Airoha IoT SDK Open Source Software Guide

Version: 3.20

Release date: 13 October 2022

Document Revision History

Revision	Date	Description
1.0	7 March 2016	Initial release
2.0	2 May 2016	Added the FatFs module.
3.0	30 June 2016	Added CMSIS and LZMA.
3.1	2 September 2016	Added the illustration of FreeRTOS memory configuration and settings.
3.2	13 January 2017	Added the mDNS and WebSocket.
3.3	5 May 2017	Updated DHCPD
3.4	1 August 2017	Updated license information
3.5	15 September 2017	Added AWS IoT SDK support
3.6	7 January 2019	Added micro-ecc and OPUS OGG Codec. Updated license information.
3.7	7 March 2019	Added a table for modules on different chips
3.8	4 September 2019	Added support modules on AG3335
3.9	19 September 2019	Added nanopb support
3.10	26 September 2019	Updated cJSON, MQTT, nghttp2
3.11	23 October 2019	Added SRP
3.12	22 November 2019	Fixed link error
3.13	2 June 2020	Added nanopb & mSBC encoder support
3.14	29 July 2020	Updated RTOS software license in Table 1 Added pre_libloader and Gsensor-key support
3.15	6 July 2021	Updated nanopb/Fatfa/mbedtls/LZMA
3.16	24 December 2021	Added support modules on AG3352
3.17	5 January 2022	Added support modules on AB158x
3.18	21 April 2022	Added nanopb, newlib, newlib-nano and Libgcc license information Removed Wi-Fi related open source modules
3.19	26 August 2022	Updated FatFs license disclaimer link
3.20	13 October 2022	Updated FreeRTOS license and version information in Table 1 & Table 2 Updated mbedtls/CMSIS/pre_libloader version in Table 2

Table of Contents

Lists of tables and figures

1. Overview

This guide provides information about the open-source software bundled in the SDK. It provides information about the open-source software packages and guides the developers in designing, prototyping, and implementing projects in a convenient environment.

As an open-source software package, the information is already available in various sources. This guide is an easy reference to module descriptions, including the official web site and the hardware or software integration versions. Developers can access the latest source code from the official website and merge or replace the code in the package with a corresponding version. From the footprint statistics result, developers can easily estimate the code size for setting up the flash memory layout. Before commencing your own IoT project or application development, look for example applications and their source code. Finally, the troubleshooting and limitations chapter provides more detailed information on reported issues that may be encountered during the application development.

1.1. Open source software resources for the SDK

The open-source software packages in the SDK are listed in Table 1 and for developer's reference only. The developer must acknowledge that such listed open-source software may be supplemented or amended by Airoha from time to time. The developer must also comply with all licensing terms applicable to such open-source software. Airoha makes the following disclaimers regarding the open-source software on behalf of itself, and the copyright holders, contributors, and licensors of the listed open-source software: TO THE FULLEST EXTENT PERMITTED UNDER APPLICABLE LAW, THE OPEN SOURCE SOFTWARE ARE PROVIDED BY THE COPYRIGHT HOLDERS, CONTRIBUTORS, LICENSORS, AND AIROHA "AS IS" AND ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER ORAL OR WRITTEN, WHETHER EXPRESS, IMPLIED, OR ARISING BY STATUTE, CUSTOM, COURSE OF DEALING, OR TRADE USAGE, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, ARE DISCLAIMED. IN NO EVENT WILL THE COPYRIGHT OWNER, CONTRIBUTORS, LICENSORS, OR AIROHA BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE OPEN SOURCE SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table 1. Open source packages

Module	Open source software licenses	Comments
RTOS	FreeRTOS License: • Modified GPL (for v8.2.0) • MIT license (for v10 and later version) Please refer to license disclaimer in kernel/rtos/FreeRTOS/Source/include/FreeRTOS.h For Cadence FreeRTOS porting layer, please refer to license disclaimer in dsp/kernel/service/context_switch/<IC_CONFIG>/inc/xtensa_context.h	Market leading, de-facto standard OS for embedded systems.

