Airoha IoT SDK for BT Audio Get Started Guide

Version: 1.5

Release date: 3 December 2020

Document Revision History

Revision	Date	Description
1.0	1 October 2019	Initial release Note. Revised from Airoha_IoT_SDK_Get_Started_Guide v4.15
1.1	11 November 2019	• Fixed a typo • Rename reference document
1.2	16 June 2020	• Revise supported Linux versions
1.3	23 June 2020	• Added support for AB1565/AB1568
1.4	16 September 2020	• Revise project folder name
1.5	3 December 2020	• Update Figure 2. Architecture layout of AB1565/AB1568 platform

Table of Contents

Lists of Tables and Figures

1. Overview

Airoha IoT SDK for BT Audio provides the software and tools for your application development on the EVK. The SDK includes drivers for hardware abstraction layer, peripherals, connectivity, such as Bluetooth/Bluetooth Low Energy and other third party features. It also provides battery management, Firmware update Over-The-Air (FOTA) and FreeRTOS.

This get started guide provides quick steps on how to use the SDK and its supported features on GCC environment.

1.1. Architecture of the platform

The three-layer architecture of the platform including **BSP**, **Middleware** and **Application** with underlying components is shown in Figure 1.

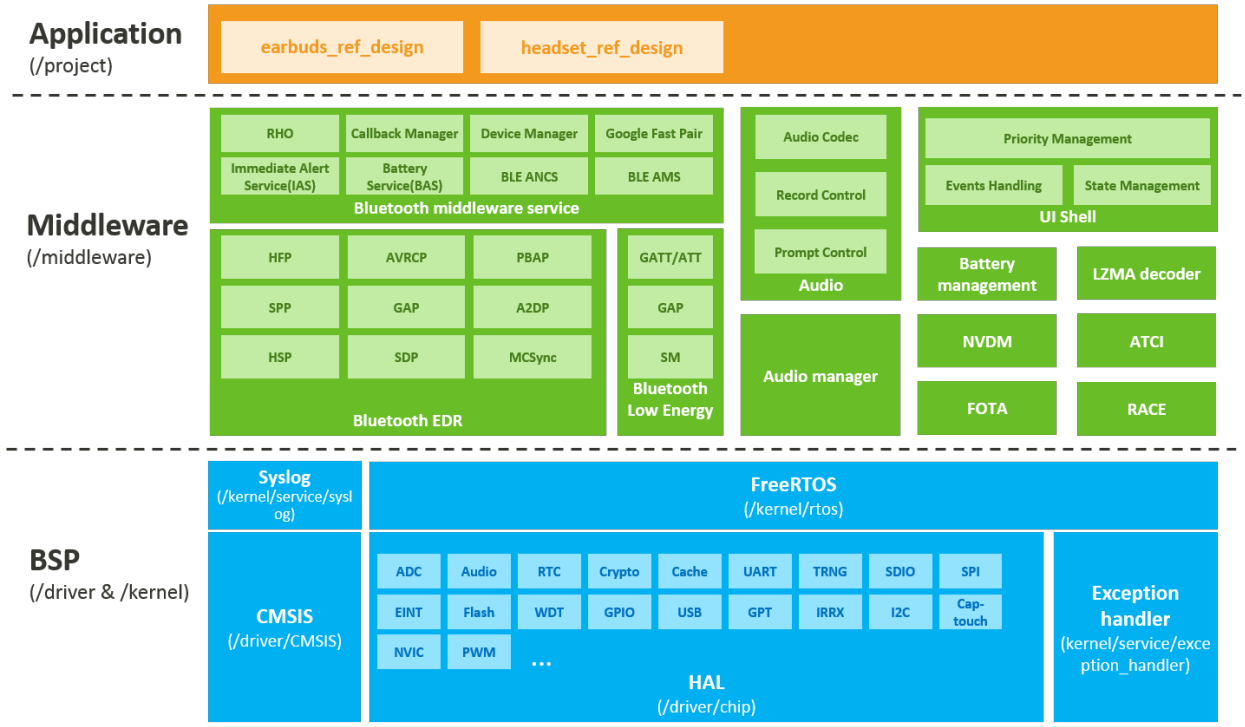

Figure 1. Architecture layout of the AB155x platform

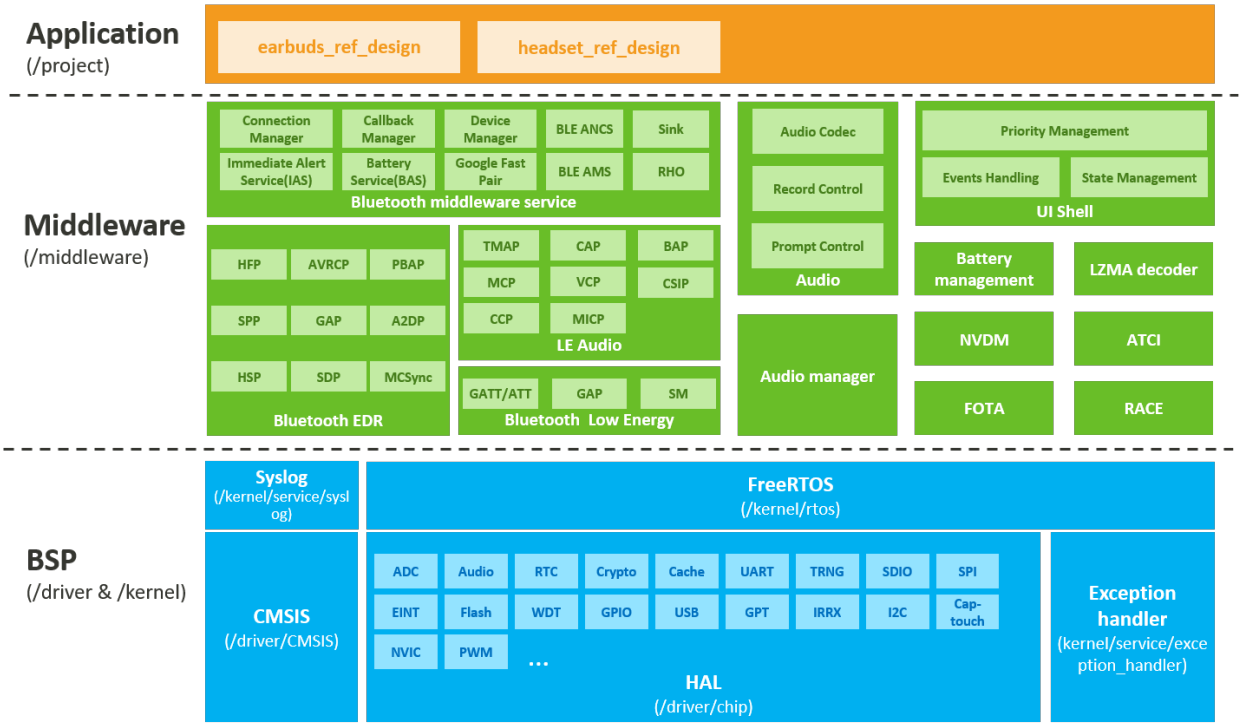

Figure 2. Architecture layout of AB1565/AB1568 platform

A brief description of the layers is provided below:

- **BSP**
  - Hardware drivers. Provide peripheral drivers for the platform, such as ADC, I2S, I2C, SPI, RTC, GPIO, UART, Flash, Security Engine, TRNG, GDMA, PWM, WDT and IRDA TX/RX.
  - Hardware Abstraction Layer (HAL). Provides the driver Application Programming Interface (API) encapsulating the low-level functions of peripheral drivers for the operating system (OS), **Middleware** features and **Application**.
  - The hardware components located at `<sdk_root>\mcu\driver\board\component` are used by the EVK (`<sdk_root>\mcu\driver\board\xxx_evk`). For example, the eint key driver is located under `<sdk_root>\mcu\driver\board\component\eint_key` folder and is available when you select the `<chip>` but the GPIO pins need to be configured at `<sdk_root>\mcu\driver\board\<chip>\eint_key` and also the source files should be included under the component folder.
  - FreeRTOS. An OS with the open source software for **Middleware** components and **Application**.
  - Syslog. This module implements system logging for development and debugging.
- **Middleware**
  - Bluetooth EDR/Bluetooth Low Energy. Provides stack and protocol-layer access profiles for data transferring and management control, such as Generic Access Profile (GAP), Serial Port Profile (SPP), Generic Attribute Profile (GATT) and Security Manager (SM).
  - Bluetooth service. This module is the Bluetooth common services implementation.
  - Audio. This module is for audio middleware implementation.
  - Audio manager. This module is the Audio Manager control implementation including all main audio behavior management and most of the control for DSP.

