REALTEK

AN0300

AmebaPro application note

Abstract

AmebaPro is a high-integrated IC. Its features include 802.11 Wi-Fi, H.264 video codec, Audio Codec.
This manual introduce users how to develop AmebaPro , including SDK compiling and downloading image to AmebaPro.

Table of Contents

1 Compiling and downloading

This chapter introduces users how to develop AmebaPro. AmebaPro is composed of one main board, one sensor board, and one daughter board with LED, light sensor, and IR-LED. AmebaPro SDK provides all the example source code for the function mentioned above.

To get start, users will need to set up the software to program the board.

IAR IDE provides the toolchain for AmebaPro. It allows users to write programs, compile and upload them to your board. Also, it supports step-by-step debug. Realtek also provides Image Tool for users to do downloading code process.

> ✍ *Note : Please use IAR version 8.30.*

1.1 SDK Project introduction

Currently users can use ignore secure mode. Project_is(ignore secure) is the project without Arm TrustZone technology. This project is easier to develop and suit for first-time developer.

1.2 Compile program

AmebaPro use the newest Big-Little architecture. Big CPU is 300MHz, supporting high speed function like WiFi, ISP, Encoder and Codec. Little CPU is 4MHz, supporting low power peripheral function. Big CPU supports power-save mode while little CPU is operating. Big CPU power-save mode can be awaked by event trigger. Since the big CPU will depend on the setting of small CPU, it is necessary to compile the small CPU before the big CPU.

> *Communication between big CPU and little CPU is based on share memory. There are some examples in SDK explaining how this works. Please refer to Inter Channel Communication.*

1.2.1 Compile big CPU

Step1. Open SDK/project/realtek_amebapro_v0_example/EWARM-RELEASE/Project_is.eww.

Step2. Confirm application_is in WorkSpace, right click application_is and choose "Rebuild All" to compile.

Step3. Make sure there is no error after compile.

1.2.2 Compile little CPU

Step1. Open SDK/project/realtek_amebapro_v0_example/EWARM-RELEASE/Project_lp.eww.

Step2. Confirm application_lp in WorkSpace, right click application_lp and choose "Rebuild All" to compile.

Step3. Make sure there is no error after compile.

1.2.3 Generating image (Bin)

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

Partition.bin stores partition table, recording the address of Boot image and firmware image. Boot.bin is bootloader image; firmware_is.bin is application image, flash_is.bin links partition.bin, boot.bin and firmware_is.bin. Users need to choose flash_is.bin when downloading the image to board by Image Tool.

1.3 Using image tool to download and concat images

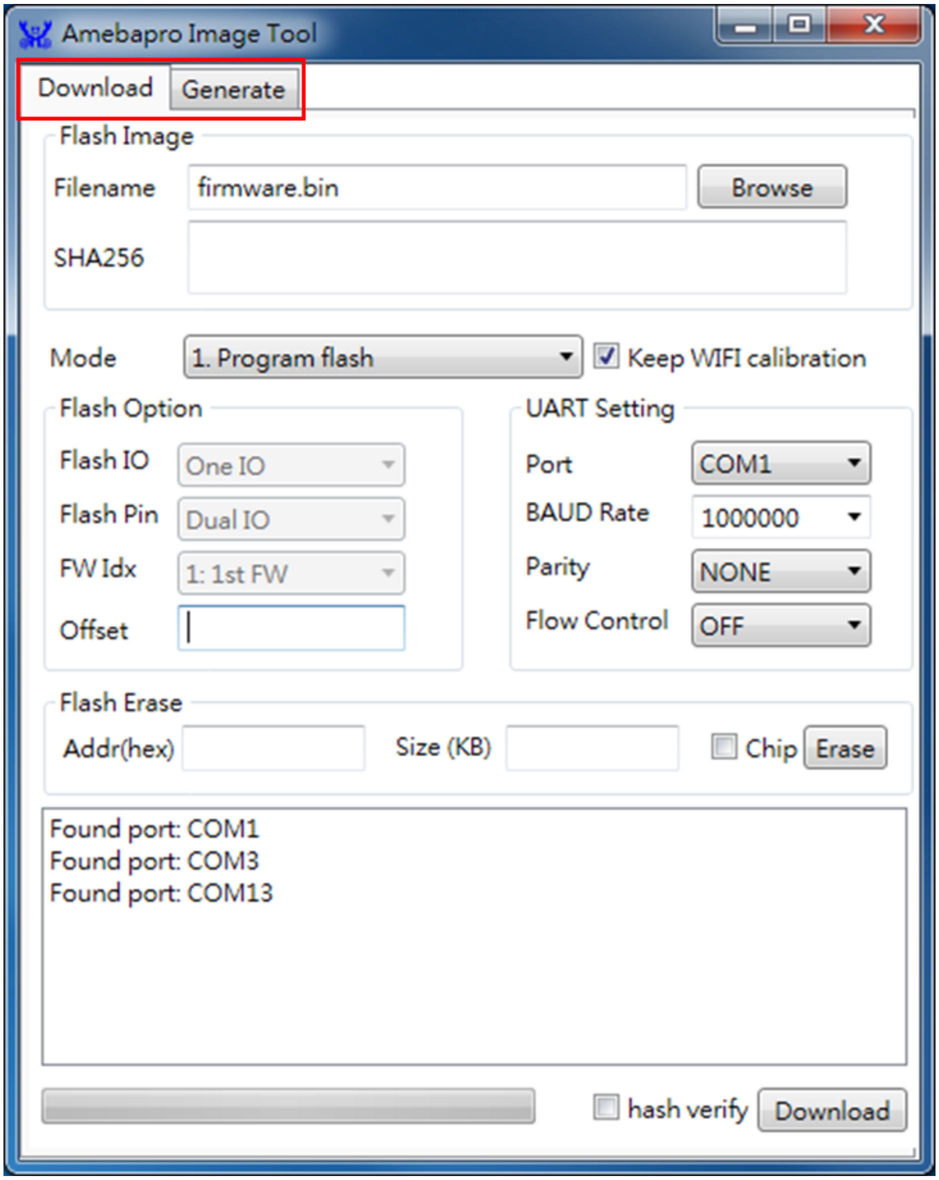
This section introduces how to use Image Tool to generate and download images.

As show in the following figure, Image Tool has two tab pages:

- Download: used as image download server to transmit images to AmebaPro through UART

- Generate: concat separate images and generate a final image

> ✍ *If you need to download code via external uart, must use* ***FT232*** *USB To UART dongle*



Filename: firmware.bin

SHA256:

Mode: 1. Program flash

☑ Keep WIFI calibration

Flash IO: One IO

Flash Pin: Dual IO

FW Idx: 1: 1st FW

Offset:

Port: COM1

BAUD Rate: 1000000

Parity: NONE

Flow Control: OFF

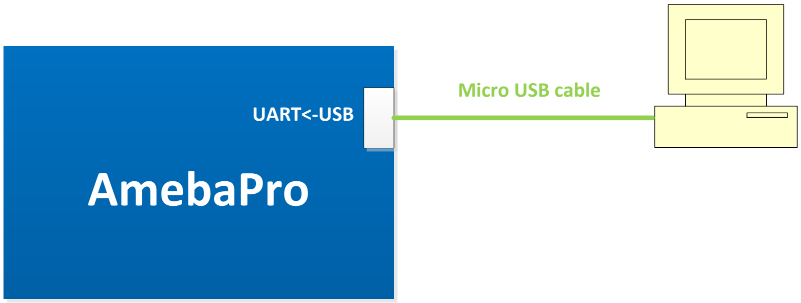
Addr(hex): Size (KB): □ Chip

```
Found port: COM1
Found port: COM3
Found port: COM13
```

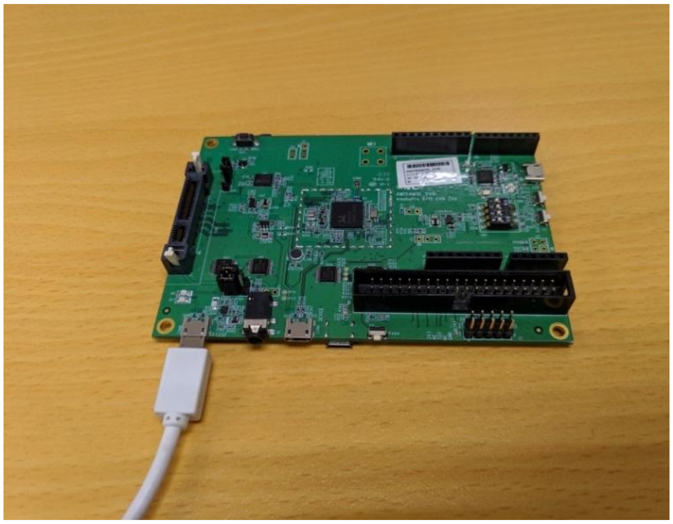
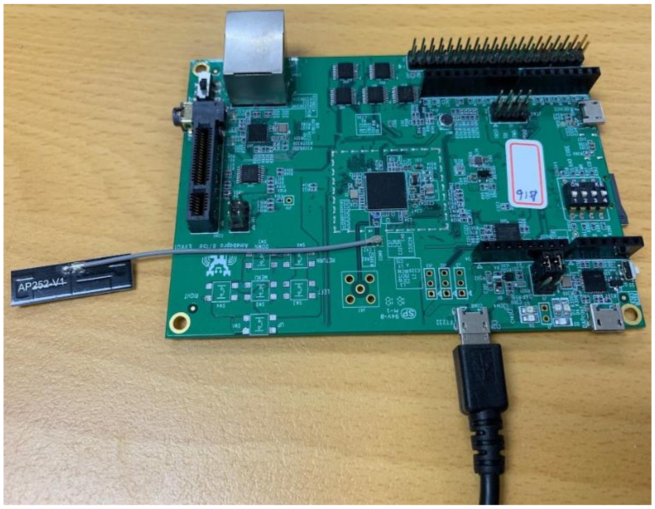
□ hash verify

1.3.1 Environment Setup

1.3.1.1 Hardware Setup



AmebaPro offers two different hardware structures, the corresponding bord number are

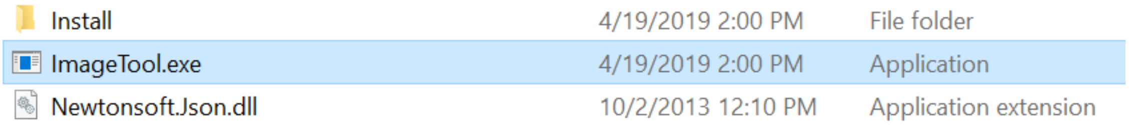
2V0 、 2V1	1V0
	

1.3.1.2 Software Setup

Execute ImageTool.exe from location

project\tools\AmebaPro\Image_Tool\ImageTool.exe.

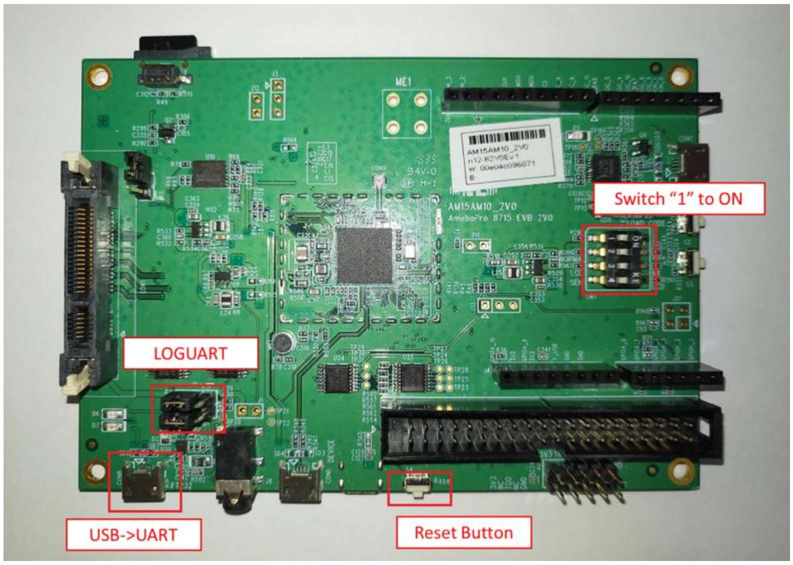
Environment Requirements: EX. Win7 Above **Microsoft .NET Framework 4.5**



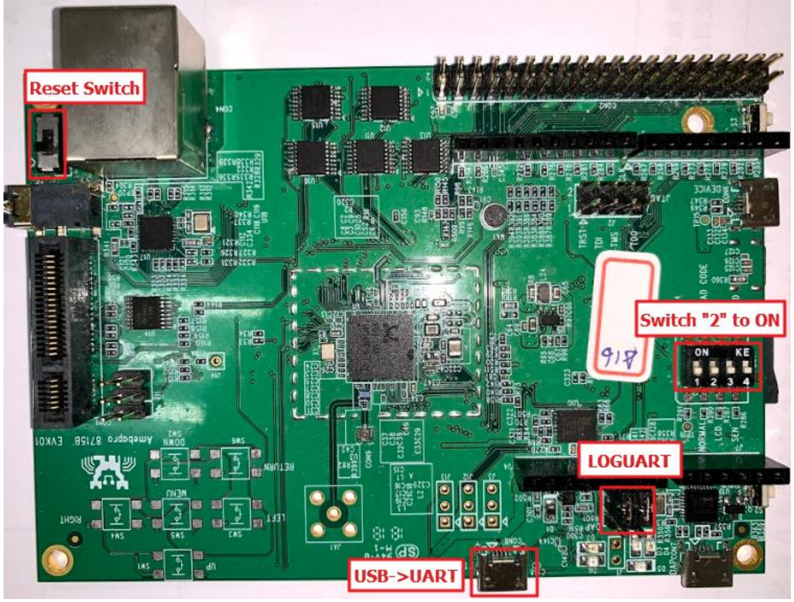
Install	4/19/2019 2:00 PM	File folder
ImageTool.exe	4/19/2019 2:00 PM	Application
Newtonsoft.Json.dll	10/2/2013 12:10 PM	Application extension

1.3.2 Image Download

Image tool use UART to transmit image to AmebaPro board. Before performing image download function, AmebaPro need to enter UART_DOWNLOAD mode first. Please follow below steps to get AmebaPro into UART_DOWNLOAD mode:



2V0、2V1



1V0

Step 1: Connect LOGUART with FT pin by jumper cap.

Step 2: Connect USB->UART to PC by using micro-USB wire.

