Airoha IoT SDK for BT Audio Build Environment Guide

Version: 1.17

Release date: 23 March 2023

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	Added a method for adding a module; Added 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 November 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 November 2019	Updated the description of the installation process.
1.8	13 December 2019	Updated examples and related description for AB155x.
1.9	16 June 2020	Updated the 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 an unused description.
1.13	21 January 2022	Added support for AB1585/AB1588.
1.14	23 February 2022	Updated the examples and description for AB158x.
1.15	10 March 2022	Added feature option improvements in Section 5 “Makefiles”.
1.16	28 December 2022	Refined language and formatting.
1.17	23 March 2023	Added support for AB1571/AB1577.

Table of contents

Lists of tables and figures

1. Overview

This guide provides information about the tools and utilities for setting up the build environment and running your projects with the Airoha Internet of Things (IoT) Software Development Kit (SDK).

It provides information about the subsequent items:

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

This information can be applied to the Airoha IoT Development Platform, including the AB155x, AB1565, AB1568, AB1571, AB1577, AB1585, and AB1588 evaluation kits (EVK).

Note: In this guide, all of the supported chips use the same installation procedure and build method as AB158x. Some of the subsequent examples use AB155x as a reference.

2. Environment

This section provides detailed information about setting up the SDK build environment with the default GNU's Not Unix (GNU) Compiler Collection (GCC) on Linux and Microsoft Windows operating systems and using the Minimal System 2 (MSYS2) cross-compilation tool.

2.1. Installing the SDK build environment on Linux

This section provides a guide to getting started with the Airoha IoT SDK for Bluetooth audio in a Linux environment. It provides information about the subsequent items:

- Preparing the Linux build environment
- Starting the Cadence toolchain
- Installing the SDK build environment

2.2. Preparing the Linux build environment

The toolchain provided with the SDK is required to set up the build environment on Linux operating systems (i.e. Ubuntu 18.10 64bit or Ubuntu 18.04 Long-term support (LTS)/20.04 LTS/22.04 LTS).

Table 1. Recommended Build Environment

Item	Description
OS	Linux OS 18.10 or 18.04 LTS/20.04 LTS/22.04 LTS
Make	GNU make 3.81

To install the SDK and build environment on Linux:

1) Download *IoT_SDK_For_BT_Audio_and_Linux_Tools_All_In_One* from MediaTek On-Line (MOL). The package contains the Advanced RISC Machine (ARM) GCC toolchain, Cadence Digital Signal Processor (DSP) toolchain for Linux OS, and Airoha IoT SDK for BT Audio. The package also contains an install script to set up the environment.

Create a folder to be the working folder.

2) Extract the contents of `IoT_SDK_For_BT_Audio_and_Linux_Tools_All_In_One` to the working folder. The folder name must be only alphanumeric characters (i.e. A-Z, 1-0).

3) Open `install.sh` in the working folder and make the subsequent changes:

a) Select an unused network port for the Cadence compiler license-service daemon. We strongly recommend using a port number between 1024 and 9999.

b) Change the value of DSP_LICSERV_PORT. You must change this setting if you have Linux host sharing with multiple users.

```
#!/bin/bash

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

Execute ./install.sh. The installation process requires Linux root permission and an internet connection. The script performs the following functions.

a) install the Cadence license tool chain;

b) get the Cadence toolchain license; and

c) extract the Airoha IoT SDK for BT Audio.

```
./install.sh
```

The subsequent message appears when the installation process is complete:

```
...
...
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
```

Build your project. You must wait before starting your first build because the Cadence license service daemon takes a few minutes to start when the installation process is complete. Run the subsequent commands to build you project.

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

The subsequent message appears when the build process is complete.

```
$./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 installation process puts the toolchain under ~/airoha_sdk_toolchain. Do not remove or modify ~/airoha_sdk_toolchain because it breaks the build environment.

NOTE: You do not need to run the install script again for a new SDK release; Just get the SDK standalone package (i.e. `IoT_SDK_For_BT_Audio`) and extract the contents to use the new SDK package.

2.3. Starting the Cadence toolchain license service daemon

The installation process starts the Cadence license service daemon. You must restart the server after the system reboots. You can use the subsequent command under the installation working folder to restart the server. The Cadence licenser service daemon takes approximately two minutes to start the service after launch.

