Airoha IoT SDK for BT Audio Build Environment Guide

Version: 1.15

Release date: 10 March 2022

Document Revision History

Revision	Date	Description
1.0	31 March 2016	Initial release
1.2	30 June 2016	• Added support for build environment on Windows OS.
1.3	13 January 2017	Update providing: • A method to add a module. • Support for C++ source files.
1.4	5 May 2017	• Added details on the pre-built folder. • Added requirement to Install GNU Awk on LinkIt 2533 HDK.
1.5	19 July 2019	• Configuration for multi-processor build.
1.6	12 Nov 2019	• Renamed the document from “Airoha IoT SDK GCC Build Environment Guide” to “Airoha IoT SDK for BT Audio Build Environment Guide.” • Added 155x DSP build environment setup
1.7	21 Nov 2019	• Made a correction to the description of the install process
1.8	13 Dec 2019	• Revised examples and related description for AB155x.
1.9	16 Jun 2020	• Revise supported Linux versions
1.10	1 July 2020	• Added support for AB1565
1.11	24 July 2020	• Added support for AB1568
1.12	6 May 2021	• Remove unused description
1.13	21 Jan 2022	• Added support for AB1585/AB1588
1.14	23 Feb 2022	• Revised examples and related description to AB158x
1.15	10 March 2022	• Added feature option improvements in section 5 Makefiles

Table of contents

1. Overview 1
2. Environment 2
2.1. Installing the SDK build environment on Linux 2
2.2. Preparing Linux build environment 2
2.3. Start cadence toolchain license service daemon 3
2.4. Offline installation 3
2.5. Installing the SDK build environment on Microsoft Windows 4
2.6. Preparing build environment 4
2.7. Offline installation 5
2.8. Troubleshooting 6
3. Building the Project Using the SDK 8
3.1. Building projects 8
3.1. Build the project. 9
3.2. Clean the out folder. 10
3.3. Build the MCU project with the "bl" option 10
3.4. Building the project from the MCU and DSP project configuration directory 10
4. Folder Structure 12
5. Makefiles 14
5.1. Project Makefile 14
5.2. Configuration Makefiles 15
6. Adding a Module to the Middleware 17
6.1. Files to add 17
6.2. Source and header files 17
6.3. Makefiles for the module 18
6.4. Adding a module to the build flow of the project 20
7. Creating a Project 21
7.1. Using an existing project 21
7.2. Removing a module 21
7.3. User-defined source and header files 21
7.4. Test and verify 23
7.5. Troubleshooting 23
7.6. Configuration for multi-processor products 23
7.7. Creating a new build target in mapping_proj.cfg 24

Lists of tables and figures

Table 1. Recommended build environment 2

Figure 1. Install MSYS2 4
Figure 2. The SDK package folder structure 12
Figure 3. Place module source and header files under the module folder 17
Figure 4. Create a module.mk under module folder 18
Figure 5. Create a Makefile under mymodule folder 18
Figure 6. Project source and header files under the project folder 22
Figure 7. Output image file, object files and dependency files after project is successfully built 23

1. Overview

Airoha IoT SDK for BT Audio build environment guide provides tools and utilities to install the supporting build environment and run your projects.

The document guides you through:

- Setting up the build environment
- Building a project using the SDK
- Adding a module to the middleware
- Creating your own project

The build environment guide is applied to the Airoha IoT Development Platform including AB155x/AB1565/AB1568/AB1585/AB1588 EVK.

In this guide, all the supported chips uses the same installation flow and build method as AB158x. The following examples use AB158x as a reference.

2. Environment

This section provides detailed guideline on how to set up the SDK build environment with default GCC on Linux OS and on Microsoft Windows using MSYS2 cross-compilation tool.

2.1. Installing the SDK build environment on Linux

2.2. Preparing Linux build environment

The toolchain provided in the SDK is required to setup the build environment on Linux OS (Ubuntu 18.10 64bit or Ubuntu 18.04 LTS).

Table 1. Recommended build environment

Item	Description
OS	• Linux OS 18.10 or 18.04 LTS
Make	• GNU make 3.81

To install the SDK and build environment on Linux, please download IoT_SDK_For_BT_Audio_and_Linux_Tools_All_In_One from MOL. The package contains ARM GCC toolchain, Cadence DSP toolchain for linux and Airoha IoT SDK for BT Audio. The package also contains an install script to setup the environment. Use following steps to do the installation:

1) Select a folder as an installation working folder, unpack IoT_SDK_For_BT_Audio_and_Linux_Tools_All_In_One to the folder. (The folder name should not contain any special characters such as space and brace.)

2) Open install.sh in the install working folder for editing. Set a free network port (suggest use port number between 1024 and 9999) for Cadence compiler license service daemon: change the value of DSP_LICSERV_PORT. Please do the setting especially if you are working on a Linux host sharing with multiple users.

```
#!/bin/bash

...
DSP_LICSERV_PORT=6677
...
```

3) Execute ./install.sh. The installation process requires Linux root permission and internet connection. The script will do:

- Install Cadence license tool chain
- Get Cadence toolchain license
- Unzip the Airoha IoT SDK for BT Audio

```
./install.sh
```

After installation complete, you should see following messages.

```
...
...
Installation done.
!!!!! Please remember to run '~/airoha_sdk_toolchain/start_lic_server'
again if you reboot the system !!!!!
Now you can start the first build, please change directory to bta_sdk
and execute build command.
Example:
       cd bta_sdk
       ./build.sh ab1585_evk earbuds_ref_design
```

4) Build your project. (Since the Cadence license service daemon needs take a few minutes to start work after installation, please wait a while to start your first build after installation done). Run following commands to build you project.

```
cd bta_sdk
./build.sh ab1585_evk earbuds_ref_design
```

The screen output of build process should be as bellow.

```
$./build.sh ab1585_evk earbuds_ref_design
cd /home/airoha/<All in one root>/bta_sdk/dsp
============================================================
Start DSP0 Build
============================================================
output folder change to:
...
...
trigger by build.sh, skip clean_log
Assembling...
../../../../../driver/chip/ab158x/dsp0/src_core/sleep_exit_prepare.S
...
...
```

The install process puts installed toolchain under ~/airoha_sdk_toolchain. You don't need to run install script again for new SDK release. Just get SDK standalone package “IoT_SDK_For_BT_Audio” and unzip to use the new SDK package. Please do not remove or modify ~/airoha_sdk_toolchain, it will cause the build environment broken.