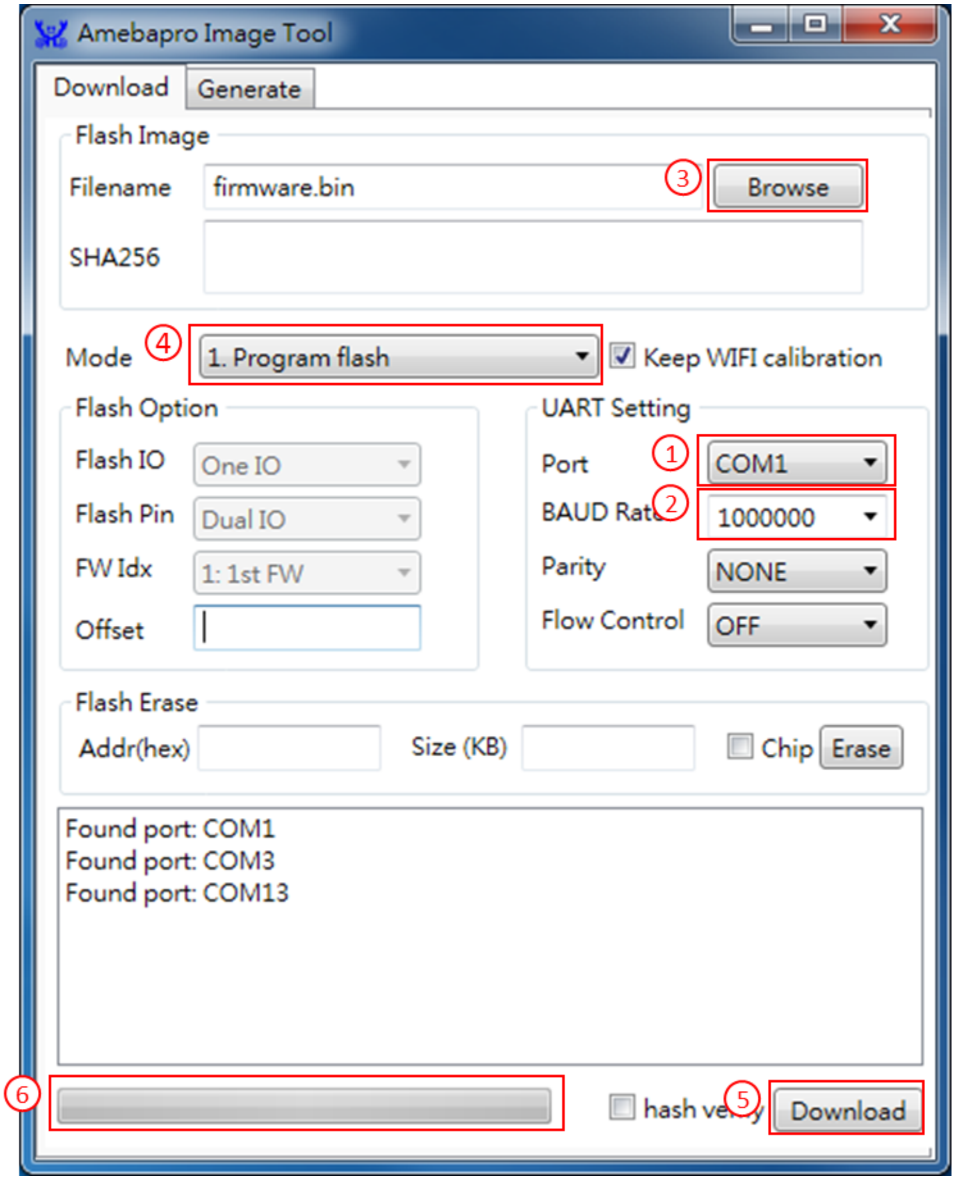
Step 3: Switch “1” to ON from SW7(2V0、2V1) or Switch “2” to ON from SW7(1V0)

Step 4: Push reset button.

After performing above steps, you should see below message from UART console, which means the board now is booted as UART_DOWNLOAD mode:

```
== Rtl8195bh IoT Platform ==
Chip VID: 0, Ver: 2
ROM Version: v3.0
Test Mode: boot_cfg1=0x0
Download Image over UART0_S1
```

1.3.2.1 Flash Download



Amebapro Image Tool

Filename: firmware.bin

SHA256:

Mode: 1. Program flash

☑ Keep WIFI calibration

Flash Option

Flash IO: One IO

Flash Pin: Dual IO

FW Idx: 1: 1st FW

Offset:

UART Setting

Port: COM1

BAUD Rate: 1000000

Parity: NONE

Flow Control: OFF

Flash Erase

Addr(hex):

Size (KB):

□ Chip

```
Found port: COM1
Found port: COM3
Found port: COM13
```

□ hash verify

To download image through Image Tool, device need to enter UART_DOWNLOAD mode first.

Steps to download flash are as following:

Step 1: Application will scan available UART ports. Please choose correct UART port.
Please close other UART connection for the target UART port.

Step 2: Choose desired baud rate between computer and AmebaPro.

Step 3: Choose target flash binary image file “flash_xx.bin”

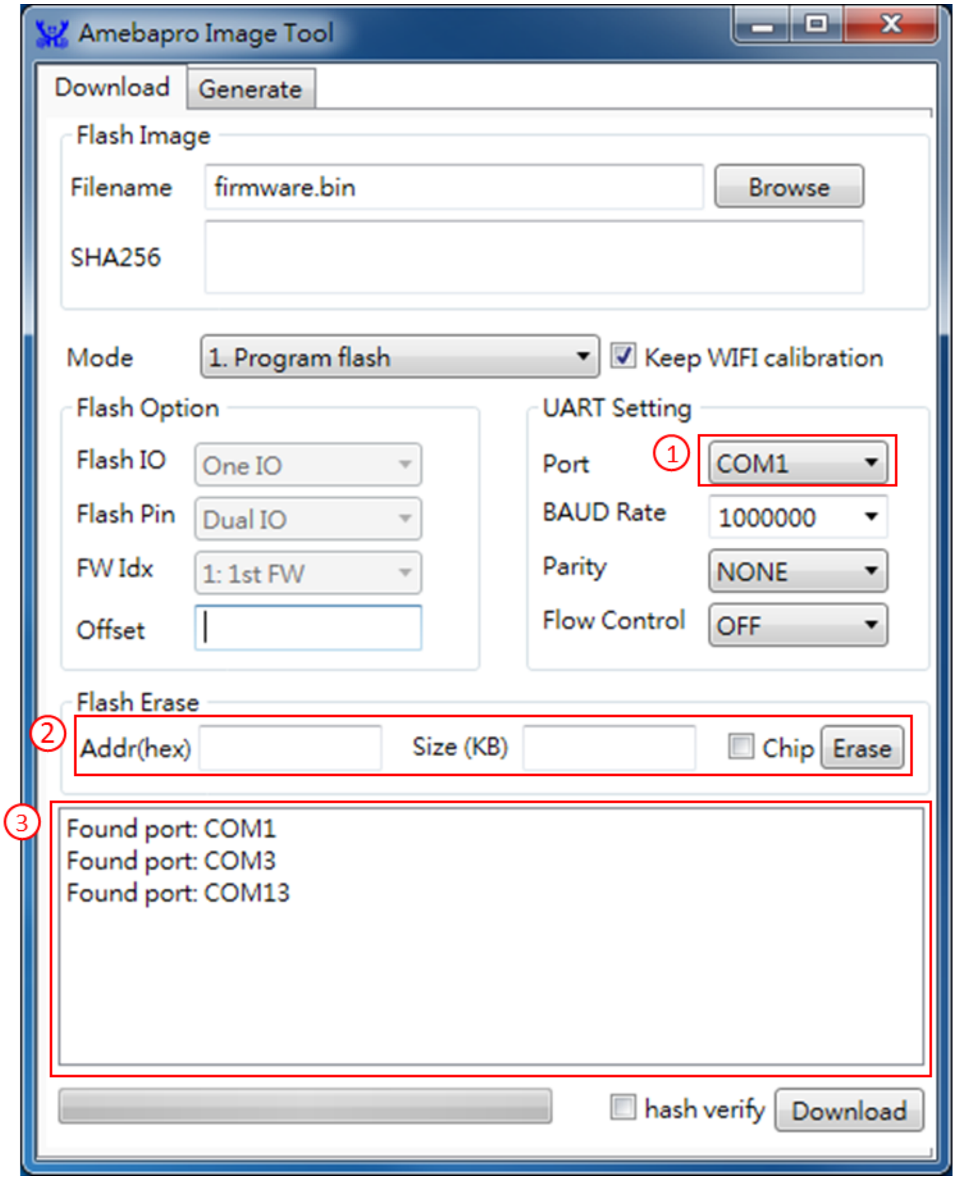
Step 4: Check Mode is “1. Program flash”

Step 5: Click “Download”

Step 6: Progress will be shown on progress bar and result will be shown after download finish.

NOTE: Other settings using default values,
Flash IO : One IO/Flash Pin : Dual IO/Parity : NONE/Flow Control : OFF

1.3.2.2 Flash Erase



Filename: firmware.bin

SHA256:

Mode: 1. Program flash

☑ Keep WIFI calibration

Flash IO: One IO

Flash Pin: Dual IO

FW Idx: 1: 1st FW

Offset:

Port: COM1

BAUD Rate: 1000000

Parity: NONE

Flow Control: OFF

```
Found port: COM1
Found port: COM3
Found port: COM13
```

To erase flash by using Image Tool, device need to enter UART_DOWNLOAD mode first. Steps to erase flash are as following:

Step 1: Application will scan available UART ports. Please choose correct UART port.

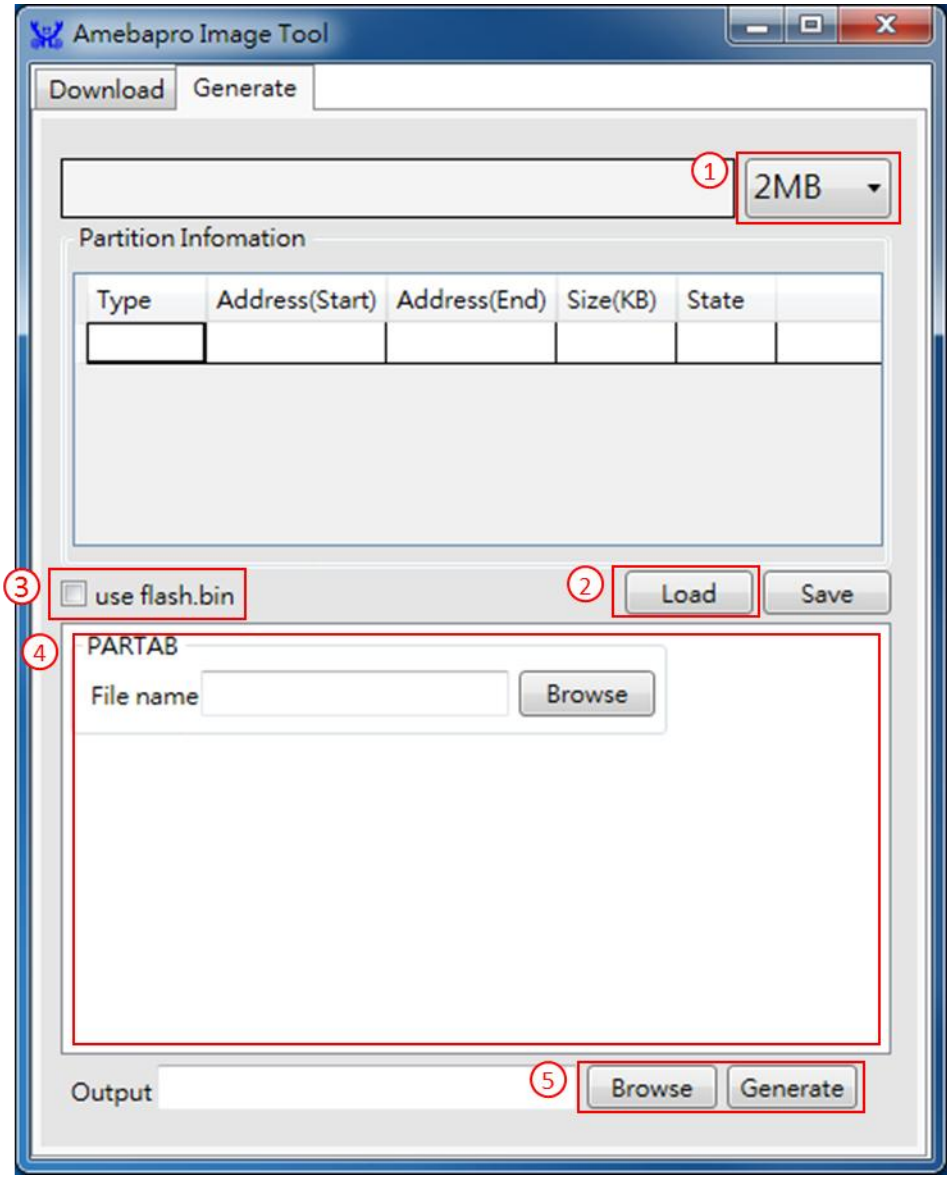
Please close other UART connection for the target UART port.

Step 2: Select erase range or using "Chip" checkbox for chip erase. Click "Erase" to perform erase.

Step 3: The result will be shown at tool console after erase progress finish.

1.3.3 Image Generation

The generate tab page can concat separate images and generate a final image. E.g. Concat Normal image and MP image to one final image.



Steps to concat images are as following:

Step 1: Select target flash size.

Step 2: Load partition table information from file. Please use “Load” button to load from partition.json from project folder. (e.g. under EWARM-RELEASE/)

Step 3: Enable “use flash.bin” checkbox.

Or not enable “use flash.bin” and choose partition.bin/boot.bin/firmware_is.bin separately.

Step 4: Select target image locations. E.g. choose flash_is.bin as FLASH.BIN and choose firmware_is.bin as FW2.

Step 5: Browse the output file location and click “Generate” for the final image.

The generated file now can be downloaded by using steps in Sec. 1.3.2.1.

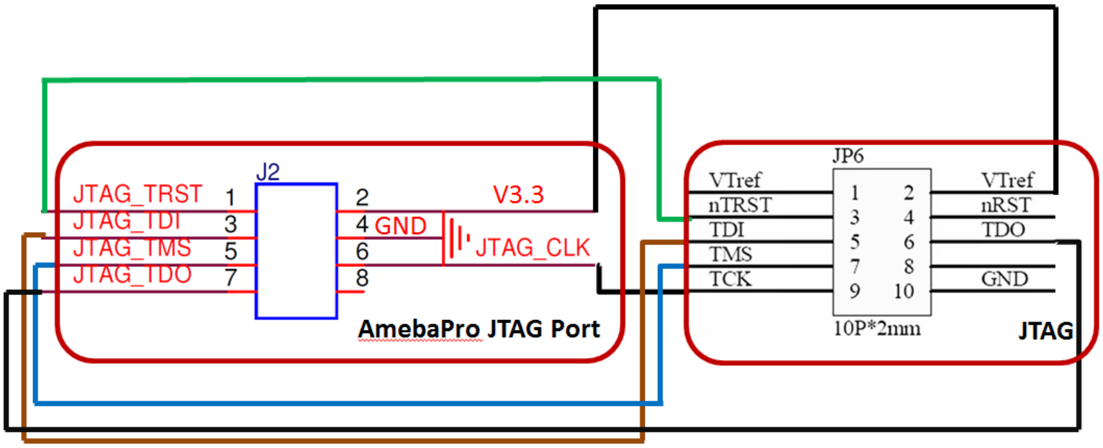
1.4 Using JTAG/SWD to debug

JTAG/SWD is a universal standard for chip internal test. The external JTAG interface has four mandatory pins, TCK, TMS, TDI, and TDO, and an optional reset, nTRST. JTAG-DP and SW-DP also require a separate power-on reset, nPOTRST. The external SWD interface requires two pins: bidirectional SWDIO signal and a clock, SWCLK, which can be input or output from the device.

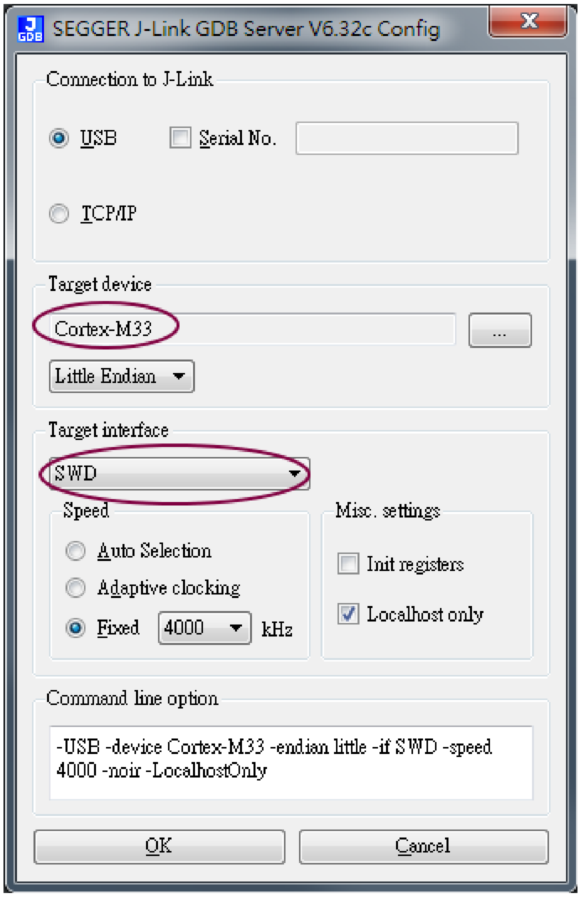
1.4.1 JTAG/SWD connection

> *If using 2V0 、2V1 version AmebaPro.*
>
> *Please check SW7 pin 3 switch to ON before connection.*

Make sure the six pin （VTref、TCK、TMS、TDI、TDO and nTRST）connect well done as the graph below, if using SWD, please check four pin (VTref[VDD]、TMS[SWDIO]、TCLK[SWCLK] and TDO[SWO]) .