Module	Open source software licenses	Comments
	Note: FreeRTOS V10 and later version are distributed under the MIT license. http://www.freertos.org/a00114.html	
TCP/IP	lwIP License: BSD License http://lwip.wikia.com/wiki/License	lwIP (lightweight IP) is a widely used open source TCP/IP stack designed for embedded systems.
SSL/TLS	mbed TLS License: Apache 2.0 License http://www.apache.org/licenses/LICENSE-2.0	Easy to use SSL/TLS library including cryptographic and SSL/TLS capabilities with small footprint.
XML	Mini-XML License: LGPLv2 with static linking exception http://michaelrsweet.github.io/mxml/	A small XML library to read and write XML documents.
FatFs	FatFs License: BSD-style license http://elm-chan.org/fsw/ff/doc/appnote.html#license	FatFs is a generic FAT file system module for small embedded systems. For compatibility and practical considerations, the SDK currently supports two FatFs versions, R0.12b and R0.14b.
CMSIS	CMSIS License: BSD License (for v4.0.0) https://developer.mbed.org/blog/entry/CMSIS-Components-BSD-Licensed/ Apache 2.0 License (for v5.7.0) https://arm-software.github.io/CMSIS_5/General/html/index.html#License	CMSIS is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces.
LZMA decoder	LZMA License: public domain http://www.7-zip.org/sdk.html	LZMA decoder is extracted from the LZMA SDK for further use.
Micro-ecc	micro-ecc License: BSD 2-Clause "Simplified" https://github.com/kmackay/micro-ecc	A small and fast ECDH and ECDSA implementation for 8-bit, 32-bit, and 64-bit processors.
OPUS Codec	OPUS License: The 3-Clause BSD License https://opus-codec.org/license/	The audio codec library to support OGG file and opus file.
nanopb	nanopb License: zlib License https://jpa.kapsi.fi/nanopb/	A C implementation of Google's Protocol Buffers
mSBC encoder	SBC encoder License: Apache 2.0 License http://www.apache.org/licenses/LICENSE-2.0	The source code is from Android source tree located in a Git repository hosted by Google.
Gsensor-key	Gsensor-key License: The 3-Clause BSD License	A module used to realize single tap and double tap, part of the source code is

Module	Open source software licenses	Comments
	https://opensource.org/licenses/BSD-3-Clause	from STMicroelectronics's MEMS standard C driver on github.
pre_libloader	pre_libloader License: MIT license https://opensource.org/licenses/MIT	A module that allows dynamically loading and unloading code to execute on an Xtensa processor without the aid of an operating system.
newlib	libm.a & libnosys.a License: Please refer to COPYING.NEWLIB https://chromium.googlesource.com/native_client/nacl-newlib/+/refs/heads/master/COPYING.NEWLIB	It's a C-standard library implementation intended for use on embedded systems.
newlib-nano	libc_s.a License: Please refer to src\newlib\COPYING.NEWLIB, in gcc-arm-none-eabi-9-2019-q4-major-src.tar.bz2 (MD5: dec65fe8c14aae90512310dd5fe88bf1).	The newlib-nano is an open-source C library (libc) targeting embedded microcontrollers (MCU).
Libgcc	libgcc.a License: GNU GPL plus the GCC Runtime Library Exception https://www.gnu.org/licenses/gcc-exception-3.1.html Please refer to Part 6 in license.txt under mcu\tools\gcc9.2.1\linux\gcc-arm-none-eabi\share\doc\gcc-arm-none-eabi\ folder.	GNU Arm Embedded Toolchain

The SDK open-source packages are implemented by the active open-source community with widely available online resources and forum support. The list of online resources is provided for each module in Table 2. The integrated versions are also included for developers to merge hot bug fixes or new features directly from the official release links.

Table 2. Online resources for each module in the SDK

Module	Official website	Integrated version	Online API reference
RTOS	http://www.freertos.org/RTOS.html	For MT2523/MT2533: v8.2.0 For AM255x/AB155x/AB1565/AB1568/AG3335/AG3352: v10.1.1 For AB158x: v10.4.5	http://www.freertos.org/a00106.html
TCP/IP	http://savannah.nongnu.org/projects/lwip/ http://lwip.wikia.com/wiki/LwIP_Wiki	2.1.2	No online API. Exported API with comments can be found at `<sdk_root>/middleware/third_party/lwip/src/include/lwip/sockets.h`.
SSL/TLS	https://tls.mbed.org/	For AB1565/AB1568/AB158x: v3.1.0	https://tls.mbed.org/api/