2.3. Start cadence toolchain license service daemon

The installation process will also start the Cadence license service daemon during install process. However, user needs to restart the server after system reboot. Please restart the server using following command under the installation working folder. The Cadence licenser service daemon needs few minutes to start service after launch.

```
~/airoha_sdk_toolchain/start_lic_server.sh
```

2.4. Offline installation

The installation process needs connect to internet to get Cadence license. For the case that user needs to install offline, user needs to manually install the license.

- Submit a JIRA request to get a **floating** license.
- Revise the install.sh

a. modify value of DSP_LIC_FILE to your license file

You can get the <license name>.lic from the Airoha eService Cadence License Request:
Airoha eService website > Project > Your Project > Customer channels > Customer portal > Software Releated > License Request.

b. remove the line CADENCE_LIC_HOST=xxxxx

```
#!/bin/bash

DSP_LIC_FILE="<license name>.lic"
...

#CADENCE_LIC_HOST=lic.airoha.com.tw
...
```

- Run the install.sh for install SDK offline.

The installation guidance video (online and offline) can be found at here.

Since the installation process of the build environment is the same, please reference to the "AB155x_Linux_Dev_Env_Setup.mp4".

2.5. Installing the SDK build environment on Microsoft Windows

To install the SDK and build environment on Windows, please download and extract the content of the IoT_SDK_For_BT_Audio_and_Windows_Tools_All_In_One on your host. The package contains MSYS, ARM GCC for Windows, and Cadence DSP toolchain for windows and Airoha IoT SDK for BT Audio. Use following instructions to do the installation:

2.6. Preparing build environment

1) Select a folder as an installation working folder, unpack IoT_SDK_For_BT_Audio_and_Windows_Tools_All_In_One to the folder. (The folder name should not contain special character such as space and brace.)

2) Execute install_msys.bat to install msys2. If you have already installed msys in your system, you can skip this step. Please notice the process needs internet connection. For the case the internet connection is not allowed, please check MSYS2 official web for offline installation.

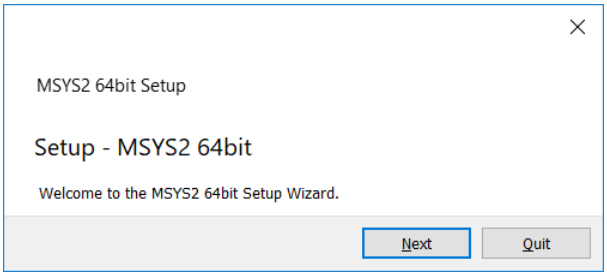

Figure 1. Install MSYS2

3) Execute msys2.exe and you can see an msys2 terminal (bash).

4) Execute ./install.sh under msys2 terminal. The installation process also requires internet connection. The script will do:

- Install required MSYS software components: diffutils, make, p7zip. tar

- Install Cadence license tool chain
- Get Cadence toolchain license
- Unzip the Airoha IoT SDK for BT Audio

The screen output of installation process should be as bellow.

```
$ ./install.sh
MSYS package installing
loading packages...
...
...
MSYS package install done
...
...
Extracting archive: IoT_SDK_for_BT_Audio_V3.0.0.7z
...
...

  0% 105 - mcu/tools/gcc/win/gcc-arm-none- ....
```

5) Build your first project

```
cd bta_sdk
./build.sh ab1585_evk earbuds_ref_design
```

The screen output of the build process should be as below.

```
$./build.sh ab1585_evk earbuds_ref_design
...
============================================================
Start DSP0 Build
============================================================
output folder change to:
...
...
trigger by build.sh, skip clean_log
Assembling...
../../../../../driver/chip/ab158x/dsp0/src_core/sleep_exit_prepare.S
...
...
```

The install process puts installed toolchain under ~/airoha_sdk_toolchain. You do not need to run the install script again for new SDK release. Just get SDK standalone package "IoT_SDK_For_BT_Audio" and unzip to use the new SDK package. Please do not remove or modify ~/airoha_sdk_toolchain because it will cause the build environment to break.

2.7. Offline installation

The installation process needs connect to internet to get Cadence license. For the case that user needs to install offline, user needs to manually install the license.

- Submit a JIRA request to get a node-locked license.
- Revise the install.sh

a. modify value of DSP_LIC_FILE to your license file

You can get the <license name>.lic from the Airoha eService Cadence License Request: https://eservice.airoha.com.tw/servicedesk/customer/portal/234

b. remove the line CADENCE_LIC_HOST=lic.airoha.com.tw

```
#!/bin/bash

DSP_LIC_FILE="<license name>.lic"
...
#CADENCE_LIC_HOST=lic.airoha.com.tw
...
```

- Run the install.sh for install SDK offline.

The installation guidance video (online and offline) can be found at here.

Since the installation process of the build environment is the same, please reference to the "AB155x_Windows_Dev_Env_Setup.mp4".