After connection, open J-Link GDB server. Choose target device Cortex-M33(for AmebaPro), and target interface JTAG / SWD. Click OK.



Connection to J-Link

● USB □ Serial No.

○ TCP/IP

Target device

Cortex-M33

Little Endian

Target interface

SWD

Speed

○ Auto Selection

○ Adaptive clocking

● Fixed 4000 kHz

Misc. settings

□ Init registers

☑ Localhost only

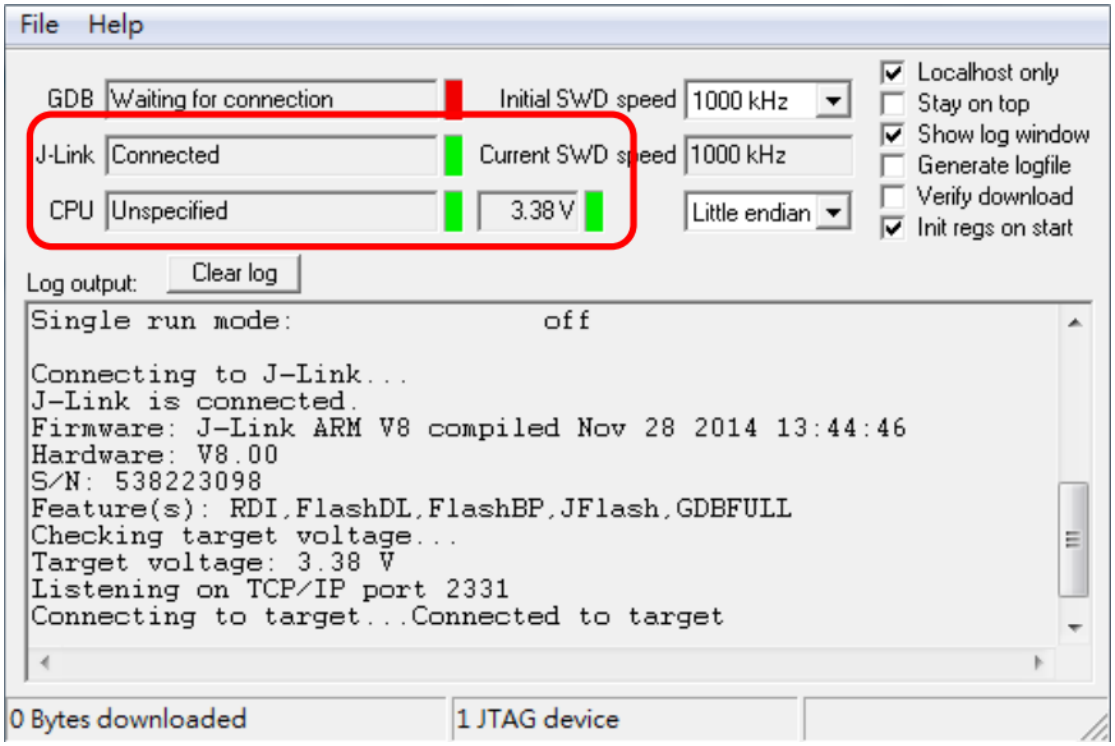
Command line option

```
-USB -device Cortex-M33 -endian little -if SWD -speed
4000 -noir -LocalhostOnly
```

Note that CMSIS-DAP have two limitations in this situation:
(1) CMSIS-DAP transmission speed is too slow.
(2) System reset cannot be completely reset system can only reset HS core.

To use WATCHDOG RESET + SYSTEM RESET, only JLINK can be used. Can’t be used in CMSIS-DAP.
It is recommended to use JLINK V9 or higher version. Currently, step in/out/over function is not supported. If want to set the interrupt position, set breakpoint directly at the point you want to see.

If connection succeeds, J-Link GDB server must show as below.



GDB Waiting for connection 🟥 Initial SWD speed 1000 kHz

J-Link Connected 🟩 Current SWD speed 1000 kHz

CPU Unspecified 🟩 3.38 V 🟩 Little endian

☑ Localhost only
☐ Stay on top
☑ Show log window
☐ Generate logfile
☐ Verify download
☑ Init regs on start

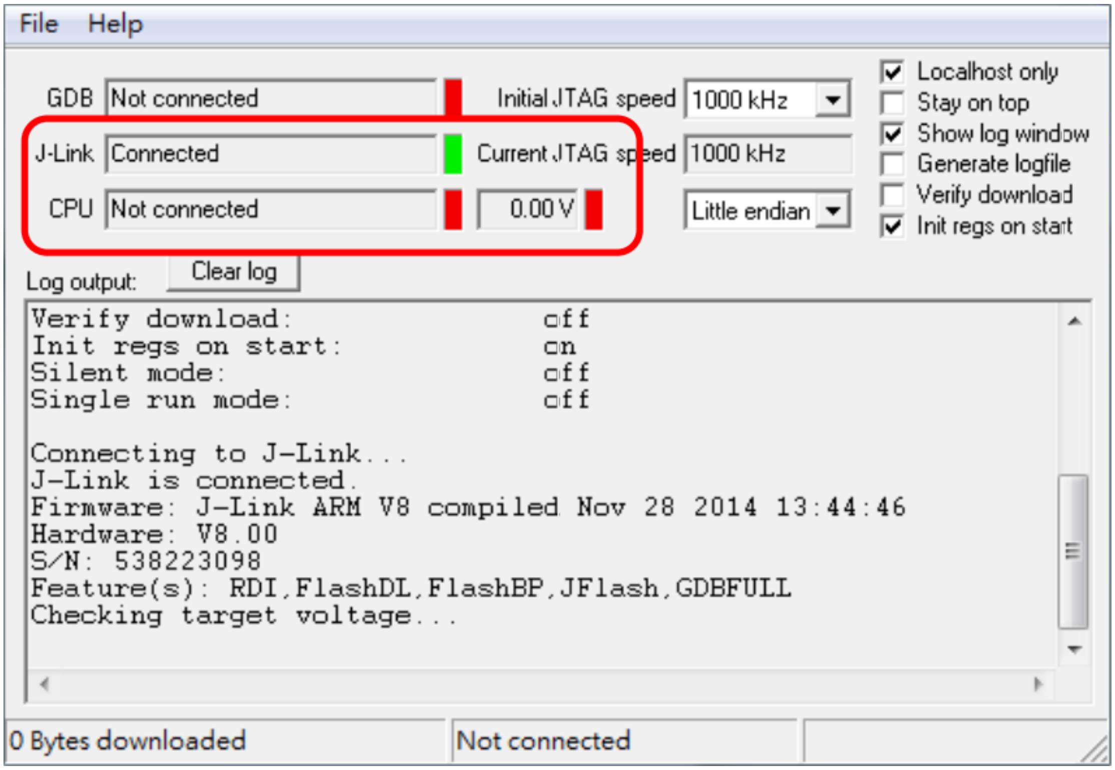
Log output:

```
Single run mode:                off

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link ARM V8 compiled Nov 28 2014 13:44:46
Hardware: V8.00
S/N: 538223098
Feature(s): RDI,FlashDL,FlashBP,JFlash,GDBFULL
Checking target voltage...
Target voltage: 3.38 V
Listening on TCP/IP port 2331
Connecting to target...Connected to target
```

0 Bytes downloaded | 1 JTAG device

If connection fails, J-Link GDB will show:



GDB Not connected 🟥 Initial JTAG speed 1000 kHz

J-Link Connected 🟩 Current JTAG speed 1000 kHz

CPU Not connected 🟥 0.00 V 🟥 Little endian

☑ Localhost only
☐ Stay on top
☑ Show log window
☐ Generate logfile
☐ Verify download
☑ Init regs on start

Log output:

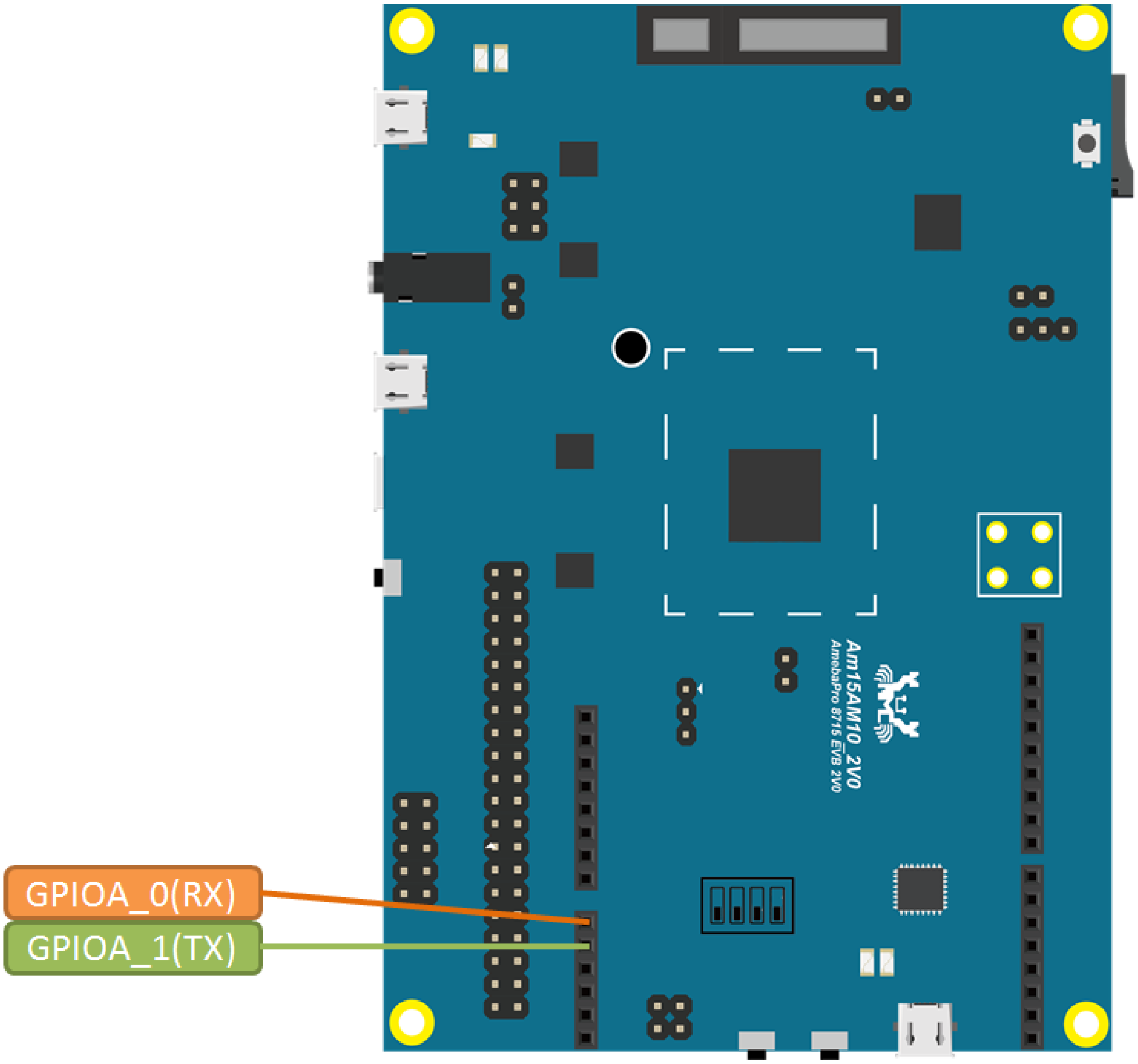
```
Verify download:                off
Init regs on start:             on
Silent mode:                    off
Single run mode:                off

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link ARM V8 compiled Nov 28 2014 13:44:46
Hardware: V8.00
S/N: 538223098
Feature(s): RDI,FlashDL,FlashBP,JFlash,GDBFULL
Checking target voltage...
```

0 Bytes downloaded | Not connected

1.5 Use uart to observe the log of the little cpu

In AmebaPro, the log of the big cpu and the little cpu is separate. The big cpu log can be observed through the FT232 or DAP, and the little cpu needs to be connected to the uart console through the A0 and A1 pins to observe the log of the small core.



2 Development Board

This chapter introduces users how to use AmebaPro Image sensor board. With correct hardware setting and software setting, image sensor board can work properly. Also, our SDK provide API to check whether sensor hardware matches software setting to prevent wrong manipulation.

2.1 Choose Image Sensor

This is sensor list that AmebaPro support:

AmebaPro EVAL	Sensor	Note
AmebaPro	OV2735	
AmebaPro	SC2232	

2.1.1 OV2735

Hardware setting

- Plug correct OV2735 sensor board

Software setting

- In project\realtek_amebapro_v0_example\inc\sensor.h
  Set the definition: #define SENSOR_USE Sensor_OV2735
- In component\soc\realtek\8195b\misc\bsp\image
  Delete original isp.bin, and rename isp_ov2735.bin to isp.bin

2.1.2 SC2232

Hardware setting

- Plug correct SC2232 sensor board

Software setting

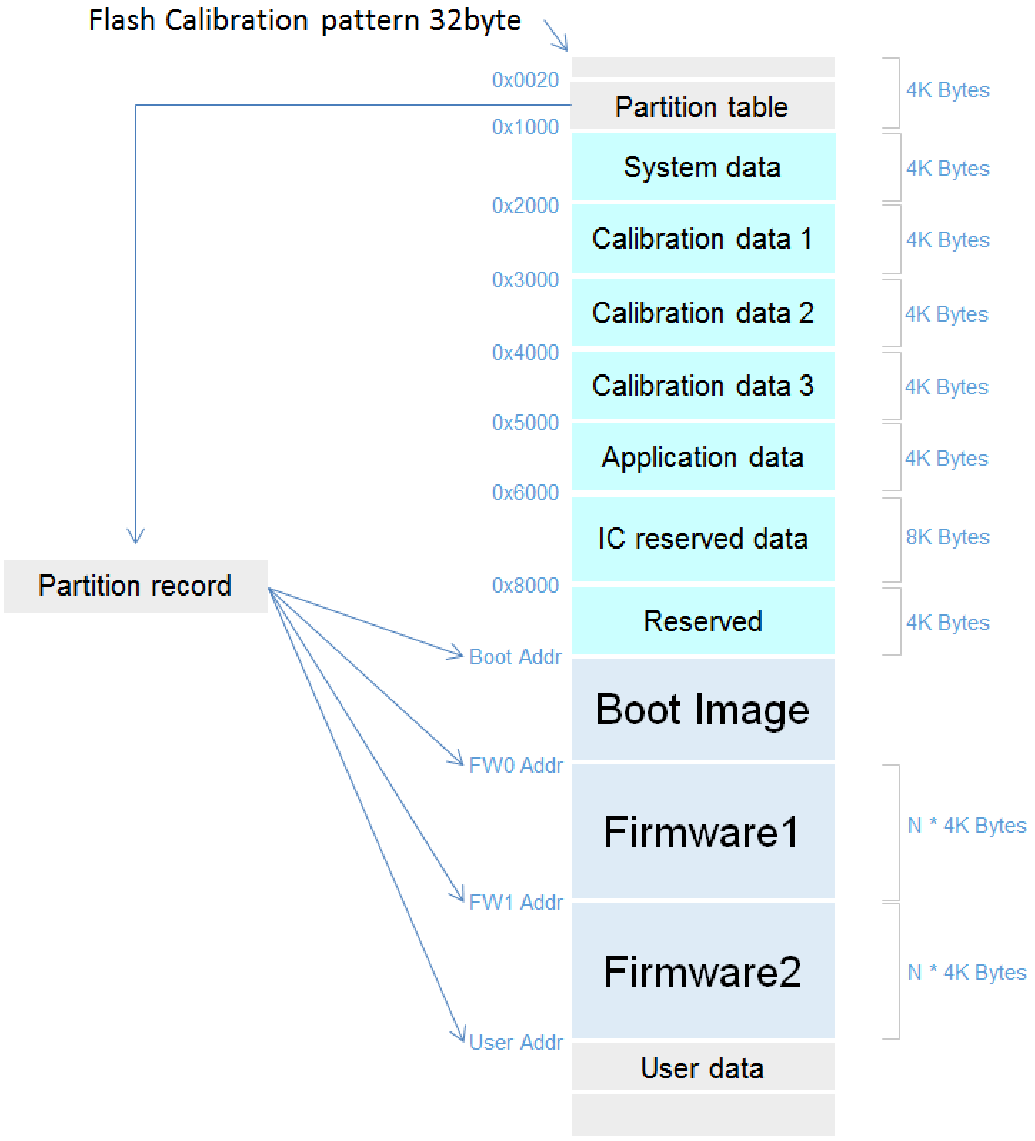
- In project\realtek_amebapro_v0_example\inc\sensor.h
  Set the definition: #define SENSOR_USE Sensor_SC2232
- In component\soc\realtek\8195b\misc\bsp\image
  Delete original isp.bin, and rename isp_sc2232.bin to isp.bin

> ✍ *If users use the wrong hardware or software setting does not match hardware setting ，API video_sensor_check(SENSOR_USE) will return -1*

3 Flash Layout

AmebaPro use Big-Little architecture and sub-mcu system design, which provide high extensibility for developers.

