AN0700
AmebaPro2 Application Note

Abstract

AmebaPro2 is a high-integrated IC. Its features include 802.11 Wi-Fi, H.264/H.265 video codec, Audio Codec.
This manual introduce users how to develop AmebaPro2, including SDK compiling and downloading image to AmebaPro2.

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

Note

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.

USING THIS DOCUMENT

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

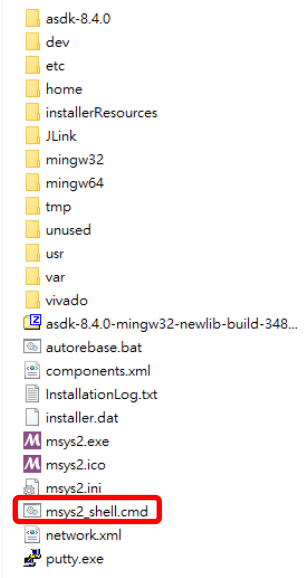
Contents

1 Building Environment

1.1 Setting up GCC Building Environment

1.1.1 Installing mingw with ASDK and setting up the CMake

- Download and extract msys64-1210.7z from tools folder
- Double click “msys2_shell.cmd” from mysys64 folder

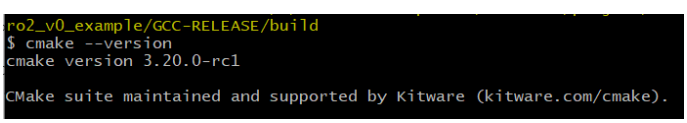


- After setting up mingw, you need to install cmake. Download cmake in https://github.com/Kitware/CMake/releases/download/v3.20.0-rc1/cmake-3.20.0-rc1-windows-x86_64.msi and install it
- Add location of cmake.exe to PATH of msys2_shell by using vim ~/.bashrc and appending path of cmake.exe to environment variable PATH
- Or using editor to directly append the path

```
if [ -d "../../asdk-8.4.0" ]; then
  echo "asdk-8.4.0 exist"
  export PATH=/asdk-8.4.0/mingw32/newlib/bin:$PATH
elif [ -d "../../asdk-8.3.0" ]; then
  echo "asdk-8.3.0 exist"
  export PATH=/asdk-8.3.0/mingw32/newlib/bin:$PATH
else
  echo "unzip asdk-8.4.0"
  cd /
  unzip asdk-8.4.0-mingw32-newlib-build-3485.zip
  export PATH=/asdk-8.4.0/mingw32/newlib/bin:$PATH
  cd ~
fi


export LANG='en_US.utf-8'
export PATH=/JLink:$PATH:/mingw64/bin
export PATH=/d/CMake/bin:$PATH
```

Note: For the first time adding the CMake PATH, after adding the PATH, you need to re-open the msys2_shell and check whether the CMake is added.

- you can check the cmake installation by “cmake --version”



```
ro2_v0_example/GCC-RELEASE/build
$ cmake --version
cmake version 3.20.0-rc1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

1.1.2 Building the project in GCC Building Environment

- Open mingw by double clicking “msys2_shell.cmd”.
- Enter the project location: project/realtek_amebapro2_v0_example/GCC-RELEASE/.
- Create folder “build” and enter “build” folder.
- Run “cmake .. -G"Unix Makefiles" -DCMAKE_TOOLCHAIN_FILE=../toolchain.cmake” to create the makefile.
- Run “cmake --build . --target flash” to build and generate flash binary.

Note: if the amebapro2 board is with ddr2 or ddr3 ram, set “#define DRAM_TYPE_DDR2 1” or “#define DRAM_TYPE_DDR3 1” in file /component/soc/8735b/fwlib/rtl8735b/include/hal_dram_init.h.

1.1.3 Building the project in GCC Building Environment (LINUX)

- Add toolchain to the linux PATH

  Extract the toolchain file (the toolchain file may provide in folder tools):
  tar -jxvf <path_to_toolcahin>/<toolchain_file(*.tar.bz2)> -C <path_to_file _extracted>

  Add the file to PATH:
  export PATH=$PATH:<path_of_the_toolchain_execute_file>

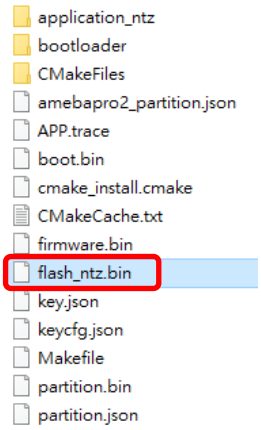
Note: make sure that all of the C compiler is under <path_of_the_toolchain_execute_file>

- Open linux terminal and enter the project location: project/realtek_amebapro2_v0_example/GCC-RELEASE/.
- Create folder “build” and enter “build” folder.
- Run “cmake .. -G"Unix Makefiles" -DCMAKE_TOOLCHAIN_FILE=../toolchain.cmake” to create the makefile.
- Run “cmake --build . --target flash” to build and generate flash binary.

Note: the folder name is not needed to be ‘build’, user can use other name.
Note: if the folder ‘build’ has been used by other platform (like linux, mingw) or other toolchain version before, suggest to remove the CMakeCache.txt and rebuild the project.

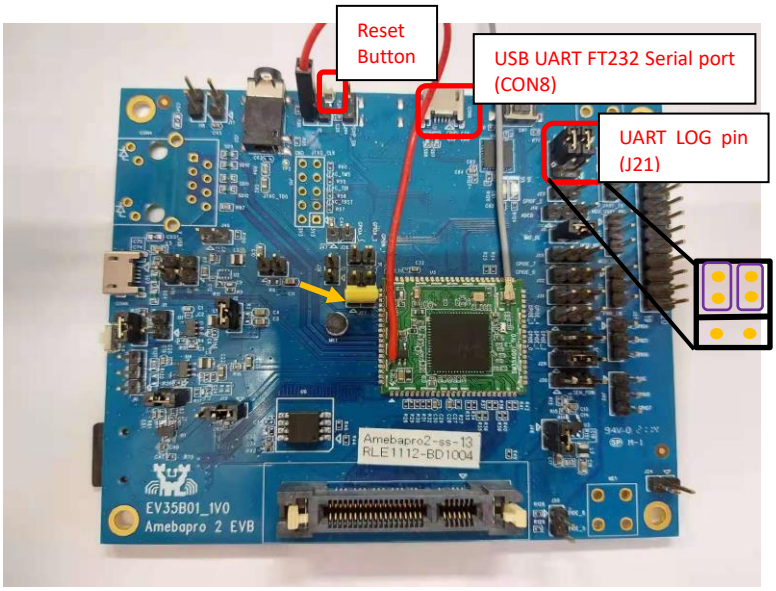
1.1.4 Choosing the fw image

- If building successfully, you can see flash_ntz.bin in the build folder. Download it to amebaPro2 device.



1.2 Log UART Settings

- To use AmebaPro2 log UART, the user needs to connect jumpers to **J21** for **FT232** (**CON8**). Remove the red line from pin J9 (close to audio jack one).

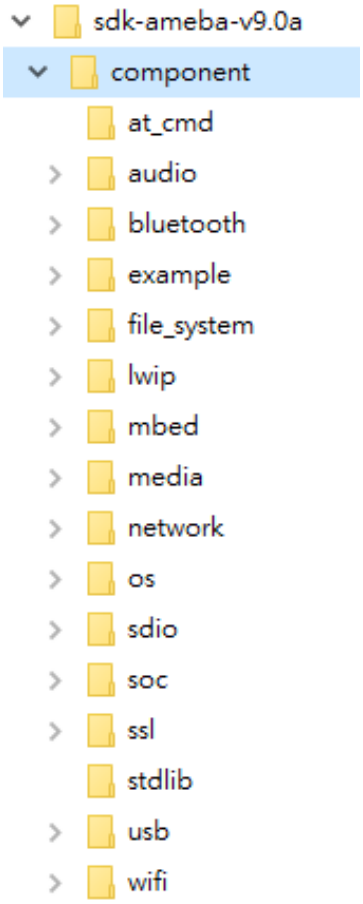


- After using CON8 to connect to PC, you can use console tools (like tera term, MoBaxterm) to get log from EVB by setting baud rate as **115200**.

2 SDK Architecture

In Amebapro2 sdk, it mainly contain four folders. The folder “component” store the main component source and the folder “project” contains the project make file, compile flag and some examples. The folder “doc” and “tool” provide the document and tools for assisting you to set up the project.

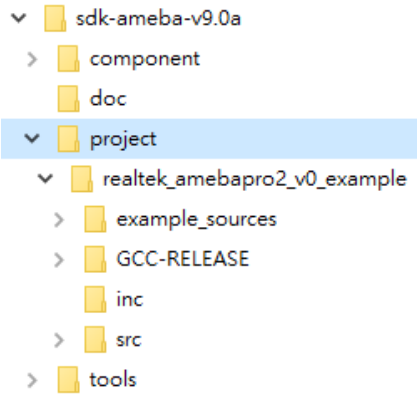
2.1 Component



Folder	Sub-folder	Description
Component	at-cmd	AT-command
	audio	ASP algorithm api audio codec
	bluetooth	bluetooth dri
	example	mmf audio examples (media_framework) utility examples: wlan_fast_connect/ssl_download/fatfs/uvc
	file_system	Fatfs DCT
	lwip	lwip API source code
	mbed	mbed API source code
	media	multi-media framework modules rtp codec for media

	network	cJSON coap dhcp http iperf mDNS mqtt ping rtsp sntp tftp websockect xml
	os	freertos: freertos source code os_dep: Realtek encapsulating interface for FreeRTOS, ram usage…
	soc	app: monitor and shell cmsis: cmsis style header file and startup file fwlib: hal drivers and nn api mbed-drivers: Mbed API source code misc: driver and utilities
	ssl	ssl stub function and ram map source code
	stdlib	stdlib header files
	usb	usb and uvc header files
	wifi	wifi api and wifi config related source code and header files

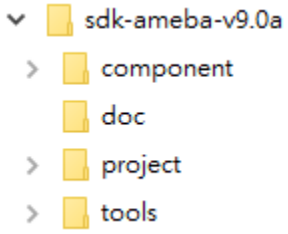
2.2 Project



Folder	Sub-folder	Description
Project	**realtek_amebapro2_v0_example**	GCC project entry for Amebapro2
	$/GCC-RELEASE	GCC cmake projects
	$/~/application_ntz	libraries for (non trust zone) GCC project

	$/~/build	pre-build image files (boot.bin) and json files place for building cmake projects and generate flash image file (flah_ntz.bin)
	$/~/ROM	ROM code libraries
	$/inc	the header files for setting the project compile flag
	$/src	the main file source code for the project
	$/~/mmfv2_video_example	source code for mmf examples with video
	$/~/nn_verify(nn)	NN model sample code
	$/~/voe_lib	voe make file
	$/example_sources	examples for peripherals

2.3 Doc and tools



3 GCC Makefile

Before building the amebapro2 project, you should install the mingw and CMake first.
After installing mingw and CMake, go to the project file located folder by command “cd <sdk-folder-location>/project/realtek_amebapro2_v0_example/GCC-RELEASE/build”.

```
asdk-8.4.0 exist

weirenchen@R074620100 MSYS ~
$ cd d:/amebapro2/sdk-ameba-v9.0a/project/realtek_amebapro2_v0_example/GCC-RELEA
SE/build
```

Use “cmake .. -G"Unix Makefiles" -DCMAKE_TOOLCHAIN_FILE=../toolchain.cmake” to generate the make files according to the CMakeLists.txt
Note: If it show the error about the file is already existed, you can delete the file “CMakeCache.txt” and type the command again.

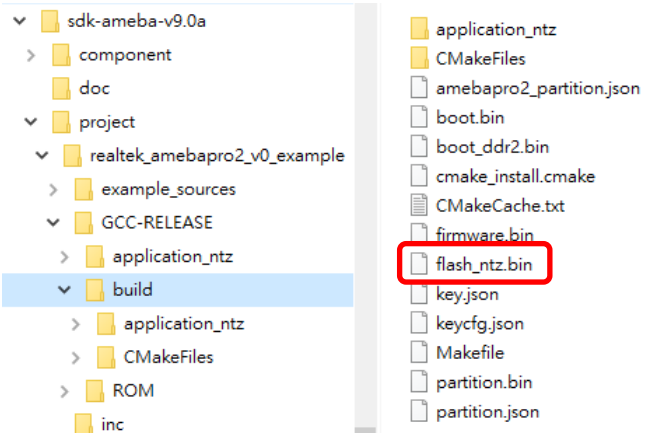
```
weirenchen@R074620100 MSYS /d/amebapro2/sdk-ameba-v9.0a/project/realtek_amebapro
2_v0_example/GCC-RELEASE/build
$ cmake .. -G"Unix Makefiles" -DCMAKE_TOOLCHAIN_FILE=../toolchain.cmake
-- The C compiler identification is GNU 8.4.0
-- The CXX compiler identification is GNU 8.4.0
```

You can see “Build files have been written to:” if build successfully. After the makefile is build, you can use “cmake --build . --target flash” to build the image file through the makefiles.

```
-- Build files have been written to: D:/amebapro2/sdk-ameba-v9.0a/project/realte
k_amebapro2_v0_example/GCC-RELEASE/build

weirenchen@R074620100 MSYS /d/amebapro2/sdk-ameba-v9.0a/project/realtek_amebapro
2_v0_example/GCC-RELEASE/build
$ cmake --build . --target flash
```

If building successfully, you can see image file “flash_ntz.bin” in bui ld folder.