2.8. Troubleshooting

Note the following caveats when building the project with MSYS2 on Windows OS.

- The folder name and file name in the Airoha IoT SDK should not contain ' ', ,'(', ')', '[' or ']' characters.
- The SDK path name should be as short as possible. Otherwise, a build error similar to the one shown below may occur due to the long path. The sum of the path length SDK installed folder and project name should than 35 characters.

```
Arm-none-eabi-gcc.exe: error:
../../../../../out/ab158x_evk/i2c_communication_with_EEPROM_dma/obj/proj
ect/ab158x/hal_examples/i2c_communication_with_EEPROM_dma/src/system_ab1
58x.o: No such file or directory
```

- The makefile in your project should not use any platform dependent commands or files, such as `stat` or `/proc/cpuinfo`.
- By default, the parallel build feature is enabled in `build.sh` to speed up the compilation. Disable the parallel build feature, if any unreasonable build error or system exception occurs.
- To disable the parallel build feature for all modules, change the value "-j" of EXTRA_VAR to "-j1" in <SDK_ROOT>/mcu/build.sh and <SDK_ROOT>/dsp/build.sh as shown below.

```
platform=$(uname)
if [[ "$platform" =~ "MINGW" || "$platform" =~ "MSYS" ]]; then
    max_jobs=$(( $(WMIC CPU Get NumberOfLogicalProcessors|tail -2|awk
'{print $1}') - 1))
else
    max_jobs=`cat /proc/cpuinfo |grep ^processor|wc -l`
fi
```

```
export EXTRA_VAR=-j$max_jobs
```

- If build errors are due to a particular module's parallel build, set the value <module>_EXTRA as "-j1" in <SDK_ROOT>/mcu/.rule.mk to disable the parallel build for that particular module, as shown below.

```
OS_VERSION := $(shell uname)
ifneq ($(filter MINGW% MSYS%,$(OS_VERSION)),)
  $(DRV_CHIP_PATH)_EXTRA    := -j1
  $(MID_MBEDTLS_PATH)_EXTRA := -j1
  $(<module>)_EXTRA         := -j1
endif...
```

Then, rebuild your project.

```
./build.sh ab1585_evk earbuds_ref_design clean
./build.sh ab1585_evk earbuds_ref_design
```

3. Building the Project Using the SDK

3.1. Building projects

Build MCU and DSP projects using the script in <sdk_root>/build.sh. To find more information about the script, navigate to the SDK root directory and execute the following command:

```
cd <sdk_root>
./build.sh
```

The outcome is:

```
================================================================
Build Project
================================================================
Usage: ./build.sh <board> <project> [clean] <argument>

Example:
       ./build.sh ab158x earbuds_ref_design
       ./build.sh ab158x earbuds_ref_design -fm=feature_ab1585_evk.mk
       ./build.sh clean
        (clean folder: out)
       ./build.sh ab158x clean
        (clean folder: out/ab158x_evk)
       ./build.sh ab158x earbuds_ref_design clean
        (clean folder: out/ab158x /earbuds_ref_design)

Argument:
       -fm=<feature makefile>
           Replace feature.mk with other makefile for mcu. For example,
           the feature_example.mk is under project folder,
           -fm=feature_example.mk will replace feature.mk with
           feature_example.mk.

       -fd0=<feature makefile>
           Replace feature.mk with other makefile for dsp0. For example,
           the feature_example.mk is under project folder,
           -fd0=feature_example.mk will replace feature.mk with
           feature_example.mk.

       -fd1=<feature makefile>
           Replace feature.mk with other makefile for dsp1. For example,
           the feature_example.mk is under project folder,
           -fd1=feature_example.mk will replace feature.mk with
           feature_example.mk.

       -mcu
           Build MCU only. (Use default dsp bin at 'dsp/prebuilt/dsp0/'
           instead of build dsp bin)."

================================================================
List Available Example Projects
================================================================
Usage: ./build.sh list
```

- List all available boards and projects.

Run the command to show all available boards and projects:

```
./build.sh list
```

The available boards and projects are listed below.

```
============================================================
Available Build Projects:
============================================================
...
ab158x_evk
  earbuds_ref_design
      MCU: earbuds_ref_design
      dsp0: dsp0_headset_ref_design
ab158x_evk
  headset_ref_design
      MCU: headset_ref_design
      dsp0: dsp0_headset_ref_design
...
```

3.1. Build the project.

To build a specific project, simply run the following command.

```
./build.sh <board> <project>
```

The output files are then put in the <sdk_root>/out/<board>/<project> folder.

Note, skip dsp build with **–mcu** option.

For example, to build a project in the AB158x EVK, run the following build command:

```
./build.sh ab158x earbuds_ref_design
```

The standard output in the terminal window is as follows:

```
$./build.sh ab158x earbuds_ref_design
cd /d/131/bta_sdk/dsp
============================================================
Start DSP0 Build
============================================================
output folder change to:
/d/131/bta_sdk/dsp/out/ab158x_evk/dsp0_headset_ref_design/feature
make -C project/ab158x_evk/apps/dsp0_headset_ref_design/XT-XCC
OUTDIR=/d/131/bta_sdk/dsp/out/ab158x_evk/dsp0_headset_ref_design/feature
-j5 FEATURE=feature.mk
OUT=out/ab158x_evk/dsp0_headset_ref_design/feature 2>>
/d/131/bta_sdk/dsp/out/ab158x_evk/dsp0_headset_ref_design/feature/log/er
r.log
make: Entering directory
'/d/131/bta_sdk/dsp/project/ab158x_evk/apps/dsp0_headset_ref_design/XT-
XCC'
trigger by build.sh, skip clean_log
Assembling... ../../../../../driver/chip/ab158x/dsp0/src_core/sleep_exit
_prepare.S
Assembling... ../../../../../kernel/service/exception_handler/src/except
ion.S
```

The output files are then put in the <sdk_root>/out/ab158x_evk/earbuds_ref_design/ folder.