Module	Official website	Integrated version	Online API reference
		For MT2523/MT2533/AM255x/AB155x: v2.26.0 For AG3335/AG3352: v2.25.0	
XML	https://www.msweet.org/mxml/	Mini-XML 2.9	https://www.msweet.org/mxml/mxml.html
FatFs	http://elm-chan.org/fsw/ff/00index_e.html	R0.14b(1)	http://elm-chan.org/fsw/ff/00index_e.html
CMSIS	https://developer.arm.com	For MT2523/MT2533/AM255x/AB155x/AB1565/AB1568/AG3335/AG3352: 4.0.0 For AB158x: 5.7.0	https://arm-software.github.io/CMSIS_5/Core/html/modules.html
LZMA decoder	http://www.7-zip.org/sdk.html	21.02	http://www.7-zip.org/sdk.html
Micro-ecc	https://github.com/kmackay/micro-ecc	https://github.com/kmackay/micro-ecc/commits/master Commit ID: 601bd11	https://github.com/kmackay/micro-ecc
OPUS Codec	http://www.xiph.org/ http://opus-codec.org/	1.1.4	No online API. Exported API with comments can be found at <sdk_root>/prebuilt/middleware/third_party/audio/celt_codec/inc/opuscelt_api.h. <sdk_root>/prebuilt/middleware/third_party/audio/ogg_codec/inc/oggcelt_api.h.
nanopb	https://jpa.kapsi.fi/nanopb/	0.4.5(2)	https://jpa.kapsi.fi/nanopb/docs/reference.html
mSBC encoder	https://android.googlesource.com/platform/system/bt/+/refs/tags/android-wear-5.1.0_r1/embdrv/sbc/encoder	5.1.0	https://android.googlesource.com/platform/system/bt/+/refs/tags/android-wear-5.1.0_r1/embdrv/sbc/encoder
Gsensor-key	https://github.com/STMicroelectronics/STMems_Standard_C_drivers/tree/master/lis2dw12_STdC https://github.com/STMicroelectronics/STMems_Standard_C_drivers/tree/master/lis2ds12_STdC	1.0.0	https://github.com/STMicroelectronics/STMems_Standard_C_drivers/tree/master/lis2dw12_STdC https://github.com/STMicroelectronics/STMems_Standard_C_drivers/tree/master/lis2ds12_STdC
pre_libloader	NA(From Cadence)	For AM255x/AB155x/AB1565/AB1568: RG-2019.12	No online API. Please find exported API at

Module	Official website	Integrated version	Online API reference
		For AB158x: RI-2021.8	<sdk_root>\dsp\kernel\service\pre_libloader\inc\preloader_pisplit.h
newlib	https://launchpad.net/gcc-arm-embedded/9.0/9-2019-q4-major	3.1.0	https://chromium.googlesource.com/native_client/nacl-newlib/+/e14046f93c76ef701d8ad133d0ea2b96d3c1b578
newlib-nano	https://launchpad.net/gcc-arm-embedded/9.0/9-2019-q4-major	3.1.0	Included in source package gcc-arm-none-eabi-9-2019-q4-major-src.tar.bz2 (MD5: dec65fe8c14aae90512310dd5fe88bf1)
Libgcc	https://launchpad.net/gcc-arm-embedded/9.0/9-2019-q4-major	ARM/arm-9-branch revision 277439	gcc-arm-none-eabi-9-2019-q4-major-src.tar.bz2 (MD5: dec65fe8c14aae90512310dd5fe88bf1)

(1) FatFs version is R0.12b before SDK version 2.7.0 and add supported for R0.14b after SDK version 2.7.0.
(2) GSound library (add-on release) also includes nanopb (version v0.4.3) in its library.

The source location for the modules listed in Table 2 is shown in Figure 1 and . The FreeRTOS for this SDK release is under the `kernel` directory and other middleware can be found under the `middleware` directory.