```
[INFO]
PTAB ==> partition.bin
BOOT ==> boot.bin
FW1 ==> firmware.bin
[100%] Built target flash
```



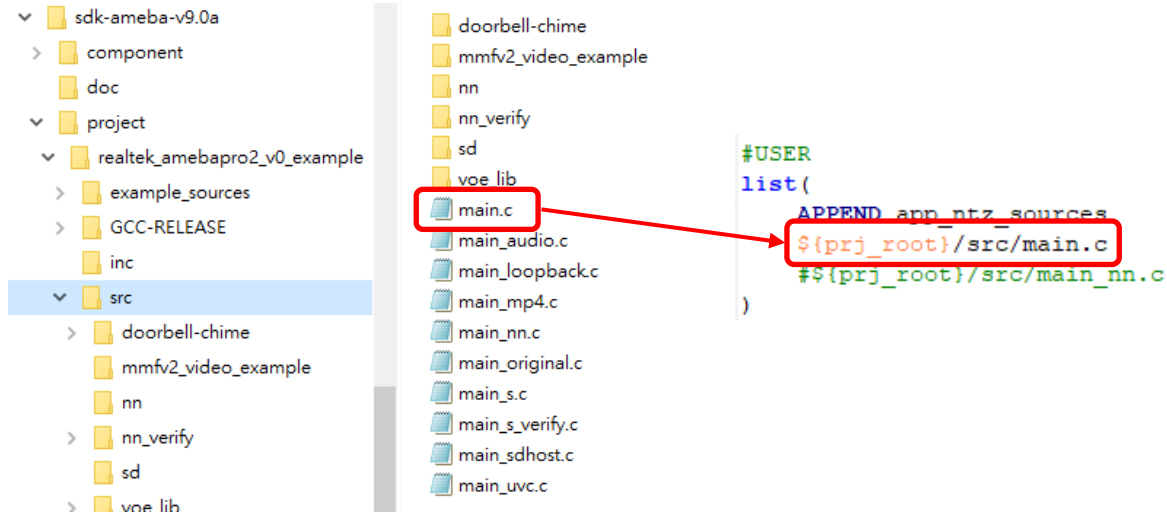
3.1 Adding a file in CMake project

In the section, it will introduce how to add exact files to Amebapro2 project, including adding source, header files, creating and linking the library files.

3.1.1 Adding a source code file or header file

- Adding Source code file

Open the CMakeLists.txt of ntz at "/project/realtek_amebapro2_v0_example/GCC-RELEASE/application_ntz/".
Add the source code by append source code to app_ntz_sources ("list{ APPEND app_ntz_sources <path to source code> }").



- Adding header file

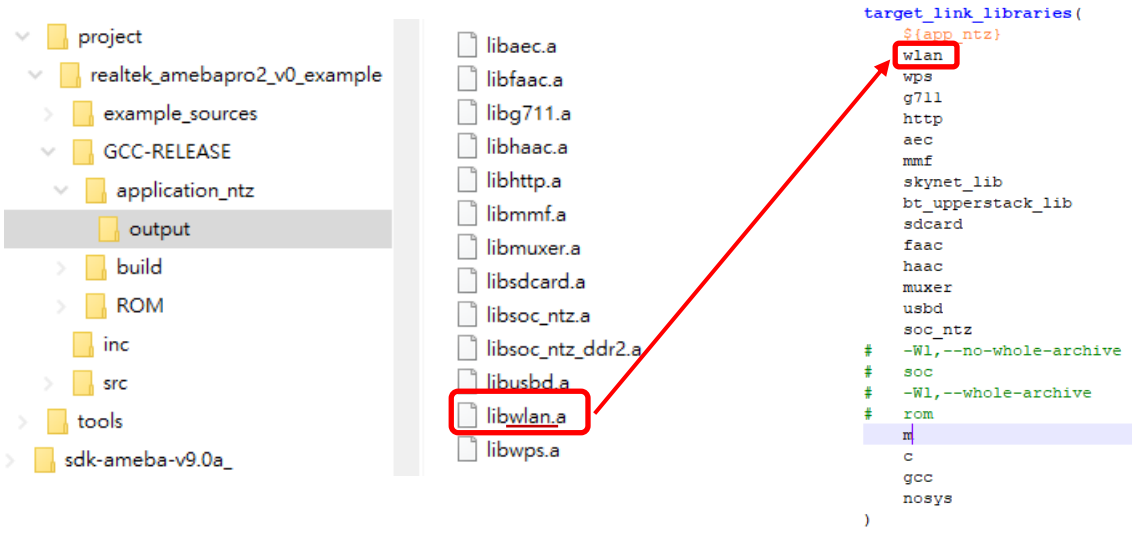
Open the includepath.cmake at "/project/realtek_amebapro2_v0_example/GCC-RELEASE /".
Add the header files by append header files location to inc_path ("list{ APPEND inc_path "<path to header folder>" }").
Also add directory to be included by include_directories (<path to header folder>).

3.1.2 Adding a library file

- Method 1:

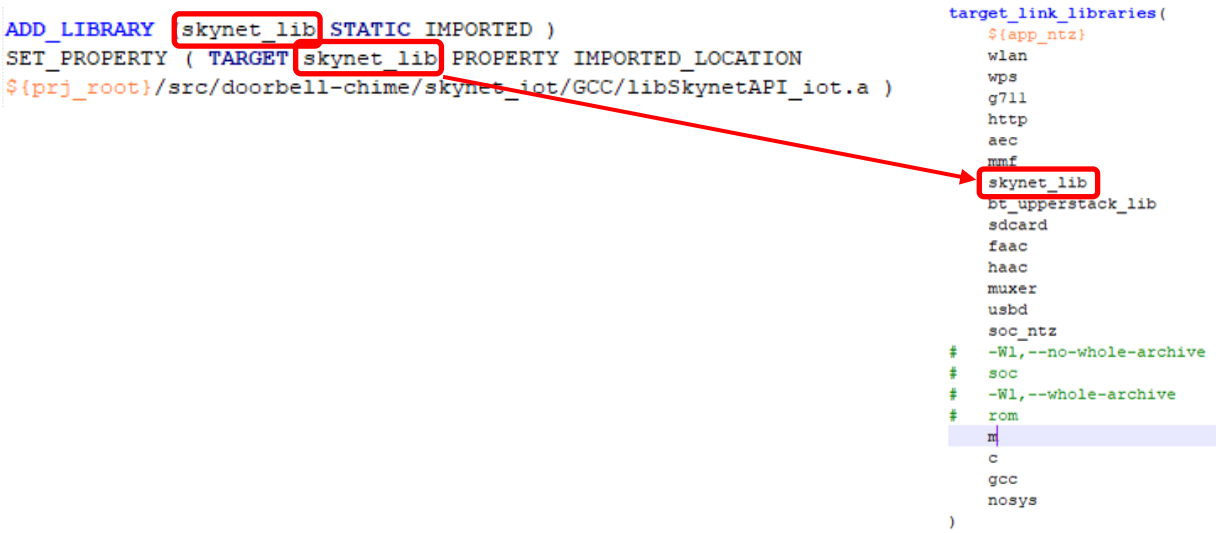
You can place the library under "/project/realtek_amebapro2_v0_example/GCC-RELEASE/application_ntz/output" and the name need to be libxxx.

Then, you can add the library xxx by writing the name xxx inside the “target_link_libraries”.



- Method 2:

  Use “ADD_LIBRARY (<lib name> STATIC IMPORTED)” first to declare the lib name and type you will be added, where <lib_name> is the label for your library and do not need to be the same as your real library.
  Then, use “SET_PROPERTY (TARGET <lib name> PROPERTY IMPORTED_LOCATION <path-to-yourlibrary>)” or “set_target_properties (<lib name> PROPERTY IMPORTED_LOCATION <path-to-yourlibrary>)” to set up the location of the library.
  Finally, you can add the library xxx by writing the name xxx inside the target_link_libraries.



3.1.3 Building a library

- Create a cmake file for the library

  Set up required cmake version (the cmake version is just use to ensure the current cmake version) and the project name. Here the output library file name will be libtest.a or libtest.so.

```
cmake_minimum_required(VERSION 3.6)

project(test)

set(test test)
```

Add the source code by append the list

```
list(
    APPEND test_sources
        ${prj_root}/src/test/test0.c
        ${prj_root}/src/test/test1.c
        ${prj_root}/src/test/test2.c
)
```

Select the library type, STATIC means that the library will be built as static-link library (*.a), while SHARED means that the library will be built as dynamic-link library (*.so).

```
add_library(
    ${test} STATIC
    ${test_sources}
)
```

Add the compile flag for the library

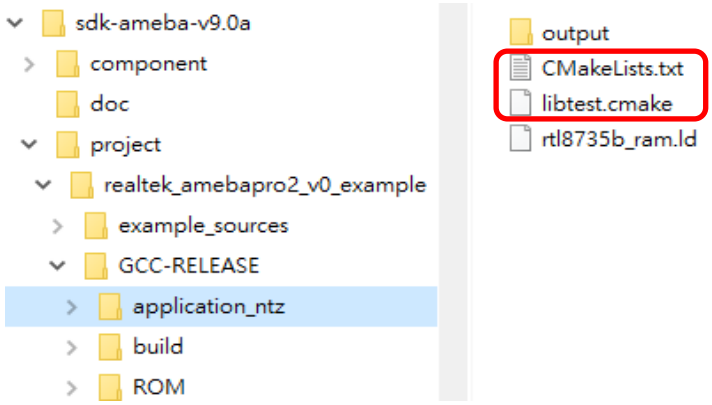
```
list(
    APPEND test_flags
    CONFIG_BUILD_ALL=1
    CONFIG_BUILD_LIB=1
    TEST_FLAGS
)

target_compile_definitions(${test} PRIVATE ${test_flags})
```

Add the header files need to be included in the library

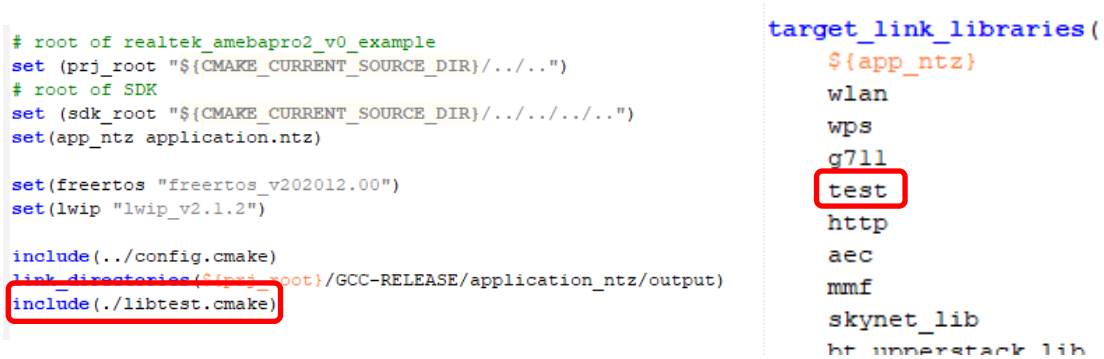
```
include(../includepath.cmake)
target_include_directories(
    ${test}
    PUBLIC

    ${inc_path}
    ${prj_root}/test/include
)
```

- Add the cmake and link the library to the project

Place the cmake file (eg.libtest.cmake) and open the CMakeLists.txt of ntz at "/project/realtek_amebapro2_v0_example/GCC-RELEASE/application_ntz/".



Add the library cmake file by "include (./<cmake_filename>.cmake)" and also in target_link_libraries.