3.1 Flash Layout overview



3.2 Flash used by Application

Some functions or applications in SDK may use the flash sections. If some of the following functions or applications enabled, please be careful that the section it used to avoid the reuse or conflict with your application’s setting. If it is necessary, the flash section defined could be modified.

Warning: The section of those data is based on default image condition. If overall image size is larger, the application data, for example, DCT_BEGIN_ADDR (0x200000) will overwrite image in flash. Please move around according to the flash size chosen.
Now the following sections will be used if the function is enabled. Those definitions are specified in platform_opts.h.

1. UART AT Command. Default disabled.

```
#define UART_SETTING_SECTOR                    0x000FC000
```

2. Fast Connect: To store the AP information. Default enabled.

```
#define FAST_RECONNECT_DATA                    0x5000
```

3.DCT: device configuration table API usage. Default disabled.

```
#define DCT_BEGIN_ADDR                         0x200000
```

4 How to use example source code

4.1 Application example source

The examples for AmebaPro application is the SDK/common/example file. All the example provide related files including .c,.h, and readme. The readme file explains how to compile and important parameter.
After opening IAR, the first step is adding example source code(.c) into application_is -> utilities -> example(right click example and choose Add -> Add Files or drag-and-drop the file into it).
After adding example code, user should use platform_opts.h to switch on the example. For example, if users are going to use DCT function, compile flag CONFIG_EXAMPLE_DCT should be set to 1, which means

```
#define CONFIG_EXAMPLE_DCT 1
```

In platform_opts.h so that the example function in example_entry will execute .
After this procedure, rebuild application_is project to execute the example.

4.2 Peripheral example source

Peripheral example source can help us utilize peripheral function. Peripheral example source code locates in SDK/project/realtek_amebapro_v0_example/example_sources. There are main.c and readme.txt in each example file. The main.c in the example should be used to replace original main.c(in SDK/project/realtek_amebapro_v0_example/src). The readme file explains how to compile and important parameter.
After that, rebuild application_is project to execute the Peripheral example.

4.3 Wi-Fi example source

4.3.1 Use AT command to connect WLAN

AmebaPro provide AT command for user to test and develop. Users can key in AT command to connect WLAN by the console in PC. AT command can be referenced in AN0025 Realtek at command.pdf.
Wi-Fi direct GO and Concurrent mode are still under development, we will release in future version.

4.3.2 WLAN scenario example

AmebaPro provide WLAN scenario example for users to develop a variety of WiFi function, the path if example is \component\common\example\wlan_scenario\example_wlan_scenario.c. This example provides many features including station mode, AP mode, Concurrent mode, WPS and P2P GO. The detail and the usage of the example can be read in readme.txt.
Wi-Fi direct GO and Concurrent mode are still under development, we will release in future version.

4.4 Video example source

AmebaPro provide Multimedia Framework v2. Video example code is based on this architecture. Any detail about Multimedia Framework v2 architecture can be referenced in document in UM0301.

5 Memory configuration and usage

This chapter introduces the memory in AmebaPro, including OM, RAM, SRAM, TCM, DRAM, Flash. Also, this chapter provides the guide that users can place their program to the specific memory to fit user's requirement. However, some of program is fixed in specific memory and cannot be moved. This program is also discussed in this chapter. Reference the chapter Flash Layout if the memory is related to Flash.

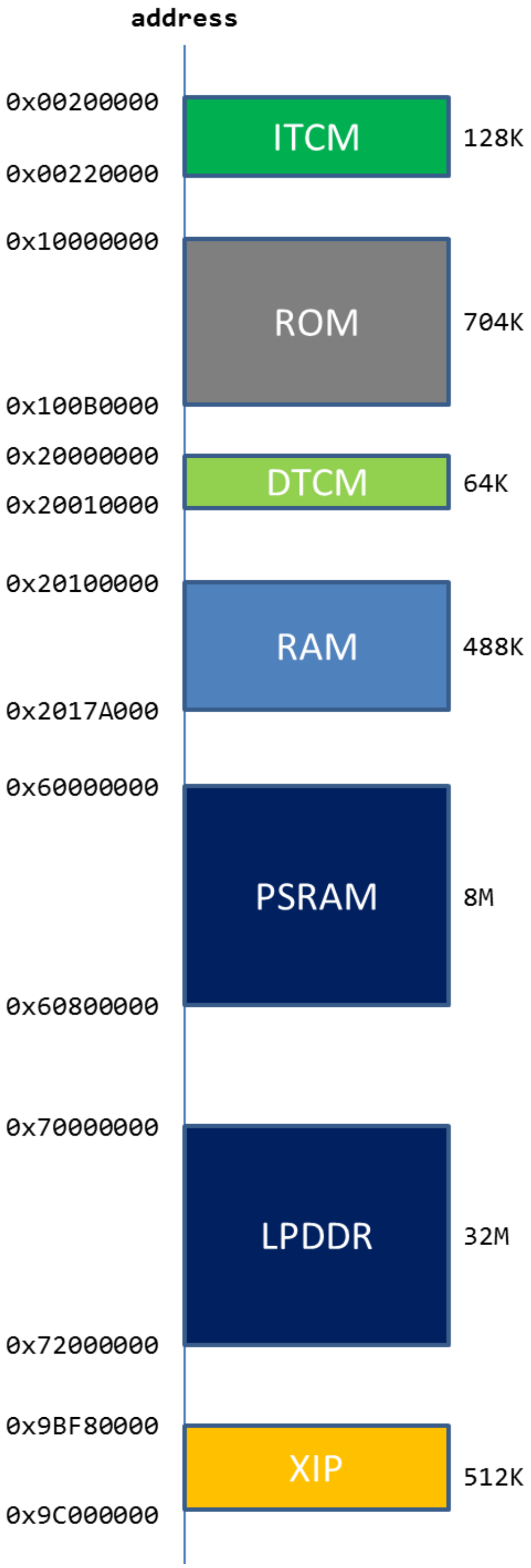
5.1 Memory type

5.1.1 The size and configuration in AmebaPro(big CPU)

The size and configuration in AmebaPro(big CPU) is as shown below

	Size(bytes)	Description
ITCM	128K	can place instruction
ROM	704K	
DTCM	64K	can place Read/write data
RAM	488K	SRAM which size is 128K
PSRAM	8M	Choose either PSRAM or LPDDR. Evaluation Reference Board is attached with LPDDR。
LPDDR	32M	Choose either PSRAM or LPDDR. Evaluation Reference Board is attached with LPDDR。 。LPDDR is also called as DRAM
XIP	512K	Execute In Place, text section in Flash can be placed in XIP

The graph of configuration in AmebaPro big CPU is as shown below:

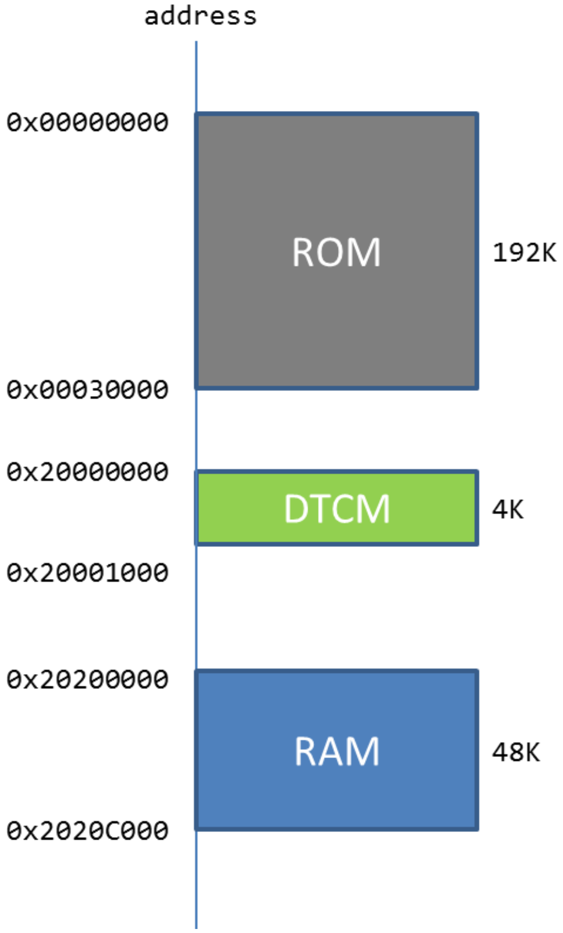


5.1.2 The size and configuration in AmebaPro(little CPU)

The size and configuration in AmebaPro(little CPU) is as shown below

	Size(bytes)	Description
ROM	192K	
DTCM	4K	can place Read/write data
RAM	48K	Is also known as SRAM

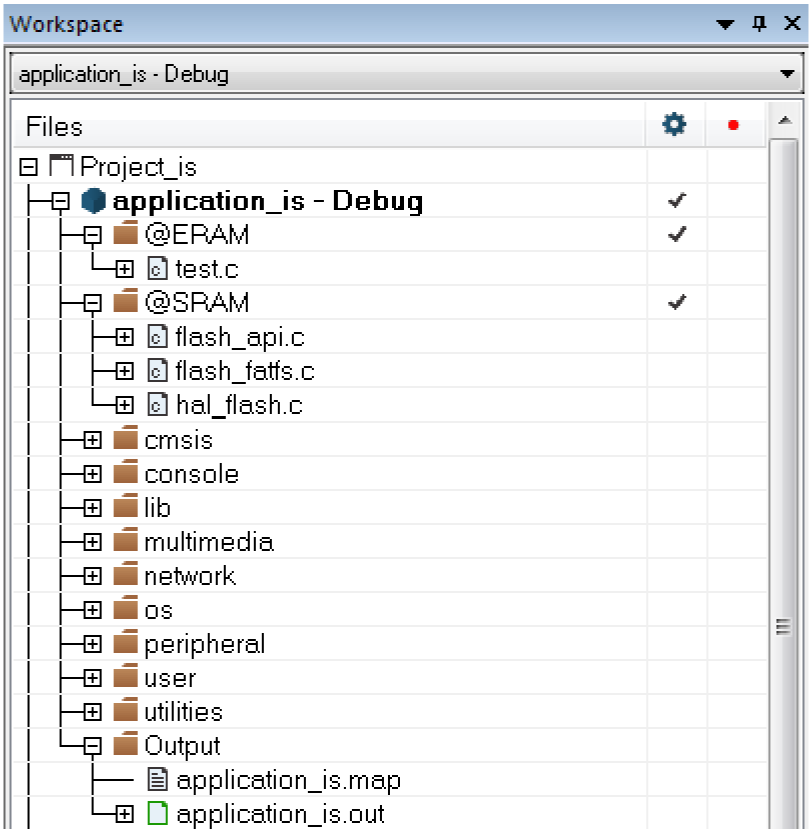
The graph of configuration in AmebaPro little CPU is as shown below:



5.2 Memory Configuration

5.2.1 Configure memory in IAR

In IAR Workspace, there are “@ERAM” and “@SRAM” group. “@ERAM” represents external RAM, which can be known as PSRAM or LPDDR. “@SRAM” represents SRAM. Except “@ERAM” and “@SRAM” group, the rest of text section will be placed in XIP.



If users want to place specific text section of source file into PSRAM/LPDDR, users can drag-and-drop the files into “@ERAM”. For example, text section of “test.c” will be placed in PSRAM/LPDDR as the graph above. 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 above.

5.2.2 Configure memory in ICF file

IAR uses ICF (IAR Configuration File) to configure memory allocation so users can configure memory allocation by ICF file.
ICF file of Big CPU in AmebaPro locates:
“SDK/project/realtek_amebapro_v0_example/EWARM-RELEASE/ application_is.icf”
Open “application_is.icf” with editor. There are some memory regions in it, which is:

- ITCM_RAM_region
- DTCM_RAM_region
- RAM_region
- RAM_NC_region
- ERAM_region
- XIP_FLASH_region

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

5.2.3 Memory overflow

In default, AmebaPro place text section in XIP area. If XIP does not have enough space, it will show the errors as below when linking.

```
Error[Lp011]: section placement failed
        unable to allocate space for sections/blocks with a total estimated minimum
size of 0xa0051 bytes (max align 0x8) in <[0x9bf80140-0x9bffffff]> (total uncommitted
space 0x7fec0).
```

The solution is to move the program in XIP to other region.

5.3 Video in DRAM

Video source example costs a lot of memory:

- Stream 1: H264 1080p, 15fps, bitrate 2M + 8K AAC
- Stream 2: 720p, 30fps, bitrate 1M + 8K AAC
- Snapshot mode: 720p, JPEG Level 5

EX: Video costs **30.58MB** in DRAM

- 1080p H264: Encoder: 6.54MB, ISP buffer: 2.97 * 3 = 8.91MB; Encoder buffer pool: 2.97MB , **total 18.42MB**
- 720p H264: Encoder: 2.92MB, ISP buffer: 1.32 * 4 = 5.28MB; Encoder buffer pool: 1.32MB , **total 9.52MB**
- 720p JPEG: Encoder: 3.21KB, ISP buffer: 1.32 MB; Snapshot buffer: 1.32MB , **total 2.64MB**

5.4 Unmovable program

5.4.1 XIP-related Flash API

Because XIP use flash interface so that flash api are placed in SRAM. Users cannot move these file to other place, including:

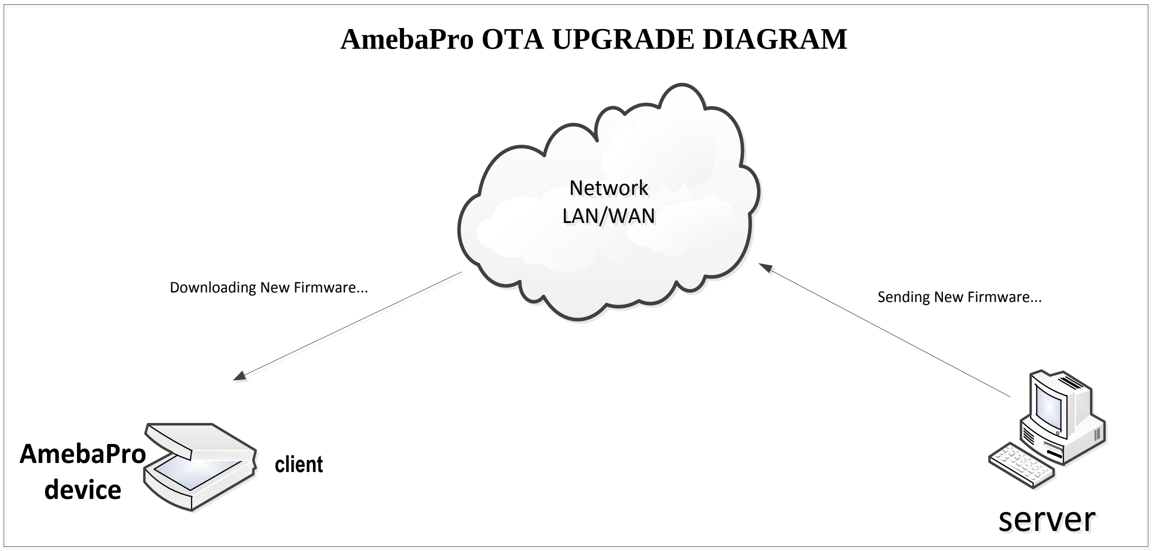
- flash_api.c
- flash_fatfs.c
- hal_flash.c.