3.2. Clean the out folder.

The build script <sdk_root>/build.sh provides options for removing the generated output files, as shown below.

Clean the <sdk_root>/out folder.

```
./build.sh clean
```

Clean the <sdk_root>/out/<board> folder.

```
./build.sh <board> clean
```

Clean the <sdk_root>/out/<board>/<projecct> folder.

```
./build.sh <board> <project> clean
```

The output folder is defined under variable BUILD_DIR in the Makefile in <sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC:

```
BUILD_DIR = $(PWD)/Build
PROJ_NAME = $(shell basename $(dir $(PWD)))
```

A project image earbuds_ref_design.bin is generated under <sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC/Build.

3.3. Build the MCU project with the "bl" option

By default, the pre-built bootloader image file is copied to the <sdk_root>/mcu/out/<board>/<project>/ folder after the project is built. The main purpose for the bootloader image is to download the Flash Tool.

Apply the "bl" option to rebuild the bootloader and use the generated bootloader image file instead of the pre-built one, as shown below.

```
./build.sh <board> <project> bl
```

To build the project on the AB158x EVK:

```
cd <sdk_root>/mcu
./build.sh ab158x earbuds_ref_design bl
```

The output image file of the project and the bootloader, along with the merged image file flash.bin, will be placed under <sdk_root>/mcu/out/ab158x_evk/earbuds_ref_design folder.

3.4. Building the project from the MCU and DSP project configuration directory

To build the project:

1) Change the current directory to project source directory where the SDK is located.
2) There are makefiles provided for the project build configuration. For example, the MCU project earbuds_ref_design is built by the project makefile under <sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC.

   The DSP project dsp0_headset_ref_design is built by the project makefile under

   <sdk_root>/dsp/project/ab158x_evk/apps/dsp0_headset_ref_design/XT-XCC.
3) Navigate to the example project's location.

For MCU project earbuds_ref_design:

```
cd <sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC
```

For DSP project dsp0_headset_ref_design:

```
cd <sdk_root>/dsp/project/ab158x_evk/apps/dsp0_headset_ref_design/XT-XCC
```

4) Run the `make` command.

```
Make
```

The MCU project output folder is defined under variable `BUILD_DIR` in the `Makefile` located at `<sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC`:

```
BUILD_DIR = $(PWD)/build
PROJ_NAME = earbuds_ref_design
```

A project image `earbuds_ref_design.bin` is generated under `<sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC/build`.

The DSP project output folder is defined under variable `OUT_DIR` in the `Makefile` located at `<sdk_root>/dsp/project/ab158x_evk/apps/dsp0_headset_ref_design/XT-XCC`:

```
OUTDIR       :=  $(abspath $(strip $(ROOTDIR))/out/$(strip
$(BOARD))/$(strip $(PROJ_NAME))/$(strip $(FEATURE_BASENAME)))
PROJ_NAME :=  $(notdir $(shell cd ../ ; pwd))
```

A project image `dsp0_headset_ref_design.bin` is generated under `<sdk_root>/dsp/out/ab158x/dsp0_headset_ref_design/feature`.

4. Folder Structure

This section shows the structure of the SDK and introduces the content of each folder. The SDK package is organized in a folder structure, as shown in Figure 2.

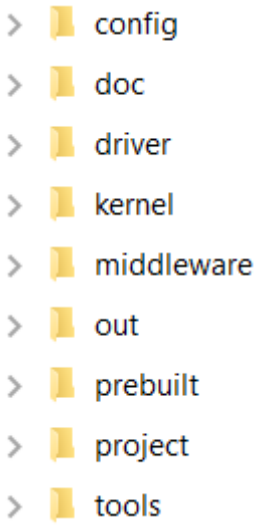

Figure 2. The SDK package folder structure

This package contains the source and library files of the main features, the build configuration, related tools and documentation. A brief description of the layout of these files is provided below:

`config` — includes make and compile configuration files for compiling a binary project.

`doc` — includes SDK related documentation, such as developer and SDK API reference guides.

`driver` — includes common driver files, such as board drivers, peripheral and CMSIS-CORE interface drivers.

`kernel` — includes the underlying RTOS and system services for exception handling and error logging.

`middleware` — includes software features for HAL and OS, such as network and advanced features.

`out` — contains binary files, libraries, objects and build logs.

`prebuilt` — contains binary files, libraries, header files, makefiles and other pre-built files.

`project` — includes pre-configured example and demo projects using Wi-Fi, HTTP, HAL, and more.

`tools` — includes tools to compile, download and debug projects using the SDK.

To enable building several different projects simultaneously, each project's output files are placed under the corresponding `<sdk_root>/out/<board>/<project>/` folder. The dependency of the which mcu/dsp side project be used to build the bin in the `<sdk_root>/out/<board>/<project>/` folder can reference to the 7.6 Configuration for multi-processor products .

For AB158x, the target folder path is `<sdk_root>/out/ab158x_evk/earbuds_ref_design/` folder.

A brief description of the files is provided below:

Binary files

- `project image` — the naming rule of the image file is `PROJ_NAME.bin`, which in this case is `earbuds_ref_design.bin`. The variable `PROJ_NAME` is defined in `Makefile`, see section 5, "Makefiles".
- `bootloader image` — the naming rule of the image file is `bootloader.bin`.

`elf file` — contains information about the executable, object code, shared libraries and core dumps.