```
# root of realtek_amebapro2_v0_example
set (prj_root "${CMAKE_CURRENT_SOURCE_DIR}/../..")
# root of SDK
set (sdk_root "${CMAKE_CURRENT_SOURCE_DIR}/../../../..")
set(app_ntz application.ntz)

set(freertos "freertos_v202012.00")
set(lwip "lwip_v2.1.2")

include(../config.cmake)
link_directories(${prj_root}/GCC-RELEASE/application_ntz/output)
include(./libtest.cmake)
```

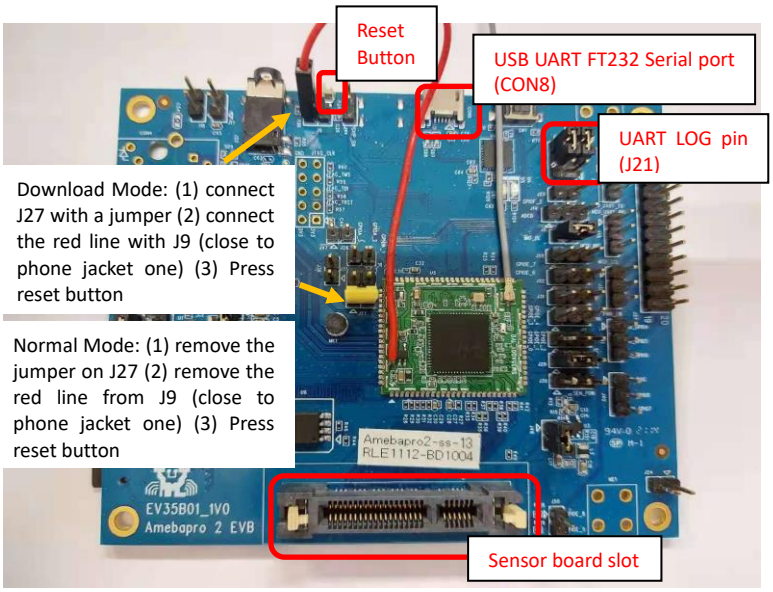
```
target_link_libraries(
    ${app_ntz}
    wlan
    wps
    g711
    test
    http
    aec
    mmf
    skynet_lib
    ht_upperstack_lib
```

After building you can get the lib file under "/project/realtek_amebapro2_v0_example/GCC-RELEASE/build/application_ntz/". You can move it to "/project/realtek_amebapro2_v0_example/GCC-RELEASE/application_ntz/output/" and remove the statement of "include (./<cmake_filename>.cmake)" in CMakeLists.txt.
Note if you need to rebuild the library, you need to remove the previous built library file.

4 Demo Board

4.1 Demo Board overview

- Hardware: 8735 EVB * 1

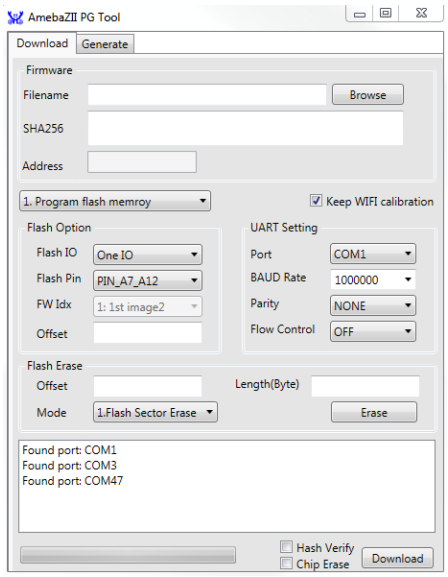


5 Image Tool

5.1 Introduction

This chapter will introduce how to use Image Tool to generate and download images. The image tool - AmebaZII_PGTool can be find in tools folder and it has two menu pages:

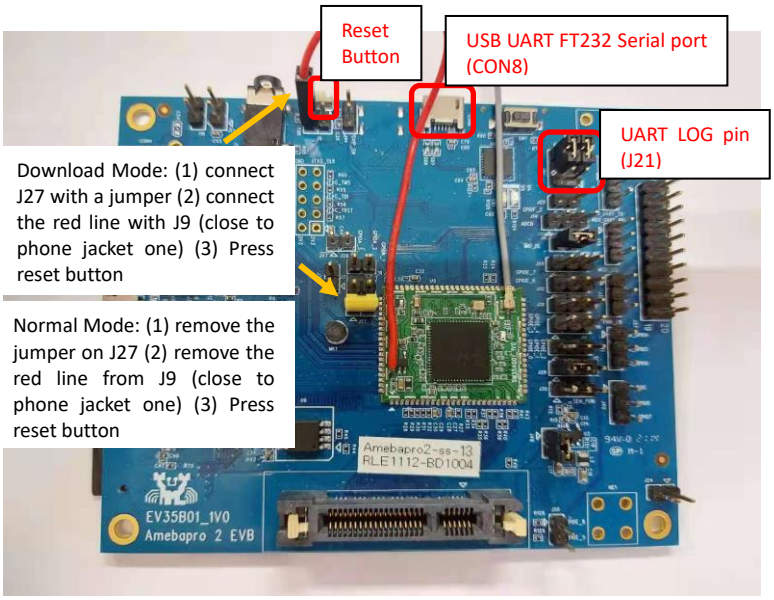
- Download: For downloading image to an amebapro2 device through UART.
- Generate: For contacting individual images to a composite image.



5.2 Download Environment Setup

5.2.1 Hardware Setup

To download image, the device must to be boot as download mode. Users need to first set up the UART with PC by connecting J21 with jumpers and CON8 with PC. Then, connect J27 with a jumper, connect the red line with J9 (close to phone jacket one) and press reset button to enter download mode.



5.2.2 Software Setup

- PC environment requirements: Windows 7 above with FT232 driver.
- AmebaZII_PGTool_vx.x.x.exe

5.3 Image Download

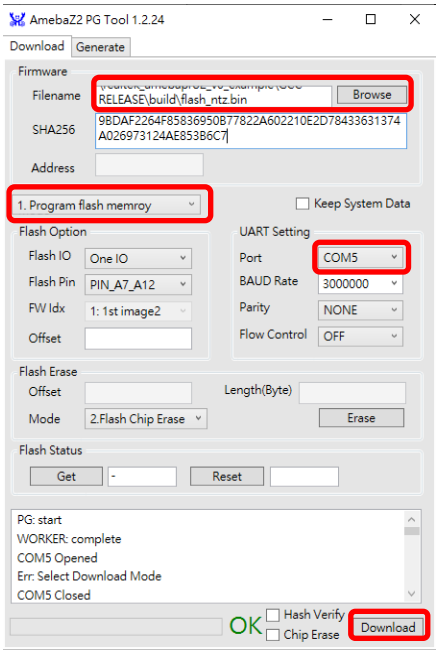
User can download the image to demo board by following steps:

- Trigger AmebaPro2 into **download mode**

  Note: after checking the device into download mode, remember to disconnect the log UART console before using Image Tool to download)

```
== Rtl8735b IoT Platform ==
Chip VID: 0, Ver: 0
ROM Version: v1.0
Test Mode: boot_cfg1=0x0

[test mode PG]
test_mode_img_download
Download Image over UART1[tx=5,rx=4] baud=115200
```

- Open the PG tool

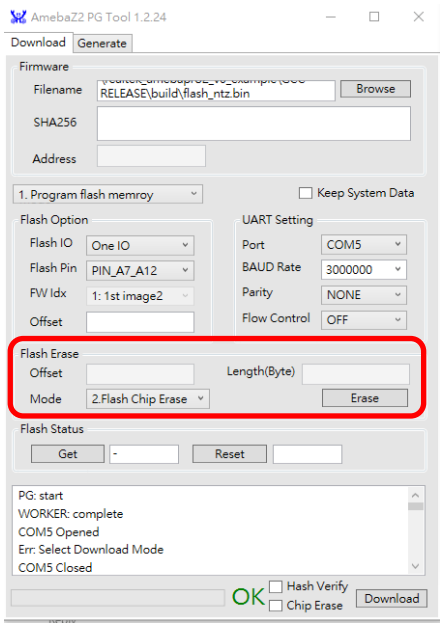


- "**Browse**" to choose the image will be downloaded (flash_ntz.bin)
- Choose "**1. Program flash memory**" and Select the correct CON port
- After the above steps are done, press the Download button and the image will start to be downloaded to Amebapro2 device. (Flash Pin will be set automatically, so we do not need to set.)

5.4 Erase Flash of device

For the image tool, it also provide user to erase the flash of the device and it provides two mode:

- Flash Sector Erase: erase the flash from the Offset with a Length of Byte that the user set.
- Flash Chip Erase: erase the whole flash



Note

Note: After you press the erase button, it needs time for erasing the flash. Please wait until the PG tool shows the “erase successfully” message.

6 How to use example source code

In this chapter, it will describe how to use the example source code for AmebaPro2.

6.1 Application example source

The AmebaPro2 application's example source codes can be found under folder sdk/component/example. The examples typically provide three related files, source code (*.c), header (*.c) and readme. The readme file demonstrates how to compile and important parameter. User can easily build up the example through it. Some example may need user to add the header and source code files first, it can refer to 3.1.1Adding a source code file or header file and add files to the project.

After adding the files, user should use inc/platform_opts.h to switch on the example. For instance, if the user want to compile fatfs example, set the parameter CONFIG_EXAMPLE_FATFS to 1 (#define CONFIG_EXAMPLE_FATFS 1) and the project will execute the example.

```
/* 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
#define FATFS_DISK_SD   1
#define FATFS_DISK_FLASH    0
#endif
#endif
```

6.1.1 MMF example source

In sdk/component/example/media_framework, it provide audio-only MMF examples. The example are based on the Multimedia Framework Architecture and the detail can refer to 7 Multimedia Framework Architecture.

6.2 Peripheral example source

The peripheral example source is for helping user utilize peripheral function. The example source is located at the folder SDK/project/realtek_amebapro_v0_example/example_sources and basically provide mian.c and readme file. The mian.c file contains the usage of peripheral function and user should replace it with the original main.c (in SDK/project/realtek_amebapro_v0_example/src). On the other hand, like application example source, the method to compile example and adjust the important parameters is described in the readme file. After the setting, user can rebuild the project with peripheral example.

6.3 Wi-Fi example source

6.3.1 Use AT command to connect WLAN

For user's test and development, we provide AT command in AmebaPro2. 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.

7 Multimedia Framework Architecture

The Multimedia Framework Architecture version 2(MMFv2) is responsible for handling the connection and management of different media resources on AmebaPro2.

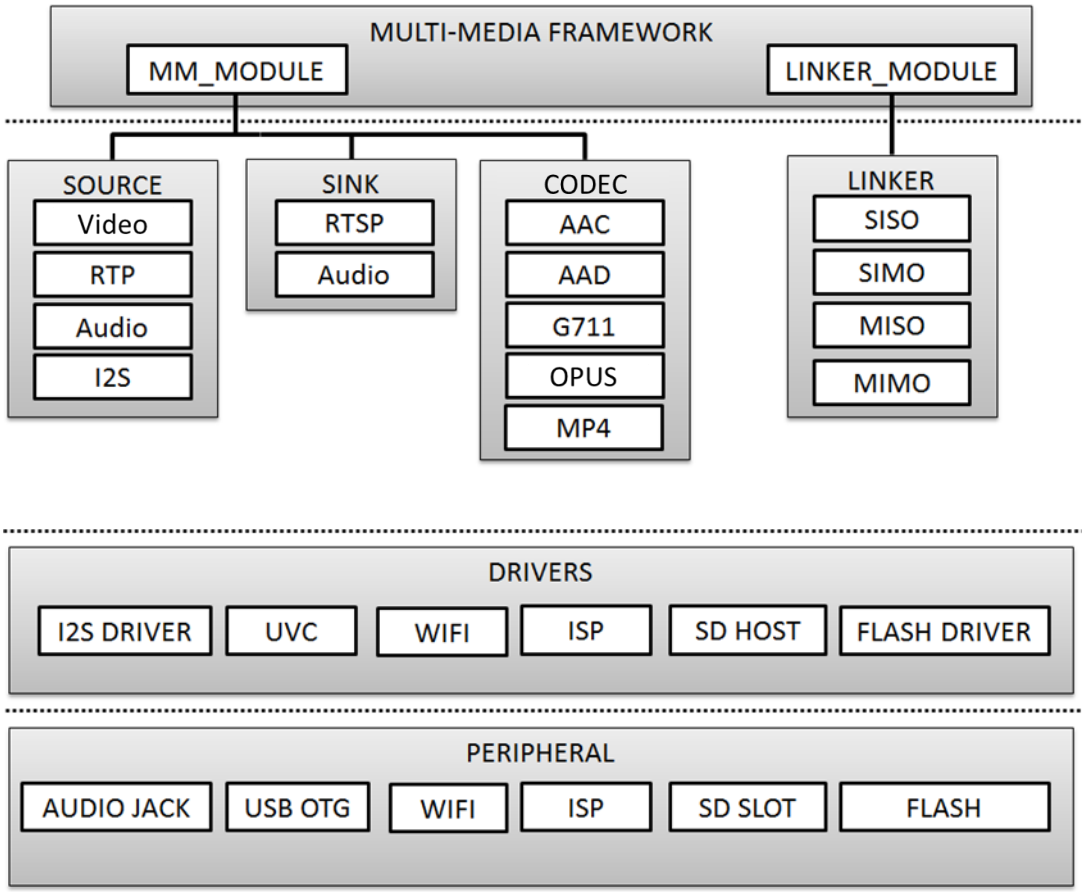
7.1 Architecture

The structure of MMFv2 is as shown in the following chart and there are two important entities in the MMFv2, **MM_MODULE** and **LINKER_MODULE**:

MM_MODULE includes the media source, sink and the codec modules.

- Source module: produce resource, it can be the file input, microphone, camera, or storage. Note that the video modules uses VOE to contain the process of sensor catching, ISP and video encoding algorithms (jpeg, H264, HEVC (H265)…).
- Codec module: mainly provide the audio codec, AAC, G711 or opus for customers to do audio encode or decode before sending streaming to sink module. In the mp4 module, it will automatically send the result into storage, SD card or ram disk.
- Sink module: consume resource from the source modules or after encoded/decoded by codec modules, like RTSP or other steaming.

LINKER_MODULE connect different type of module and deal with inter module communication, included siso, simo, miso and mimo.

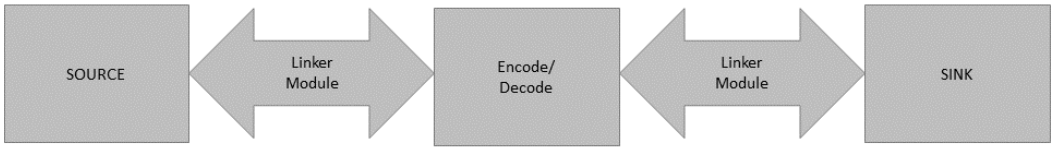


In order to use the MMFv2, here are some aspects must to be followed.

- Define valid source

- Define valid sink
- Define valid codec (encode/decode) if needed.
- Define valid linker modules to link the above media modules.

The following picture shows the main usage flow to initialize different **MM_MODULE**, and connect different **MM_MODULE** through **LINKER_MODULE**.



7.1.1 MM_Module Prototype

MMFv2 allows users to define customized source, sink and encoder/decoder modules depending on the application. Although implementation details may be different, basic rules of the MMF structure are similar. The MMFv2 requires users to predefine both source and sink modules through implement create, destroy, control, handle, new_item, del_item and rsz_item function callbacks. The structure mmf_module_t provides the interface for communication between mmf modules. In order to maintain the flexibility and convenience between modules, modules only retain the interface of each type to provide module to access. Function's constant of each module is defined by module itself.