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

2.4. Offline installation

The installation process requires an internet connection to get the Cadence license. You must manually install the license if you are installing offline.

To manually install the license offline:

1) Submit a JIRA request to get a floating license and put the license in same folder of install.sh.
2) Modify the `install.sh`.
3) Modify the DSP_LIC_FILE value in your license file.
4) Go to the Airoha eService website and go to `Project > Your Project > Customer channels > Customer portal > Software Related > License Request`.
5) Submit an **Airoha eService Cadence License Request** to get `<license name>.lic`.
6) Remove CADENCE_LIC_HOST=xxxxx from the license file as shown below.

```
#!/bin/bash

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

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

8) Run the `install.sh` to install the SDK offline.

This video shows the online and offline installation process. Refer to `AB155x_Linux_Dev_Env_Setup.mp4` for more information. The installation process for this build environment is the same.

2.5. Installing the SDK build environment on Microsoft Windows

To install the SDK and build environment on Windows, you must download and extract the contents of `IoT_SDK_For_BT_Audio_and_Windows_Tools_All_In_One`. The package contains MSYS, ARM GCC for Windows, the Cadence DSP toolchain for Windows, and Airoha IoT SDK for BT Audio.

The subsequent sections show the instructions for installing the SDK and build environment.

2.6. Preparing the build environment

To prepare a build environment:

1) Select a folder as an installation folder. The folder name must not contain special characters.
2) Extract the contents of the `IoT_SDK_For_BT_Audio_and_Windows_Tools_All_In_One` file to the folder.
3) Run the `install_msys.bat` file to install msys2. You can skip this step if you have already installed msys. This step requires an internet connection. Refer to the MSYS2 page here for information about performing an offline installation.

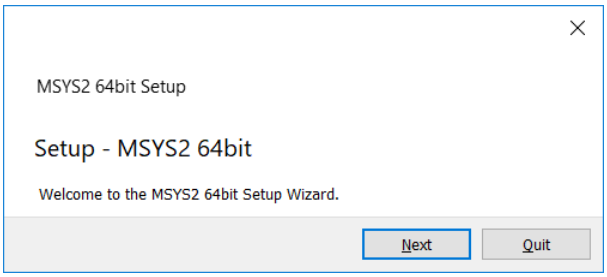

Figure 1. Installing MSYS2

4) Execute msys2.exe. An msys2 terminal window appears on the screen.

5) Key ./install.sh into the msys2 terminal. The installation process requires an internet connection. The script performs the following actions:

   a) Install the required MSYS software components (i.e. diffutils, make, p7zip., and tar).

Install the Cadence license tool chain.

6) Get the Cadence toolchain license.

7) Extract the contents of `Airoha IoT SDK for BT Audio`.

The following output appears in the msys2 terminal when the installation process is complete:

```
$ ./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- ....
```

Build your first project.

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

The following output appears in the msys2 terminal when the build process is complete:

```
$./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 installation process puts the toolchain under `~/airoha_sdk_toolchain`.

NOTES:

Do not remove or modify `~/airoha_sdk_toolchain` because it breaks the build environment.

You do not need to run the install script again for a new SDK release; Just get the SDK standalone package `IoT_SDK_For_BT_Audio` and extract the contents to use the new SDK package.

2.7. Offline installation

The installation requires an internet connection to get the Cadence license.

If you are installing the build environment offline, you must:

1) Manually install the license;
2) Submit a JIRA request to get a node-locked license; and
3) Modify `install.sh`

Change the `DSP_LIC_FILE` value to be the same as your license file. You can get the `<license name>.lic` from the Airoha eService Cadence License Request. Go to the Airoha eService website and go to `Project > Your Project > Customer channels > Customer portal > Software Related > License Request`.

4) Remove the line `CADENCE_LIC_HOST=lic.airoha.com.tw` as shown below.

```
#!/bin/bash

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

5) Run `install.sh` to install the SDK offline.

This video (i.e. `AB155x_Windows_Dev_Env_Setup.mp4`) shows the online and offline installation process. The installation process for your build environment is the same.

2.8. Troubleshooting

The subsequent information is important for building your project with MSYS2.

- The folder name and file name in the Airoha IoT SDK must not contain ‘ ’, ‚’(’, ’)’, ‘[‘ or ‘]’ characters.
- The sum of the path length SDK installed folder and project name should less than 35 characters in length. Otherwise, a build error similar to the one shown below may occur.