  - FOTA. Provides a mechanism to update the firmware.
  - Battery management. Provides the charging flow control and precise information on battery.
  - File system. Provides APIs to control data storage and retrieval in a file system.
  - UI Shell. It is the UI framework which helps application developers to design and implement applications. Please refer to Airoha_IoT_SDK_UI_Framework_Developers_Guide under <sdk_root>/mcu/doc folder to get more information.
  - Other features. Non-Volatile Data Management (NVDM), Race command, LZMA decoder and other features that are dependent on **HAL** and **FreeRTOS**. The Airoha IoT SDK also supports AT command interface (ATCI) as an advanced feature.

- **Application**
  - Pre-configured projects using **Middleware** components, such as earbuds_ref_design and headset_ref_design.

The application layer enables running the projects that are based on **Middleware**, **FreeRTOS** and **HAL** layers. These layers provide rich features for application development, such as **Middleware** provides the Bluetooth features, Audio features, and the OS provides the underlying real-time operating system.

The supported HAL features on different chipsets are listed in Table 1.

To use the HAL features, enable the compile options of the corresponding modules.

1) Open the header file `hal_feature_config.h`, located under `inc` folder of each example project.
2) Edit and define the compile options as needed.
3) Include the corresponding module header files, located at `<sdk_root>\mcu\driver\chip\inc`, in the project source files.

Table 1. HAL features on different chipsets

Feature	Compile option
ADC	HAL_ADC_MODULE_ENABLED
AUDIO	HAL_AUDIO_MODULE_ENABLED
Cache	HAL_CACHE_MODULE_ENABLED
Crypto	HAL_AES_MODULE_ENABLED HAL_DES_MODULE_ENABLED HAL_MD5_MODULE_ENABLED HAL_SHA_MODULE_ENABLED
Clock	HAL_CLOCK_MODULE_ENABLED
CAP-TOUCH	HAL_CAPTOUCH_MODULE_ENABLED
EINT	HAL_EINT_MODULE_ENABLED
Flash	HAL_FLASH_MODULE_ENABLED
GDMA	HAL_GDMA_MODULE_ENABLED
GPIO	HAL_GPIO_MODULE_ENABLED
GPT	HAL_GPT_MODULE_ENABLED
I2C Master	HAL_I2C_MASTER_MODULE_ENABLED
NVIC	HAL_NVIC_MODULE_ENABLED
PWM	HAL_PWM_MODULE_ENABLED
RTC	HAL_RTC_MODULE_ENABLED
SPI Master	HAL_SPI_MASTER_MODULE_ENABLED
SPI Slave	HAL_SPI_SLAVE_MODULE_ENABLED
SD	HAL_SD_MODULE_ENABLED
Sleep Manager	HAL_SLEEP_MANAGER_MODULE_ENABLED
TRNG	HAL_TRNG_MODULE_ENABLED
UART	HAL_UART_MODULE_ENABLED
USB	HAL_USB_MODULE_ENABLED
WDT	HAL_WDT_MODULE_ENABLED

The supported middleware features on different chipsets are listed in Table 2. There is a `readme.txt` under the root directory of each middleware module. It contains the information about the module dependency, feature options, notes and brief introduction. To learn more about the usage of the middleware modules, refer to the `readme.txt` file under each module path.

Table 2. Middleware features on different chipsets

Feature	Module path
Bluetooth stack	middleware\MTK\bluetooth
Bluetooth profile	middleware\MTK\bluetooth
Bluetooth Low Energy stack	middleware\MTK\bluetooth
Bluetooth Low Energy profile	middleware\MTK\bluetooth
Bluetooth AWS MCE	middleware\MTK\bluetooth

Feature	Module path
Bluetooth AWS report	middleware\MTK\bt_aws_mce_report
Bluetooth RHO	middleware\MTK\bt_role_handover
Blueooth sink service	middleware\MTK\sink
Air pairing	middleware\MTK\sink
Fast pairing	middleware\MTK\bt_fast_pair
TLS	middleware\third_party\mbedtls
NVDM	middleware\MTK\nvdm
Audio	middleware\MTK\audio
ANC	middleware\MTK\audio
Pass through	middleware\MTK\audio
mp3 codec	middleware\MTK\audio
prompt control	middleware\MTK\audio
record control	middleware\MTK\audio
PEQ	middleware\MTK\audio_manager
Audio management	middleware\MTK\audio_manager
ATCI	middleware\MTK\atci
Race Command	middleware\MTK\race
read-only file system	middleware\MTK\rofs
File system	middleware\third_party\fatfs
Battery management	middleware\MTK\battery_management
FOTA	middleware\MTK\fota
LZMA decoder	middleware\third_party\lzma_decoder
UI framework	middleware\third_party\ui_shell

Note, the file system does not work without SD/eMMC.

1.2. Supported key components

The platform offers rich connectivity options, such as Bluetooth, peripheral drivers and other advanced components. This section introduces each of these components.

1.2.1. Bluetooth and Bluetooth Low Energy

Bluetooth with Enhanced Data Rate (EDR) and Bluetooth Low Energy (LE) are key features in the Airoha IoT SDK. The details are listed in Table 3. The SDK API and module descriptions can be found in the API reference guide and Bluetooth developer's guides at `<sdk_root>\mcu\doc`. In addition, find more details on how to include the Bluetooth module in `<sdk_root>\mcu\middleware\MTK\bluetooth\readme.txt`.

Table 3. Bluetooth/Bluetooth Low Energy features

Item	Features
EDR-A2DP	• Advanced Audio Distribution Profile
EDR-AVRCP	• Audio/Video Remote Control Profile (CT:v1.3/TG:v1.0)
EDR-HFP/HSP	• Hands-Free Profile v1.7 or Headset Profile
EDR-PBAP	• Phone Book Access Profile (PBAP) o Defines the procedures and protocols to exchange Phonebook objects between devices.
EDR-SPP	• Serial Port Profile
EDR-GAP	• Generic Access Profile
BLE-GAP	• Generic Access Profile
BLE-GATT/ATT	• Generic Attribute Profile
BLE-SMP	• Low Energy Security Manager Protocol
Multipoint Support	• Supports multipoint Bluetooth access in EDR. o Two HFP (HF) o Two A2DP (Sink) o Two AVRCP (CT) o Two SPP server/client • Supports multipoint Bluetooth access in Bluetooth Low Energy. o Four Bluetooth Low Energy links.

