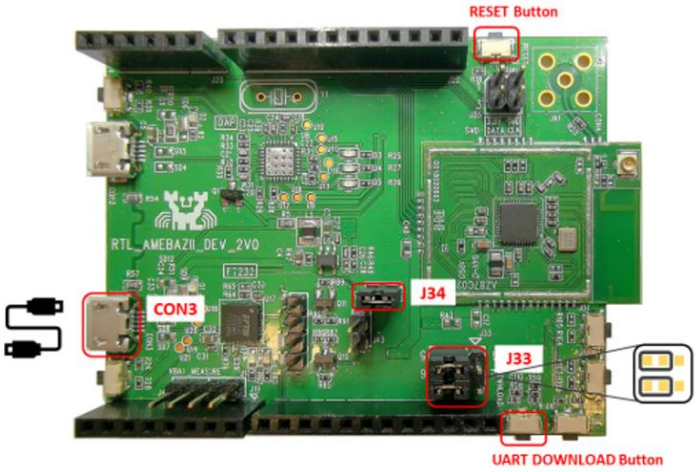


Figure 8-4 Power consumption measurement

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

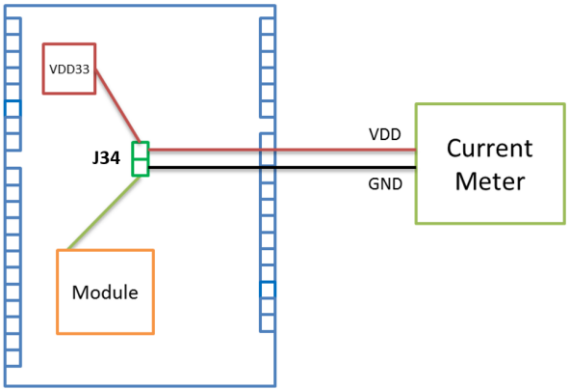


Figure 8-5 Measure power consumption from micro usb

8.2.2 Build SDK

Below are suggesting 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 eFuse

9.1 Introduction

eFuse belongs to One Time Programmable (OTP) technology, its default value is '1', and can only be changed from '1' to '0'. eFuse can be used to hold the individual and stable data such as key, calibration data, MAC address and specific setting.
The total size of physical eFuse is 512 bytes and divided into two parts by software. The first 256 bytes of physical eFuse are used for logical mapping, which can be mapped to logical eFuse by some algorithm and can be programmed multiple times. The remaining 256 bytes of the physical eFuse are defined by Realtek.
Logical eFuse program will program header and package, and the package is in word, so it takes at least three bytes a time. Software reserves one byte for isolation, so when the space of eFuse for logical mapping is less than four bytes, it cannot be programmed in any mode.

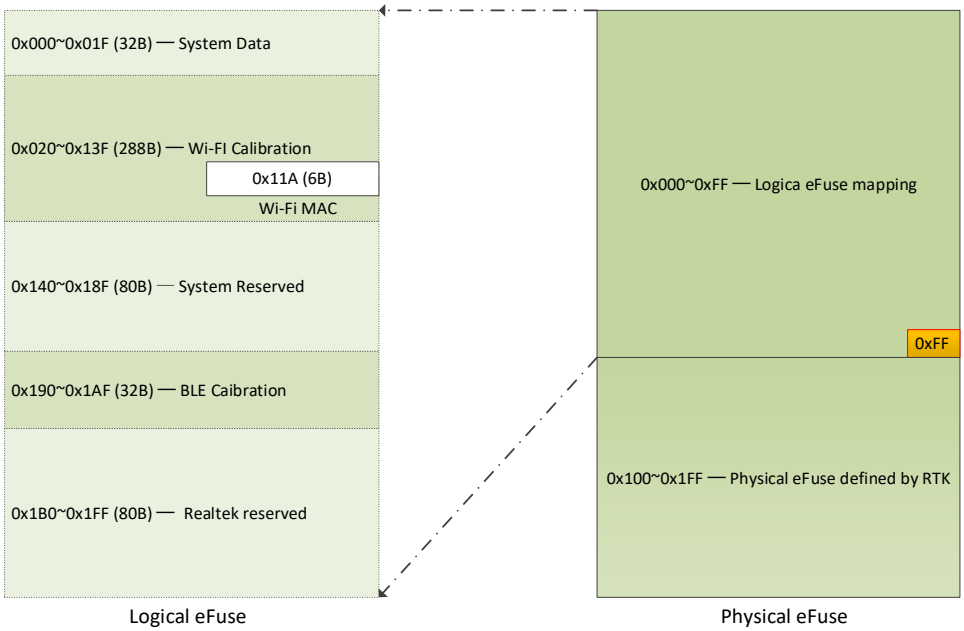


Fig 9-1 eFuse Diagram

9.2 Power Requirement

The power requirement of eFuse operation is listed in the Table 9-1 below.