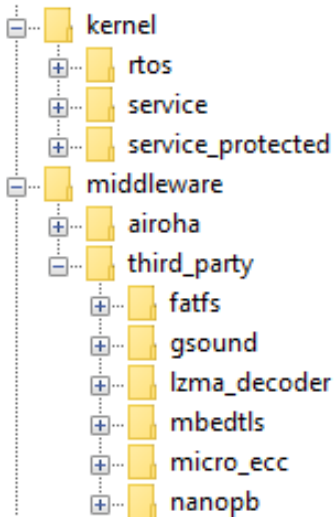

Figure 1. Source location of Airoha IoT SDK for BT Audio and Smart MCU open source software

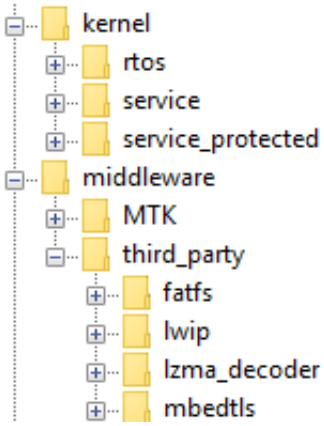

Figure 2. Source location of Airoha IoT SDK for Location open source software

The supported modules on different chips are listed in Table 3; chips of each product line are shown below.

- Airoha IoT SDK for Smart MCU: MT2523/MT2533/AM255x
- Airoha IoT SDK for BT Audio: AB155x/AB1565/AB1568/AB158x
- Airoha IoT SDK for Location: AG3335/AG3352

Table 3. Modules based on different chips

Module	Airoha IoT SDK for Smart MCU	Airoha IoT SDK for BT Audio	Airoha IoT SDK for Location
RTOS	✓	✓	✓
TCP/IP			✓
SSL/TLS		✓	✓
XML	✓		
FatFs	✓	✓	✓
CMSIS	✓	✓	✓
LZMA decoder	✓	✓	✓
Micro-ecc		✓	
OPUS Codec		✓	
nanopb		✓	
mSBC encoder		✓	
Gsensor-key	✓	✓	
pre_libloader	✓	✓	
newlib	✓	✓	✓
newlib-nano	✓	✓	✓
Libgcc	✓	✓	✓

The following sections give a more detailed description of each module.

2. RTOS: FreeRTOS

FreeRTOS is a real-time OS that manages a multitasking and multiprocessing system environment. It has a scheduler to manage user-created tasks. FreeRTOS provides APIs for users to control, synchronize, and communicate among various tasks.

2.1. Features

FreeRTOS supports the following five features to accomplish event/task scheduling and multitasking:

- Task and Scheduler – Each application consists of tasks or threads controlled by the operating system. The multitasking operation is implemented with a scheduler. The scheduler is in the kernel and manages the task execution at a specific time. The kernel can suspend and resume a task many times during lifecycle of the task execution.
- Queue – Queues are the primary forms of inter-task communications. In most cases they are used as thread safe first-in-first-out (FIFO) buffers.
- Semaphore – Threads use semaphores to control the access to shared resources.
- Software Timer – A software timer allows a function to be invoked at a predefined time. The timer callback functions are executed within the timer service task. It is essential to ensure the timer callback functions perform lightweight operations and return as quick as possible to avoid blocking the system resources.
- Event Group (enabled after FreeRTOS version 8.0.0) – An event group is a set of event bits. Individual event bits within an event group are referenced by a bit number. Event bits are used to indicate if an event has already occurred or not. Event groups can also be used to synchronize tasks.
- For more information on FreeRTOS and its features, please refer to the official website.

2.2. Memory usage

Brief details on the memory usage can be found in FreeRTOS FAQ - Memory Usage, Boot Times & Context Switch Times website, under the following sections:

- How much RAM does FreeRTOS use?
- How much ROM/Flash does FreeRTOS use?

2.3. Heap service

`heap_4.c` is used as the heap service on the SDK. The implementation is extended with more algorithms that are not supported by the official FreeRTOS. More details can be found in the header file `<sdk_root>\kernel\rtos\FreeRTOS\Source\include\portable.h`.

Several APIs are described in Table 4.

Table 4. Heap service APIs

Prototype	Description
`void *pvPortCalloc(size_t nmemb, size_t size)`	Provides the same functionality as the ISO C function `calloc()`.