`map file` — contains the link information of the project libraries.

`lib folder` — contains module libraries.

`log folder` — contains build log including build information, timestamp and error messages.

`obj folder` — contains object and dependency files.

5. Makefiles

The SDK package contains several Makefiles, including the following:

- Project Makefile
    - Makefile - Mainly used to generate the project image.
- Configuration Makefiles
    - feature.mk - Feature configuration file
    - chip.mk - Chip configuration file

We have made some improvements to the AB158x series as listed below.

- Simplified feature_*.mk - Removed unused feature options and rules, synchronized option names with other Airoha chips and made the optional names consistent with the same functionality between MCU and DSP sides. Adjustments use one feature option for only one feature whenever possible.
- Comment - Added comments to describe the usage and notes about the feature options in feature_*.mk to make it easy for customers to understand.
- Dependency check - Sets the dependency rules for the build flow of the project, to check feature option settings.

Please use the **earbuds_ref_design** project as a reference for more details on the usage and relationship of each Makefile.

5.1. Project Makefile

Project Makefile is placed under the <sdk_root>/mcu/project/<board>/apps/<project_name>/GCC/ folder. Its purpose is summarized below:

- Configures project settings, including the root directory, project name, project path, and more.
- Includes other Makefiles for the configuration, such as `feature.mk`, `chip.mk` and `module.mk`.
- Sets the file path of the project's source code.
- Sets the include path of the project's header files.
- Sets the dependency rules for the build flow of the project.
    - Add a make rule to check for feature options conflict and stop the build if there is a conflict. (Error message tag "**[Conflict feature option**]")
        - For example, environment detection (ED) and ANC have dependency; If we want to enable ED but forget to enable the ANC feature, the message will show "[Conflict feature option] To enable AIR_ANC_ENVIRONMENT_DETECTION_ENABLE must support AIR_ANC_ENABLE." to notify the user to enable ANC.
    - Add make rule to check for add-on feature options. (Error message tag "**[Addon feature option fail]**") In this case, please contact Airoha PLM to request the add-on package.
- Sets the module libraries to link when creating the image file.

- Triggers a make command for each module to create a module library.

5.2. Configuration Makefiles

This section provides more details on the configuration Makefiles, `feature.mk` and `chip.mk`.

The `feature_*.mk` is located in the <sdk_root>/mcu/project/<board>/apps/earbuds_ref_design/GCC/ folder. Users can turn on/off or set a value for a specific feature by simply changing it in the feature_*.mk and get more information on the usage of feature option from option comments. The template is as follows.

```
# Give a brief introduction to this feature option.
# (By case) It must be turned on/off for both DSP and MCU, otherwise, it
will not work.
# (By case) Dependency description
# (By case) For value case, describe each value of the feature option.
<feature_option> = y/n or value
```

The `IC_CONFIG` and `BOARD_CONFIG` are also defined in the `feature.mk`.

```
IC_CONFIG                               = <chip>
...
BOARD_CONFIG                            = <board>
...

# This option is used to enable/disable User Unaware adaptive ANC.
# It must be turned on/off for both DSP and MCU, otherwise, it will not
work.
# Dependency: AIR_ANC_ENABLE must be enabled when this option is set to
y.
AIR_ANC_USER_UNAWARE_ENABLE = n
...

# This option is to choose the uplink rate. Default setting is none.
# It must be set to the same value for both DSP and MCU, otherwise, it
will not work.
# Up Link Rate : none, 48k
#                none : uplink rate will be handled by scenario itself.
#                48k  : uplink rate will be fixed in 48k Hz.
AIR_UPLINK_RATE   = none
...
```

The `chip.mk` is located at <sdk_root>/mcu/config/chip/<board>/chip.mk and defines the common settings, compiler configuration, include path and middleware module path of the chip. The major functions of the chip.mk are described below:

1) Configures the common settings of the chip.

2) Defines the `CFLAGS` macro.

3) Sets the include path of the kernel and the driver header file.

4) Sets the module folder path that contains the `Makefile`.

The `chip.mk` includes the `CFLAGS`, including the paths and module folder paths, as shown below.

```
...
MTK_SYSLOG_VERSION_2                    ?= y
MTK_SYSLOG_SUB_FEATURE_STRING_LOG_SUPPORT = y
```

```
MTK_SYSLOG_SUB_FEATURE_BINARY_LOG_SUPPORT = y
MTK_SYSLOG_SUB_FEATURE_USB_ACTIVE_MODE = y
MTK_SYSLOG_SUB_FEATURE_OFFLINE_DUMP_ACTIVE_MODE = y
MTK_CPU_NUMBER_0                        ?= y
FPGA_ENV                                ?= n
...

AR      = $(BINPATH)/arm-none-eabi-ar
CC      = $(BINPATH)/arm-none-eabi-gcc
CXX     = $(BINPATH)/arm-none-eabi-g++
OBJCOPY = $(BINPATH)/arm-none-eabi-objcopy
SIZE    = $(BINPATH)/arm-none-eabi-size
OBJDUMP = $(BINPATH)/arm-none-eabi-objdump

...

COM_CFLAGS += $(ALLFLAGS) $(FPUFLAGS) -ffunction-sections -fdata-
sections -fno-builtin -Wimplicit-function-declaration
COM_CFLAGS += -gdwarf-2 -Os -Wall -fno-strict-aliasing -fno-common
COM_CFLAGS += -Wall -Wimplicit-function-declaration -
Werror=uninitialized -Wno-error=maybe-uninitialized -Werror=return-type
COM_CFLAGS += -DPCFG_OS=2 -D_REENT_SMALL -Wno-error -Wno-switch
COM_CFLAGS += -DPRODUCT_VERSION=$(PRODUCT_VERSION)
COM_CFLAGS += -D$(TARGET)_BOOTING

...

#Incldue Path
COM_CFLAGS += -I$(SOURCE_DIR)/middleware/third_party/mbedtls/include
COM_CFLAGS += -I$(SOURCE_DIR)/middleware/third_party/mbedtls/configs

CFLAGS     += -std=gnu99 $(COM_CFLAGS)
CXXFLAGS   += -std=c++11 $(COM_CFLAGS)

...
```

6. Adding a Module to the Middleware

This section provides details on adding a module or a custom defined feature into an existing project. Added module will be compiled, archived and linked with other libraries to create the final image file during the project build. The following example shows how to add a module named `mymodule` into `earbuds_ref_design` project on the AB158x EVK development board.