Supply Voltage (V)	Operation
1.146	Only read operation is allowed
2.5	Program eFuse

Table 9-1 Operation under different voltage

Note: If you want to program eFuse, you must switch the power supply to 2.5V

9.3 eFuse Auto-load

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

- In the logical eFuse, make sure that the third and fourth bytes of System Data area are written to 0x10 and 0x87 correctly.
- Program the corresponding bytes in word. If just programming in byte, another byte will load the eFuse default value 0xFF, which may make mistakes.
- Reboot the chip, and the System Configure Registers will load the new values.

9.4 Logical eFuse

9.4.1 Logical eFuse Layout

There is total 512 bytes logical eFuse in Ameba-ZII, as shown in the Table 9-2 below. Most of the logical eFuse space are reserved for extension, so the 512 bytes of logical eFuse can be easily mapped to 256 bytes of physical eFuse.

Start Address	End Address	Description
0x000	0x01F	System Data to be auto-loaded
0x020	0x13C	Wi-Fi Calibration data
0x140	0x18F	System reserved
0x190	0x1AD	Bluetooth Calibration data
0x1B0	0x1FF	Realtek reserved

Table 9-2 Logical eFuse Layout

9.4.2 Wi-Fi 2.4G Power Index

The addresses and specifications of 2.4G power index are shown in Fig 9-2 and Fig 9-3.

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

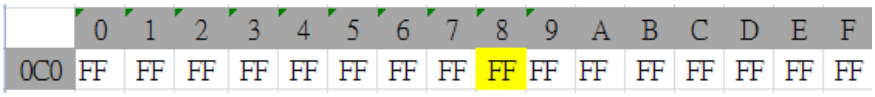
Fig 9-2 2.4G Wi-Fi power index address

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

Fig 9-3. 4G Wi-Fi power index specifications

9.4.3 Wi-Fi Channel Plan

The address of Wi-Fi channel plan is shown in Fig 9-4.



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

Fig 9-4 Address of Wi-Fi channel plan

eFuse	Name	Bit	Default	Comment
0x0C8	Channel Plan	[7]	0h	Software configure mode • 0h: Enable software configure (refer to Channel Plane Domain Code) • 1h: Disable software configure(cannot change Channel Plan Setting)
		[6:0]	7Fh	Channel Plan • 20h: 2G worldwide13, channel 1~13 • 21h: 2G Europe, channel 1~13 • 22h: 2G US, channel 1~11 • 23h: 2G Japan, channel 1~13,14 • 24h: 2G France, channel 10~13 • 2Ah: 2G US, channel 1~13 • 58h: 2G Japan, channel 1~13

Frequently used country codes and the corresponding channel plans are listed in Table 9-3 and Table 9-4.

Country	Abbreviation	Enumeration Variable name	Channel Plan
America	US	RTW_COUNTRY_US	0x76
Australia	AS	RTW_COUNTRY_AS	0x2A
Chile	CL	RTW_COUNTRY_CL	0x2D
China	CN	RTW_COUNTRY_CN	0x48
Europe	EU	RTW_COUNTRY_UA	0x26
Japan	JP	RTW_COUNTRY_JP	0x27
Mexico	MX	RTW_COUNTRY_MX	0x4D
Ukraine	UA	RTW_COUNTRY_UA	0x35
Default	-	RT_CHANNEL_DOMAIN_REALTEK_DEFINE	0x7F

Table 9-3 Relationship between country code and channel plan

Channel Plan	2.4G Channel Definition	Channels	Passive
0x76	2G_02	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	-
0x2A	2G_02	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	-
0x2D	2G_01	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	12,13
0x48	2G_01	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	12,13
0x26	2G_01	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	12,13
0x27	2G_04	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	-
0x4D	2G_02	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	-
0x35	2G_01	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	12,13
0x7F	2G_01	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	12,13

Table 9-4 2.4G channel map of the above channel plan

Note: Active scan channels are channel listed in table except the passive scan channels.

9.4.4 Wi-Fi Crystal Calibration

The address of Wi-Fi crystal calibration is shown in Fig 9-5.

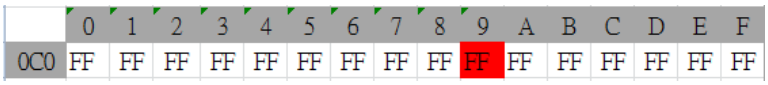


Fig 9-5 Crystal calibration address

eFuse	Name	Bit	Default	Comment
0x0C9	Crystal Calibration	[7]	0h	• Reserved
		[6:0]	3Fh	XTAL_K Value • Xi=Xo, range 0~7Fh

9.4.5 Wi-Fi Thermal Meter

The address of Thermal Meter is shown in Fig 9-6

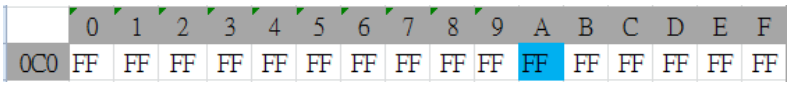


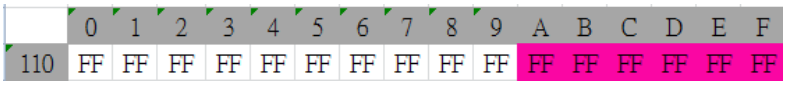
Fig 9-6 Adress of Thermal Meter

eFuse	Name	Bit	Default	Comment

0x0CA	Thermal Meter	[7:0]	20h	Thermal Meter Default Value • System maker will calibrate a value and save it in EEPROM. • Bit[7:0]: Thermal Meter Value

9.4.6 Wi-Fi MAC Address

The Wi-Fi MAC address is shown in Fig 9-7.