Prototype	Description
	Allocates memory for an array "nmemb" with elements in bytes and returns a pointer to the allocated memory. The memory is set to zero. This function is available in SDK v1 and later.
void *pvPortRealloc(void *pv, size_t size)	Provides the same functionality as the ISO C function realloc(). Changes the size of the memory block pointed by "pv" to "size" in bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will not be initialized. This function is available in SDK v1 and later.
void *pvPortMallocNC(size_t xWantedSize)	This function is similar to pvPortMalloc(). Allocates "xWantedSize" bytes and returns a pointer to the allocated memory, except the allocated memory is in non-cacheable region. This function is available in SDK v3 and later.
void vPortFreeNC(void *pv)	This function is similar to vPortFree(). Frees the memory space pointed by "pv". All cache related operations are directly implemented in the API and is transparent to users. This function is available in SDK v3 and later.

Heap is in cacheable region by default for performance critical operations, such as task stack. The functions pvPortMallocNC() and vPortFreeNC() are implemented for temporary non-cacheable memory requirements, such as when using DMAs.

- For convenient migration of third-party software and libraries, the SDK also supports C standard library heap implementation, including malloc(), calloc(), realloc() and free(). However, these functions are wrapped in FreeRTOS heap service functions pvPortMalloc(), pvPortCalloc(), pvPortRealloc() and vPortFree(), and require building the project with a FreeRTOS module.

2.4. Examples

The source code and API documentation of the FreeRTOS can be found here.

2.5. Customization

Users can provide custom configuration for the FreeRTOS in FreeRTOSConfig.h header file. The configurable parameters include OS tick frequency, maximum priorities, disabled functions, etc.

For more details, please refer to the online documentation.

2.6. Limitations

Airoha does not impose any limitation on the integrated FreeRTOS. Please refer to the official online resources for more detailed information.

2.7. Developer notes

- There is no software restriction on the number of tasks to create.
- The tasks can share the same priority.
- If the configuration `USE_PORT_OPTIMISED_TASK_SELECTION` is enabled, the maximum number of task priorities will be 32 (0 to 31) in the ARM Cortex-M4 with floating point porting.
- Only API functions that end in "`FromISR`" can be called from within an interrupt service routine. Please refer to Open RTOS API documentation for more details.
- APIs that can potentially cause a context switch must not be called while the scheduler is suspended.
- APIs that can potentially cause a context switch must not be called from within a critical section.

3. TCP/IP: lwIP

TCP/IP (Transmission Control Protocol/Internet Protocol) is a communication internet protocol and can also be used in a private network, either an intranet or an extranet. It provides specifications on how data should be packetized, addressed, transmitted, routed, and received at the destination. The current IoT standard is moving towards IP communication and transporting data over various physical layers such as Wi-Fi, IEEE 802.15.4, and Bluetooth.

3.1. Features

lwIP is a widely used open-source TCP/IP stack designed for embedded systems. It includes the IP, ICMP, TCP, UDP, IGMP, ARP, AutoIP, DHCP, DNS, and SNMP protocols. The SDK provides the following supported features for these protocols.

- IPv4 (LWIP_IPV4).
- UDP (LWIP_UDP) – User Datagram Protocol, the widely adopted connectionless transmission protocol.
- TCP (LWIP_TCP) – Transmission Control Protocol, a widely used transport protocol providing reliable and in-order delivery.
- ARP (LWIP_ARP).
- ICMP (LWIP_ICMP).
- DHCP (LWIP_DHCP).
- DNS (LWIP_DNS).
- NETCONN (LWIP_NETCONN).
- Socket (LWIP_SOCKET).

3.2. Memory usage

The MEM_SIZE parameter defines the size of the heap memory. PBUF_RAM, stores the sent and received data. If the application requires more data to send, the value of the parameter must be set higher.

The values of different control blocks are configurable in lwIP. For example, MEMP_NUM_NETDB sets the concurrent Domain Name Resolution connections. Table 5 lists the current configuration values of these blocks in the SDK. These pools are configured and allocated from a buffer reserved only for the lwIP. The values are set to pass internal performance test and are expected to fulfill most common use cases. However, the number of configured control blocks could limit the maximum concurrent connections created by the network applications in the system. Developers can configure these values for a specific use case.