6.1. Files to add

6.2. Source and header files

Create a module folder with module name under `<sdk_root>/mcu/middleware/third_party/` folder to place the module files. Module source and header files should be placed under the `"src"` and the `"inc"` folders, respectively, as shown in Figure 3.

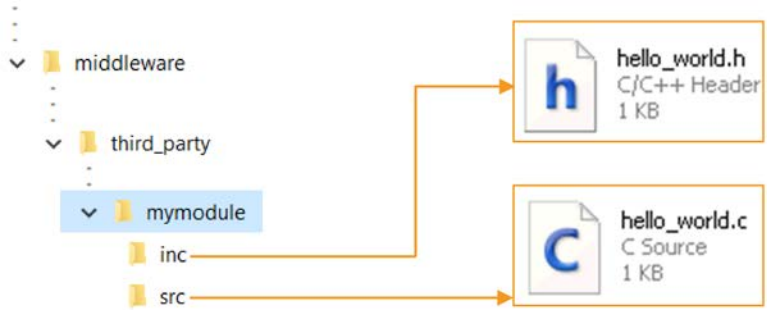

Figure 3. Place module source and header files under the module folder

The sample source code `hello_world.c` and header file `hello_world.h` with their corresponding locations are shown below:

`<sdk_root>/mcu/middleware/third_party/mymodule/src/hello_world.c`

```
#include "hello_world.h"

void myFunc(void)
{
    printf("%s", "hello world\n");
}
```

`<sdk_root>/mcu/middleware/third_party/mymodule/inc/hello_world.h`

```
#ifndef __HELLO_WORLD__
#define __HELLO_WORLD__

#include <stdio.h>

void myFunc(void);

#endif
```

6.3. Makefiles for the module

Create a makefile under the mymodule folder (see Figure 4) named `module.mk`. It defines the module path, module sources that need to be compiled and include path for the compiler to search for the header files during compilation.

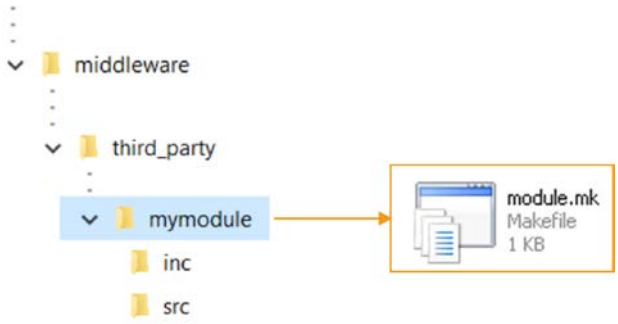

Figure 4. Create a module.mk under module folder

In this example, the `module.mk` is at `<sdk_root>/mcu/middleware/third_party/mymodule/module.mk`. `C_FILES` and `CFLAGS` are built-in variables that store the module's .c source files and include paths, respectively. The corresponding built-in variables to support compiling source files of a module (`.cpp`) are `CXX_FILES` and `CXXFLAGS`.

```
#module path
MYMODULE_SRC = middleware/third_party/mymodule

#source file
C_FILES  +=  $(MYMODULE_SRC)/src/hello_world.c
CXX_FILES+=

#include path
CFLAGS   += -I$(SOURCE_DIR)/middleware/third_party/mymodule/inc
CXXFLAGS +=
```

Besides **`module.mk`**, another makefile under `mymodule` folder named **`Makefile`** (`<sdk_root>/mcu/middleware/third_party/mymodule/Makefile`) is required to generate a module library, as shown in Figure 5.

Most of the dependency rules and definitions in the file are written for a common use. Simply copy the code below and modify the value of the variable `PROJ_PATH` and `TARGET_LIB`. The variable `PROJ_PATH` is the path to the project folder that contains the **`Makefile`** and the variable `TARGET_LIB` is the library for the added module.