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

Fig 9-7 MAC address

eFuse	Bit	Default	Comment
0x0CA	[47:0]	FFFFFFFFFFFF	After the auto-load command or hardware reset, Ameba-ZII loads MAC addresses to MACID of the I/O registers

9.4.7 BLE

Address Offset	Bit	Default	Description
190h ~ 0x195h	[47:0]	FFFFFFFFFFFFh	BT address
196h	[7:0]	08h	Function Enable • Bit[7:0] : Default is 0x08. • Bit[0] : Fix 0. • Bit[1] : Tx gain K valid bit* . • Bit[2] : Flatness K valid * . • Bit[3] : Fix 1. • Bit[7:4] : Fix 0. **Note:** • After Tx gain K flow please remember to enable this bit which 0x196 will be 0x0A. • After Tx flatness K flow, please remember to enable this bit which 0x196 will be 0x0C.
197h	[7:0]	FFh	Bit[7:0] : Tx Gain K. 0.5dBm/step. **Note:** • It's reminded that customer need to write Tx Gain K value if Tx gain K valid bit is enabled.
198h~199h	[7:0]	FFh	Flatness K
19Ah~19Bh	[7:0]	FFh	Reserved for Realtek
19Ch	[7:0]	2Ah	Max TX Gain LE1M
19Dh	[7:0]	FFh	Reserved for Realtek
19Eh~19Fh	[15:0]	FFFFh	Reserved for Realtek
1A0h	[7:0]	FFh	BT Thermal Meter
1A1h	[7:0]	FFh	Logic map IQK/KOK valid bit • FF: Disable logic map IQK/KOK values • FE: Enable logic map IQK/KOK values

1A2h~1A5h	[31:0]	FFFFFFFFh	Reserved for Realtek
1A6h~1A9h	[31:0]	FFFFFFFFh	Logic map IQK values
1AAh~1ABh	[15:0]	FFFFh	Logic map LOK values

9.5 eFuse PG APIs

Items	API	Description
Mbed API	efuse_get_remaining_length	Get remaining efuse length
	efuse_otp_read	Read efuse OTP contant
	efuse_otp_write	Write user's contant to OTP efuse
	efuse_otp_chk	Check user's contant to OTP efuse
	efuse_disable_jtag	Disable jtag
	efuse_disable_sec_jtag	Disable secure jtag
	efuse_disable_nonsec_jtag	Disable nonsecure jtag
	efuse_sec_key_write	Write secure key to physical efuse
	efuse_susec_key_write	Write super secure key to physical efuse
	efuse_s_jtag_key_write	Write secure j-tag key to physical efuse
	efuse_ns_jtag_key_write	Write non-secure j-tag key to physical efuse
	efuse_lock_susec_key	Lock super secure key
Low Level API	efuse_logical_read	Read efuse content on logical map
	efuse_logical_write	Write user's content to efuse on logical map
	efuse_fw_verify_enable	To enable secure boot
	efuse_fw_verify_check	To check the secure boot is enabled or not

9.5.1 Mbed APIs

9.5.1.1 efuse_get_remaining_length

Items	Description
Introduction	Get remaining efuse length
Parameters	
Return	• remaining efuse length

9.5.1.2 efuse_otp_read

Items	Description
Introduction	Read efuse OTP content.
Parameters	• address: Specifies the offset of the OTP. • len: Specifies the length of readback data. buf: Specified the address to save the readback data.
Return	• 0: Success • -1: Failure

9.5.1.3 efuse_otp_write

Items	Description
Introduction	Write user's content to OTP efuse
Parameters	• address: Specifies the offset of the programmed OTP.

	• len: Specifies the data length of programmed data. buf: Specified the data to be programmed.
Return	• 0: Success • -1: Failure

9.5.1.4 efuse_otp_chk

Items	Description
Introduction	Check user's content to OTP efuse
Parameters	• buf: Specified the data to be programmed. • len: Specifies the data length of programmed data.
Return	• 0: Success • -1: Failure

9.5.1.5 efuse_disable_jtag

Items	Description
Introduction	Disable jtag
Parameters	
Return	• 0: Success

9.5.1.6 efuse_disable_sec_jtag

Items	Description
Introduction	Disable secure jtag
Parameters	
Return	• 0: Success

9.5.1.7 efuse_disable_nonsec_jtag

Items	Description
Introduction	Disable nonsecure jtag
Parameters	
Return	• 0: Success

9.5.1.8 efuse_sec_key_write

Items	Description
Introduction	Write secure key to efuse.
Parameters	• buf: specified the 32-byte security key to be programmed. • key_num: select key number.
Return	• 0 Success • -1 Failure

9.5.1.9 efuse_susec_key_write

Items	Description

Introduction	Write super secure key to efuse
Parameters	• buf: Specified the 32-byte super security key to be programmed.
Return	• 0 Success • -1 Failure