Table 5. Number of control blocks and buffers in lwIP

Name	Current value
MEMP_NUM_UDP_PCB	4
MEMP_NUM_TCP_PCB	8
MEMP_NUM_TCP_PCB_LISTEN	16
MEMP_NUM_REASSDATA	5
MEMP_NUM_NETBUF	2

Name	Current value
MEMP_NUM_NETCONN	10
MEMP_NUM_TCPIP_MSG_API	8
MEMP_NUM_TCPIP_MSG_INPKT	8
MEMP_NUM_SYS_TIMEOUT	16
MEMP_NUM_NETDB	1
MEMP_NUM_PBUF	16
PBUF_POOL_SIZE	10

The required code size of lwIP is listed in Table 6. This information is gathered from an ARM Cortex M4 targeted configuration using the `gcc -Os` optimization. The feature set is IPv4, TCP, UDP, DHCP client, ICMP, RAW, NETCONN and Sockets and DNS client. The footprint is shown below.

Table 6. Footprint of the lwIP

Static footprint	ROM (bytes)	RAM (bytes)
Debug release	45541	51861

Note: If the options `LWIP_DEBUG`, `LWIP_ERROR`, `LWIP_ASSERTS` or `LWIP_STATS` are enabled, then the application has a significantly larger code footprint. Similarly, if the application is built with the compiler `-O0` optimization flag on, the footprint is again significantly affected.

3.3. Examples

1) File: `<sdk_root>/project/<chip>/apps/lwip_socket/src/main.c.`
2) Application Name:
   a) IPv4 UDP Client/Server test + IPv4 TCP Client/Server test,
   b) IPv6 UDP Client/Server test + IPv6 TCP Client/Server test by disabling MLD.
3) Application Overview. This is a reference application to demonstrate the usage of socket functions. The UDP can be used as a client or a server. When operating as a client, it can send a data packet to a remote UDP server. When operating as a server, it can receive data packets from remote UDP clients.
4) Similar tests can be carried out on TCP. When TCP operates as a client, it can connect to and communicate with a remote TCP server. When it operates as a server, it listens to and waits for incoming connections from remote TCP clients. After a connection is established, it can communicate with the peer clients.

3.4. Customization

The custom settings and configuration can be applied in `otp.h` file located under `<sdk_root>/middleware/third_party/lwip/src/include/lwip/`. This file is fully commented and it is clear which options are defined, enabled, or disabled.

Developers can also customize the project settings in the `lwipopts.h` file located under `<sdk_root>/project/<chip>/apps/<project>/include/lwipopts.h`. For more information, please refer to introduction to lwIP on the Wikia website.

To enable or disable a feature, simply change the configuration parameters in `lwipopts.h`. For example, if you would like to disable DNS and enable DHCP, please modify or add the following lines in `lwipopts.h` header file.

- //Disable DNS

```
#define LWIP_DNS 0
```

- //Enable DHCP

```
#define LWIP_DHCP 1
```

Here are the major features in lwIP that can be customized (enabled or disabled) by developers:

- `LWIP_IPV4`: Enables IPv4 protocol.
- LWIP_IPV6: Enables IPv6 protocol. In this release, the SDK supports only IPv4. Developers can manually enable it. The basic functions work with IPv4/IPv6 dual stack support, but it cannot pass the IPv6 conformance test.
- `LWIP_UDP`: Enables the UDP.
- `LWIP_TCP`: Enables the TCP.
- `LWIP_RAW`: The raw API is an event-driven API designed to be used without an operating system that implements zero-copy send and receive operation. However, the SDK suggests using BSD socket APIs (defined by `LWIP_SOCKET`) to make the socket application portable. If you are interested in the raw API, please refer to the lwip native API documentation for more details.
- `LWIP_ARP`: Enables the ARP functionality.
- `LWIP_ICMP`: Enables the ICMP module in the IP stack.
- `LWIP_IGMP`: Enables the IGMP module (multicast support).
- `LWIP_SNMP`: Enables the lwIP SNMP agent. The UDP must be available for the SNMP transport.
- `LWIP_DHCP`: Enables the DHCP client module.
- `LWIP_AUTOIP`: Enables the AUTOIP module and the RFC 3927 - Dynamic Configuration of IPv4 Link-Local Addresses.
- `LWIP_DNS`: Enables the DNS module and requires UDP for DNS transport.
- `LWIP_NETCONN`: Enables Netconn API, requires using `api_lib.c` source file.
- `LWIP_SOCKET`: Enables Socket API, require using `sockets.c` to include a set of APIs compatible with POSIX-/BSD sockets.
- `LWIP_STATS`: Enables statistics collection in `lwip_stats`.
- `LWIP_DEBUG`: Enables debugging.
- `LWIP_ERROR`: Enables error logging.
- `LWIP_NOASSERT`: Disables LWIP_ASSERT checks.