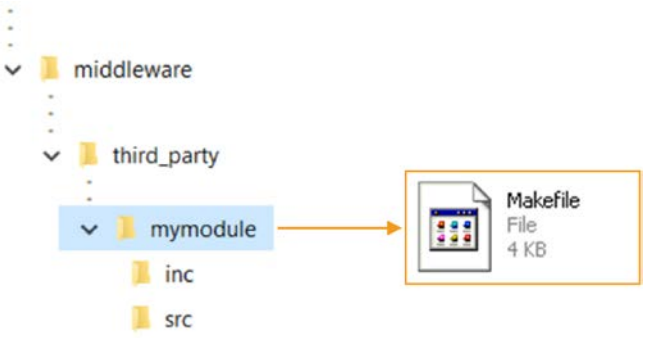

Figure 5. Create a Makefile under mymodule folder

```
SOURCE_DIR  = ../../..
BINPATH     = ~/gcc-arm-none-eabi/bin
PROJ_PATH   = ../../../project/ab158x_evk/apps/earbuds_ref_design/GCC
CONFIG_PATH ?= .

CFLAGS += -I$(PROJ_PATH)/../inc
CFLAGS += -I$(SOURCE_DIR)/$(CONFIG_PATH)


FEATURE    ?= feature.mk
include $(PROJ_PATH)/$(FEATURE)

# Global Config
-include $(SOURCE_DIR)/.config
# IC Config
-include $(SOURCE_DIR)/config/chip/$(IC_CONFIG)/chip.mk
# Board Config
-include $(SOURCE_DIR)/config/board/$(BOARD_CONFIG)/board.mk

# Project name
TARGET_LIB=libmymodule

BUILD_DIR = Build
OUTPATH   = Build

# Sources
include module.mk

C_OBJS   = $(C_FILES:%.c=$(BUILD_DIR)/%.o)
  CXX_OBJS = $(CXX_FILES:%.cpp=$(BUILD_DIR)/%.o)

.PHONY: $(TARGET_LIB).a

all: $(TARGET_LIB).a
    @echo Build $< Done

include $(SOURCE_DIR)/.rule.mk

clean:
    rm -rf $(OUTPATH)/$(TARGET_LIB).a
    rm -rf $(BUILD_DIR)
```

6.4. Adding a module to the build flow of the project

The rules to compile module sources to a single library are now complete and the module is ready to build. To add the module into the project's build flow, modify the `Makefile` under the project folder. In this example, it is at `<sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC/Makefile`.

In the `Makefile`, there's a section with rules defined as `include XXXX/module.mk`. This section defines the modules required by the project when creating an image file. Add a new line into the section to include the module in the project.

Include your module's `module.mk` path in the `Makefile`, as shown below.

```
…
################################################################################
######
#
# SDK source files
#
################################################################################
######
#include cjson
include $(SOURCE_DIR)/middleware/third_party/cjson/module.mk

#include xml
include $(SOURCE_DIR)/middleware/third_party/xml/module.mk
#include mymodule
include $(SOURCE_DIR)/middleware/third_party/mymodule/module.mk
…
```

After the module is successfully built, the object and the dependency files of the added module can be found under `<sdk_root>/mcu/out/<board>/<project>/obj/middleware/third_party/mymodule` folder. In this example the file path is `<sdk_root>/mcu/out/<board>/<project>/obj/middleware/third_party/mymodule/hello_world.o` and `<sdk_root>/mcu/out/<board>/<project>/obj/middleware/third_party/mymodule/hello_world.d`.

You've successfully added a module to an existing project. The next section describes how to create a project.

7. Creating a Project

This section provides details on how to use an existing project and create your own project named `my_project` on AB158x EVK using MCU earbuds_`ref_design` project as a reference.

7.1. Using an existing project

Apply an existing project as a reference design for your own project development.

Copy the folder `<sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design` to a new directory `<sdk_root>/mcu/project/ab158x_evk/apps/` and rename `earbuds_ref_design` to the new project name `my_project`.

7.2. Removing a module

The copied project has modules that could be removed in order to have a clean start for your project development. After the previous steps, a project with the same features has been created. It can be built to generate image file as the original project.

To remove a module:

1) Open the project `Makefile` from

   `<sdk_root>/mcu/project/ab158x_evk/apps/my_project/GCC/Makefile.`

2) Locate the module include list of the project and remove any unwanted module by removing or commenting out the corresponding include statement.

```
############################################################
...
# Bluetooth module
include $(SOURCE_DIR)/middleware/airoha/bluetooth/module.mk

# BT callback manager
include $(SOURCE_DIR)/middleware/airoha/bt_callback_manager/module.mk

# BT connection manager
include $(SOURCE_DIR)/middleware/airoha/bt_connection_manager/module.mk
...
```

7.3. User-defined source and header files

User defined project source and header files should be put under the `src` and the `inc` folder respectively shown in Figure 6.

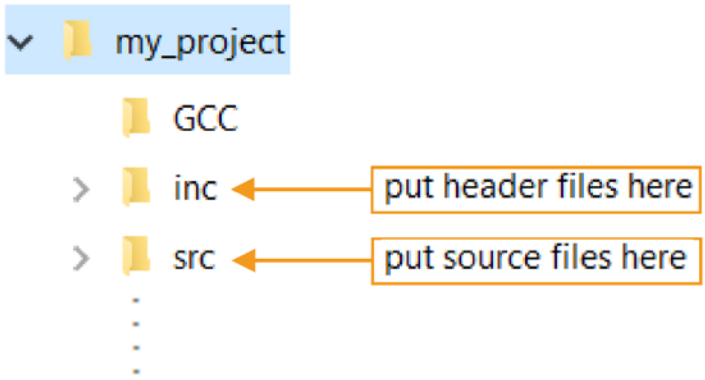

Figure 6. Project source and header files under the project folder

To compile the added source code, simply add the .c source files to variable "C_FILES" and the header search path to variable "CFLAGS" in the project Makefile, as shown below. The corresponding variables to support compiling the source files (.cpp) of the module are CXX_FILES and CXXFLAGS).