9.5.1.10 efuse_lock_susec_key

Items	Description
Introduction	Lock super secure key
Parameters	
Return	• 0 Success • -1 Failure

9.5.1.11 efuse_s_jtag_key_write

Items	Description
Introduction	Write secure j-tag key to efuse
Parameters	• buf: Specified the 16-byte S-JTAG key to be programmed.
Return	• Success • -1 Failure

9.5.1.12 efuse_ns_jtag_key_write

Items	Description
Introduction	Write non-secure j-tag key to efuse
Parameters	• buf: Specified the 16-byte S-JTAG key to be programmed.
Return	• Success • -1 Failure

9.5.2 Low Level APIs

9.5.2.1 efuse_logical_read

Items	Description
Introduction	Read efuse content on logical map
Parameters	• laddr: address on logical map • size: size of wanted data • pbuf: buffer of read data
Return	• return number of used bytes

9.5.2.2 efuse_logical_write

Items	Description
Introduction	Write user's content to efuse on logical map
Parameters	• addr: address on logical map • cnts: how many bytes of data

	• data: data need to be written
Return	• 0 Success • <0 Failure

9.5.2.3 efuse_fw_verify_enable

Introduction	To enable secure boot
Parameters	
Return	• 0 Success • <0 Failure

9.5.2.4 efuse_fw_verify_check

Introduction	To check the secure boot is enabled or not
Parameters	
Return	• 1 Success • 0 Failure

10 Bluetooth

10.1 Features

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

10.2 BT Wi-Fi Coexist

Since Wi-Fi and BT share same RF block, so make sure do not enable wifi power save when BT is enabled. When BT is enabled, wifi_disable_powersave() API will be called, and do not call wifi_enable_powersave() API when BT is on.

10.3 Memory Usage

Since Wi-Fi and BT share same RF block, so to enable BT, it is required to enable Wi-Fi. The following is the memory usage of Wi-Fi only and Wi-Fi + BT.

10.3.1 Wi-Fi Only

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

10.3.2 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

10.4 Examples

10.4.1 ble_peripheral

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

10.4.1.1 *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.

10.4.1.2 *Test Procedure*

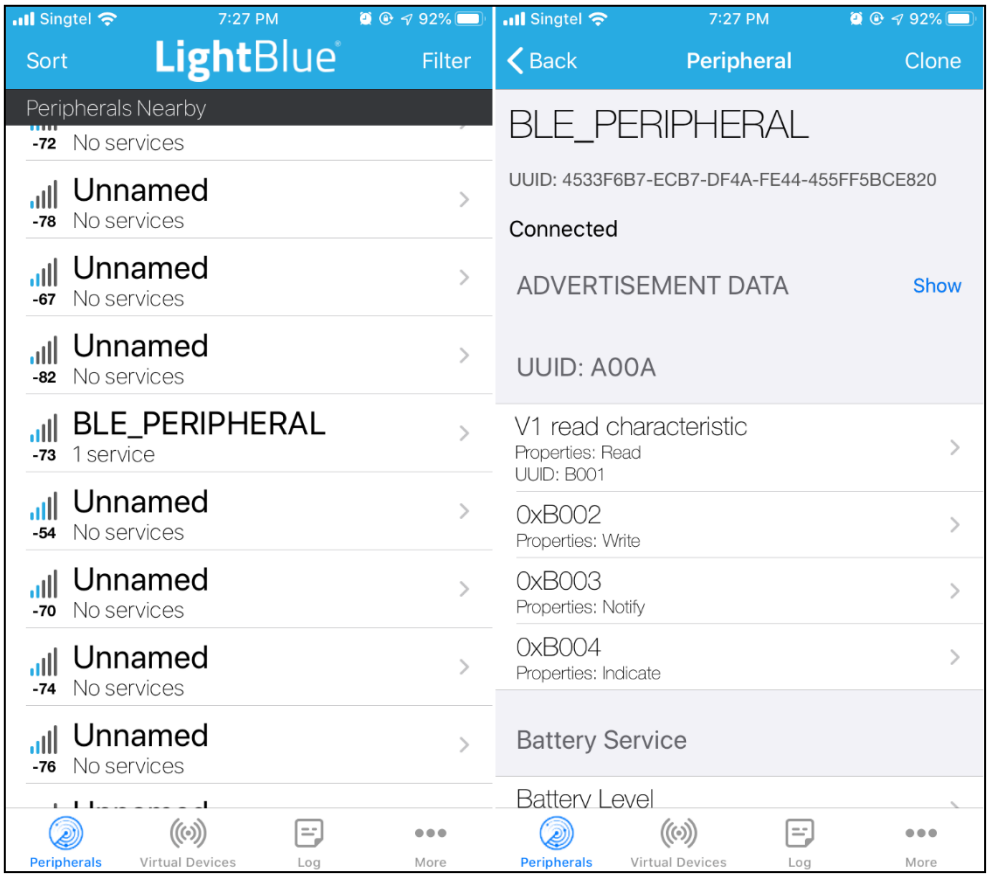
(1) After download image to your Ameba-ZII board, reset it. The default device name is BLE_PERIPHERAL.