```
typedef struct mm_module_s {
        void*       (*create)(void *);
        void*       (*destroy)(void *);
        int         (*control)(void *, int, int);
        int         (*handle)(void *, void *, void *);
        void*       (*new_item)(void *);
        void*       (*del_item)(void *, void *);
        void*       (*rsz_item)(void *, void *, int);
        void*       (*vrelease_item)(void *, void *, int);

        uint32_t    output_type;
        uint32_t    module_type;
        char *      name;
} mm_module_t;
```

7.1.1.1 Function description

- create

  Pointer to the function that loads and initializes the module that you wish to add. For example, for Audio source, it points to the function in which the Audio driver is initialized and the corresponding context is returned.

- destroy

  Pointer to the function that de-initializes module instance and releases resource. For example, for Audio source, it points to function in which Audio driver is initialized and the corresponding context is released.

- control

Pointer to function that sends the control command to the MMF module layer (see **mm_module_ctrl**) or a specific module. For example, for Audio source, it points to function that controls Audio parameters (“sample rate”, “word length”, “mic gain”, etc.) and MMFv2 service task on or off.

- handle

  Pointer to the function that manipulates media data (how to produce data in source or how to consume data in sink). Data is transferred from source to sink and vice versa by means of OS message queue. Please note that MMF service task reacts differently based on message exchange buffer status.

- new_item

  Pointer to the function that create queue item that will be send to input and output queue, only will be used when setting MM_CMD_INIT_QUEUE_ITEMS to MMQI_FLAG_STATIC.

- del_item

  Pointer to the function that destroy queue item, only will be used when setting MM_CMD_INIT_QUEUE_ITEMS to MMQI_FLAG_STATIC.

- rsz_item

  Pointer to the function decrease memory pool size, only will be used when video (H264, HEVC (H265)…) and AAC module is created.

- output_type and module_type

  Output_type indicates output mode. There are MM_TYPE_NONE, MM_TYPE_VSRC, MM_TYPE_ASRC, MM_TYPE_VDSP, MM_TYPE_ADSP, MM_TYPE_VSINK, MM_TYPE_ASINK, and MM_TYPE_AVSINK can be used, corresponding to different module usage scenarios, let application know which mode the output is. module_type represents the identity of the module, and there are three option can be used MM_MASK_SRC, MM_MASK_DSP and MM_MASK_SINK.

- name

  Pointer to the module name.

7.1.1.2 mm_module_ctrl

Here lists some command defined MMF module layer. Call by mm_module_ctrl (mm_context_t *ctx, int cmd, int arg) to use them.

- MM_CMD_INIT_QUEUE_ITEMS: initialize static queue item.
- MM_CMD_SET_QUEUE_LEN: Set one queue's length.
- MM_CMD_SET_QUEUE_NUM: Set number of queue, not more than 3.
- MM_CMD_SELECT_QUEUE: select queue from multi queues.
- MM_CMD_CLEAR_QUEUE_ITEMS: clear queue item.

7.1.2 Context

MMFv2 context supply message transfer between different modules. It contains mm_module_t, and queue that used to pass data. There are 6 types of status that mm_context support (MM_STAT_INIT, MM_STAT_READY, MM_STAT_ERROR, MM_STAT_ERR_MALLOC, MM_STAT_ERR_QUEUE, MM_STAT_ERR_NEWITEM), these status are responsible for maintaining the module state to ensure the program runs smoothly.

```
typedef struct mm_contex_s {
        union {
```

```
        struct {
                xQueueHandle    output_ready;
                xQueueHandle    output_recycle;
                int32_t         item_num;
        };
        mm_conveyor_t       port[4];
};

mm_module_t*    module;

void*           priv;                   // private data structure for created
instance

// module state
uint32_t        state;
int32_t         queue_num;      // number of queue
int32_t         curr_queue;
} mm_context_t;
```

The mm_context is responsible for maintaining each module entity. MMFv2 default support these modules (video, AAC_encoder, AAC_decoder, audio, g711, opus, mp4, rtp, rtsp). Each module is independent and corresponding to the individual input/ output queue, state and in the mm_context of the module to update parameters and delivery entities.

7.1.3 Module Inter Connection

This section introduces mm_siso_t, mm_simo_t, mm_miso_t, mm_mimo_t and its corresponding create, delete, ctrl, start, stop, pause, resume function, which is responsible for connection and control between modules in mmfv2.

7.1.3.1 SISO module (Single Input Single Output)

The SISO module is a unidirectional interface between modules. Input and output are independent. The status of the SISO module is responsible for determining the correct process. The stack_size is used to determine the size of the handler, while xTaskHandle task, task_priority and taskname are reserved to control the use of the task, task priority and task name.

```
typedef struct mm_siso_s {
        mm_context_t *input;
        mm_context_t *output;
        int         input_port_idx;
// default is 0, can be set to 1 or 2 or 3 if source module support 2 or more output
queue

        uint32_t        status;
        uint32_t        stack_size;
        uint32_t  task_priority;
        char       taskname[16];
        xTaskHandle task;
} mm_siso_t;
```

There are some functions in the SISO module responsible for the module inter-connection. By these functions, it will be simple to update the status of the task and are handed over to the task handler for the main

processing:

- siso_create

  Pointer to the function that siso_create declares the space of mm_siso_t and returns mm_siso_t entity after initialization.

- siso_delete

  Pointer to the function stops SISO execution and free space of mm_siso_t entity.

- siso_ctrl

  Pointer to the function sends the control command to siso module.
  MMIC_CMD_ADD_INPUT link the input module to the input of the siso module.
  MMIC_CMD_ADD_OUTPUT link the output module to the output of the siso module.
  MMIC_CMD_SET_TASKPRIORITY set the task priority for the linker task. If setting as 0, it will be configured to tskIDLE_PRIORITY + 1 automatically.
  MMIC_CMD_SET_TASKNANE set the task names for the linker task.
  MMIC_CMD_SET_STACKSIZE add size to the stack_size of siso.
  Note: for consistency, the setting task size will be divided by 4. Make sure setting an enough and valid stack_size for the task.

- siso_start

  Pointer to the function checks whether there is anything in the input and output module before siso start. If the answer is yes, siso task will create a task handler to send data from input module to the output module.

- siso_stop

  Pointer to the function updates status to MMIC_STAT_SET_EXIT and wait for task handler to switch status to MMIC_STAT_EXIT.

- siso_pause

  Pointer to the function updates status to MMIC_STAT_SET_PAUSE and wait for task handler to switch status to MMIC_STAT_PAUSE.

- siso_resume

  Pointer to the function updates status to MMIC_STAT_SET_RUN and wait for the task handler to switch status to MMIC_STAT_RUN.

7.1.3.2 SIMO module (Single Input Multiple Output)

The SIMO module is a unidirectional interface between modules. Input and output are independent, and output_cnt represents the number of simultaneous output modules. The array – status[4] maintains the state of the SIMO module to confer the process is correct in the middle of the transfer, stack_size is used to determine the size of the handler task for intermediate transfers. Similarly, it also provides xTaskHandle task, task_priority, taskname for xTaskCreate. Note that each output will be served by one unique task and pause mask will control which output will be blocked.

```
typedef struct mm_simo_s {
        mm_context_t *input;
        int                 output_cnt;
        mm_context_t *output[4];
        // internal queue to handle reference count and usage log
        mm_simo_queue_t queue;
```

```
        uint32_t      pause_mask;
    uint32_t          status[4];;
    uint32_t          stack_size;
    uint32_t   task_priority;
    char         taskname[4][16];
    xTaskHandle task[4];
} mm_simo_t;
```

There are some functions in the SIMO module responsible for the module inter-connection. By these functions, it will be simple to update the status of the task and are handed over to the task handler for the main processing:

- simo_create

  Pointer to the function that simo_create declares the space of mm_simo_t entity and returns mm_siso_t after initialization, and simo_create crate a queue head and a queue lock to protect the results of multiple outputs.

- simo_delete

  Pointer to the function calls simo_stop() to stop SIMO execution and free space.

- simo_ctrl

  Pointer to the function sends the control command to simo module.
  MMIC_CMD_ADD_INPUT link the input module to the input of the simo module.
  MMIC_CMD_ADD_OUTPUT0, MMIC_CMD_ADD_OUTPUT1, MMIC_CMD_ADD_OUTPUT2, MMIC_CMD_ADD_OUTPUT3 link output module to the corresponding output and increase the output_cnt to record number of output modules.
  MMIC_CMD_SET_TASKPRIORITY set the task priority for the linker task. If setting as 0, it will be configured to tskIDLE_PRIORITY + 1 automatically.
  MMIC_CMD_SET_TASKNANEx set the task names for the linker task corresponding to MMIC_CMD_ADD_OUTPUTx (x = 0~3).
  MMIC_CMD_SET_STACKSIZE add size to simo stack_size.
  Note: for consistency, the setting task size will be divided by 4 and it means each task will only have task_size/4 for task stack size. Make sure setting an enough and valid stack_size for the task.

- simo_start

  Pointer to the function that simo_start will create corresponding number of task handlers based on simo -> output_cnt, and each task handler will be used to send the received data.

- simo_stop

  Pointer to the function that simo_stop sets each simo status to MMIC_STAT_SET_EXIT,and waits for the task handler to switch each status to MMIC_STAT_EXIT.

- simo_pause

  Pointer to the function that simo_pause will set each simo -> status to MMIC_STAT_SET_PAUSE according to pause_mask, and wait for the task handler to switch each status to MMIC_STAT_PAUSE.

- simo_resume

  Pointer to the function that simo_resume will set each simo -> status to MMIC_STAT_SET_RUN, and wait for the task handler to switch each status to MMIC_STAT_RUN.

7.1.3.3 MISO module (Multiple Input Single Output)

The MISO module is a unidirectional interface between modules. Input and output are independent, and input_cnt represents the number of simultaneous input modules. The status maintains the state of the MISO module to confer the process is correct in the middle of the transfer, stack_size is used to determine the size of the handler task for intermediate transfers, and finally the xTaskHandle task, task_priority and taskname are reserved for xTaskCreate to control the use of the task. The pause_mask can be controlled to block the inputs or the single output.

```
typedef struct mm_miso_s {
        int                 input_cnt;
        mm_context_t *input[4];  // max 4 input
        int                 input_port_idx[4];

        mm_context_t *output;

        uint32_t        pause_mask;
        uint32_t        status;
        uint32_t        stack_size;
        uint32_t        task_priority;
        char        taskname[16];
        xTaskHandle   task;
} mm_miso_t;
```

There are some functions in the MISO module responsible for the module inter-connection. By these functions, it will be simple to update the status of the task and are handed over to the task handler for the main processing:

- miso_create

  Pointer to the function that space of mm_miso_t is declared in miso_create and initialized to return mm_miso_t entity.

- miso_delete

  Pointer to the function that calls miso_stop() to stop MISO and free space.

- miso_ctrl

  Pointer to the function sends the control command to miso module.
  MMIC_CMD_ADD_INPUT0, MMIC_CMD_ADD_INPUT1, MMIC_CMD_ADD_INPUT2, MMIC_CMD_ADD_INPUT3 couple input modules to the corresponding miso input and increase the value of input_cnt for number of input module.
  MMIC_CMD_ADD_OUTPUT links the output module to the output of the miso module.
  MMIC_CMD_SET_TASKPRIORITY set the task priority for the linker task. If setting as 0, it will be configured to tskIDLE_PRIORITY + 1 automatically.
  MMIC_CMD_SET_TASKNANE set the task names for the linker task.
  MMIC_CMD_SET_STACKSIZE add size to miso stack_size.
  Note: for consistency, the setting task size will be divided by 4. Make sure setting an enough and valid stack_size for the task.

- miso_start

  Pointer to the function checks whether there is anything in the input and output module before starting. If the answer is yes, a task handler will be created, and the data of the input module will be sent to the output module.

- miso_stop

  Pointer to the function sets the miso status to MMIC_STAT_SET_EXIT and wait for the task handler to switch the status to MMIC_STAT_EXIT.

- miso_pause

  Pointer to the function that miso_pause will set miso -> status to MMIC_STAT_SET_PAUSE according to pause_mask, waiting for the task handler to switch status to MMIC_STAT_PAUSE.

- miso _resume

  Pointer to the function that miso_resume will set miso -> status to MMIC_STAT_SET_RUN, waiting for the task handler to switch each status to MMIC_STAT_RUN.

7.1.3.4 MIMO module (Multiple Input Multiple Output)

The MIMO module is a unidirectional interface between modules, Input[4] and output[4] represent input and output modules respectively, and input_cnt represents the number of simultaneous input modules. Input and output support up to 4 outputs at the same time, MIMO module also needs mm_mimo_queue_t queue[4] to maintain the synchronization problem of each input queue. Each mm_mimo_queue_t has a lock and head to record the beginning of each queue and whether a program is already in use. The array, status[4], maintains the state of the MIMO module to determine the correct process in the middle of the transfer, stack_size is used to determine the size of the handler task for the intermediate transfer, and the xTaskHandle task of xTaskCreate is reserved to control the use of the task. The array, pause_mask[4], is use to control the input or output streaming for each task.