5.5 Counting system of memory

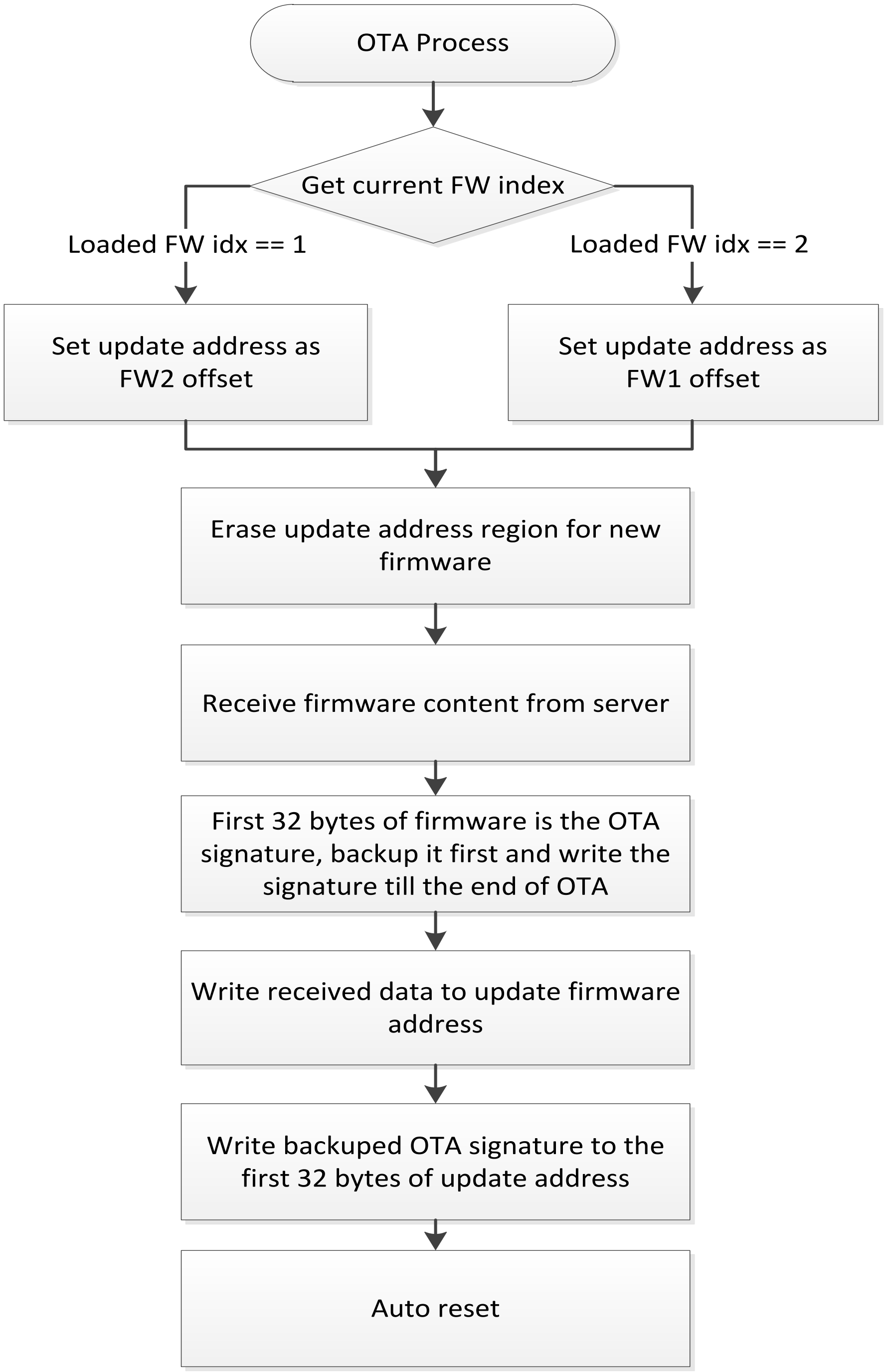
Please refer to UM0070.

6 OTA

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

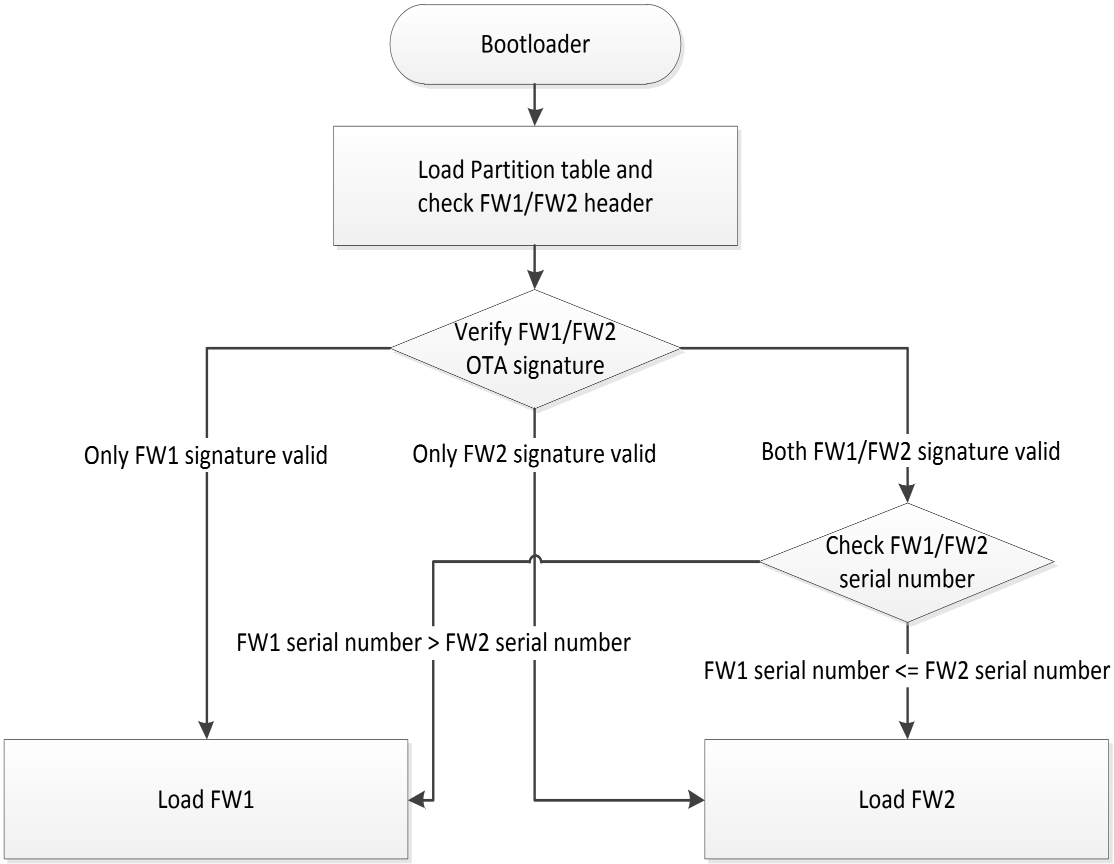


6.1 OTA operation flow


OTA Process
Get current FW index
Loaded FW idx == 1
Loaded FW idx == 2
Set update address as FW2 offset
Set update address as FW1 offset
Erase update address region for new firmware
Receive firmware content from server
First 32 bytes of firmware is the OTA signature, backup it first and write the signature till the end of OTA
Write received data to update firmware address
Write backuped OTA signature to the first 32 bytes of update address
Auto reset

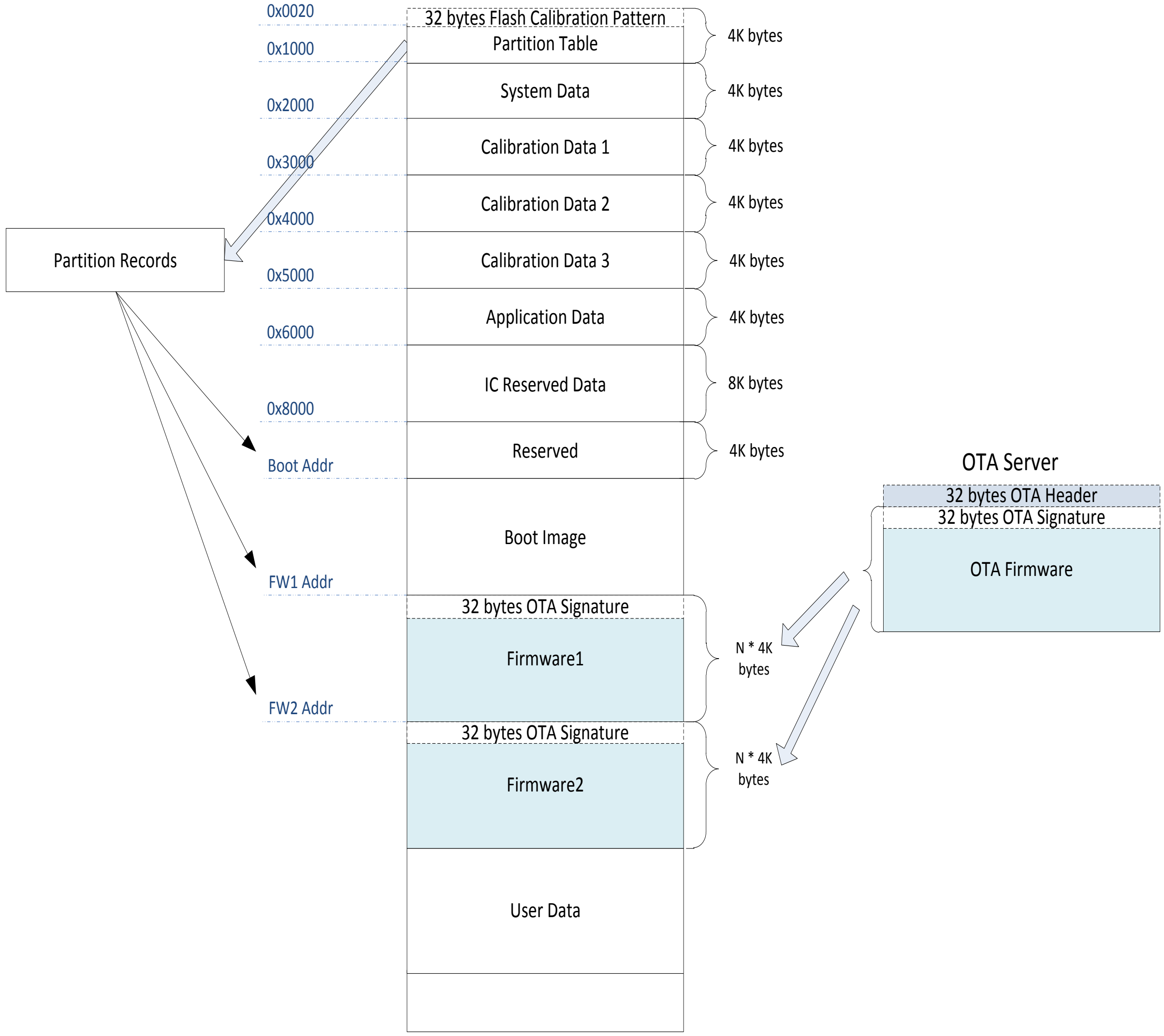
> *During the step of "Write received data to update firmware address", the 32 bytes OTA signature need set to 0xff, which is invalid signature. The correct OTA signature needs to be appended at the end of OTA process to prevent device booting from incomplete firmware.*

6.2 Boot process flow


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

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

6.3 Upgraded partition



In AmebaPro OTA update procedure, Firmware1 and Firmware2 are swapped each other. The Firmware1/Firmware2 addresses are stored in partition records, defined in *partition.json* under *project\realtek_amebapro_v0_example\EWARM-RELEASE\*. Please adjust it according to your firmware size. Also, please note that the Firmware1/Firmware2 address is required set to a 256KB aligned address, e.g. 0x40000 or 0x240000, due to the restriction of XIP remapping architecture.

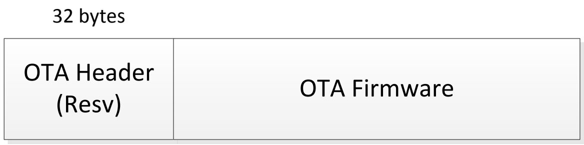
```
"fw1":{
        "start_addr" : "0x40000",
        "length" : "0x200000",
        "type": "FW1",
        "dbg_skip": false,
        "hash_key":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F"
},
"fw2":{
        "start_addr" : "0x240000",
        "length" : "0x200000",
        "type": "FW2",
        "dbg_skip": false,
        "hash_key":"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E5F"
}
```

6.4 Firmware image output

After building project source files in SDK, it would generate ota_is.bin, which is the OTA Firmware as mentioned earlier. This ota_is.bin can be used for remote OTA server. When device executes the OTA procedure, it would determine target OTA firmware and only would program its corresponding content into the flash.

6.4.1 OTA file format

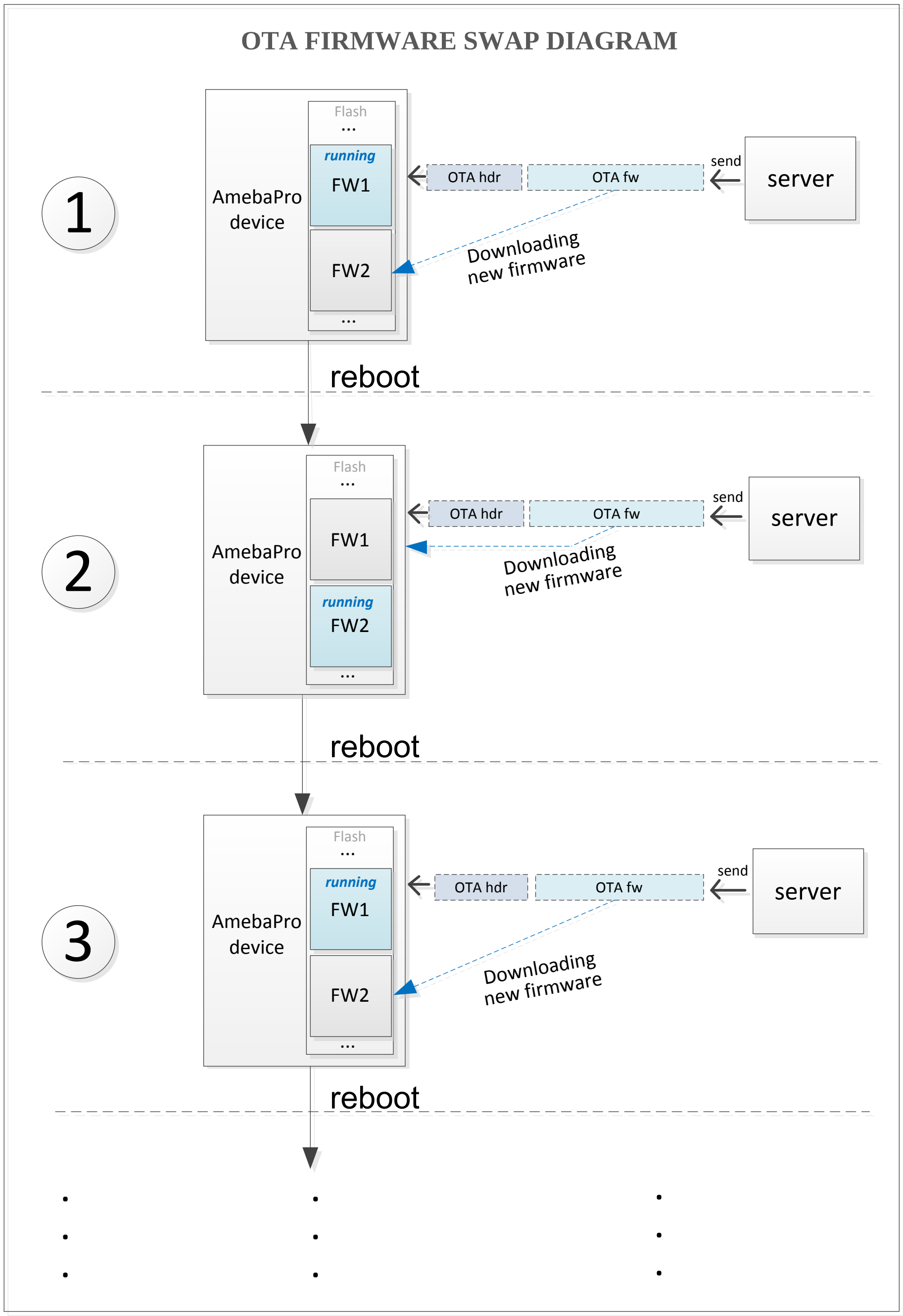
Below is the format of ota_is.bin used by OTA server:



ota_is.bin includes OTA firmware and a 32 bytes OTA Header. Currently, the OTA Header is reserved for future extension.