(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.



10.4.2 ble_central

This example shows how to discover service on GATT server.

10.4.2.1 *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
```

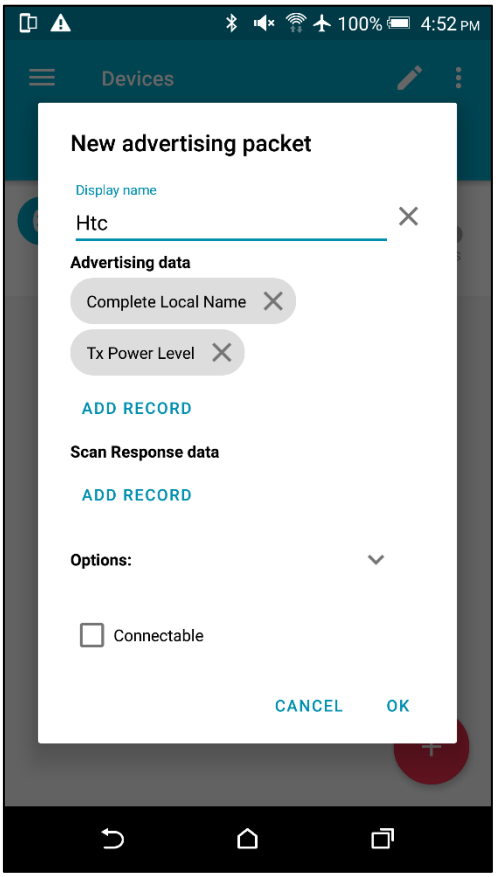
(2) Build image and download image to your board.

10.4.2.2 *Test Procedure*

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

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

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



(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***'.

10.4.3 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.

10.4.3.1 *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.

10.4.3.2 *Test Procedure*

(1) After download 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***'.

10.4.4 bt_beacon

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

10.4.4.1 *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
```

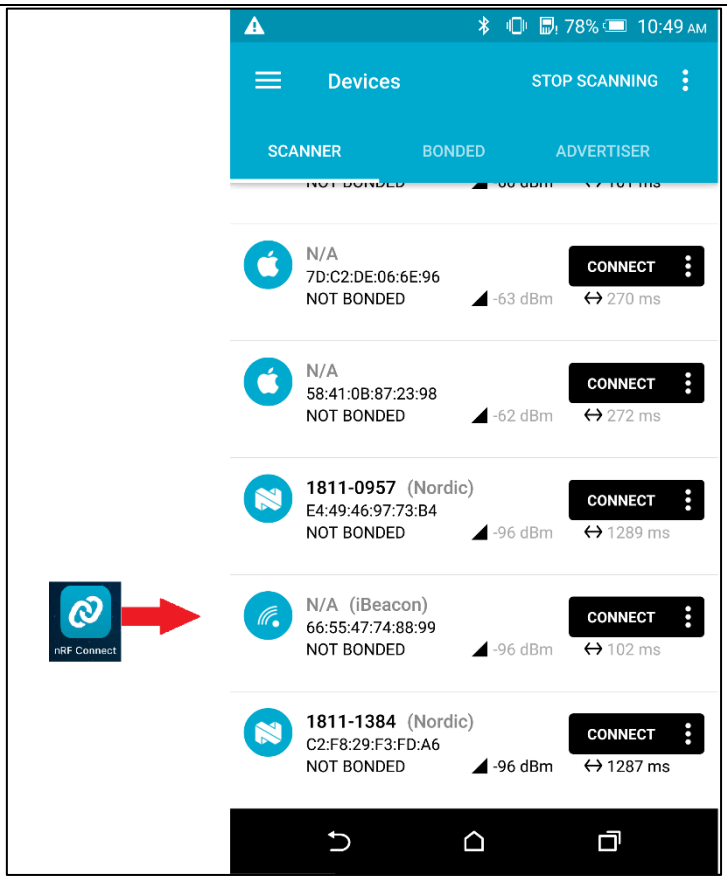
(2) Build image and download image to your board.

10.4.4.2 *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".



10.4.5 bt_config

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

10.4.5.1 *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.

10.4.5.2 *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.

10.4.5.3 *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”.