```
typedef struct mm_mimo_s {
        int             input_cnt;

        // depend on intput count
        mm_context_t*         input[4];
        mm_mimo_queue_t    queue[4];

        int                   output_cnt;

        // depend on output count
        uint32_t              pause_mask[4];
        mm_context_t*  output[4];       // output module context
        uint32_t          output_dep[4];  // output depend on which input, bit mask
        uint32_t              input_mask[4]; // convert from output_dep, input
referenced by which output, bit mask
        uint32_t          status[4];
        uint32_t          stack_size;
        uint32_t          task_priority;
        char              taskname[4][16];
        xTaskHandle       task[4];
} mm_mimo_t;
```

There are some functions in the MIMO module responsible for the module inter-connection. By these functions, it will be simple to update the status of the task and are handed over to the task handler for the main processing:

- mimo_create

Pointer to the function mimo_create declares the space of mm_mimo_t entity and returns mm_mimo_t after initialization.

- mimo_delete

  Pointer to the function calls mimo_stop() to stop the mimo module and free space.

- mimo_ctrl

  Pointer to the function sends the control command to miso module.
  MMIC_CMD_ADD_INPUT0, MMIC_CMD_ADD_INPUT1, MMIC_CMD_ADD_INPUT2, and MMIC_CMD_ADD_INPUT3 link input module to the input corresponding to the mimo module and increase the value of input_cnt to record the number of input modules.
  MMIC_CMD_ADD_OUTPUT0, MMIC_CMD_ADD_OUTPUT1, MMIC_CMD_ADD_OUTPUT2, and MMIC_CMD_ADD_OUTPUT3 couple the output module to the output of the mimo module and increase the value of output_cnt to record the number of output modules. The inputs corresponding to outputs modules can be set by arg2 of mimo_ctrl using the union of MMIC_CMD_ADD_INPUTx.
  MMIC_CMD_SET_TASKPRIORITY set the task priority for the linker task. If setting as 0, it will be configured to tskIDLE_PRIORITY + 1 automatically.
  MMIC_CMD_SET_TASKNANE set the task names for the linker task.
  Note that, for consistency, the setting task size will be divided by 4 and it means each task will only have task_size/4 for task stack size. Make sure setting an enough and valid stack_size for the task.

- mimo_start

  Pointer to the function that mimo_start will generate corresponding task handler according to output_cnt to transfer the received data.

- mimo_stop

  Pointer to the function that mimo_stop will set the mimo status to MMIC_STAT_SET_EXIT according to output_cnt, and waiting for the task handler switch the status to MMIC_STAT_EXIT.

- mimo_pause

  Pointer to the function that miso_pause will set each mimo -> status to MMIC_STAT_SET_PAUSE according to pause_mask, and waiting for the task handler to switch status to MMIC_STAT_PAUSE.

- mimo_resume

  Pointer to the function that mimo_resume will set mimo -> status in the task of MMIC_STAT_PAUSE for each status to MMIC_STAT_SET_RUN, and waiting for the task handler to switch each status to MMIC_STAT_RUN.

7.2 Using the MMF example

Describe how to use the sample program to construct the data stream required by the terminal application. In this section, there will be an introduction to correctly select the mmfv2 sample program and adjust the parameters.

7.2.1 Selecting and setting up sample program

For audio only samples, they are in function example_mmf2_audio_only while video joined samples are listed in example_mmf2_video_surport. Pick the example want to open before using it, remove the comment, and recompile. Opening more than two examples at the same time will result in unpredictable program execution results.

7.2.1.1 Requisites and Setup

Pre-requisites:

- AmebaPro2 board
- Camera sensor board
- Micro USB cable
- WIFI (for transferring rtsp stream)
- MicroSD card (for saving the mp4 data)

Hardware setup:

- Connect the camera sensor board to the AmebaPro2's camera sensor board slot (CON1).
- Connect the PC with the AmebaPro2 CON8 port by the Micro USB cable.
- Insert the MicroSD card to the AmebaPro2's SD card slot.

Software setup:

- In project\realtek_amebapro2_v0_example\inc\platform_opts.h

  Audio only example: switch CONFIG_EXAMPLE_MEDIA_FRAMEWORK to 1

  Video joined example: switch both CONFIG_EXAMPLE_MEDIA_FRAMEWORK and CONFIG_EXAMPLE_MEDIA_VIDEO to 1

  If example with SD card, also check FATFS_DISK_SD

```
/* For MMFv2 example */
#define CONFIG_EXAMPLE_MEDIA_FRAMEWORK  1
#if (defined(CONFIG_EXAMPLE_MEDIA_FRAMEWORK) &&
CONFIG_EXAMPLE_MEDIA_FRAMEWORK)
#define CONFIG_EXAMPLE_MEDIA_UVCD   0
#define CONFIG_EXAMPLE_MEDIA_VIDEO  1
#define CONFIG_EXAMPLE_MEDIA_CLOUD  0
#define CONFIG_EXAMPLE_MEDIA_NN     0
#define FATFS_DISK_SD         1
#endif

#define SENSOR_GC2053 1
#define SENSOR_PS5258 2

//SENSOR_GC2053, SENSOR_PS5258
#define USE_SENSOR SENSOR_GC2053
```

- The sample program is located at:

  Audio only: \component\example\media_framework\ example_media_framework.c

  Video joined: \project\realtek_amebapro2_v0_example\src\mmfv2_video_example\ video_example_media_framework.c

For example: open mmf2_video_example_joint_test_rtsp_mp4_init

```
// Joint test RTSP MP4
// H264 -> RTSP (V1)
// H264 -> MP4 (V2)
// AUDIO -> AAC  -> RTSP and mp4
// RTP   -> AAD  -> AUDIO
```

```
//mmf2_video_example_joint_test_rtsp_mp4_init();
```

Uncomment the example want to execute