In current Makefile, there are two intermediate define “APP_FILES” and “SYS_FILES”. Both of them are added in C_FILES. This line “include $(SOURCE_DIR)/$(APP_PATH_SRC)/apps/module.mk” which is in Makefile includes the C files in folder <my_projet>/src/apps

<sdk_root>/mcu/project/ab158x_evk/apps/my_project/GCC/Makefile

```
...
APP_FILES       += $(APP_PATH_SRC)/main.c
APP_FILES       += $(APP_PATH)/GCC/syscalls.c
APP_FILES       += $(APP_PATH_SRC)/regions_init.c
...
SYS_FILES       += $(APP_PATH_SRC)/system_ab158x.c
...
CXX_FILES       += ...
...
C_FILES         += $(APP_FILES) $(SYS_FILES)
...
```

7.4. Test and verify

After the MCU project is successfully built, the final image file can be found under `<sdk_root>/mcu/out/<board>/<project>/` folder. In this example, it's `<sdk_root>/mcu/out/ab158x_evk/my_project/`.
The object and the dependency files of your project can be found under `<sdk_root>/mcu/out/<board>/<project>/obj/project/<board>/apps/<project>/src/` folder. In this example it's `<sdk_root>/mcu/out/ab158x_evk/my_project/obj/project/ab158x_evk/apps/my_project/src/`.
The location of the image file, object and dependency files after the example project is built, as shown in Figure 7.

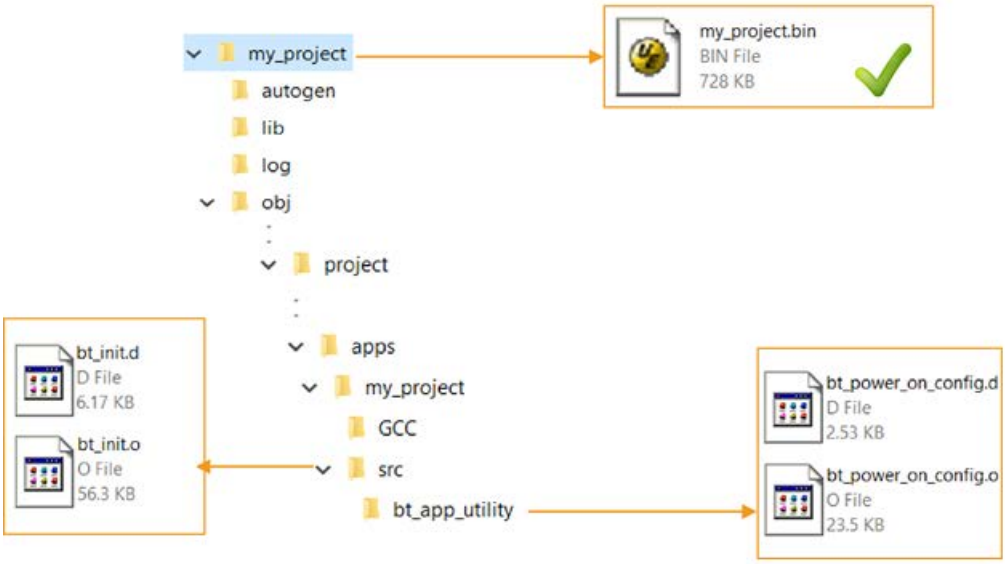

Figure 7. Output image file, object files and dependency files after project is successfully built

7.5. Troubleshooting

When a build process is failed as shown below, the error messages are written to the `err.log` file under `<sdk_root>/out/<board>/<project>/log/` folder. Please see the `err.log` file for more information.

```
…
TOTAL BUILD: FAIL
```

7.6. Configuration for multi-processor products

Some Airoha products, such as AB1585, may have embedded digital signal processors as well as MCU processor. Those processors have independent project files and build process. We integrated the build processes of the processors into one build.sh. Users need to configure the build mapping relation of each processor in advance to let the build.sh work correctly.

This section describes adding a new build target for multi-processor projects:

- 7.7 Creating a new build target in mapping_proj.cfg

Note: The multi-processors build script for AB158x series is in `<sdk_root>/build.sh`. It can build both the MCU and DSP projects.

7.7. Creating a new build target in mapping_proj.cfg

To create a new build target, please edit the config file at `<sdk_root>/mcu/tools/scripts/build/co_build_script/mapping_proj.cfg`.

Add a new shell index array at the end of mapping_proj.cfg. The name of the array must be map**__<target _board>__<target_project>**. The <target_board> and <target_project> should be the expected first and second parameters while executing build.sh. The value of the array elements define the mapping relation of physical board, project and feature mapping of each processor. The array element definition is as follows:

```
0: borad_folder - folder name of board folder (under <sdk_root>/<processor>/project/)

1: MCU_project_folder – folder name of MCU project (under
<sdk_root>/MCU/project/<board_folder>/)

2: dsp0_project_folder – folder name of dep0 project (under
<sdk_root>/dsp0/project/<board_folder>/)

3: dsp1_project_folder – folder name of dep1 project (under
<sdk_root>/dsp1/project/<board_folder>/)

4: MCU_project_feature_mk – Make file name of feautre definition for MCU project

5: dsp0_project_feature_mk – Make file name of feautre definition for dsp0 project

6: dsp1_project_feature_mk – Make file name of feautre definition for dsp1 project
```

Example:

```
  map__ab1585_evk__earbuds_ref_design=( \
  [0]="ab158x_evk" \
  [1]="earbuds_ref_design" \
  [2]="dsp0_headset_ref_design" \
  [4]="feature_ab1585_evk.mk" \
  [5]="feature_ab1585_evk.mk" \
)
```

To build the newly added target project, run the following command.

```
  cd <sdk_root>
  ./build.sh <target_board> <target_project>
```

For example, run the following command under `<sdk_root>`:

```
  ./build.sh my_board my_project
```

This command causes the following command to run in both the mcu and dsp part.

For the mcu part:

```
  cd <sdk_root>/mcu/
```

```
./build.sh ab158x earbuds_ref_design –f=feature_ab1585_evk.mk
```

For the dsp part:

```
cd <sdk_root>/dsp/
./build.sh ab158x dsp0_headset_ref_design
-f=feature_ab1585_evk.mk
```