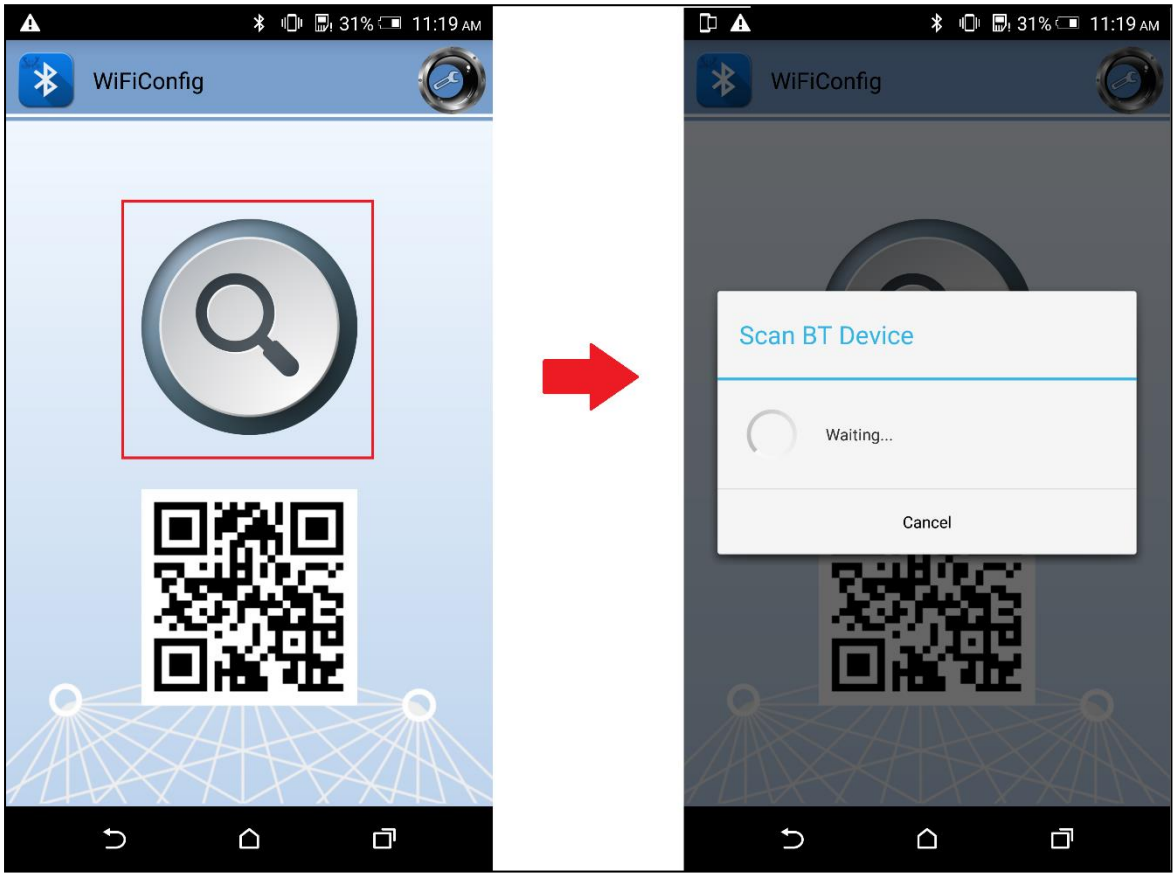
(3) Once see the following message, you can open BT Config APP to associate AP.

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:

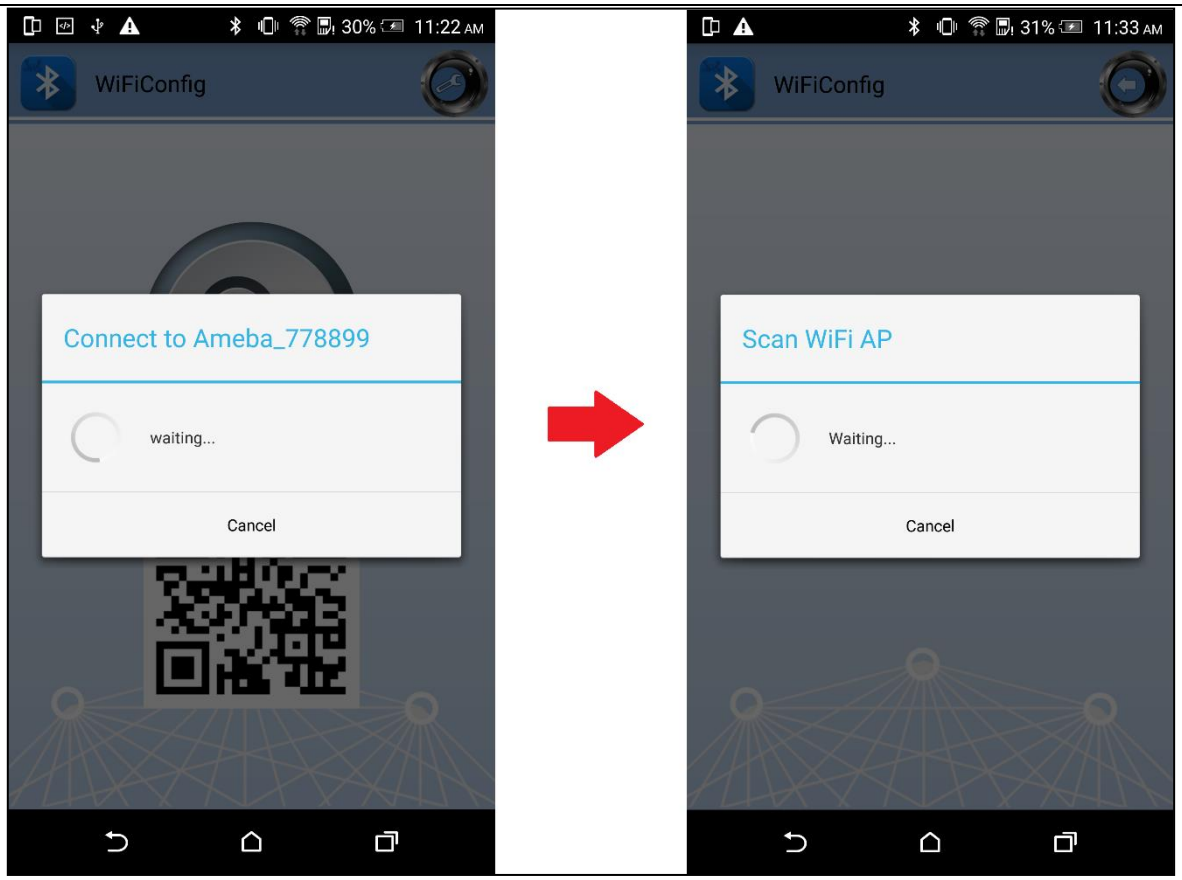


(5) Once BT Config APP connected to AmebaZII, below log will be show. When connection is established AmebaZII will start searching for AP.

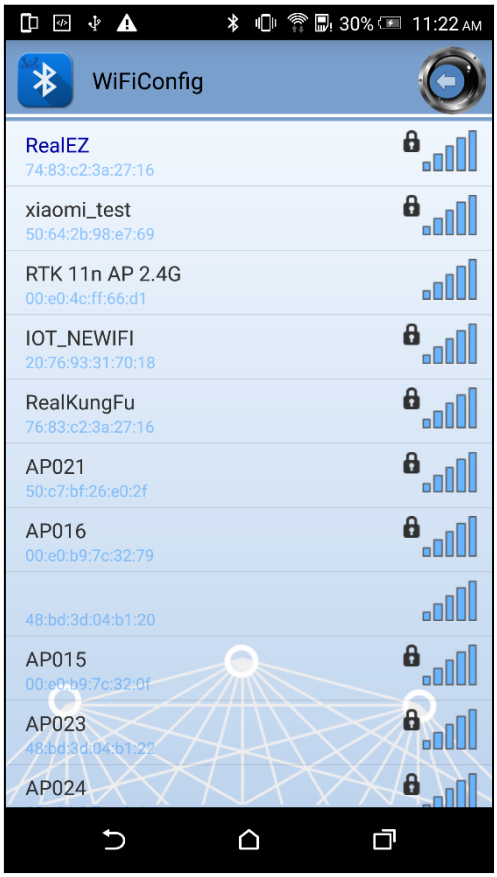
BT Connection log:

```
[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:



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

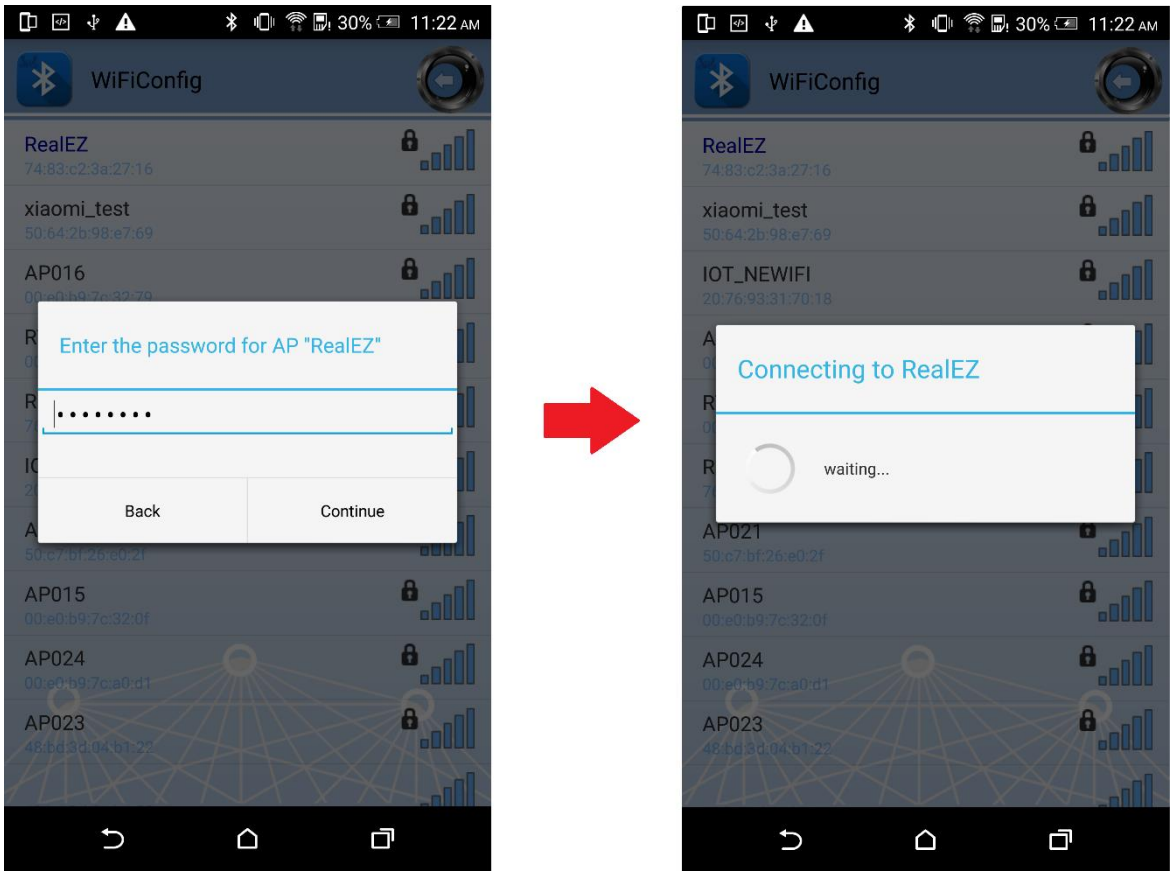


(6) Select an AP to connect to and input password (if any).

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.
```