1.2.2. FOTA

The detailed list of FOTA features is provided in Table 4. The API and module descriptions can be found in Airoha IoT SDK API Reference Manual and Airoha IoT SDK Firmware Update Developer's Guide under `<sdk_root>\mcu\doc`. In addition, find more information on how to include this module in `<sdk_root>\mcu\middleware\MTK\fota\readme.txt`.

Table 4. FOTA features

Item	Features
FOTA	• Full binary update mechanism

Item	Features
	• Package data compression • Integrity check • Power loss protection • FOTA packaging tool

1.2.3. Peripheral drivers

The detailed list of peripheral drivers is provided in Table 5. The APIs for the drivers can be found in the Airoha IoT SDK API Reference Manual under `<sdk_root>\mcu\doc`. To include HAL module, include `<sdk_root>\mcu\driver\chip\<chip>\module.mk` in project makefile for the EVK.

Table 5. Supported peripheral drivers

Item	Features
ADC	• ADC module.
CACHE	• The maximum size of the cache is 32kB.
EINT	• External interrupt controller. • Processes the interrupt request from an external source or a peripheral device.
Flash	• Supports execute in place (XIP) and programming flash by software. • Default 4MB system in package (SiP) flash on the AB155x and AB1565/AB1568EVK, and AB1568 have 8MB SiP flash. • Supports external flash up to 16MB on Airoha IoT SDK platform HDKs
GPIO	• GPIO mode (in or out) • Set Pull Up/Down for GPIO IN mode
GPT	• General Purpose Timer. • Supports 32kHz and 1MHz clock sources, repeat and one-shot modes for timing events and delays in µs or ms.
PWM	• Range is 256 duty cycles • 32kHz, 2MHz, XTAL clock for PWM frequency reference
UART	• Three full set (TX/RX) UART support on the EVK • Baud rate of up to 3000000
I2C Master	• Two I2C interfaces • Supports 50/100/200/400kHz transmission rate
I2S Master	• I2S master is capable of servicing an external codec component. • Supports 8/11.025/12/16/22.05/24/32/44.1/48 kHz audio sampling rates in stereo mode.
ISINK	• Current sink

Item	Features
	• Adjustable backlight current
MPU	• Memory Protection Unit
IrDA	• RX (NEC, RC5, RC6, SIRC,RCMM)
WDT	• Supports hardware, software watchdog • Supports system reset
I2S-Slave	• Supports sample rates: 8,12, 16, 24, 32, 48 kbits • Supports mono and stereo mode
SPI-Master	• Serial Peripheral Interface
RTC	• Real-Time Clock
GDMA	• General Purpose DMA
Security	• SHA1, SHA2 (256, 384, 512), AES
TRNG	• Truly Random Number Generator • Generates 32bit random number
Charger	• Supports single-cell Li-Ion battery charging
Cap-touch	• Cap-touch key detect • Supports Cap-touch for key event

1.2.4. Battery management

The battery management features are listed in Table 6. The battery management APIs are found in API Reference Manual under `<sdk_root>\mcu\doc`. Find more information on how to include this module in `<sdk_root>\mcu\middleware\MTK\battery_management\readme.txt`.

Table 6. Battery management features

Item	Features
Battery management	• Charging flow control mechanism. • Algorithm for battery capacity measurement. • Precise information on the battery, including temperature and battery level.

1.2.5. Advanced features and components

The advanced features and components included in the platform are listed in Table 7.

Table 7. Advanced features and components

Item	Features
ATCI	• AT command parser
File system	• Windows compatible • Platform independent • Very small footprint for code and work area • Multiple volumes • Multiple ANSI/OEM code pages including DBCS

Item	Features
	• Long file name support in ANSI/OEM or Unicode • FreeRTOS support for multitasking • Multiple sector size support up to 4kB • Read-only, minimized API, I/O buffer and more

1.3. Folder structure

The SDK is delivered as a single package organized in a folder structure, as shown in Figure 3.

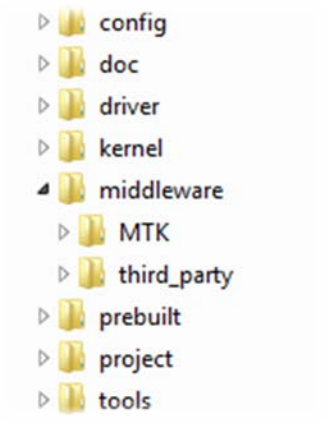

Figure 3. Folder structure

This package contains the source and library files of the major components, build configuration, related tools and documentation. A brief description on 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.
- `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.

The main components that belong to middleware are in the `middleware` folder:

- MTK
  - atci. Provides the interface for a target communication using AT commands though UART.
  - audio. This module is for audio middleware implementation.
  - audio_fft. This module is the Fast Fourier Transform (FFT) implementation for analyzing data.
  - audio_manager. This module is the Audio Manager control implementation including all main audio behavior management and most of the control for DSP.

- battery_management. Includes battery monitor, charging flow control and battery capacity algorithms.
- ble_ancs. This module is the ble_ancs middleware implementation. Apple Notification Center Service (ANCS) is a way to access the notifications generated on iOS devices by a Bluetooth low-energy link. It can be connected with iOS devices only.
- ble_bas. This module is the Battery Service (BAS) implementation. BAS exposes the Battery State and Battery Level to the peer device by a Bluetooth low-energy link. This module supports Battery Level notification to the peer device and manages Battery Level related read events from the peer device.
- ble_dis. This module is the Device Information Service (DIS) implementation. DIS exposes manafacturer and/or vendor information about a device.a cntrol point to allow a peer device to cause the local device to immediately alert by a Bluetooth low-energy link. This module manages all user's registered callbacks and notifies all users when an Alert Level related write event occurs from the peer device.
- ble_ias. This module is the Immediate Alert Service (IAS) implementation. IAS exposes a cntrol point to allow a peer device to cause the local device to immediately alert by a Bluetooth low-energy link. This module manages all user's registered callbacks and notifies all users when an Alert Level related write event occurs from the peer device.
- bluetooth. Bluetooth/Bluetooth Low Energy provides three profiles (GAP, GATT, SM) to discover and connect Bluetooth devices and to transfer and control data securely through a Bluetooth connection.bluetooth_service. This module is the Bluetooth common services implementation. It includes 2 services: one is Device Manager Service, which manages bonded peer device's security information; the other is GATT Server Service, which supports GAP service and characteristics configuration feature.
- bluetooth_service. This module is the Bluetooth common services implementation. It includes 2 services: one is Device Manager Service, which manages bonded peer device's security information; the other is GATT Server Service, which supports GAP service and characteristics configuration feature.
- bt_air. This module is the BT AIR Service implementation. It includes 3 modules (BLE AIR, SPP AIR, AirUpdate). It can help the AIR application communicate with peer device by low-energy link or Bluetooth SPP profile or AirUpdate Fixed channel. This module manages all user's registered callbacks and notifies all users when a RX data event occurs from the peer device.
- bt_aws_mce_report. This module is to manage all Bluetooth aws mce report users. User can register and deregister their callback functions to complete sending or receiving app report info.
- bt_callback_manager. This module is to manage the callback function of the Bluetooth stack. User can register and deregister the EDR/BLE callback function to handle different callback functions.
- bt_connection_manager. This module is to manage all Bluetooth role handover users. User can register and deregister their callback functions to complete the role handover procedure.
- bt_fast_pair. This module is used to provide google fast pair 2.0 feature. User can call API or send action provide by this module to fast pair 2.0 feature.
- bt_role_handover. This module is to manage all Bluetooth role handover users. User can register and deregister their callback functions to complete the role handover procedure.
- fota. FOTA provides firmware update functionality.
- key. Airo key is a common upper layer for different types of keys, including captouch_key, eint_key, gsensor key and powerkey. This module has a common interface and common event type.
- mfi_coprocessor. This module is for middleware MFI coprocessor porting layer implementation.
- module_log. This module is the module_log implementation.