6.4.2 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.



6.4.3 Configuration for building OTA firmware

Since 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.

6.4.3.1 Serial number

AmebaPro OTA use serial number to decide the boot sequence if the signature of both firmware are valid. Hence before building the project, please make sure the serial number is correctly configured. 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 *project\realtek_amebapro_v0_example\EWARM-RELEASE\amebapro_firmware_is.json*.

```
"FIRMWARE":{
        "images":[
                {"img": "XIP", "offset":"0x00"},
                {"img": "ISP", "offset":"0x00"},
                {"img": "CINIT", "offset":"0x00"},
                {"img": "FWLS", "offset":"0x00"},
                {"img": "FWHS", "offset":"0x00"},
                {"img": "WLAN", "offset":"0x00"},
                {"img": "WOWLAN", "offset":"0x00"}
        ]
```

For this example, the XIP 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 "**XIP**":

```
"XIP": {
        "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": 100
        },
```

The Serial number is stored as 4 byte digital number and is valid from 0. Please modify it according to your firmware version.

Step 3: After building project source files in SDK, it should automatically generate *SDK_folder/project/project_name/EWARM-RELEASE/Debug/Exe/ota_is.bin*, which is the application only firmware as mentioned as OTA Firmware. The serial information would also be included in this firmware.

6.5 Implement OTA over Wi-Fi

6.5.1 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.

6.5.1.1 Build OTA application image

Turn on OTA command

The flag defined in *\project\realtek_amebapro_v0_example\inc\platform_opts.h*.

```
#define CONFIG_OTA_UPDATE         1
```

Check current loaded firmware index and acquire the upgraded firmware address

```
/* ota_8195b.c */
uint32_t update_ota_prepare_addr(void){
        uint32_t NewFWAddr;
        fw_img_export_info_type_t *pfw_image_info;

        pfw_image_info = get_fw_img_info_tbl();

        printf("₩n₩r[%s] Get loaded_fw_idx %d₩n₩r", __FUNCTION__, pfw_image_info-
>loaded_fw_idx);
        if(pfw_image_info->loaded_fw_idx == 1)
                NewFWAddr = pfw_image_info->fw2_start_offset;
        else if(pfw_image_info->loaded_fw_idx == 2)
                NewFWAddr = pfw_image_info->fw1_start_offset;
        else {
                printf("₩n₩r[%s] Unexpected index %d", __FUNCTION__, pfw_image_info-
>loaded_fw_idx);
                return -1;
        }

        printf("₩n₩r[%s] NewFWAddr %08X₩n₩r", __FUNCTION__, NewFWAddr);
        return NewFWAddr;
}
```

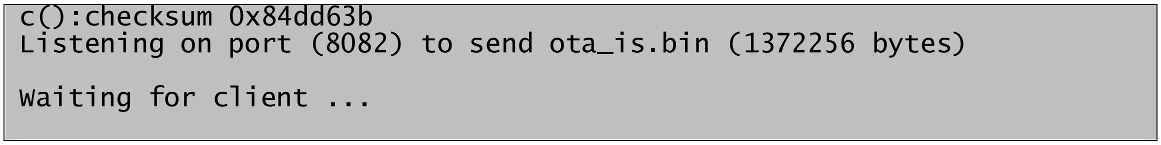
6.5.1.2 Setup local download server

Step 1: Build new firmware ota_is.bin and place to tools\DownloadServer folder.

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

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

Step 3: Execute start.bat.

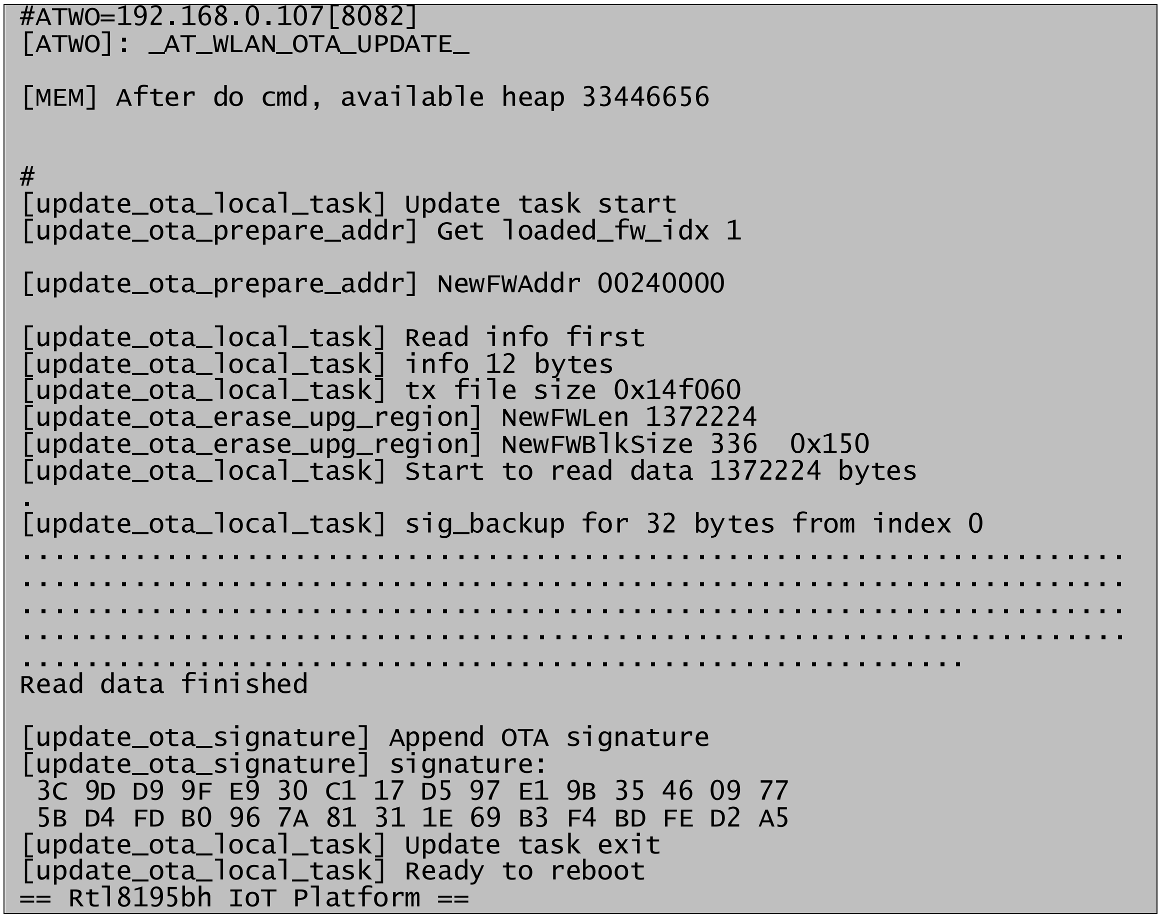


```
c():checksum 0x84dd63b
Listening on port (8082) to send ota_is.bin (1372256 bytes)

Waiting for client ...
```

6.5.1.3 Execute OTA procedure

After device connect to AP, enter command: ATWO=IP[PORT]



```
#ATWO=192.168.0.107[8082]
[ATWO]: _AT_WLAN_OTA_UPDATE_

[MEM] After do cmd, available heap 33446656


#
[update_ota_local_task] Update task start
[update_ota_prepare_addr] Get loaded_fw_idx 1

[update_ota_prepare_addr] NewFWAddr 00240000

[update_ota_local_task] Read info first
[update_ota_local_task] info 12 bytes
[update_ota_local_task] tx file size 0x14f060
[update_ota_erase_upg_region] NewFWLen 1372224
[update_ota_erase_upg_region] NewFWBlkSize 336  0x150
[update_ota_local_task] Start to read data 1372224 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:
 3C 9D D9 9F E9 30 C1 17 D5 97 E1 9B 35 46 09 77
 5B D4 FD B0 96 7A 81 31 1E 69 B3 F4 BD FE D2 A5
[update_ota_local_task] Update task exit
[update_ota_local_task] Ready to reboot
== Rtl8195bh IoT Platform ==
```

Local download server success message:

```
c():checksum 0x84dd63b
Listening on port (8082) to send ota_is.bin (1372256 bytes)

Waiting for client ...
Accept client connection from 192.168.0.110
Send checksum and file size first
Send checksum byte 12
Sending file...
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
......................................................................
..............................
Total send 1372256 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.

6.5.2 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 *ota_is.bin* after receiving the http request.

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

6.5.2.1 Build OTA application image

Turn on OTA command

The flags defined in *\project\realtek_amebapro_v0_example\inc\platform_opts.h* and *\component\soc\realtek\8195b\misc\platform\ota_8195b.h*.

```
/* platform_opts.h */
#define CONFIG_OTA_UPDATE           1
#define CONFIG_EXAMPLE_OTA_HTTP         1
```

```
/* ota_8195b.h */
#define HTTP_OTA_UPDATE
```

Define Server IP and PORT in example_ota.c file

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

```
#define PORT            8082
#define IP             "192.168.0.107"
#define RESOURCE        "ota_is.bin"
```

Example:

SERVER: http://m-apps.oss-cn-shenzhen.aliyuncs.com/051103061600.bin

Setting: #define PORT 80

#define HOST "m-apps.oss-cn-shenzhen.aliyuncs.com"

#define RESOURCE "051103061600.bin"

Check current loaded firmware index and acquire the upgraded firmware address

```
/* ota_8195b.c */
uint32_t update_ota_prepare_addr(void){
        uint32_t NewFWAddr;
        fw_img_export_info_type_t *pfw_image_info;

        pfw_image_info = get_fw_img_info_tbl();

        printf("₩n₩r[%s] Get loaded_fw_idx %d₩n₩r", __FUNCTION__, pfw_image_info-
>loaded_fw_idx);
        if(pfw_image_info->loaded_fw_idx == 1)
                NewFWAddr = pfw_image_info->fw2_start_offset;
        else if(pfw_image_info->loaded_fw_idx == 2)
                NewFWAddr = pfw_image_info->fw1_start_offset;
        else {
                printf("₩n₩r[%s] Unexpected index %d", __FUNCTION__, pfw_image_info-
>loaded_fw_idx);
                return -1;
        }

        printf("₩n₩r[%s] NewFWAddr %08X₩n₩r", __FUNCTION__, NewFWAddr);
        return NewFWAddr;
}
```

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.

6.5.2.2 Setup local HTTP download server

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

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

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

Step 3: Execute start.bat.

```
<Local HTTP Download Server>
Listening on port (8082) to send ota_is.bin (1365984 bytes)

Waiting for client ...
```

6.5.2.3 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] Get loaded_fw_idx 1

[update_ota_prepare_addr] NewFWAddr 00240000


[http_update_ota] Download new firmware begin, total size : 1365984

[update_ota_erase_upg_region] NewFWLen 1365952
[update_ota_erase_upg_region] NewFWBlkSize 334  0x14e.
[http_update_ota] sig_backup for 32 bytes from 0 index
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.........................................................
[http_update_ota] Download new firmware 1365952 bytes completed

[update_ota_signature] Append OTA signature
[update_ota_signature] signature:
 47 8C 7E E0 31 01 F1 FA 5E 81 58 88 B9 70 20 65
 D5 91 B1 CA 3A 2E F1 9B D5 BD 13 54 09 F1 11 5A
[http_update_ota_task] Update task exit
[http_update_ota_task] Ready to reboot
== Rtl8195bh IoT Platform ==
```

Local download server success message:

```
<Local HTTP Download Server>
Listening on port (8082) to send ota_is.bin (1365984 bytes)

Waiting for client ...
Accept client connection from 192.168.0.110
Waiting for client's request...
Receiving GET request, start sending file...
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
........................
Total send 1366028 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.

6.6 OTA signature

To Clear or Recover OTA signature for verification via UART at command, please refer to AN0025.

7 Power Management

7.1 AmebaPro Power Save Modes

AmebaPro provide several power modes. Table 7.1 describes DeepSleep and Stanby:

Mode	DeepSleep	Standby
Initiator/domain	LS	LS
WLAN wakeup	No	Yes
STimer wakeup	Yes	Yes
GTimer wakeup	No	Yes
HSTimer wakeup	No	Yes
GPIO wakeup	Yes (GPIOA group)	Yes (GPIOA group)
ADP wakeup	Yes	No
RTC wakeup	Yes	No
PWM wakeup	No	Yes
UART wakeup	No	Yes
MII wakeup	No	No
I2C wakeup	No	Yes
ADC wakeup	No	Yes
COMP wakeup	No	Yes
USB wakeup	No	No
SDIO wakeup	No	No
SGPIO wakeup	No	Yes

Table 7.1 AmebaPro Power mode and wakeup source

There are some features needs to be noticed:

- DeepSleep saves the most power among all power modes
- After resume, DeepSleep and Standby running code from initial, not from the position where program suspend.

7.1.1 DeepSleep

DeepSleep save the most power among all power modes. HS is turned off. LS only keep resources that are required by the wakeup source.

DeepSleep needs to be invoked from LS. If user wants to invoke DeepSleep from HS, he needs to design his own ICC command/message and implement related code.

7.1.2 Standby

In Standby mode, HS is turned off. LS only keep resources that are required by the wakeup source.

In most of wakeup source except WLAN wakeup, the power consumption is stable and can be measured from current meter with low sample rate.

If user chooses wakeup from WLAN, user need configures WLAN part before call Standby. Please reference “Wakeup from WLAN” for more information.

Standby needs to be invoked from LS. If user wants to invoke Standby from HS, he needs to design his own ICC command/message and implement related code.

7.1.3 Wakeup from WLAN

If user choose wakeup from WLAN in Standby mode, he needs to configure wlan before invoke Standby.
After user configure wlan and invoke Standby or SleepPG, system would enter suspend state. In this state, LS MCU would wakeup/suspend periodically to check wlan state and decide if it needs to wakeup HS MCU. After wlan receive the wakeup packet that matches the wakeup pattern, then LS MCU would wakeup HS.

The whole progress is like below diagram:

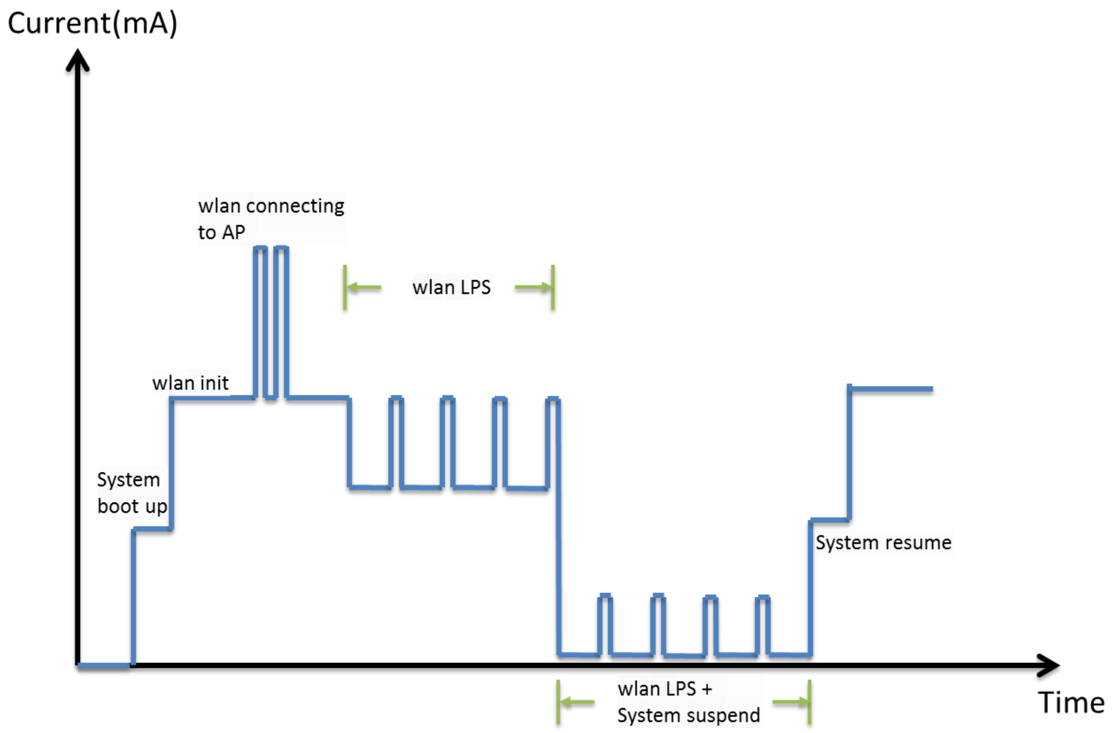


Diagram 7.1 power consumption of wake on wlan

1. At first system boot up, initialize wlan, connecting to AP.
2. After connected to AP, wlan would enter LPS state if there is no heavy data traffic. In LPS state, wlan would listen beacon for every 100ms (if DTIM is 1). So you could see power consumption rise for every 100ms. Power consumption would drop after wlan receive the beacon and the TIM field has no packet for this device, then wlan would turn off RF and try to keep in low power state.

3. If user try to make system save more power with wlan associate idle, he could invoke Standby. You could see the power consumption drops more in wlan LSP with system suspend.

Please note that if you want to measure the power consumption when system suspend with wlan LPS, you have to make sure the voltage regulator of power supply and current meter could handle the voltage drop and rise between hundreds of micro amp and dozens of milliamp.

7.2 Examples

7.2.1 Example: power_save

7.2.1.1 Abstract

This example add an AT command “PS” that user can test power saving related function included “Deepsleep”, “Standby”, “wake on wlan”, “tcp keep alive”, “wake up pattern”, and wake up reason.

7.2.1.2 Setup example

The example is located in:

“\project\realtek_amebapro_v0_example\example_sources\power_save\”

Copy HS & LS folder to src folder, compile LS & HS project, and the download the binary.

7.2.1.3 Deepsleep

Type below command in HS console:

```
PS=deepsleep
```

It would show below logs in LS console:

```
deepsleep wake event:0x0001
```

You can see current changes if you have connected current meter. System would resume after 60s. HS & LS would run from the start just like reboot. In the log in LS, “wake event” is the wake up source. You can reference “power_mode_api.h” for correspond wake up source.

Type below command in HS console:

```
PS=deepsleep,stimer=10
```

This would set stimer wake up in deepsleep for 10s.

Type below command in HS console:

```
PS=deepslee,gpio
```

This would set GPIOA_13 wake up in deepsleep.

7.2.1.4Standby

Type below command in HS:

```
PS=standby
```

It would show below logs in LS console:

```
Standby wake event:0x0001
```

You can see current changes if you have connected current meter. System would resume after 60s. HS & LS would run from the start just like reboot. In the log in LS, "wake event" is the wake up source. You can reference "power_mode_api.h" for correspond wake up source.

After system resume, it would show wake reason in LS log:

```
wake from AON_TIMER
```

Type below command in HS:

```
PS=standby,stimer=10
```

This would set stimer wake up in standby for 10s.

Type below command in HS:

```
PS=standby,gpio=2
```

This would set GPIOA_2 wake up in standby.

Type below command in HS

```
PS=standby,stimer=10,stimer_wake=0
```

This would set stimer wakeup in standby for 10s. When time reaches 10s, LS won't wake up HS, and LS would invoke standby. This command demonstrates how to setup up a periodical task in LS when HS is suspended.

Type below command in HS

```
PS=standby,stimer=10,stimer_wake=5
```

This would set stimer wakeup in standby for 10s. When time reaches 10s, LS won't wake up HS, and LS would invoke standby. Repeat this for 5 times, then LS would power on HS. In other means, LS suspend 10s for 5 times, and HS suspend for 10s*5=50s.

Type below command in HS:

```
PS=standby,gpio=2,gpio_wake=0
```

This would set GPIOA_2 wake up in standby. When user triggers GPIOA_2 and wake up LS, LS won't wake up HS, and LS would invoke standby. This command demonstrates if GPIO is triggered from PIR or push button, the LS can check PIR or button state, then LS can enter suspend without power on HS.

Type below command in HS:

```
PS=standby,gpio=2,gpio_wake=5
```

This would set GPIOA_2 wake up in standby. When user triggers GPIOA_2 and wake up LS, LS would invoke standby. Repeat this for 5 times, then LS would power on HS. In other means, LS is wake up for 5 times, and HS is wake up for 1 time.

7.2.1.5LS wake reason

As system resume from standby, HS may need to know the wake up source in LS. You can type below command in HS console:

```
PS=ls_wake_reason
```

It would show below log in HS log:

```
wake from AON_TIMER
```

7.2.1.6Wake on wlan

Type below command in HS console:

```
PS=wowlan,your_ssid,your_pw
```

System would try to connect specified AP with your_ssid & your_pw. It would enter wake on wlan mode after connected. The wake on wlan mode use standby with WLAN wake up source.

In wake on wlan mode, LS would keep restart correspond to the AP beacon interval. So you can see boot up log in LS periodically.

Wlan would wake system after it receive wake up packet. (PING, by default)

Type below commands in HS console:

```
PS=standby,wowlan,stimer=60
PS=wowlan,your_ssid,your_pw
```

The first command is similar to previous standby command except that it adds "wowlan" in standby's parameter. This would set standby parameter to LS without doing standby. The second command is the same as previous one.

After type these 2 commands, LS can be wake up by 2 sources: STIMER and WLAN. If LS doesn't receive wake up packet interrupt from WLAN within 60s, then LS would be wakeup by stimer, and then LS power on HS.

Type below commands in HS console:

```
PS=standby,wowlan,stimer=5,stimer_wake=0
PS=wowlan,your_ssid,your_pw
```

These commands are similar to previous one except that the stimer would wake LS every 5s, then LS invoke standby. These commands demonstrate that in wake on wlan mode, LS can check status every 5s.

Type below commands in HS console:

```
PS=standby,wowlan,stimer=5,stimer_wake=0,gpio=2
PS=wowlan,your_ssid,your_pw
```

These commands are similar to previous one. It can be wake up by 3 sources: STIMER, GPIO and WLAN. Under wake on wlan mode, LS is wake up by stimer every 5s and won't wake up HS. LS can also be wake up by GPIO trigger, then LS power on HS.
These commands demonstrate that in wake on wlan mode, LS check status every 5s, and boot up if user press a button.

7.2.1.7 Wake on wlan with wakeup pattern

Type below commands in HS console:

```
PS=wowlan_pattern
PS=wowlan,your_ssid,your_pw
```

The first command configures a wake on wlan pattern which matches TCP packet with destination port 80. In other means, if user opens a browser, type STA's IP under same router, then the HTTP request (TCP SYN) would wake up STA.

7.2.1.8 TCP keep alive

Type below command in HS console to setup TCP keep alive:

```
PS=tcp_keep_alive,192.168.1.100,5566
PS=wowlan,your_ssid,your_pw
```

In first command, the last 2 parameter is the IP & port of TCP server. The setting would become effective after system enter wake on wlan mode. By default it sends a TCP packet every 30s, and resend after 10s if STA does not receive TCP ACK. You can modify the interval in the code:

```
static uint32_t interval_ms = 30000;
static uint32_t resend_ms = 10000;
```

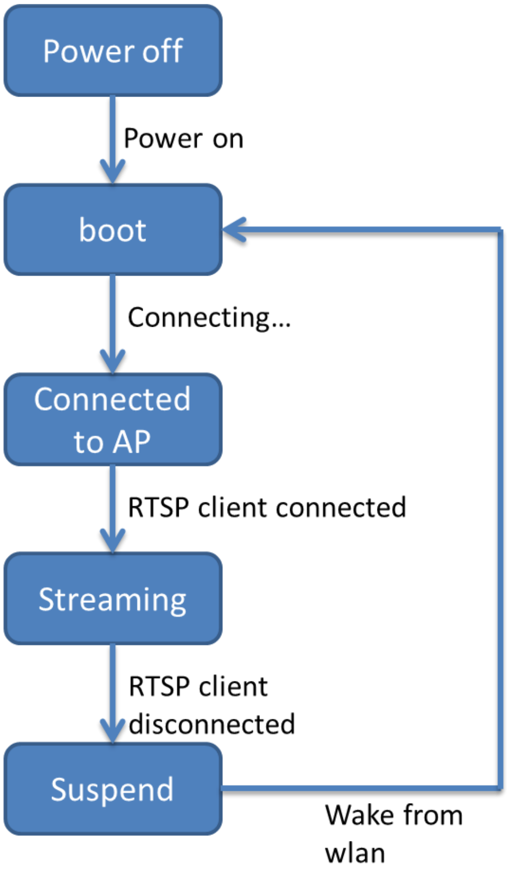
In the code, it also adds a wake on wlan pattern that matches TCP packet with same source port in tcp keep alive, and match TCP flag with PSH+ACK. These setting would allow TCP server wakeup STA by sending a packet to STA.

7.2.2 Example: system suspend with wake on wlan after streaming stop

7.2.2.1 Abstract

This example demonstrate (1) how to get first video frame after boot/resume, and (2) how to make system suspend and able to be waked from wlan.

The life cycle is shown as below diagram:



- At first, system is boot from power on or wake from wlan(describe later)
- Then AmebaPro try connecting to AP either by fast connect feature or AT command.

- After AmebaPro connected to AP, AmebaPro would start RTSP server and wait for RTSP client.
- User may use VLC or other RTSP client connects to AmebaPro. User should see streaming content **which is buffered after boot up**. So user can evaluate the time from boot up to get the first frame.
- After user stop VLC client, AmebaPro would enter system suspend with wlan association idle.
- User can wake Ameba by sending ping to AmebaPro.

7.2.2.2 Setup example

You need to use power_save example in example_sources.

You also needs modify file **"platform_opts.h"** (placed in "project\realtek_amebapro_v0_example\inc\platform_opts.h")

And enable these compile flags:

You can reference example **"media_framework"** (placed in "component\common\example\media_framework")

```
#define CONFIG_EXAMPLE_MEDIA_FRAMEWORK                  1
#if CONFIG_EXAMPLE_MEDIA_FRAMEWORK
#define FATFS_DISK_SD                                   1
#define ISP_BOOT_MODE_ENABLE                           1
#if ISP_BOOT_MODE_ENABLE
#define ISP_BOOT_MP4                                    0
#define ISP_BOOT_RTSP                                  1
#if (ISP_BOOT_MP4 == 1 && ISP_BOOT_RTSP == 1) || (ISP_BOOT_MP4 == 0 && ISP_BOOT_RTSP == 0)
#error "It only can select the mp4 or rtsp"
#endif
#endif
#endif
```

This would enable media framework example. You can reference "example_media_framework.c" (placed in "\component\common\example\media_framework\example_media_framework.c").

After compile and flash image, boot up AmebaPro and should see logs as below:

```
WIFI initialized

init_thread(53), Available heap 0x9b0f60
use ATW0, ATW1, ATWC to make wifi connection
wait for wifi connection...
```

Then you can use AT command to connect to AP.

After connected to AP, AmebaPro would start RTSP server.

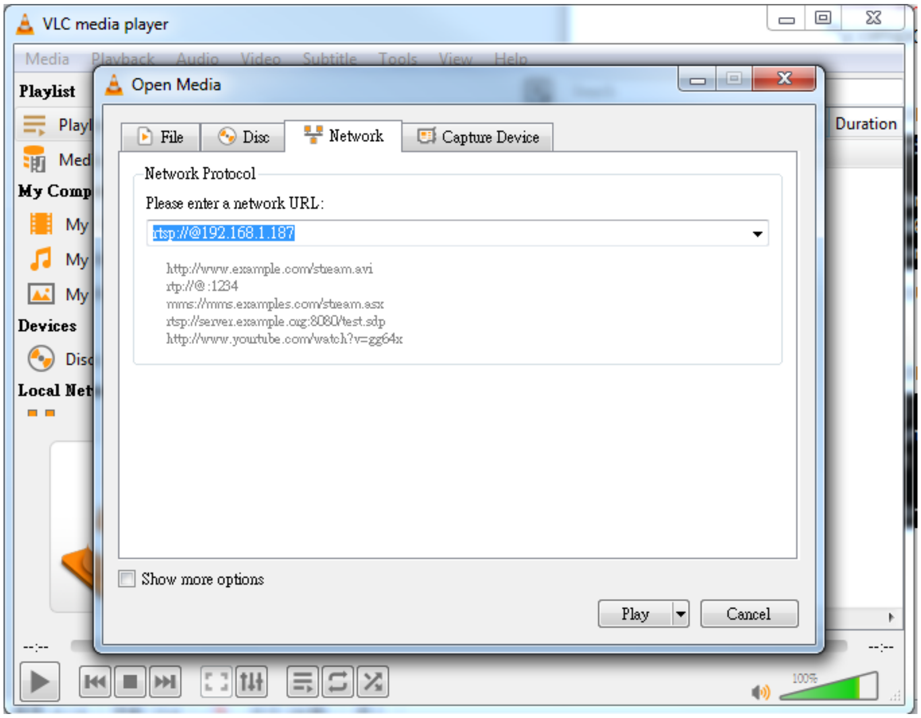
```
RTSP[0] port: 554
```