```
// Joint test RTSP MP4
// H264 -> RTSP (V1)
// H264 -> MP4 (V2)
// AUDIO -> AAC  -> RTSP and mp4
// RTP   -> AAD  -> AUDIO
mmf2_video_example_joint_test_rtsp_mp4_init();
```

Note: uncomment two media examples in the same time may cause unexpected result.

- Compile and execute firmware. The compilation and execution can refer to the previous chapter.

7.2.1.2 Currently supported example

- **Audio only examples**:

Example	Description	Result
mmf2_example_a_init	audio -> AAC -> RTSP(A)	AmebaPro2's AAC sound stream over the network. The sound received by AmebaPro2 is encoded by AAC and then streamed through the network (rtsp).
mmf2_example_audioloop_init	PCM audio -> PCM audio , audio loopback	The sound received by AmebaPro2 can be broadcast from the 3.5 audio channel of AmebaPro2, and the PCM transmission is directly used in the procedure.
mmf2_example_g711loop_init	audio -> G711E -> G711D -> audio	The sound received by AmebaPro2 can be broadcast from the 3.5 audio channel of AmebaPro2. PCM is encoded by G711 and transmit, then decoded by G711 and playback.
mmf2_example_aacloop_init	audio -> AAC -> AAD -> audio	The sound received by AmebaPro2 can be broadcast from the 3.5 audio channel of AmebaPro2. PCM is encoded by AAC and transmit, then decoded by AAD and playback.
mmf2_example_rtp_aad_init	RTP -> AAD -> audio	Stream AAC sound over the network to AmebaPro2 for playback. Streaming audio is decoded by AAD and played through 3.5 audio jack.
mmf2_example_2way_audio_init	audio -> AAC -> RTSP RTP -> AAD -> audio	Stream AAC sound to AmebaPro2's audio jack via the

		network and transmit the sound received by AmebaPro2 over the network simultaneously.
mmf2_example_pcmu_array_rtsp_init	ARRAY (PCMU) -> RTSP (A)	Transmitting PCMU sound arrays within AmebaPro2 over the network.
mmf2_example_aac_array_rtsp_init	ARRAY (AAC) -> RTSP (A)	Transfer AAC sound arrays in AmebaPro2 over the network.
mmf2_example_opusloop_init	audio -> OPUSC -> OPUSD -> audio	The sound received by AmebaPro2 can be broadcast from the 3.5 audio channel of AmebaPro2. PCM is encoded by OPUS and transmit, then decoded by OPUS and playback.
mmf2_example_a_opus_init	Audio -> OPUSC -> RTSP(A)	AmebaPro2's OPUS sound stream over the network. The sound received by AmebaPro2 is encoded by OPUSC and then streamed through the network (rtsp).
mmf2_example_rtp_opusd_init	RTP -> OPUSD -> audio	Stream OPUSC sound over the network to AmebaPro2 for playback. Streaming audio is decoded by OPUSD and played through 3.5 audio jack.
mmf2_example_2way_audio_opus_init	audio -> OPUSC -> RTSP RTP -> OPUSD -> audio	Stream OPUS sound to AmebaPro2's audio jack via the network and transmit the sound received by AmebaPro2 over the network simultaneously.

- **Video only examples**:

Example	Description	Result
mmf2_video_example_v1_init	CH1 Video -> H264/HEVC -> RTSP	Transfer AmebaPro2's H264/HEVC video stream over the network. Video default format: 720P 30FPS.
mmf2_video_example_v2_init	CH2 Video -> H264/HEVC -> RTSP	Transfer AmebaPro2's H264/HEVC video stream over the network. Video default format: 1080P 30FPS.
mmf2_video_example_v3_init	CH3 Video -> JPEG -> RTSP	Transfer AmebaPro2's JPEG video stream over the network. Video default format: 1080P 30FPS.

mmf2_video_example_v1_shapshot_init	CH1 Video -> H264/HEVC -> RTSP + SNAPSHOT	Transfer AmebaPro2's H264/HEVC video stream over the network and snapshot (JPEG) while streaming.
mmf2_video_example_simo_init	1 Video (H264/HEVC) -> 2 RTSP (V1, V2)	Transmitting two H264/HEVC video streams from AmebaPro2 over the network, the source of the video is the same video stream. Video default format: 1080P 30FPS.
mmf2_video_example_array_rtsp_init	ARRAY (H264/HEVC) -> RTSP (V)	Transfer H264/HEVC stream array in AmebaPro2 over the network. Video default format: 25FPS.
mmf2_video_example_v1_param_change_init	CH1 Video -> H264/HEVC -> RTSP (parameter change)	Transfer AmebaPro2's H264/HEVC video over the network and support dynamic adjustment of video parameters. The parameters of dynamic adjustment are Resolution, Rate Control Mode, Bit Rate in order.
mmf2_video_example_h264_array_mp4_init	ARRAY (H264/HEVC) -> MP4 (SD card)	AmebaPro2 will record H264/HEVC stream array to the SD card for 30 second. Video default format: 25FPS.

- **Video + Audio examples**:

Example	Description	Result
mmf2_video_example_av_init	1 Video (H264/HEVC) and 1 Audio -> AAC -> RTSP	Transfer AmebaPro2's H264/HEVC video and AAC sound stream over the network. Video default format: 1080P 30FPS.
mmf2_video_example_av2_init	2 Video (H264/HEVC) and 1 Audio -> AAC -> 2 RTSP (V1+A, V2+A)	Transmitting two H264/HEVC videos and AAC audio streams from AmebaPro2 over the network. The source of the videos is different ISP channel. The videos formats are set to 1080P 30FPS (V1) and 720P 30FPS (V2) respectively.
mmf2_video_example_av21_init	1 Video (H264/HEVC) and 1 Audio -> 2 RTSP (V+A)	Transfer two copies of AmebaPro2's H264/HEVC video (1080P 30FPS) and AAC sound stream through the network, the video source is the same ISP channel.

Note

mmf2_video_example_av_mp4_init	1 Video (H264/HEVC) and 1 Audio -> MP4 (SD card)	AmebaPro2 will record three videos (1080P 30FPS) to the SD card for 30 seconds each The default storage name is : AmebaPro2_recording_0.mp4 AmebaPro2_recording_1.mp4 AmebaPro2_recording_2.mp4
mmf2_video_example_av_rtsp_mp4_init	Video (H264/HEVC) -> RTSP and mp4 AUDIO -> AAC -> RTSP and MP4	(1) Transfer AmebaPro2's H264/HEVC video and AAC sound stream over the network. Video default format: 1080P 30FPS. (2) AmebaPro2 will record three videos (1080P 30FPS+AAC) to the SD card for 30 seconds each. The default storage name is : AmebaPro2_recording_0.mp4 AmebaPro2_recording_1.mp4 AmebaPro2_recording_2.mp4 (3) Streaming AAC sounds to AmebaPro2 via the network. Note: (1) video source of (2) is from the same ISP channel.
mmf2_video_example_joint_test_init	Video (H264/HEVC) -> RTSP (V1+A) Video (H264/HEVC) -> RTSP (V2+A) AUDIO -> AAC -> RTSP RTP -> AAD -> AUDIO	(1) Transmitting two H264/HEVC video streams from AmebaPro2 over the network, the source of the video is the different video stream. Video default format: 1080P 30FPS (V1) and 720P 30FPS (V2). (2) Streaming two copies of AAC sounds to AmebaPro2 via the network.
mmf2_video_example_joint_test_rtsp_mp4_init	Video (H264/HEVC) -> MP4 (V1+A) Video (H264/HEVC) -> RTSP (V2+A) AUDIO -> AAC -> RTSP and MP4 RTP -> AAD -> AUDIO	(1) Transfer AmebaPro2's H264/HEVC video and AAC sound stream over the network. Video default format: 1080P 30FPS. (2) AmebaPro2 will record three videos (720P 30FPS+AAC) to the SD card for 30 seconds each. The default storage name is : AmebaPro2_recording_0.mp4 AmebaPro2_recording_1.mp4 AmebaPro2_recording_2.mp4 (3) Streaming AAC sounds to AmebaPro2 via the network.

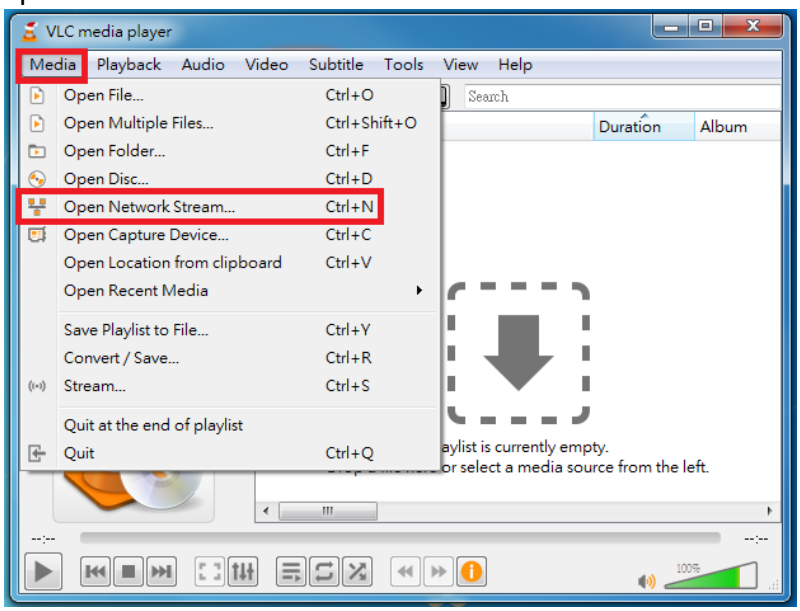
		(4) RTP send the audio stream from network to AmebaPro2 and the stream is decoded by AAD and played through 3.5 audio jack. Note: (1) video source of (2) is from different ISP channels.
mmf2_video_example_2way_audio_pcmu_doorbell_init	Video (H264/HEVC) -> RTSP (V1) AUDIO -> G711E -> RTSP RTP -> G711D -> AUDIO ARRAY (PCMU) -> G711D -> AUDIO (doorbell)	(1) Transmitting AmebaPro2's H264/HEVC stream and PCMU sound stream over the network. Video default format: 1080P 30FPS. (2) PCMU sound can be streamed to AmebaPro2 via the Internet and playback. (3) Play PCMU sound array in AmebaPro2 (default is the doorbell).
mmf2_video_example_2way_audio_pcmu_init	Video (H264/HEVC) -> RTSP (V1) AUDIO -> G711E -> RTSP RTP -> G711D -> AUDIO	(1) Transmitting AmebaPro2's H264/HEVC stream and PCMU sound stream over the network. Video default format: 1080P 30FPS. (2) PCMU sound can be streamed to AmebaPro2 via the Internet and playback.

7.2.2 VLC media player settings

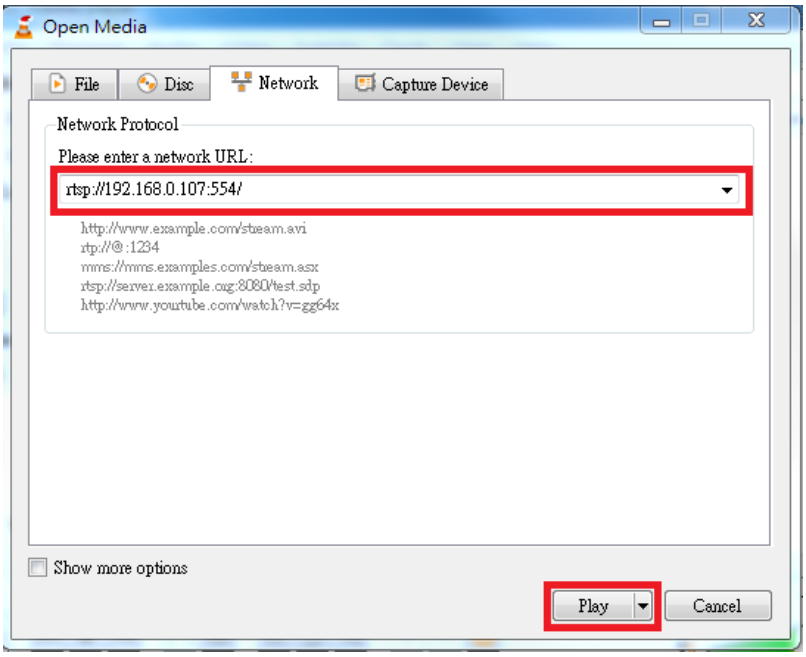
For RTSP examples, you can use VLC media player to receive or transmit the stream. Download VLC media player from website https://www.videolan.org/.

7.2.2.1 Stream audio/video from AmebaPro2 to VLC player

- Click “Media” -> “Open Network Stream”.

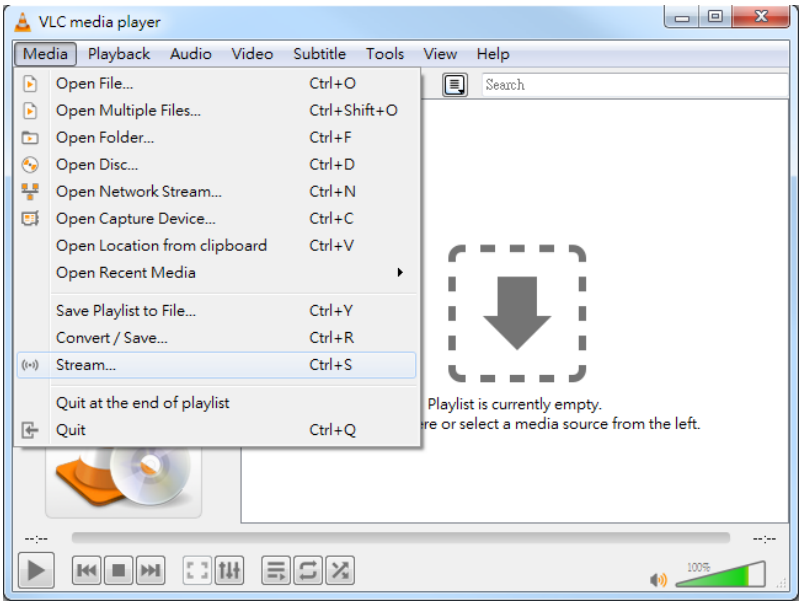


- Enter “rtsp://xxx.xxx.xxx.xxx:yyy/”, where xxx.xxx.xxx.xxxis the Ameba IP addressa and yyy is the RTSP server port (default is 554), and click “Play”.

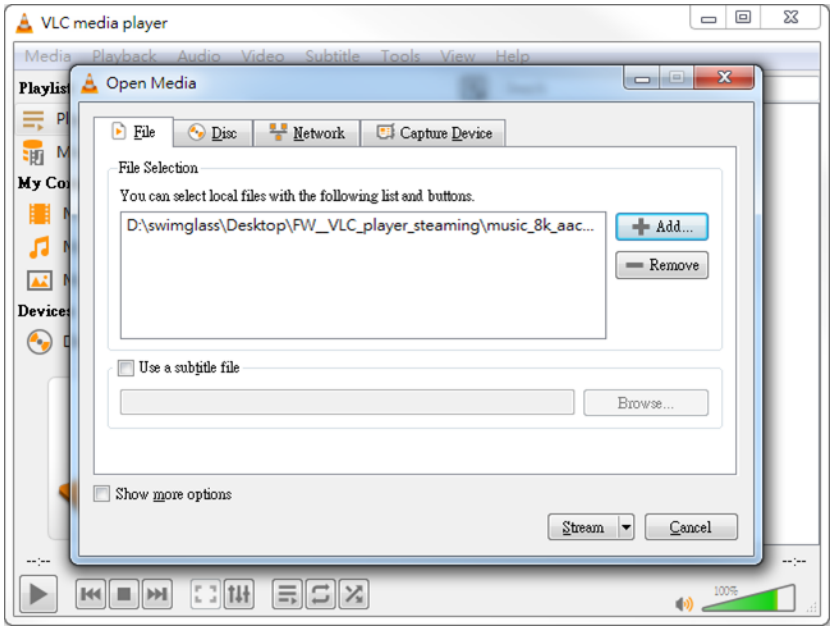


7.2.2.2 Stream audio from VLC player to AmebaPro2

- Click “Media” -> “Stream”.

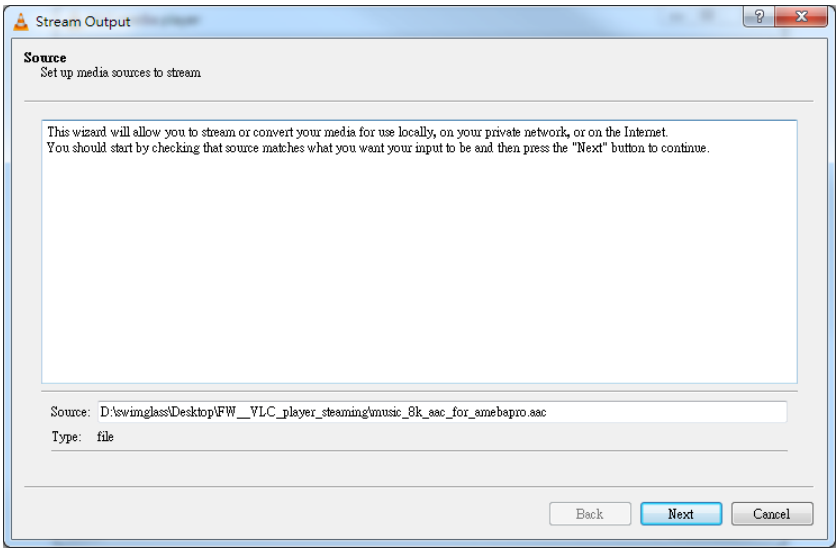


- Select “File”, choose the file by “Add” and finally chick the “Stream”. (If the startup example is RTP -> AAD -> AUDIO please select the audio file with the file name .aac (The file format must be the same as the AAC decoder setting, the default is mono, sampling rate = 8k Hz). If the startup example is RTP -> G711D -> AUDIO, please select the audio file with the file name .wav). If the startup example is RTP -> OPUSD -> AUDIO, please select the audio file with the file name .opus)

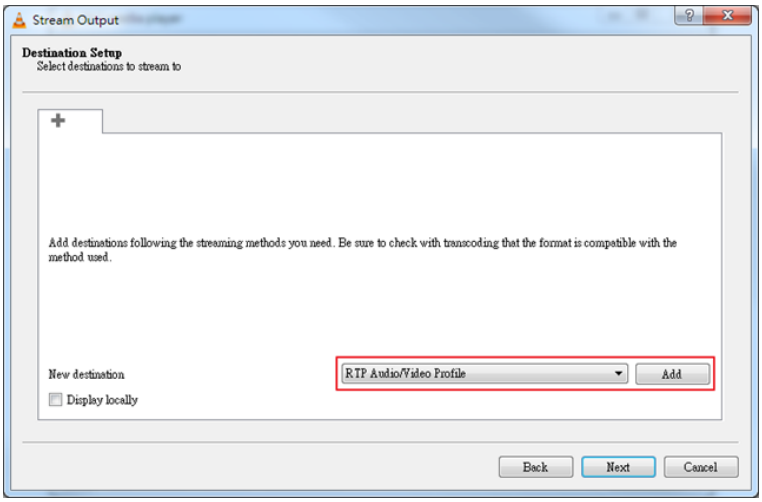


Note

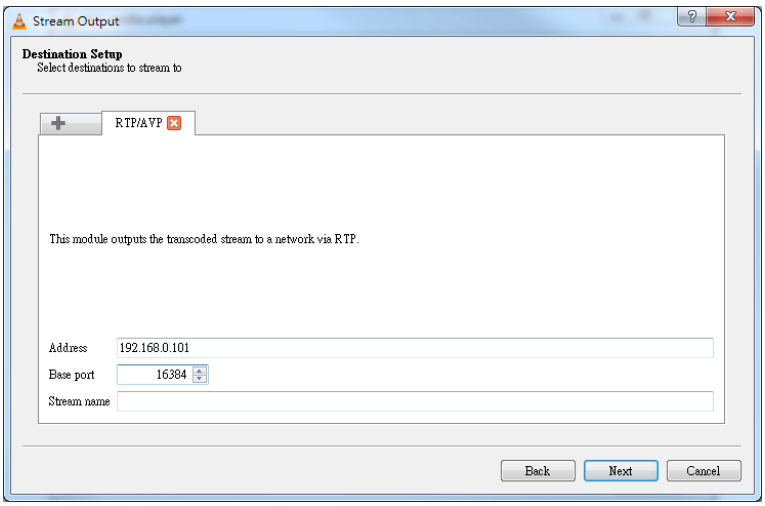
- You will see your select file after push “Stream”. Check it and click “Next”.



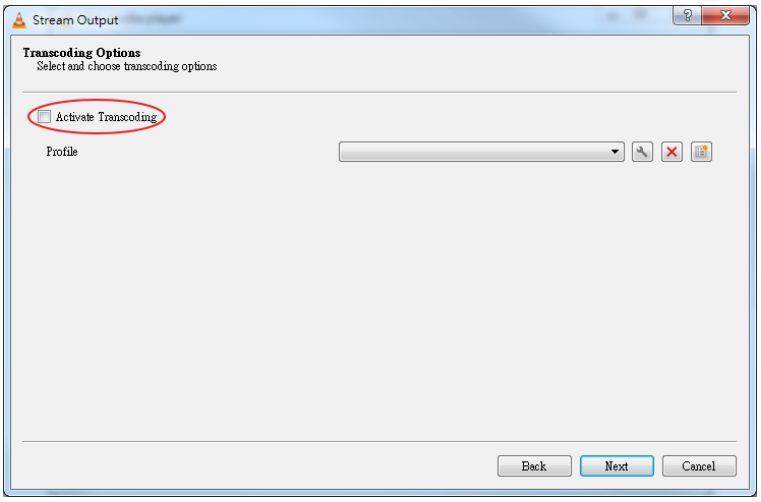
- Select “RTP Audio/Video Profile”, and click “Add”.