- nvdm. NVDM is a type of memory mechanism that retains its contents when the system power is turned off.
- port_service. This module is the port service implementation. It bases on different serial device ports and offers user with unified interface to use.
- race_cmd. This module is the Race command interface.
- rofs. This module is the ROFS (read-only file system) interface.
- serial_nand. This module is the flash disk management implementation specialized for the serial NAND flash device.
- serial_nor. This module is for access the serial NOR Flash by SPI interface.
- sink. This module is the sink service which integrates HFP, A2DP, AVRCP and PBAPC profiles. It works as a Bluetooth headset and supports the headset features, such as, answering or rejecting incoming call, getting contact name of the incoming call, playing and pausing music, moving to previous song and next song, and connection reestablishing when power on or link lost.
- smart_charger. This module is the smart charger case interface.
- ui_shell. It is the UI framework which helps application developers to design and implement applications. Please refer to Airoha_IoT_SDK_UI_Framework_Developers_Guide under <sdk_root>/mcu/doc folder to get more information.
- usb. This module is the USB class interface.

- third_party
  - `mbedtls`. Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are cryptographic protocols designed to provide communications security over a computer network. mbed TLS is an open source implementation for developers to include cryptographic and SSL/TLS capabilities in embedded products with a minimal coding footprint.
  - `fatfs`. FatFs is generic FAT file system for small embedded systems. It is used to control data storage and retrieval in a file system.
  - `lzma_decoder.`LZMA is the default and general compression method used to perform lossless data compression. LZMA is also suitable for embedded applications because it provides fast decompression and a high compression ratio.
  - `micro_ecc.`A small and fast ECDH and ECDSA implementation for 8-bit, 32-bit, and 64-bit processors.

1.4. Project source structure

The SDK provides a set of reference applications. For example, projects with a single function showing how to use drivers or other module features and others with complex functionality demonstrating how to use the middleware components.

Example applications are located in the `<sdk_root>\mcu\project\<evk>\apps` folder and they all have the same folder structure, as shown in Figure 4.

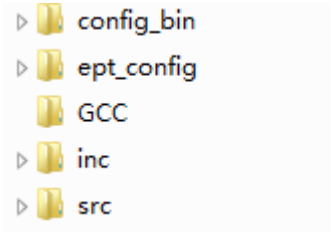

Figure 4. Project folder structure

1) `config_bin`. The filesystem.bin and the nvkey.xml.
2) `ept_config`. The *.ews files for setting GPIO of a board.
3) `GCC`. GCC related project configuration files, such as a makefile.
4) `inc`. Project header files.
5) `src`. Project source files.
6) `Readme.txt`. A brief introduction about project behavior and the required environment.

You can apply the relevant reference applications to further your development.

2. Getting Started Using Build Script

This section provides a guide to getting started with the Airoha IoT development platform for BT-Audio and covers the following items:

- Supported environments for development.
- Configuring the EVK.
- Building the project using the SDK.
- Downloading and running the project from Microsoft Windows.
- Debugging the project from Microsoft Windows.
- Creating your own project.

The chips of Airoha IoT SDK for BT Audio product line are shown below.

- Airoha IoT SDK for BT Audio: AB155x/AB1565/AB1568

2.1. Environment

The SDK can be used on any edition of Microsoft Windows 7 and 8, and on Linux (**Ubuntu 18.10 64-bit** or **Ubuntu 18.04 LTS**). Compilers for ARM CM4 (gcc) and Cadence's Tensilica DSP are required to build the project.

- Download and extract the content of the SDK all in one package on your local PC.
  - Linux: IoT_SDK_For_BT_Audio_and_Linux_Tools_All_In_One
  - Windows: IoT_SDK_For_BT_Audio_and_Windows_Tools_All_In_One
- Follow the instructions in the readme.txt file, run install script to install the SDK.

The install.sh will install required system component (e.g. make, libc-i386, …), unpack the sdk package and configure the toolchain. For most cases, the install.sh can fully cover the environment setup process. However, if you want to customize your environment, you can refer to Airoha_IoT_SDK_for_BT_Audio_Build_Environment_Guide under <sdk_root>/mcu/doc folder and Airoha_IoT_SDK_DSP_Get_Started_Guide under <sdk_root>/dsp/doc folder. The documents contain the detail environment setup steps for manually install.

Please notice that the installation process needs an internet connection to get the license of Cadence's Tensilica tool chain from the Airoha server. When the installation is complete, the SDK can work offline. For cases that cannot connect to the internet during installation, we also provide an offline installation. Please watch the environment setup training video on Airoha eService documents for details.

2.2. Building the project using the SDK

2.2.1. Building AB155x series CM4 and DSP projects

Build CM4 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 ab155x_evk earbuds_ref_design
       ./build.sh ab155x_evk earbuds_ref_design -fm=feature_ab1552_evk.mk
       ./build.sh clean
        (clean folder: out)
       ./build.sh ab155x_evk clean
        (clean folder: out/ab155x_evk)
       ./build.sh ab155x_evk earbuds_ref_design clean
        (clean folder: out/ab155x_evk/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:
============================================================
ab1552_asia
  earbuds_ref_design
      board_share: ab155x_evk
      cm4: earbuds_ref_design
      fd0: feature_ab1552_asia.mk
      dsp0: dsp0_headset_ref_design
      fm: feature_ab1552_asia.mk
ab1552_evk
  earbuds_ref_design
```

```
      board_share: ab155x_evk
      cm4: earbuds_ref_design
      fd0: feature_ab1552_evk.mk
      dsp0: dsp0_headset_ref_design
      fm: feature_ab1552_evk.mk
...
```

- 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 AB155x EVK, run the following build command:

```
./build.sh ab155x_evk earbuds_ref_design
```

The standard output in the terminal window is as follows:

```
$./build.sh ab155x_evk earbuds_ref_design
cd /usr/home/user_name/SDK_vn.n.n/dsp
============================================================
Start DSP0 Build
============================================================
FEATURE = feature.mk
make -C project/ab155x_evk/apps/dsp0_headset_ref_design/XT-XCC OUTDIR=/
usr/home/user_name/SDK_vn.n.n/dsp/out/ab155x_evk/dsp0_headset_ref_design
2>>
/usr/home/user_name/SDK_vn.n.n/dsp/out/ab155x_evk/dsp0_headset_ref_design/
log/err.log
make: Entering directory
`/usr/home/user_name/SDK_vn.n.n/dsp/project/ab155x_evk/apps/dsp0_headset_r
ef_design/XT-XCC'
  ...
```

The output files are then put in the <sdk_root>/out/ab155x_evk/earbuds_ref_design/ folder.

- 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/ab155x_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/ab155x_evk/apps/earbuds_ref_design/GCC/Build.

2.3. Developing on AB155x EVK

2.3.1. Configuring the AB155x EVK

The AB155x EVK has two separate major boards. One board is the EVK main board. The other board is an adaptor board for each chipset. There are two kinds of adaptors available: AB1558/6 and AB1555. The front view of the AB155x EVK with the AB1555 adaptor board is shown in Figure 5.

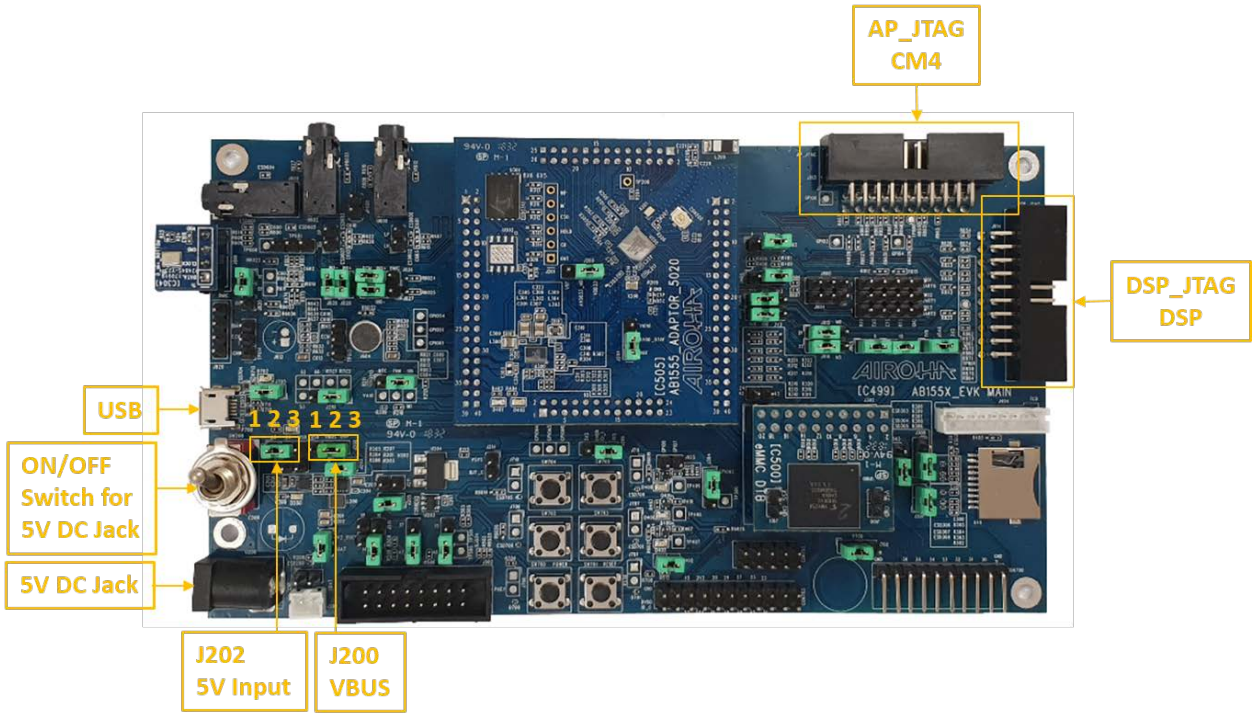

Figure 5. Front view of the AB155x EVK

The EVK can be powered by either USB VBUS or the 5V DC jack. Set the jumpers J202 2-3 on for USB VBUS (5V_USB).

A micro-USB connector is reserved on the left-side of the EVK, this port can also be used as the firmware download port. Set jumpers J200 1-2 on, the VBUS power source is from USB VBUS.

AB155x has reserved two JTAG debugging interfaces, AP_JTAG is used for embedded CM4 and DSP_JTAG is used for the embedded DSP. You can directly connect JTAG JIG with these connectors for debugging.

2.3.2. Installing AB155x Flash Tool for AB155x EVK

IoT Flash Tool is a flexible device flashing tool for application development on AB155x EVK.

To install the IoT Flash Tool:

- The IoT Flash Tool is located in <sdk_root>/mcu/tools/pc_tool/IOT_Flash_Tool
- The IoT Flash Tool is also available through MOL. Go here to download the IoT Flash Tool from the MediaTek MOL website.
    - Search for “IoT_Flash_Tool” in “Tool Name” to find the latest version of IoT Flash Tool.

The tool is a setup free package. You can start the Flash Tool by clicking the `FlashTool.exe` inside the folder.

2.3.3. Installing the AB155x EVK drivers on Microsoft Windows

This section describes how to install AB155x EVK USB drivers on PCs running Microsoft Windows. Complete the following procedure to install them.

To install the MediaTek USB Port driver AB155x **USB** port on the EVK on Windows 7 or other Windows:

1) Install the MediaTek USB Port driver from `MS_USB_ComPort_Driver/v3.16.46.1` folder located in AB155x _FlashTool folder.