```
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
```

- You must not use any platform dependent commands or files in the project makefile (e.g., `stat` or `/proc/cpuinfo`).
- The parallel build feature is enabled in `build.sh` by default to make the compilation process faster. Disable the parallel build feature if there are any irregular build error or system exceptions.
- To disable the parallel build feature for all modules, change the -j value of the `EXTRA_VAR` variable 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 the parallel build of a specific module causes build errors, set the value `<module>_EXTRA` to `-j1` in `<SDK_ROOT>/mcu/.rule.mk` to disable the parallel build for the specific 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

Use <sdk_root>/build.sh to build Microcontroller Unit (MCU) and DSP projects. For more information about the script, go to the SDK root directory and execute the following command:

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

The following output appears on the screen:

```
============================================================
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
```

Execute the subsequent command to get a list of all of the available boards and projects.

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

The output appears as follows.

```
============================================================
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. Building the project.

Execute the subsequent command to build a specific project:

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

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

NOTE: Use the **–mcu** option to skip the DSP build.

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

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

The standard output appears 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 generated files are added to the <sdk_root>/out/ab158x_evk/earbuds_ref_design/ folder.

3.2. Cleaning the output folder

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

Execute the following command to clean the <sdk_root>/out folder:

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

Execute the following command to clean the <sdk_root>/out/<board> folder:

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

Execute the following command to clean the <sdk_root>/out/<board>/<projecct> folder:

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

The output folder is defined by the BUILD_DIR variable 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)))
```

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

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

The pre-built bootloader image file is copied to the <sdk_root>/mcu/out/<board>/<project>/ folder by default when the project is built. The primary purpose of the bootloader image is to download the Flash Tool.

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

```
./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 generated image file of the project, the bootloader, and the merged image file flash.bin are put under <sdk_root>/mcu/out/ab158x_evk/earbuds_ref_design.

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

To build the project:

1) Change the directory to the project source directory where the SDK is located.

   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.

2) Go to the example project's location. For the MCU project earbuds_ref_design:

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

For DSP project, go to dsp0_headset_ref_design:

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

3) Execute the make command.

```
Make
```

   The MCU project output folder is defined under variable BUILD_DIR in the makefile at <sdk_root>/mcu/project/ab158x_evk/apps/earbuds_ref_design/GCC as shown below.

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

   The 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 at <sdk_root>/dsp/project/ab158x_evk/apps/dsp0_headset_ref_design/XT-XCC as shown below.

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

The 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 directory of the SDK package is organized into the folder structure shown in Figure 2.

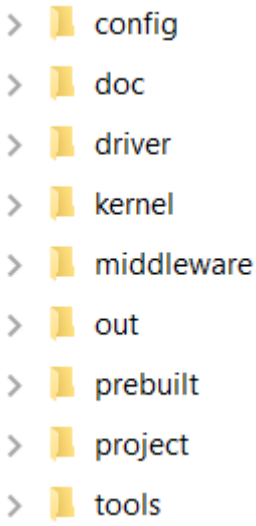

Figure 2. SDK package folder structure

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 generated files are put under the corresponding `<sdk_root>/out/<board>/<project>/` folder. The dependency of mcu/dsp side projects can be used to build the bin in the `<sdk_root>/out/<board>/<project>/` folder. Refer to Section *7.6* for more information.

The target folder path for AB158x is the `<sdk_root>/out/ab158x_evk/earbuds_ref_design/` folder. The following list provides a brief description of the binary files:

- `project image` – the naming rule of the image file is `PROJ_NAME.bin`, e.g., `earbuds_ref_design.bin`. The variable `PROJ_NAME` is defined in the makefile (refer to Section *5* for more information.
- `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 the build log which shows build information, timestamps, and error messages.
- `obj folder` – contains object and dependency files.

5. Makefiles

You can only change the feature options in feature*.mk and the project Makefile under `<sdk_root>/mcu/project/<chip>/apps/<project_name>/GCC` folder in the SDK build process. Do not make changes to the other .mk files because it may cause the build process to break.