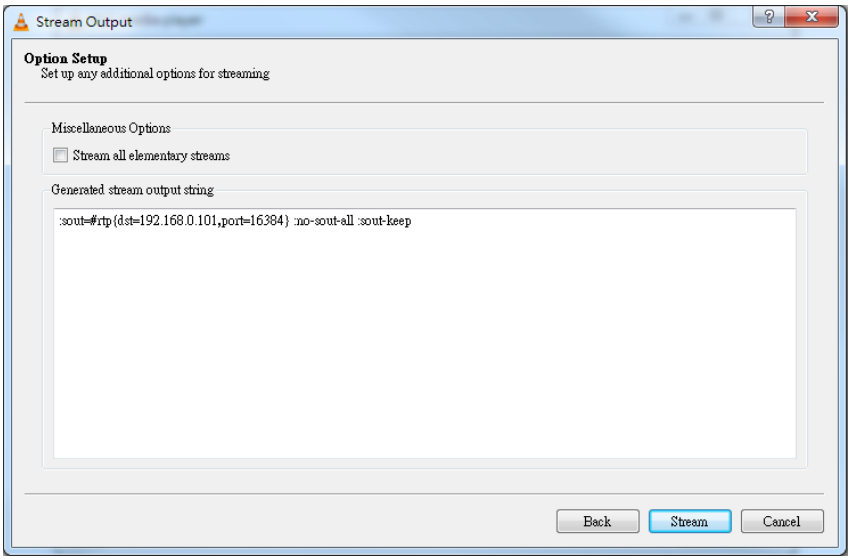


- Enter AmebaPro's IP Address in “Address” field, with “Base port” set to 16384, and click “Next”.



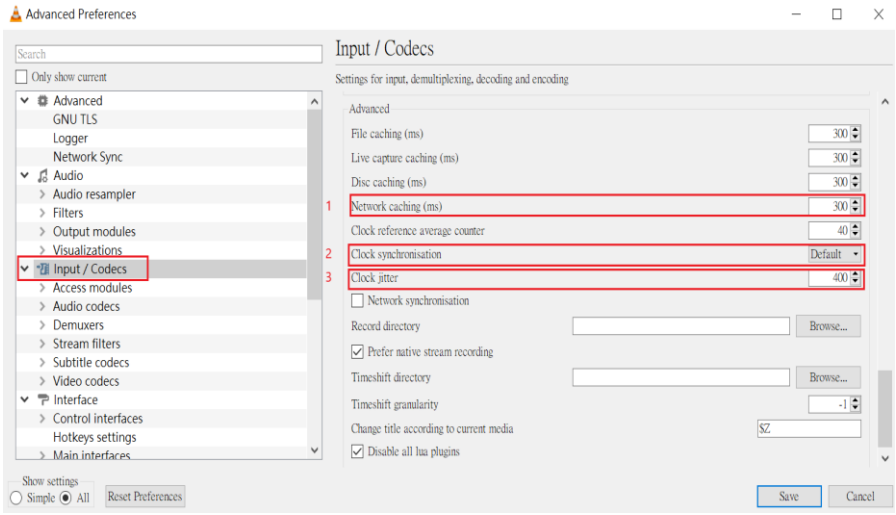
- Confirm “Activate Transcoding” is unchecked, and click “Next” -> “Stream”. Then the sound can be heard on AmebaPro2 3.5mm audio jack.



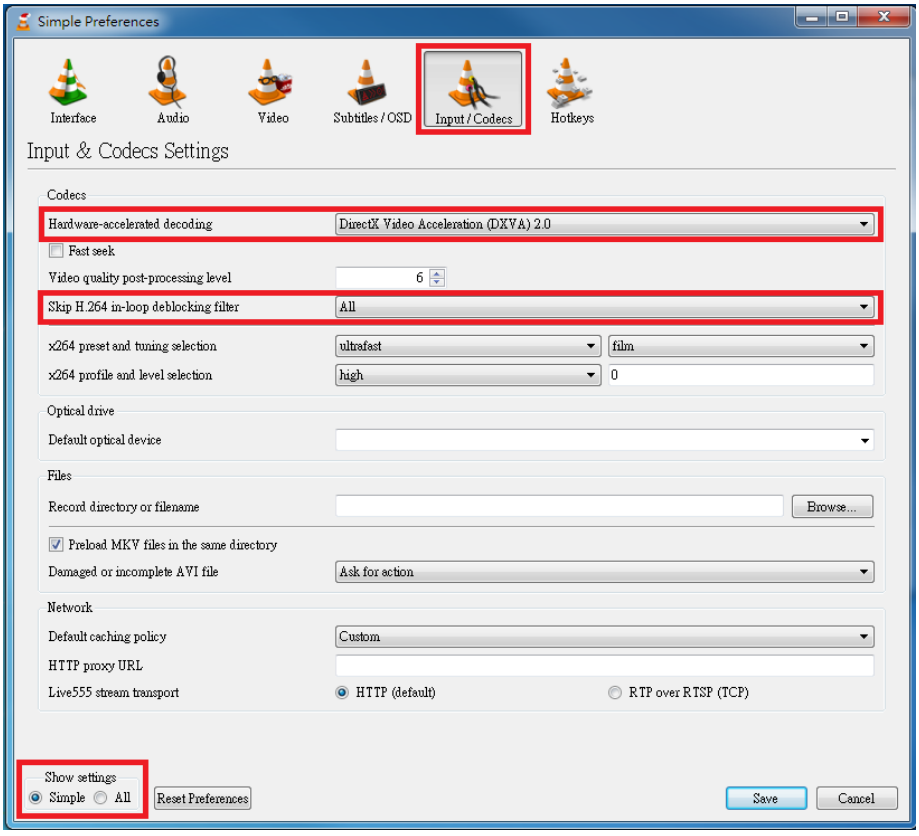


7.2.2.3 Adjust latency (buffer) related settings

- Click “Tools” -> “Preferences” -> “Show settings: All” (lower left corner) -> “Input/ Codecs”, (1) set “Network caching” to 300ms (recommended), (2)set “Clock synchronisation” to Default, (3) set “Clock jitter” to 400ms (recommended).



- Click “Tools” -> “Preferences” -> “Show settings: Simple” (lower left corner) -> “Input/ Codecs”. Enable “Hardware-accelerated decoding” if available, and set “Skip H.264 in-loop deblocking filter” to “All”.



- VLC have a pts_delay buffer by "network buffer" and "clock jitter". The maximum value of this buffer is equal to "network buffer" plus "clock jitter". The video display on the VLC side will delay due to the increase of pts_delay buffer. By reducing the "network cache" and "clock jitter" can achieve the effect of shortening the delay.

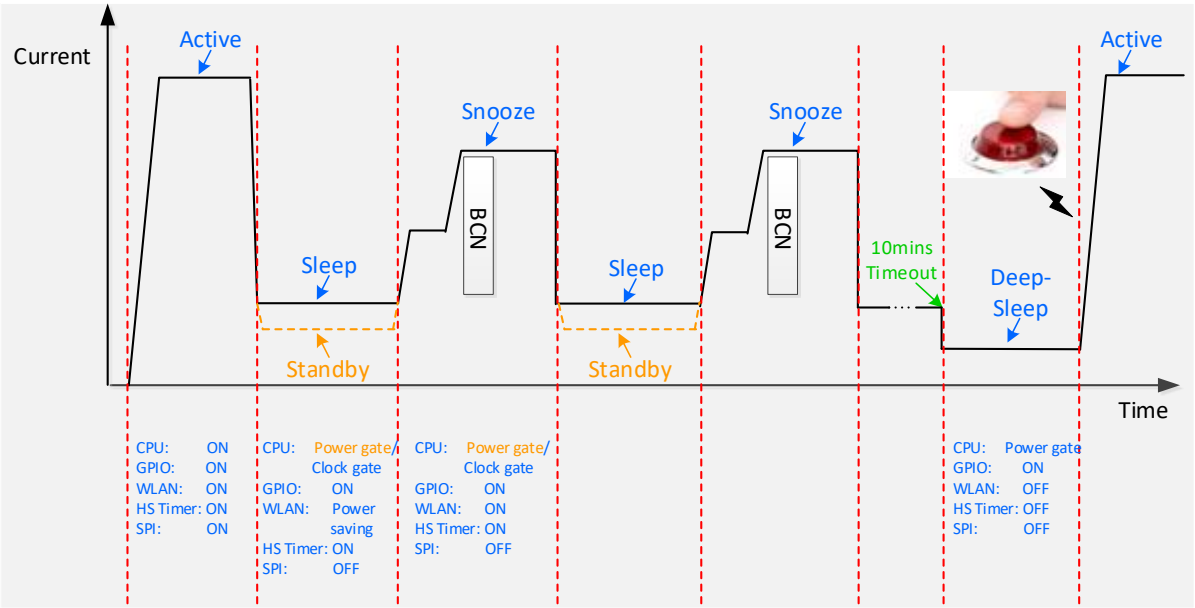
8 Power Save

8.1 Overview

8.1.1 Application Scenario

Ameba-Pro II achieves low power consumption with a combination of several proprietary technologies. The power-saving architecture features six reduced power modes of operation: active, snooze, sleep, standby, deepsleep, shutdown mode. With the elaborate architecture, the battery life of whole IOT system could be extended.

For reading pen application, it can divide into three-scenario. First, press the power button to power on reading pen to active mode and then connect to the cloud to download data. Second, once the reading pen without any activity for 2 minutes, the system will go to sleep mode (for system fast resume and keep WIFI connect) or standby mode (for lower power consumption and keep WIFI connect) and regularly wake up to receive WLAN beacon while into snooze mode. At last, without using the reading pen exceeds 10 minutes, the system will into deep sleep mode and waiting for any button signals to wake up system into active mode.

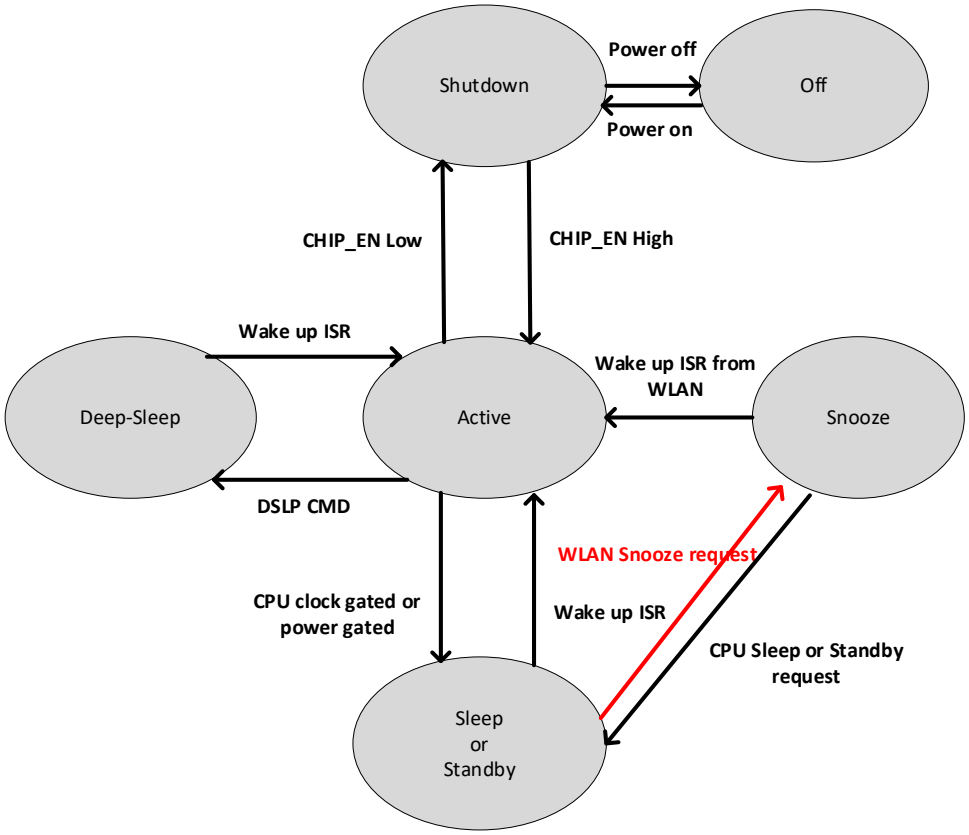


8.1.2 Features

- **Active Mode**: The CPU in active mode and all peripherals are available.
- **Snooze Mode**: In this mode system can regularly wake up to receive WLAN beacon without software intervention. The significant difference between Snooze and Sleep/Standby mode is WLAN capability and could be receive and transmit beacon in this state.
- **Sleep Mode**: The CPU in clock-gated and can be woken up by most of peripherals. The system resume time could be much faster than the Standby mode and the WLAN could be ON or power saving mode in this state.

- **Standby Mode**: The CPU in power-gated and can be woken up by most of peripherals. The power consumption could be lower than the Sleep mode and the WLAN could be ON or power saving mode in this state.
- **DeepSleep Mode**: The lowest power consumption than the other power mode except for shutdown mode, it can be only woken up by LP Timer or GPIO.
- **Shutdown Mode:** The CPU will be shutdown while CHIP_EN was Low.

8.1.3 Power Mode and Power Consumption



In Figure, the power mode can be divided into 6 states except for “off” state and the each power In Figure, the power mode can be divided into 6 states except for “off” state and the each power consumption was shown in Table. The introduction of each power mode, clock-gated and power-gated state will be in the following sections. Clock/power gated state could be regarded as a status of any hardware.

8.2 Deep Sleep Mode

- CHIP_EN keeps high. User can invoke Deep Sleep API to force into deep sleep mode. By using specified interrupts to wake up system.
- The following wake flow: Wake up ISR is high -> PMC -> enable CPU -> Reboot flow.

8.2.1 Wakeup Source

Aon GPIO, RTC, comparator, Aon Timer
Aon GPIO: GPIOA0~GPIOA3
Comparator: GPIOA0~GPIOA3

8.3 Standby Mode

- CHIP_EN keeps high. User can invoke Standby API to force into deep sleep mode. By using specified interrupts to wake up system.
- The following wake flow: Wake up ISR is high -> PMC -> enable CPU -> Fast reboot flow.

8.3.1 Wakeup Source

Aon GPIO, RTC, comparator, Aon Timer, Gtimer0, PWM, Pon GPIO, Uart0, Wlan
Aon GPIO: GPIOA0~GPIOA3
Pon GPIO: GPIOF0~GPIOF17
Comparator: GPIOA0~GPIOA3

8.4 Sleep Mode

- CHIP_EN keeps high. User can invoke Sleep API to force into deep sleep mode. By using specified interrupts to wake up system.
- The following wake flow: Wake up ISR is high -> PMC -> enable CPU -> Execution of instructions continues.

8.4.1 Wakeup Source

Aon GPIO, RTC, comparator, Aon Timer, Gtimer0, PWM, Pon GPIO, Uart0, Wlan
Aon GPIO: GPIOA0~GPIOA3
Pon GPIO: GPIOF0~GPIOF17
Comparator: GPIOA0~GPIOA3

8.5 Snooze Mode

- CHIP_EN keeps high. By using specified interrupts to wake up system.
- The following wake flow: WLAN power on request-> Receive particular beacon-> Wake up ISR is high -> PMC -> enable CPU -> Execution of instructions continues or fast reboot flow.

8.5.1 Wakeup Source

In snooze mode, the only wake up source is WLAN. The wakeup condition could be configured by WLAN driver according to system application. Once the event takes place, WLAN hardware would raise interrupt to PMC that could change hardware state.

9 Audio optimization

The following chapters describe the software and hardware optimization solutions of AmebaPro2 audio.

9.1 Audio setting

9.1.1 Gain setting

The audio input gain can be namely divided into digital gain and analog gain.
As for the analog gain configuration, it support 0 ~ 40 dB to support the gain optimization. The corresponding configuration method can refer to Mic Gain. Recommend that customers can first configure the analog gain. If the audio signal gain need to increase but the analog gain achieve the maximum range, then configure the digital gain.
For digital gain configuration, the suggestion gain configuration is 0dB (0x2F) ~ 4dB (0x3C), though the range is -17.625dB (0x00) ~ 30dB (0x7F). If the digital gain is too large (more than 12dB (0x4F)), digital gain will affect the sound effect and noise will be obvious. ADC Volume shows the corresponding detail setting method of the digital gain.

- **CMD_AUDIO_SET_ADC_GAIN** can be used to set the input digital gain – ADC Volume.

A digital gain configuration is offered to control the audio output gain. Customers can set a reasonable gain value via DAC Volume to obtain the appropriate audio output volume according to their needs. Basically setting the gain to 0dB (0xAF), the output amplitude will meet the board audio output volume requirements. Note that a sound breakage will happen when the output gain is setting too large.

- **CMD_AUDIO_SET_DAC_GAIN** can be used to set the output digital gain – DAC Volume.

9.1.2 Other setting

Here are some command about the audio setting:

- **CMD_AUDIO_SET_RESET** will be re-initialize the audio setting and also the ASP algorithms. If you do some change need to reset the audio configuration, like change the sample rate, reset the audio to switch the configuration.
- **CMD_AUDIO_SET_SAMPLERATE** can set the sample rate. After using this command, a reset is needed to apply the sample rate configuration on audio and ASP algorithms.
  Note: If using audio codec, be sure the sample rate is fitting the sample rate used in audio codec.
- **CMD_AUDIO_SET_TRX** provide a way to stop and re-start the audio without re-initialize the audio system and ASP algorithms. Set 0 to stop the tx and rx progresses or 1 to start them.

9.2 Audio ASP algorithm

The following table shows some common audio problem with their causes and also the adjustment using ASP algorithm.

Situation	Algorithm	Influence End	Causes
Distortion	AGC	transmitting end	The ambient sound is too high Headphone preGain Compression_gain_db of AGC is too large

Low audio volume	AGC	transmitting end	The original input volume is too low Compression_gain_db of AGC is too small AGC is not working properly
Echo or howling	AEC	transmitting end	Too close between transmitting and receiving end device Volume too large or mic too sensitive AEC is not turn on AEC parameters is not setting correctly (frame_size, sample rate, set_sndcard_delay_ms)
Intermittent voice	AEC、NS	transmitting end	NS or AEC suppression
Noise floor	NS	transmitting end	NS mode setting too low Caused by the environment, NS can't do well
Mechanical sound	Network、Device	Receiving end	Poor network environment Device sampling is unstable or device hardware problem

9.2.1 Open ASP algorithm

The codes and functions related to the ASP algorithm are shows in the table.
Enable ENABLE_ASP in module_audio.h and use the 3A (AGC: Automatic gain control; ANS: Adaptive noise suppression; AEC: Acoustic echo cancellation) and VAD (Voice Activity Detection) algorithms to obtain better audio effects.

- The parameters, sample_rate and mic_gain, and the initialization of NS (enable_ns), AEC (enable_aec), AGC (enable_agc), VAD (enable_vad) and other algorithms will be setting at CMD_AUDIO_APPLY and CMD_AUDIO_SET_RESET.