2) Execute `InstallDriver.exe` to install the driver.
3) Use a USB cable to connect the AB155x **USB** connector on the AB155x EVK to your computer's USB port.

2.3.4. Flashing the image to AB155x EVK

Before using the IoT Flash Tool, it is necessary to use a pre-built project file (.cfg) or build your own project to get one (see section 2.2, "Building the project using the SDK").

To download the firmware to the target device, use the **AB155x USB** interface:

1) Power off the target (you must disconnect the USB cable).
2) Launch IoT Flash Tool, and click **Download** on the left panel of the main GUI.
3) Select **USB** from the **COM Port** drop down menu. If you do not have the adapter or battery, click the **Enable Download without Battery** option.

Click **Open** to provide the configuration file, which is usually named as `flash_download.cfg` and is generated after build process. If it loads successfully, **Download Information** is displayed, including **Name**, **Length** and **File Path** of the firmware binary.

4) Click **Start** to start downloading.
5) Connect the USB cable to power on the HDK through the **AB155x USB** connector. The process automatically starts.

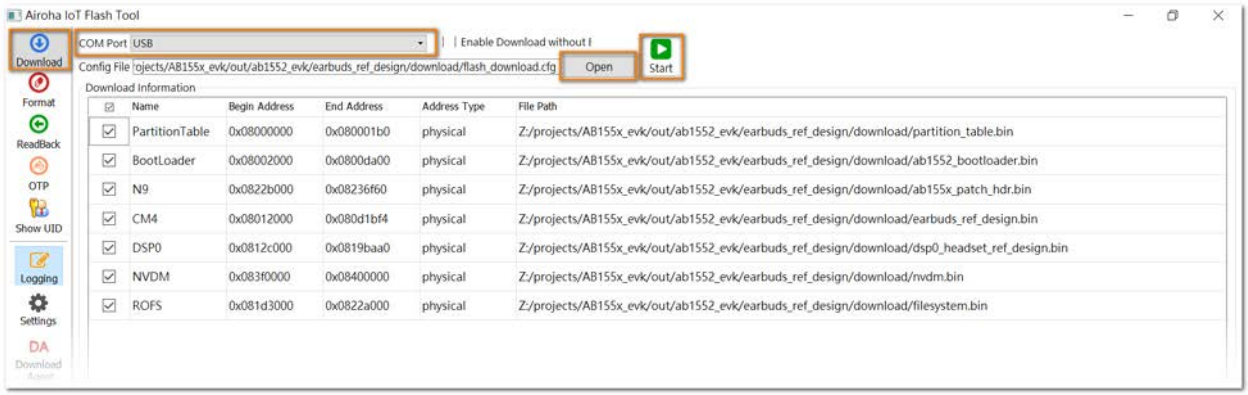

Figure 6. Download the firmware to a target device using USB connection

2.3.5. Running the project on AB155x EVK

You must complete the following procedure to use Logging tool:

1) Launch Logging tool.
2) Click "Serial port" as shown in Figure 7.

3) Select log binary file as shown in Figure 8
4) Set the Serial port & Baud Rate is shown in Figure 9.
5) Click “Start” as shown in Figure 10.
6) Reset the target. The log is shown in the Log window.

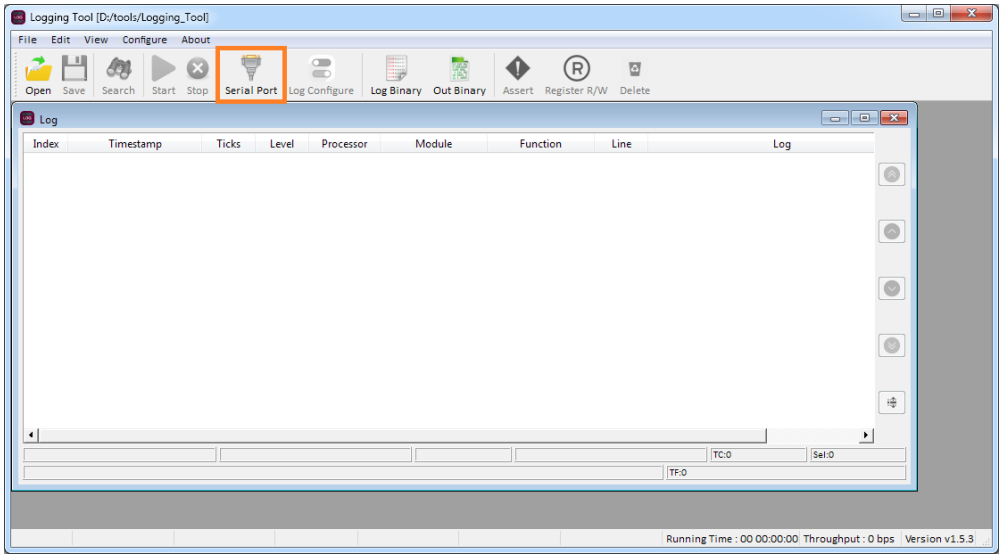

Figure 7. Serial Port Button

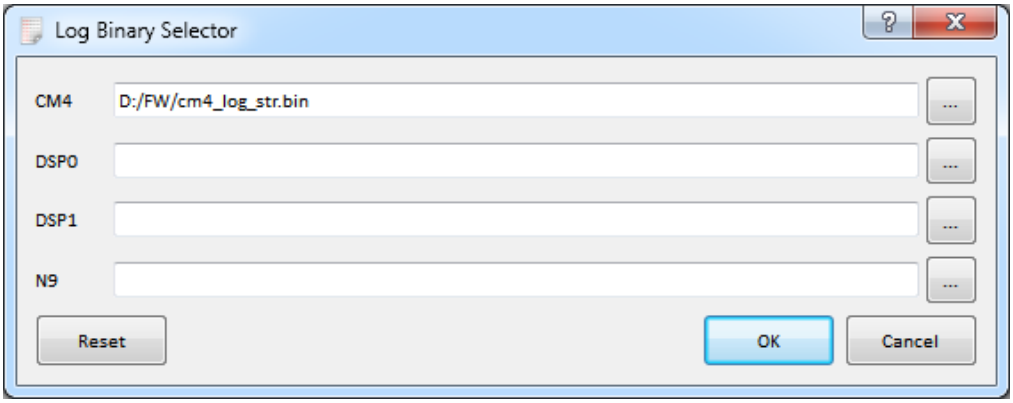

Figure 8. Select Log Binary

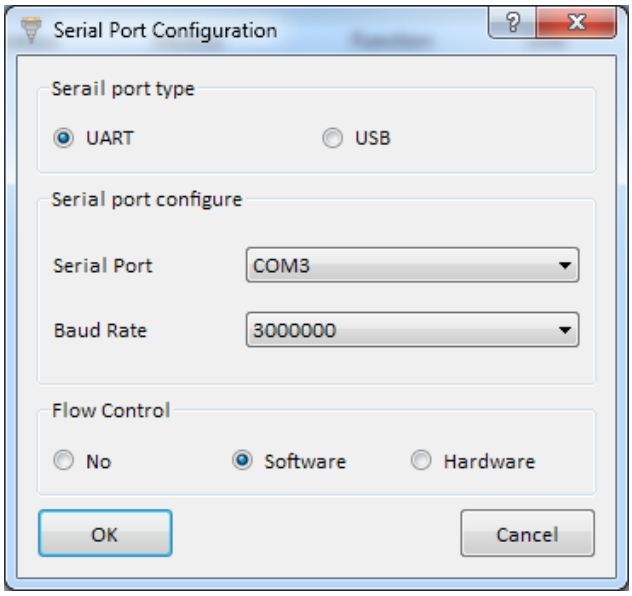

Figure 9. Serial Port Configuration Dialog

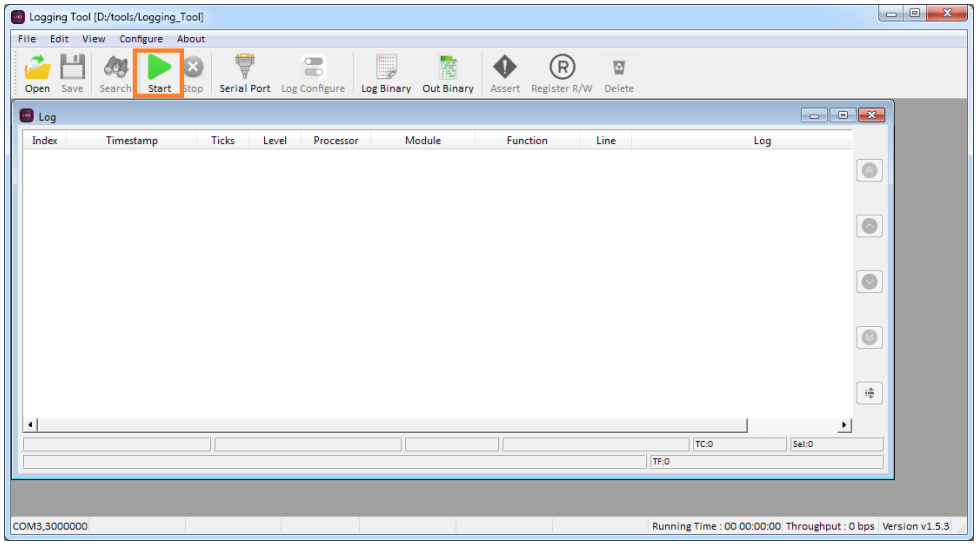

Figure 10. Start Button

2.3.6. Debugging with the AB155x EVK from Microsoft Windows

This section shows how to debug a project built with the GCC compiler using openOCD debugger tool.

Before commencing project debugging, install the supporting software on Windows OS.

1) Download openocd-0.10.0 from here and unzip it into the `<openocd_root>` folder.
   - Download the GCC toolchain for your specific version of Windows from here, and unzip it into the `<gcc_root>` folder.
2) Install the mbed serial port driver, if the mbed serial port driver is not installed (see section 2.3.3, “Installing the AB155x EVK drivers on Microsoft Windows”).
3) Create a board configuration file named `ab155x.cfg` and copy the following content to the file:

```
puts "Load AB155x configuration"
```

```
#source [find interface/cmsis-dap.cfg]
source [find interface/jlink.cfg]
transport select swd
source [find target/swj-dp.tcl]

set _CHIPNAME AB155x
set _TARGETNAME $_CHIPNAME.CM4
set _CPUTAPID 0x3ba02477

swj_newdap $_CHIPNAME DAP -irlen 4 -expected-id $_CPUTAPID
target create $_TARGETNAME cortex_m -chain-position $_CHIPNAME.DAP

adapter_khz 1000
reset_config srst_only

$_TARGETNAME configure -event gdb-attach {
    global _TARGETNAME
      targets $_TARGETNAME
      halt
}

$_TARGETNAME configure -event gdb-detach {
    global _TARGETNAME
      targets $_TARGETNAME
      resume
}

puts "AB155x configuration done"
```

4) Put the board configuration file under `<openocd_root>\share\openocd\scripts\board.`

Start debugging with AB155x EVK:

5) Copy the project `.elf` file from project `Build` folder to GCC tool path, such as `<gcc_root>` (for example, `<sdk_root>\out\ab1552_evk\ earbuds_ref_design\debug\ earbuds_ref_design.elf`.)
6) Open the command window for openOCD.
7) Change the directory in the command window to the openOCD tool folder, such as `<openocd_root>\bin`.
8) Disconnect the micro-USB cable from the board to completely power off the board.
9) Use a simulator (such as J-Link) to connect to JTAG pin (TMS, TCLK, VCC and GND) or SWD pin (SWDIO, SWCLK, VCC and GND) of AB155x.
10) Reconnect the micro-USB cable to the board to power-on the board.
11) Run the command to start the openOCD.