Category	File naming	Description	Location	Attrribute
Feature	feaure_*.mk	feature config for project	<sdk_root>/<mcu or dsp>/project/<chip>/apps/<project_name>/<GCC or XT-XCC>	user configuration
Chip	chip.mk	chip rule	<sdk_root>/<mcu or dsp>/config/chip/<chip>	read only
Board	module.mk	board setting	<sdk_root>/<mcu or dsp>/config/chip	read only
Module	module.mk	module makefile	under each module folder	read only
Project makefile	Makefile	main makefile for project build	<sdk_root>/<mcu or dsp>/project/<chip>/apps/<project_name>/<GCC or XT-XCC>	user configuration

We have made the following improvements to the AB158x series chipset:

- 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.
- Added comments to describe the usage and notes about the feature options in `feature_*.mk` to make it easier for you to understand.
- Added a dependency check.
- Sets the dependency rules for the build flow of the project to check feature option settings.

You can use the earbuds_ref_design project as a reference for more information about the usage and relationship of each makefile.

5.1. Project Makefile

The project makefile is under `<sdk_root>/mcu/project/<chip>/apps/<project_name>/GCC/`. The purpose of the project makefile is as follows:

- Configure project settings, including the root directory, project name, project path, and more.
- Include other makefiles for the configuration (i.e. `feature.mk`, `chip.mk` and `module.mk`).
- Set the file path of the project's source code.
- Set the include path of the project's header files.
- Set 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 (i.e. the error message tag `[Conflict feature option]`).

- For example, environment detection (ED) and Active noise cancellation (ANC) are dependent. If you want to enable ED but forget to enable the ANC feature, the following message tells you to enable ANC: `[Conflict feature option] To enable AIR_ANC_ENVIRONMENT_DETECTION_ENABLE must support AIR_ANC_ENABLE.`

o Add a make rule to check for add-on feature options (i.e. the error message tag is `[Addon feature option fail]`). If this occurs, contact Airoha (CPM) team to request the add-on package.

- Set the module libraries to link when creating the image file.
- Trigger a command for each module to generate a module library.

5.2. Configuration makefiles

This section provides more information about the configuration makefiles, `feature.mk` and `chip.mk`.

The `feature_*.mk` file is in the `<sdk_root>/mcu/project/<chip>/apps/earbuds_ref_design/GCC/` folder. You can set a value for a specific feature or turn it on or off by making a change in the `feature_*.mk`. The comments provide more information about using the feature. The template for comments 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` variables are also defined in `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` makefile is at `<sdk_root>/mcu/config/chip/<board>/chip.mk` and defines the common settings, compiler configuration, the path and middleware module path of the chip. The major functions of `chip.mk` are as follows:

- Configure the common settings of the chip.
- Define the CFLAGS macro.
- Set the include path of the kernel and the driver header file.
- Set the module folder path that contains the makefile.

The `chip.mk` makefile 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 detailed information about adding a module or a custom defined feature to an existing project. The added module is compiled, archived, and linked with other libraries to generate the final image file during the project build. The following example shows how to add the `mymodule` module to the `earbuds_ref_design` project on the AB158x EVK development board.

6.1. Files to add

[TEXT PLACEHOLDER]

6.2. Source and header files

Create a module folder with the module name under `<sdk_root>/mcu/middleware/third_party/` folder. This folder will contain the module files. The module source and header files must be put under the `src` and `inc` folders as shown in Figure 3.

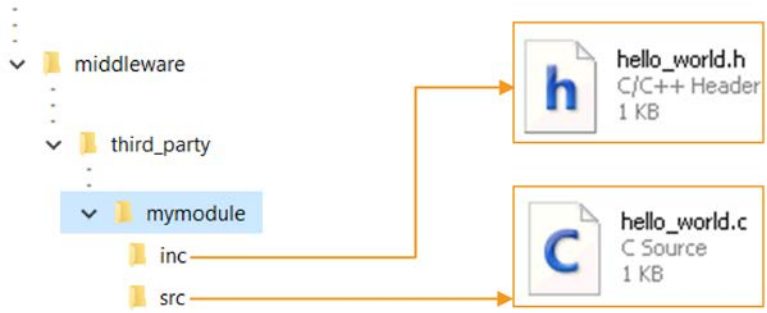

Figure 3. Module source and header files under the module folder

The sample source code `hello_world.c` and header file `hello_world.h` and their 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 named `module.mk` (as shown in Figure 4). The makefile defines the module path and module sources that must be compiled. It must include the path that the compiler uses to find the header files during compilation.