```
================== Open ASP algorithm (module_audio.h) ===============
#define ENABLE_ASP          1

typedef struct audio_param_s {
        audio_sr            sample_rate;     // ASR_8KHZ
        audio_wl            word_length;     // WL_16BIT
```

```
        audio_mic_gain      mic_gain;          // MIC_40DB

        int             channel;        // 1
        int             enable_aec;     // 0: off  1: on
        int             enable_ns;      // 0: off, 1: out 2: in 3: in/out
        int             enable_agc;     // 0: off, 1: output agc
        int             enable_vad;     // 0: off  1: input vad
        int             mix_mode;       // 0
        //...
} audio_params_t;

typedef struct audio_ctx_s {
        void                *parent;
        audio_t             *audio;
        audio_params_t      params;
        uint8_t             inited_aec;
        uint8_t             inited_ns;
        uint8_t             inited_agc;
        uint8_t           inited_vad;
        uint8_t           run_aec;
        uint8_t           run_ns;
        uint8_t           run_agc;
        uint8_t           run_vad;

        uint32_t            sample_rate;
        uint8_t           word_length;          // Byte
        // for AEC
        TaskHandle_t      aec_rx_task;
        xSemaphoreHandle  aec_rx_done_sema;
} audio_ctx_t;

=================== ASP algorithm function (AEC.h) ================
void AEC_init(int16_t frame_size , int32_t sample_freq , AEC_CORE aec_core,
                int speex_filter_length, int16_t agc_mode, int16_t
compression_gain_db, uint8_t limiter_enable,
                int ns_mode, float snd_amplification);

int AEC_set_level(int level);
int AEC_process(const int16_t* farend, const int16_t* nearend, int16_t* out);
void AEC_destory();

void AGC_init(int32_t sample_freq, int16_t agc_mode, int16_t
compression_gain_db, uint8_t limiter_enable);
void AGC_destory(void);
void AGC_process(int16_t frame_size, int16_t* out);
```

```
void AGC2_init(int32_t sample_freq, int16_t agc_mode, int16_t
compression_gain_db, uint8_t limiter_enable);
void AGC2_destory(void);
void AGC2_process(int16_t frame_size, int16_t* out);

void NS_init(int32_t sample_freq, int16_t ns_mode);
void NS_destory(void);
void NS_process(int16_t frame_size, int16_t* out);

void NS2_init(int32_t sample_freq, int16_t ns_mode);
void NS2_destory(void);
void NS2_process(int16_t frame_size, int16_t* out);

void VAD_init(int32_t sample_freq, int16_t vad_mode);
void VAD_destory(void);
int VAD_process(int16_t frame_size, int16_t* out);
```

9.2.1.1 ASP algorithm usage

- 8K and 16K audio sample rate are supported in the ASP algorithms.
- When enable_ns is set, 0 is to disable NS, 1 is to enable NS_init() for Speaker, 2 means enable NS2_init() for MIC, 3 is to enable both directions.
- When enable_agc is set, 0 is to disable AGC, 1 is to enable AGC_init() for Speaker, 2 is to enable AGC2_init() for MIC, and 3 is to enable both directions.
- Set enable_aec 0/1 to disable/enable the AEC_init().
- In AEC_process, it will also run NS and AGC algorithms, so it will be unnecessary to apply additional NS and AGC for MIC (NS2_init() and AGC2_init()) if open enable_aec.
- Set enable_vad to 0/1 to disable/enable VAD_init().

9.2.1.1.1 AEC setting

The AEC algorithm includes three parts: delay adjustment strategy, linear echo estimation, and nonlinear echo suppression.

- Use CMD_AUDIO_RUN_AEC to dynamically switch the use of AEC_process().
- Use CMD_AUDIO_SET_AEC_ENABLE to determine whether AEC_init() is enabled during audio reset.
- CMD_AUDIO_SET_AEC_LEVEL can set the strength of cancellation.
- Note that if using WEBRTC_AECM (default) as AEC_CORE to do initialization, the AEC strength level is 0 ~ 4 and the minimum strength is 0; while using WEBRTC_AEC as AEC_CORE to do initialization, the AEC strength level is 0 ~ 2 and the minimum strength is 0.
- Default value of WEBRTC_AECM is 3, and default value of WEBRTC_AEC is 1
- Please make sure that the level setting is after that the audio is initialed.
- The parameters agc_mode, compression_gain_db and limiter_enable of AEC_init are for setting the agc processing mode, compression gain (default 18dB) and limiter (default 0 => disable) of AGC algorithm in AEC_process.
- The parameters ns_mode of AEC_init are for setting the NS strength of NS algorithm in AEC_process.

9.2.1.1.2 AEC effect

Here is the estimation result of AEC algorithm on the device.

- For the audio setting, the MIC gain, ADC gain and DAC gain are set as 40dB, 12dB (0x4F) and 0dB (0xAF).
- Only the NS process with NS mode level 3 is applied before AEC algorithm to decrease the noise of the rx input.
- The ACOM is obtained by the average 1 minutes speech result.

9.2.1.1.3 NS setting

The NS algorithm is armed to decrease the noise or environment sound, so it is recommend to use it before other ASP algorithms.

- Use CMD_AUDIO_SET_NS_ENABLE to determine whether NSx_init() is enabled during audio reset.
- Setting parameter NS_MODE in NSx_init() to adjust ns aggressive level. The value is from 0 to 3 and the default value is 3.
- Use CMD_AUDIO_RUN_NS to dynamically switch the use of NSx_process().

9.2.1.1.4 AGC setting

The AGC algorithm is used to balance the audio volume of signal streaming.

- Use CMD_AUDIO_SET_AGC_ENABLE to determine whether AGCx_init() is enabled during audio reset.
- The parameter of agc_mode can choose the AGC mode to initial, compression_gain_db (default 18/24 for in/not in the AEC) can setting max gain of AGC and limiter_enable (default 0) to restrict the signal level.
- Use CMD_AUDIO_RUN_AGC can dynamically switch the use of AGCx_process().

9.2.1.1.5 VAD setting

The VAD algorithm is applied to do voice enhancement.

- Use CMD_AUDIO_SET_VAD_ENABLE to determine whether VAD_init() is enabled during audio reset.
- The parameter VAD MODE can set the aggressive level of VAD. The value can be configured from 0 to 3and default is 1.
- Use CMD_AUDIO_RUN_VAD to dynamically switch the use of VAD_process().