```
openocd.exe -s ..\share\openocd\scripts -f board\ab155x.cfg
```

12) Open the command window for GNU project debugger (GDB).
13) Change the directory in the command window to tool folder, such as `<gcc_root>\bin`.
14) Run the command to start the GDB.

```
arm-none-eabi-gdb.exe <gcc_root>\earbuds_ref_design.elf
(gdb) target remote localhost:3333
(gdb) monitor reset init
(gdb) load
```

```
(gdb) info registers
(gdb) x/10i $pc
```

You now have the openOCD debugging software running on your system.

Note: openOCD is a free third-party debugging tool (GPL license). To resolve any issues or perform troubleshooting, please refer to the openOCD official forum. In addition, openOCD debugging cannot work if the system goes into sleep mode. For more detail information, please refer to the document Airoha IoT SDK Power Mode Developers Guide under `<sdk_root>/mcu/doc`.

2.4. Developing on AB1565/AB1568 EVK

2.4.1. Configuring the AB1565/AB1568 EVK

The AB1565/1568 EVK has two separate major boards. One board is the EVK main board. The other board is an adaptor board for each chipset. There are two kinds of adaptors available: AB1565 and AB1568. The front view of the AB1565/AB1568 EVK with the AB1568 adaptor board is shown in Figure 5.

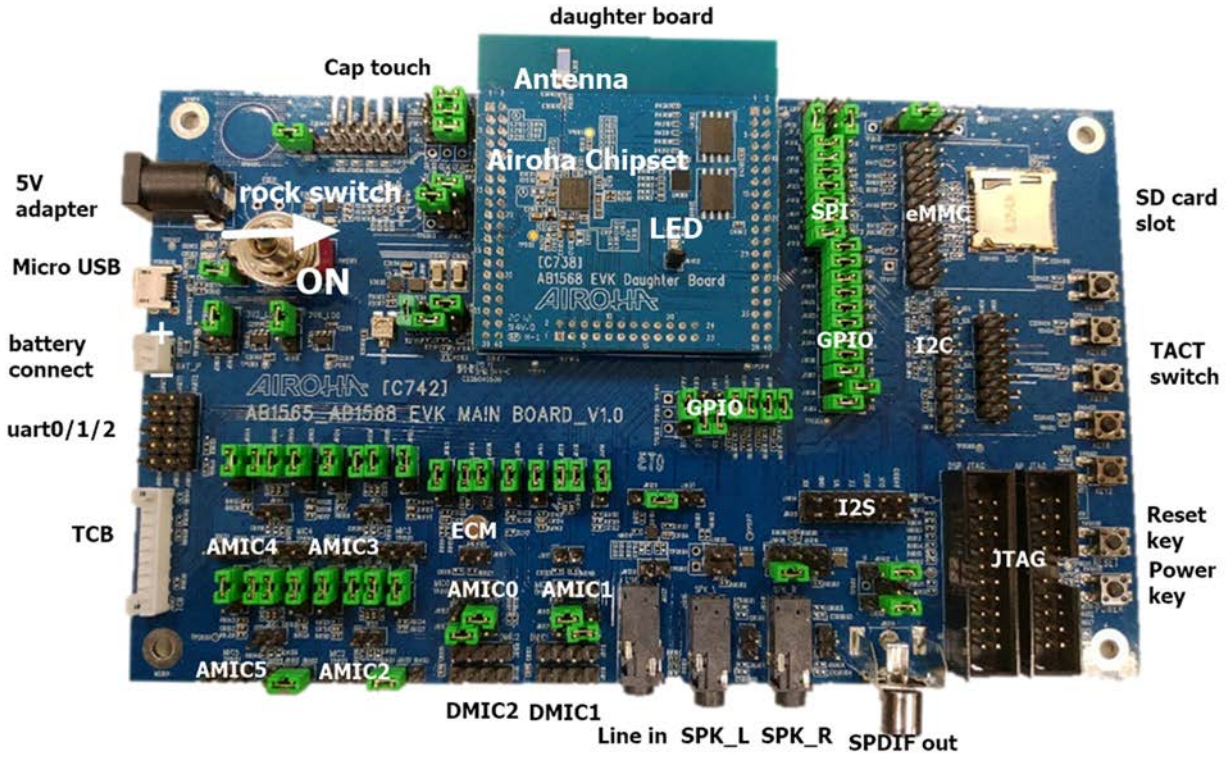

Figure 11. Front view of the AB1565/AB1568 EVK

The EVK can be powered by either USB VBUS or the 5V DC jack. Set the jumpers J2001 1-2 on for 5V DC jack.

A micro-USB connector is reserved on the left-side of the EVK, this port can also be used as the firmware download port. Set jumpers J2001 1-2 on, the VBUS power source is from USB VBUS.

AB1565/8 has reserved two JTAG debugging interfaces, AP_JTAG is used for embedded CM4 and DSP_JTAG is used for the embedded DSP. You can directly connect JTAG JIG with these connectors for debugging.

2.4.2. Installing AB1565/AB1568 Flash Tool for AB1565/AB1568 EVK

Same as 155x, please refer to section 2.3.2 Installing AB155x Flash Tool for AB155x EVK and replace 155x to 1565/1568.

2.4.3. Installing the AB1565/AB1568 EVK drivers on Microsoft Windows

Same as 155x, please refer to section 2.3.3 Installing the AB155x EVK drivers on Microsoft Windows and replace 155x to 1565/1568.

2.4.4. Flashing the image to AB1565/AB1568 EVK

Same as 155x, please refer to section 2.3.4 Flashing the image to AB155x EVK and replace 155x to 1565/1568.

2.4.5. Running the project on AB1565/AB1568 EVK

You must complete the following procedure to use Logging tool:

1) Launch Logging tool from Airoha.Tool.Kit.exe.
2) Select log binary file as shown in Figure 7.
3) Set Baud Rate is shown in Figure 8.
4) Select “Serial port” and click connect as shown in Figure 9.
5) Click “Wireshark” as shown in Figure 10.
6) Reset the target. The log is shown in the Log window.

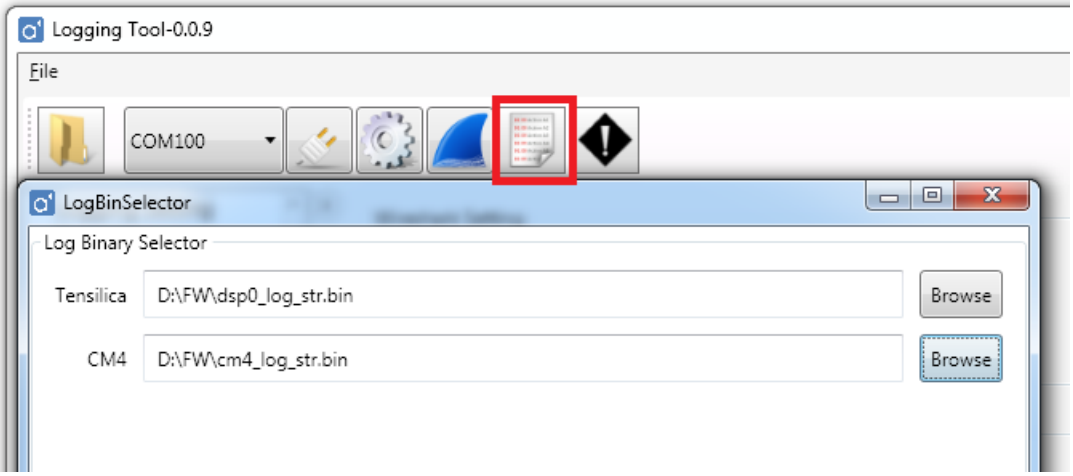

Figure 12. Select Log Binary

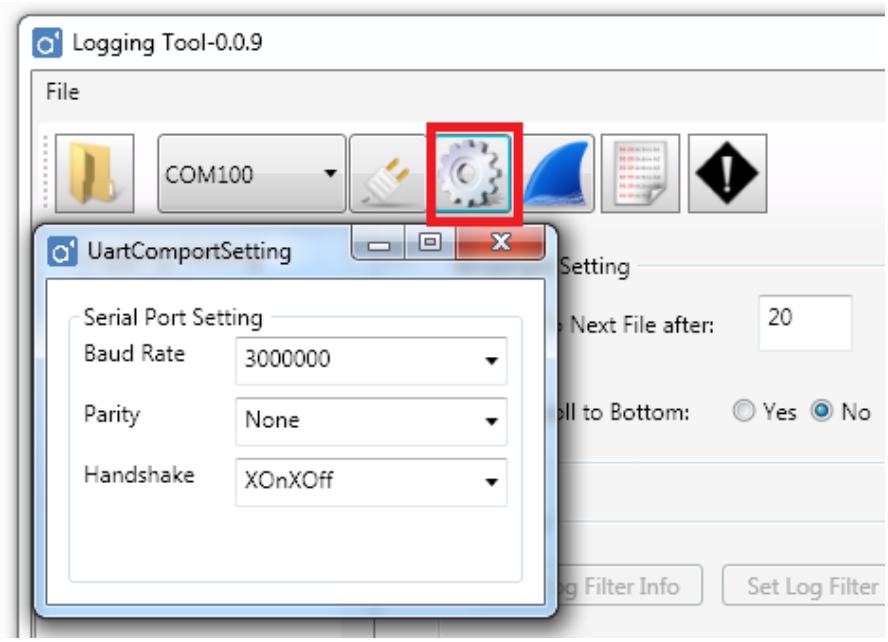

Figure 13. Serial Port Configuration Dialog

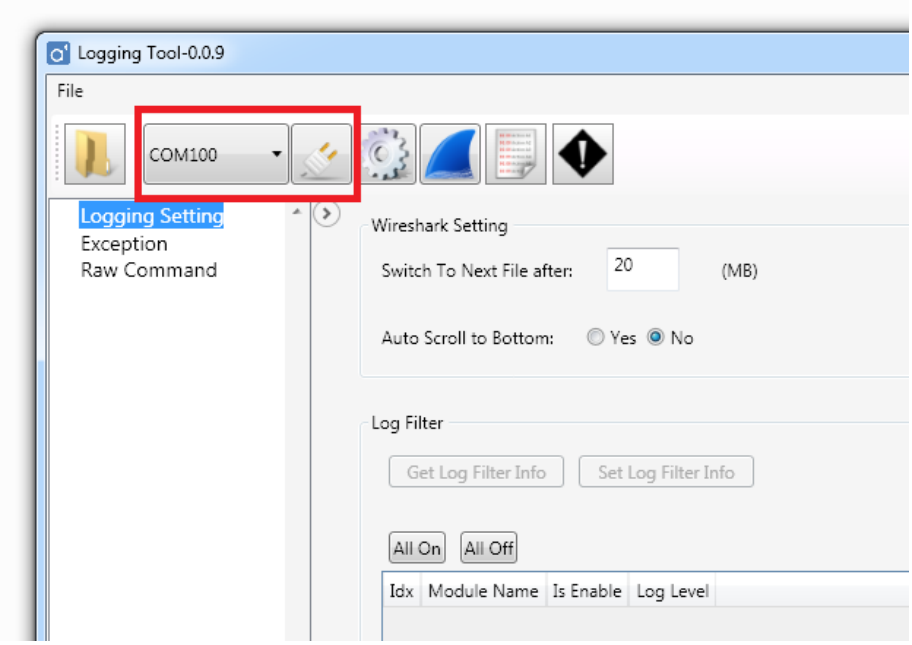

Figure 14. Serial Port Connect Button

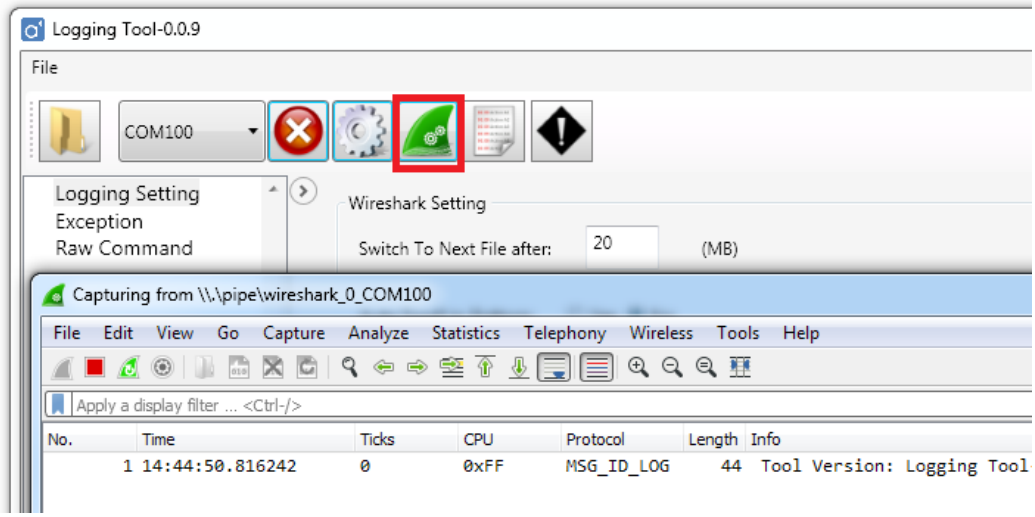

Figure 15. Start Wireshark Button

2.4.6. Debugging with the AB1565/AB1568 EVK from Microsoft Windows

Same as 155x, please refer to section 2.3.6 Debugging with the AB155x EVK from Microsoft Windows and replace 155x to 1565/1568.

2.5. Create your own project

This section provides details on how to use an existing project and create your own project named `my_project` on AB1565/AB1568 EVK using `earbuds_ref_design` project as a reference.

2.5.1. Using an existing project

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

Copy the folder `<sdk_root>/mcu/project/ab1565_ab1568_evk/apps/earbuds_ref_design` to a new directory `<sdk_root>/mcu/project/ab1565_ab1568_evk/apps/` and rename `earbuds_ref_design` to the new project name `my_project`.

2.5.2. 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/ab1565_ab1568_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/MTK/bluetooth/module.mk

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

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

2.5.3. Add the source and header files

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

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/ab1565_ab1568_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_ab155x.c
...
CXX_FILES       += ...
...
C_FILES         += $(APP_FILES) $(SYS_FILES)
...
```