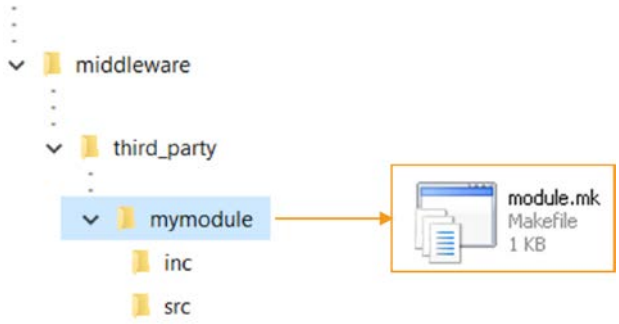

Figure 4. module.mk under the module folder

In this example, `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 the paths. The corresponding built-in variables `CXX_FILES` and `CXXFLAGS` support compiling the source files (e.g., .cpp) of the module.

```
#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 +=
```

In addition to the module.mk makefile, another makefile under the `mymodule` folder named `Makefile` (i.e. `<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 makefile are written for a common use. You can copy the subsequent code and make any necessary changes to the values of the `PROJ_PATH` and `TARGET_LIB` variables. The `PROJ_PATH` variable is the path to the project folder that contains the makefile. The `TARGET_LIB` variable is the library for the added module.

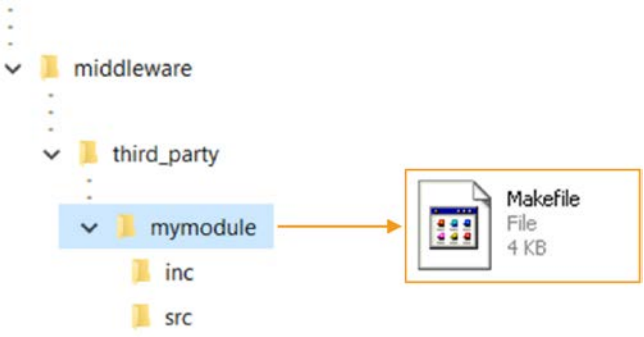

Figure 5. Creating a makefile under the mymodule folder

```
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 for compiling 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`.

There is a section in the XXXX/`module.mk` makefile that defines the rules. This section also defines the modules required by the project when generating 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
…
```

When the module is successfully built, the object and the dependency files of the added module are 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 have now successfully added a module to an existing project. The subsequent section shows how to create a project.

7. Creating a Project

This section provides detailed information about creating your own project from an existing project. The following example shows how to create a new project named `my_project` on AB158x EVK using MCU earbuds_`ref_design` project as a reference.

7.1. Using an existing project

To use an existing project as a reference design for your own project development:

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

7.2. Removing a module

When you use an existing project to create a new project, the new project keeps the same features. You can remove the modules to have a clean start for your project development.

To remove a module:

1) Open the project `makefile` at `<sdk_root>/mcu/project/ab158x_evk/apps/my_project/GCC/Makefile`.
2) Locate the module include list of the project and remove any unwanted modules by removing or commenting out the corresponding statements. For example…

```
############################################################
...
# 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 must be put under the `src` and the `inc` folder as shown in Figure 6.

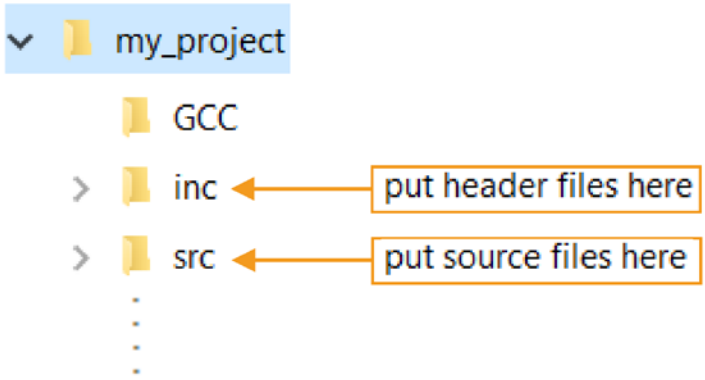

Figure 6. Project source and header files under the project folder

To compile the added source code, add the .c source files to the C_FILES variable and the header search path to the CFLAGS variable in the project makefile as shown below. The CXX_FILES and CXXFLAGS variables provide support for compiling the module source files (i.e. `.cpp`).

In the current makefile, there are two intermediate definitions, APP_FILES and SYS_FILES. Both of them are added in `C_FILES`. The line `include $(SOURCE_DIR)/$(APP_PATH_SRC)/apps/module.mk` in the 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. Testing and verification

When the MCU project is successfully built, the final image file is in the `<sdk_root>/mcu/out/<board>/<project>/` folder. In this example, it is `<sdk_root>/mcu/out/ab158x_evk/my_project/`.

The object and the dependency files of your project are under `<sdk_root>/mcu/out/<board>/<project>/obj/project/<chip>/apps/<project>/src/` folder. In this example, they are under `<sdk_root>/mcu/out/ab158x_evk/my_project/obj/project/ab158x_evk/apps/my_project/src/`.

Figure 7 shows the location of the image file, object and dependency files when the example project is built.

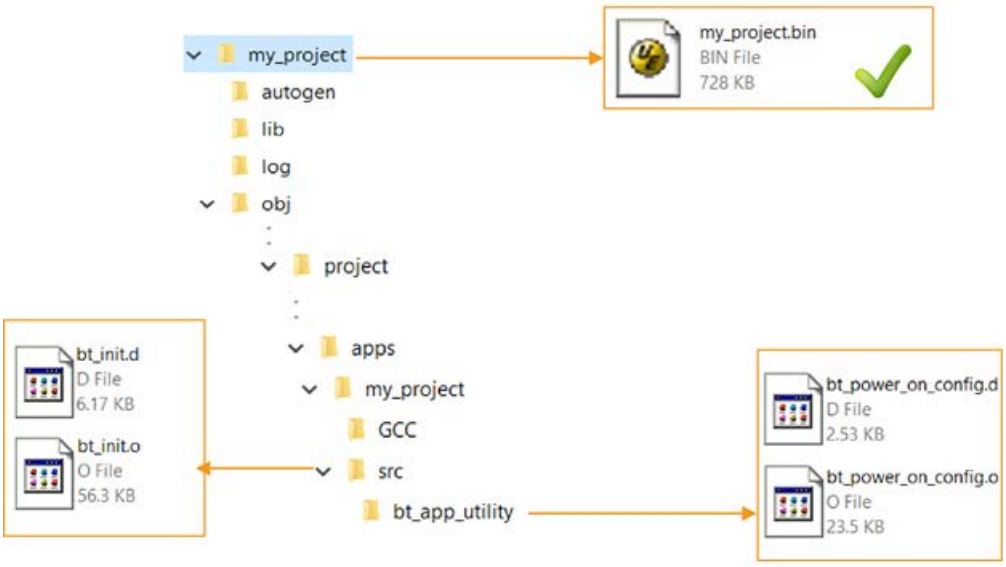

Figure 7. Location of the generated files when a project is built

7.5. Troubleshooting

When a build process fails (as shown below), the error messages are written to the `err.log` file under `<sdk_root>/out/<board>/<project>/log/` folder. Refer to the `err.log` file for more information.

```
…
TOTAL BUILD: FAIL
```

7.6. Configuring multi-processor products

Some Airoha products, such as AB1585, may have embedded digital signal processors in addition to an MCU processor. These processors have independent project files and build processes. We have integrated the build processes for these processors into one `build.sh`. You must first configure the build-mapping relation of each processor to make the build.sh operate as expected.

Section *7.7* shows how to add a new build target for multi-processor projects.

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, you must modify the config file at `<sdk_root>/mcu/tools/scripts/build/co_build_script/mapping_proj.cfg`.

Add a new shell index array to the end of `mapping_proj.cfg`. The name of the array must be `map__<target_board>__<target_project>`. The <target_board> and <target_project> must be the expected first and second parameters when executing the `build.sh` file. The value of the array elements define the mapping relation of the physical board project and feature mapping of each processor. The array element definition is as follows:

- `0: board_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` – Makefile name of feature definition for an MCU project.
- `5: dsp0_project_feature_mk` – Makefile name of feature definition for a dsp0 project.
- `6: dsp1_project_feature_mk` – Makefile name of feature definition for a dsp1 project.

For 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" \
)
```

Execute the following command to build the newly added target project:

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

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

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

This command causes the subsequent command to execute in the MCU and DSP parts.

For example, in the MCU part:

```
cd <sdk_root>/mcu/
./build.sh ab158x earbuds_ref_design –f=feature_ab1585_evk.mk
```

And in the the DSP part:

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