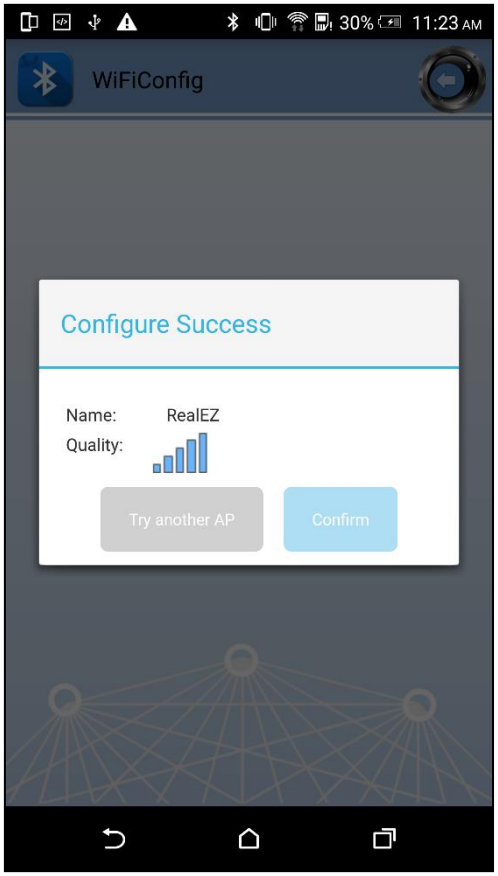
Display on BT config app:



(7) When AmebaZII is connected to an AP, user 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:



(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.

10.4.6 128-bit UUID Configuration

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

};
```

10.5 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.

10.6 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.

11 Troubleshooting

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

11.1 Hard Fault

AmebaZ2 platform provides a detail back trace information when a hard fault exception happens. Please refer to the following approach to see the full back trace which will help debugging a lot.

11.1.1 IAR Environment

If you are using IAR IDE to develop, build project and then encounter a hard fault error. You need to install additional GCC toolchain in order to utilize '**arm-none-eabi-addr2line.exe'** to back trace hard fault error based on the generated back trace information.

11.1.1.1 Download and Install GCC Toolchain for Windows

1). Please refer to the link and follow the step to download and install GCC toolchain for Windows.
   a. https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
2). After installation, arm-none-eabi-addr2line.exe can be found in below folder.
   a. $INSTALL_PATH/bin

11.1.1.2 Trace Hard Fault

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

```
S-Domain Fault Handler: msp=0x1003f998 psp=0x1002bf70 lr=0xffffffff1
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   = 0xffffffff1
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
```

User needs to check the last sentence of the hard fault (highlighted in yellow).

1). Open **CMD** window, go to the path of "**application_is.dbg.out**" which should be under /project/realtek_amebaz2_v0_example/EWARM-RELEASE/Debug/Exe. (If for trust zone project, please replace "application_is" by "application_tz".)

2). Use installed **arm-none-eabi-addr2line.exe** to get the back trace.

$INSTALL_PATH/bin/arm-none-eabi-addr2line.exe -e application_is.dbg.out -a -f 9b01b7d0 9b0465dd 0001869b 9b005955

The result will be

```
/cygdrive/d/v7.1a/project/realtek_amebaz2_v0_example/EWARM-
RELEASE/Debug/Exe
$ /cygdrive/d/GNU Tools ARM Embedded/8 2018-q4-major/bin/arm-none-
eabi-addr2line.exe -e application_is.dbg.out -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
:205
0x0001869b
??
??:0
0x9b005955
xPortStartScheduler
D:\v7.1a\component\os\freertos\freertos_v10.0.1\Source\portable\IAR\A
RM_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 in D:\v7.1a\component\os\freertos/freertos_service.c:139.

11.1.2 GCC Environment

11.1.2.1 Install Cygwin

Please refer to section *"3.5.1 Install Cygwin"*.

11.1.2.2 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.

11.1.2.3 Trace Hard Fault

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

```
S-Domain Fault Handler: msp=0x1003f998 psp=0x1002bf70 lr=0xffffffff1
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.dbg.axf**' which under /project/realtek_amebaz2_v0_example/GCC-RELEASE/application_is/Debug/bin. (If for trust zone project, please replace "application_is" by "application_tz".)

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.dbg.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.dbg.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\IAR\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 in D:\v7.1a\component\os\freertos/freertos_service.c:139.

11.2 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.