```
time_sync_enable
connect successful sta mode

rtsp stream enabled
```

You can use ping command to (1) test AmebaPro’s connection, and (2) update PC or Smart Phone’s ARP table.

Then open VLC as RTSP client and connected to AmebaPro:



Network Protocol

Please enter a network URL:

rtsp://@192.168.1.187

http://www.example.com/stream.avi
rtp://@:1234
mms://mms.examples.com/stream.asx
rtsp://server.example.org:8080/test.sdp
http://www.youtube.com/watch?v=gg64x

You should see streaming after RTSP client connected.

After you stop streaming, system would enter power save state.

By default AmebaPro is wake by ping. So you can ping AmebaPro again to resume it.

After resume, AmebaPro is just like boot from power on, so you can exam the example again.

7.3 Programming Guide

7.3.1 Turn off wlan if using standby without wowlan

If we want to use standby but we do not need wake on wlan mode, then we must turn off wifi before enter standby mode. This scenario usually happens when AmebaPro fails to connect to AP and it still needs power save. If we do not turn off wifi and enter standby, then the device would hang.

7.3.2 Do not use ICC while HS is suspend

We cannot use ICC while HS is suspended. The code in LS should separate into 2 cases: HS is active, and HS is suspended. If user tries to initialize ICC while HS is suspended, it would result un-expected behavior.

7.3.3 Turn off Video, ISP, SD, and USB before standby

If we ever turned on video, ISP, SD, or USB, then we must turn them off before enter standby, otherwise HS would boot fail when HS is resumed.

7.3.4 Use retained memory in LS in Standby

The SRAM in LS is retained in Standby power save mode. But the BSS section in LS would be cleared when LS is resumed. If we want to use some persist data in LS, we need to put these variables in DATA section.

7.3.5 A little delay between connected to AP and wake on wlan mode

If we want to enter wake on wlan mode immediately after connected to AP, it needs to add a little delay between it. When STA connected to AP, it usually has some packets send to STA side. If we put device enter wake on wlan mode in this moment, then STA would exit LPS mode and try to receive this packet, it would takes more time to enter wake on mode. It won’t cause any issue though, but we still suggest add a little delay to make STA receive all packets, it would shorten the time to receive these packet and save more power.

7.3.6 Set LS GPIO pull control before standby

Check the LS GPIO external circuit status and set the LS GPIO states matching the external circuit to prevent leakage.

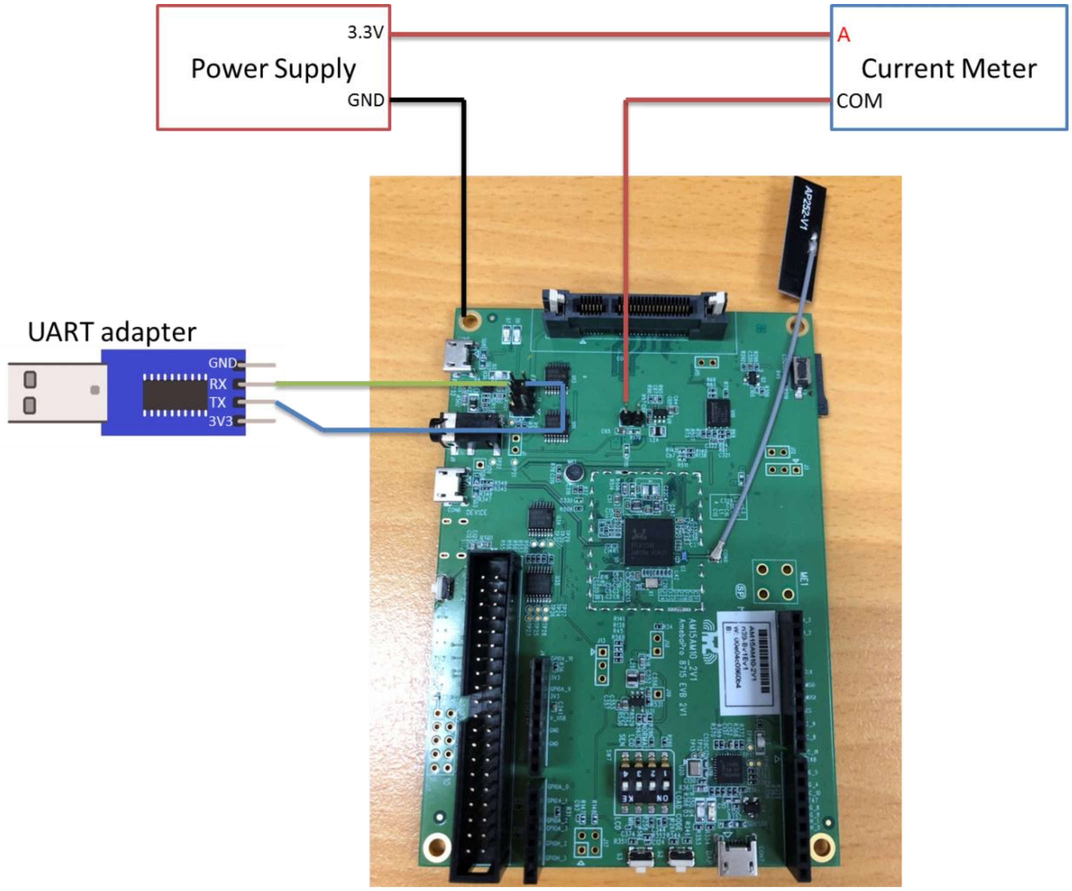
- External pull high → internal pull high
- External pull low or floating → internal pull low

```
union {
    __IOM uint32_t gpioa_pull_ctrl;  /*!< (@ 0x00000144) GPIOA Pull Control Register*/

    struct {
        __IOM uint32_t gpioa0_pull_ctrl : 2;/*!< [1..0] 2b'00: high impedence; 2b'01:
pull low; 2b'10: pull                          high; 2b'11: reserved */
        __IOM uint32_t gpioa1_pull_ctrl : 2;      /*!< [3..2] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa2_pull_ctrl : 2;      /*!< [5..4] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa3_pull_ctrl : 2;      /*!< [7..6] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa4_pull_ctrl : 2;      /*!< [9..8] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa5_pull_ctrl : 2;      /*!< [11..10] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa6_pull_ctrl : 2;      /*!< [13..12] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa7_pull_ctrl : 2;      /*!< [15..14] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa8_pull_ctrl : 2;      /*!< [17..16] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa9_pull_ctrl : 2;      /*!< [19..18] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa10_pull_ctrl : 2;     /*!< [21..20] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa11_pull_ctrl : 2;     /*!< [23..22] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
        __IOM uint32_t gpioa12_pull_ctrl : 2;     /*!< [25..24] 2b'00: high impedence;
2b'01: pull low; 2b'10: pull high; 2b'11: reserved
*/
    } gpioa_pull_ctrl_b;
  };
```

7.4 Measure Power Consumption

We can measure power consumption at EVB 2V1 board. You need to download the target image first, then remove the 2 jumpers at J9 and 1 jumper at J21. And setup equipment as below:



Here are some notice while setup these equipment:

- Power supply:

  We use 3.3V to measure power consumption. While measuring the power consumption, the voltage may drops too low in a small period of time if power supply cannot responds the current change in time. In this situation you can try to add capacitors to avoid this.
- Current meter:

If you are about to measure wake on wlan power consumption, you need to fix the current range to 1A on current meter. Otherwise the current would change from hundred microamperes to hundred milliamperes, and it would cause device hang from voltage drops.

You also need to setup the sample rate of the current meter to 1K or higher. When station listen to beacon from wifi AP, it only open wifi RF for several milliseconds. You need higher sample rate to capture these activities. You should see current/time diagram like Diagram 7.1

- UART adapter:

  UART adapter is for you to input commands. However, the TX/RX might provide some power to AmebaPro, so it is VERY IMPORTANT to remove the TX/RX line after you input commands and begin to measure power consumption.

8 System

8.1 ICC (Inter CPUs Communication)

There are 2 CPUs, HP (High-Power) CPU and LP (Low-Power) CPU, on the RTL8195B platform. The ICC (Inter CPUs Communication) hardware is designed for the purpose of making 2 CPUs can communicate each other.

8.1.1 Example: ICC

8.1.1.1Abstract

This example shows:

1. How to send ICC commands from HS/LS platform to the LS/HS platform.
2. A ICC message loop back. Icc message data is sent from HS to LS and then LS loop back ICC message to the HS.

8.1.1.2 Setup example

The example is located in:

“\project\realtek_amebapro_v0_example\example_sources\icc\”

You should copy both HS and LS folder to project folder. Compile both LS and HS code and download the program.

8.2 System reset

8.2.1 ls_sys_reset

```
/* Reset LS platform via ICC command*/
void ls_sys_reset( void )
```

8.2.2 sys_reset

```
/* Reset some register status and do HS watchdog reset */
void sys_reset( void )
```

9 Hardware Peripherals

9.1 GTimer

- 16 Gtimer supported at HS domain and 8 Gtimer supported at LP domain.
- Source clock is selected from 20MHz and moderate accuracy 32KHz.

9.1.1 gtimer_rtc

This example shows how to use general timer API to implement a software RTC.

gtimer_rtc example is located in

project\realtek_amebapro_v0_example\example_sources\gtimer_rtc

9.1.2 SNTP

This example shows system time maintained by time from NTP server.

sntp_showtime example is located in component\common\example\sntp_showtime

9.1.2.1 Software Setup

configure the SNTP example settings in *(project)\inc\platform_opts.h*:

```
/* For sntp show time example */
#define CONFIG_EXAMPLE_SNTP_SHOWTIME        1
```

9.2 RTC

RTC is invoked from LS MCU. This example shows how to init and read/write RTC.

RTC example is located in project\realtek_amebapro_v0_example\example_sources\rtc

9.3 PWM

- Support maximum 8 PWM functions at HS domain and LP domain
- 0~100% duty can be configurable
- Use selected Gtimer interrupt as counter source

9.4 Audio

- Default DAC value is 0xAF: 0dB
- Default ADC value is 0x2F: 0dB

9.4.1 DAC Volume

```
/**
  * @brief  Control the digital gain of DAC.
  * @param  obj: Audio object define in application software.
  * @param  step: The digital volume. Every Step is 0.375dB.
  *           @arg 0xAF: 0dB.
  *           @arg 0xAE: -0.375dB.
  *           @arg ...
  *           @arg 0x00: -65.625dB.
  * @retval none
  */
void audio_dac_digital_vol (audio_t *obj, u8 step);

/**
  * @brief  Control the digital mute of DAC.
  * @param  obj: Audio object define in application software.
  * @param  mute_en: To enable or disable mute.
  * @retval none
  */
void audio_dac_digital_mute (audio_t *obj, BOOL mute_en);
```

9.4.2 ADC Volume

```
/**
  * @brief  Control the digital gain of ADC.
  * @param  obj: Audio object define in application software.
  * @param  step: The digital volume. Every Step is 0.375dB.
  *           @arg 0x7F: 30dB.
  *           @arg ...
  *           @arg 0x2F: 0dB.
  *           @arg ...
  *           @arg 0x00: -17.625dB.
  * @retval none
  */
void audio_adc_digital_vol (audio_t *obj, u8 step);

/**
  * @brief  Control the digital mute of ADC.
  * @param  obj: Audio object define in application software.
  * @param  mute_en: To enable or disable mute.
  * @retval none
  */
```

```
void audio_adc_digital_mute (audio_t *obj, BOOL mute_en);
```

9.5 Efuse

- LS efuse
  - Logical area : 0x0 ~ 0x1CF , size: 464 bytes
  - Reserved area : 0x1D0 ~ 0x1FF , size: 48 bytes

  Total size : 512 bytes

- HS efuse
  - Logical area : 0x0 ~ 0x10F , size: 272 bytes
  - OTP : 0x110~0x12F = 32 bytes
  - security area 0x130~0x1AF = 128 bytes
    - security key: 0x130~0x14F
    - writable security zone : 0x150~0x16F
    - super security key: 0x170~0x18F
    - reserved: 0x190~0x1AF
  - wlan hidden area : 0x1B0 ~ 0x1FF = 80 bytes

  Total size : 512 bytes

9.5.1 User OTP

User OTP : 0x110~0x12F = 32 bytes

efuse_api

```
/ /**
  * @brief  Read efuse OTP contant
  * @param  address: Specifies the offset of the OTP.
  * @param  len: Specifies the length of readback data.
  * @param  buf: Specified the address to save the readback data.
  * @retval 0: Success
  * @retval -1: Failure
  */
int efuse_otp_read(u8 address, u8 len, u8 *buf);

/**
  * @brief  Write user's contant to OTP efuse
  * @param  address: Specifies the offset of the programmed OTP.
  * @param  len: Specifies the data length of programmed data.
  * @param  buf: Specified the data to be programmed.
  * @retval 0: Success
  * @retval -1: Failure
  */
int efuse_otp_write(u8 address, u8 len, u8 *buf);
```

9.6 Ethernet

configure the Ethernet interface settings in (project)\component\common\mbed\hal_ext\ethernet_ex_api.h:

```
/// the selection of Ethernet pinmux
#define ETH_PIN_SEL         (Eth_Pin_Sel1)//
/// the selection of the interface between MAC and PHY
#define ETH_IF_SEL          (Eth_If_Mii)
```

KEY3 of switch SW7 should be switch to MII on EVB.

9.7 SD CARD

This section explains how the SD CARD API is used.

- sdio_driver_init() :Assign the function pointer API for low level operation.
- SD_Init() :Initializes the SD card.
- SD_DeInit(): DE initialize the SD card.
- SD_Wait_Ready() :Currently only has a wait for SPI, SD card does not need.
- SD_SetCLK(enum sdioh_access_mode_e clk) :Sets SD card clock speed. The driver will fill the clock, it do not need to fill it.
- SD_Status(): Gets the current SD card status, and returns 0 for success.
- SD_GetCID(): It don't provide the information now.
- SD_GetCSD() :It don't provide the information now.
- SD_GetCapacity(): Gets the current number of sectors.
- SD_ReadBlock(u32 sector,u8 *data,u32 count) :SD card read function, parameter sector position, data processing data buffer, count how many sector count.
- SD_WriteBlock(u32 sector,u8 *data,u32 count) : SD card write function, parameter sector position, data processing data buffer, count how many sector count.

It must to call the sdio_driver_init() firstly, and then to do the SD_Init().After the action, we can do the write and read operation. If we don't need to use the SD card, it will call the SD_Deinit to close the SD card.

The speed setting is at below.

```
        SdiohSpeedDS          = 0,  // 3.3V Function 0 //Default Speed
        SdiohSpeedHS          = 1,  // 3.3V Function 1 //High Speed
        SdiohSpeedSDR12       = 2,  // 1.8V Function 0
        SdiohSpeedSDR25       = 3,  // 1.8V Function 1
        SdiohSpeedSDR50       = 4,  // 1.8V Function 2
        SdiohSpeedSDR104      = 5,  // 1.8V Function 3
        SdiohSpeedDDR50       = 6,  // 1.8V Function 4
        SdiohKeepCurSpd       = 15
```

The following are the precautions items.

- The SD card has the power control pin. The demo board and QFN128 use the different pin to control. The demo board is GPIOH_0 and the QFN128 is GPIOE_12. The pin must to pull low to give the SD card power. If we need to reset the SD card, it needs to power off for SD car. The procedure is low too high to do the reset.
- It needs to confirm that the SD card is already initialized or it will get the wrong status.
- If we need to write the SD card for several threads in fat file system, it need to open the reentrant flag to do the protect procedure.
- Do not pull the card during the card writing process; otherwise the file system will be damaged.
- In order to speed the R/W speed, it is recommended that we use the temp buffer to store the data. If the data is big enough then we write back to the SD card, the buffer is suggesting 32 or 64KB. Must be an integer multiple of 512.
- We provide the FATFS file system to do the SD card operation.

10 File system

AmebaPro support file system based on flash and SD card and AmebaPro can support these two storages simultaneously. File system AmebaPro support is FAT32 for SD card, if SD card is exFAT or other, it should be format at first. User can use FATFS API *f_mkfs* to do format procedure.

10.1 FAT Filesystem on Flash

Flash file system example is located in *component\common\example\fatfs*

10.1.1 Software Setup

First, configure the FATFS example settings in *(project)\inc\platform_opts.h*:

```
/* For FATFS example*/
#define CONFIG_EXAMPLE_FATFS               1
#if CONFIG_EXAMPLE_FATFS
#define CONFIG_FATFS_EN      1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
// fatfs disk interface
#define FATFS_DISK_USB       0 // Not supported on AmebaPro
#define FATFS_DISK_SD        0
#define FATFS_DISK_FLASH    1
#endif
#endif
```

Next, modify parameters in *component\common\file_system\fatfs\r0.10c\include\ffconf.h:*

```
…
#define_USE_MKFS             1//0   /* 0:Disable or 1:Enable */
…
…
…
#define_MAX_SS       4096//512
…
```

Please note that the flash memory base for the flash filesystem used in the Flash FATFS example is defined in the file *component\common\file_system\fatfs\disk_if\src\flash_fatfs.c:*

```
…
#define FLASH_APP_BASE  0x180000
…
```

Finally, rebuild the project and download active application to DEV board.

10.2 Dual FAT Filesystem - File system on both SD Card and Flash

Dual file system example is located in *component\common\example\fatfs*

10.2.1 Hardware Setup

Please connect your Ameba DEV board with SD/MMC card connector as described in UM0073.

10.2.2 Software Setup

First, configure the FATFS example settings in *(project)\inc\platform_opts.h*:

```
/* For FATFS example*/
#define CONFIG_EXAMPLE_FATFS                1
#if CONFIG_EXAMPLE_FATFS
#define CONFIG_FATFS_EN      1
#if CONFIG_FATFS_EN
// fatfs version
#define FATFS_R_10C
// fatfs disk interface
#define FATFS_DISK_USB       0 // Not supported on AmebaPro
#define FATFS_DISK_SD        1
#define FATFS_DISK_FLASH     1
#endif
#endif
```

> *In AmebaPro C-cut QFN128, GPIOE_12 is low active to used to control SDIO power. If users don't need to use power control, please don't control GPIOE_12.*

Next, modify parameters in *ffconf.h*:

```
…
#define_USE_MKFS      1
…
#define _VOLUMES      2
…
#define_MAX_SS        4096
…
```

Please note that the flash memory base for the flash filesystem used in the Flash FATFS example is defined in the file *component\common\file_system\fatfs\disk_if\src\flash_fatfs.c:*

```
…
#define FLASH_APP_BASE  0x180000
…
```

Finally, rebuild the project and download active application to DEV board.