3.5. Limitations

- The list of feature limitations for lwIP is provided below.

- Only supports a few ICMP packet types, such as echo reply, destination unreachable, and time exceeded. Most of the other types that are not specific to the embedded systems are ignored.
- Does not support Network Address Translation (NAT) to map IP address space into another address for forwarding the IP packets.
- Few of TCP/IP options are supported in lwIP.

3.6. Developer notes

Applications should invoke `recv()` API repeatedly until it returns `EWOULDBLOCK` indicating there is no pending data in the receiving buffer. Otherwise, more and more incoming data will exhaust the buffer.

lwIP supports both blocking and non-blocking methods in the SDK. By default, sockets are blocking, indicating that if a socket call is issued, it cannot be completed immediately. For example, in the TCP three-way handshaking process, the caller process is put to sleep while waiting for the condition to be true. With non-blocking socket operation, the socket function call can return immediately and the application can proceed with other operations. Non-blocking I/O is suitable for single-thread socket applications.

-

4. SSL/TLS: mbed TLS

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL) are cryptographic protocols designed to provide secure communication over the computer network. TLS and SSL use X.509 certificates and asymmetric cryptography to authenticate secure data communication and to negotiate the process with a symmetric session key that ensure message confidentiality.

mbed TLS offers libraries including SSL/TLS cryptographic communication capabilities for (embedded) devices and applications that provide end-to-end communication protection to the upper layer application protocols such as web browsing, email, instant messaging, and voice-over-IP (VoIP).

Starting from the version 2.1.0, mbed TLS is released under Apache 2.0 License and enables developers to use mbed TLS in both open source and closed source projects.

4.1. Features

mbed TLS is an open-source and commercial SSL library licensed under ARM Limited. This library easily integrates with new and existing (embedded) devices and applications and provides the building blocks for secure communication, cryptography, and key management. Both the client-side and the server-side APIs support current SSL and TLS standards (i.e. SSL version 3.0, TLS version 1.0, TLS version 1.1 and TLS version 1.2). The cryptographic algorithms enabled in the SDK include:

1) Symmetric encryption algorithms: AES, Triple-DES (3DES), DES, ARC4.
2) Modes of operations: Cipher Block Chaining Mode (CBC).
3) Hash algorithms: MD5, SHA-1, and SHA-256.
4) RSA/PKCS#1 v1.5.
5) Random number generation: CTR_DRBG.

4.2. Memory usage

Airoha offers two configurations in the release, basic and mini. The basic package contains mandatory cipher suites supported by TLS v1.0 through TLS v1.2. The mini package includes minimal algorithm support for key exchange, cipher and hash algorithms. If the manufactures can control the deployment on both sides of the communication line, the memory footprint can be considerably reduced by manually selecting lightweight algorithms, such as algorithms using a pre-shared key.

Please refer to the sample configuration files `config-mtk-basic.h` and `config-mtk-mini.h` located under folder `<sdk_root>/middleware/third_party/mbedtls/configs/`.

- Airoha IoT SDK provides a configuration file named `config-mtk-mini.h`. This is the minimum configuration to fulfill the requirement of TLS from versions 1.0 to 1.2. The `config-mtk-bsic.h` enables more commonly used cryptographic algorithms. The required ROM size for these two default configurations is shown in Table 7.

Table 7. Footprint of mbed TLS

Memory (Kbytes)	Basic		Mini	
	Debug	Release	Debug	Release
ROM	96	78	78	65
RAM	8.7	8.7	8.7	8.7

Memory (Kbytes)	Basic		Mini	
HEAP (peak for one connection)	26	26	26	26

4.3. Examples

mbed TLS is equipped with test cases. Developers can use them as a reference to develop and learn how to use the APIs. A few example applications can be found under `<sdk_root>/project/<chip>/apps/mbedtls/` directory.

4.4. Customization

- The default configuration file for mbed TLS is `config-mtk-basic.h` in the SDK release. The file is located under `<sdk_root>/middleware/third_party/mbedtls/configs/config-mtk-basic.h`. Developers can provide their custom configuration file under `<sdk_root>/middleware/third_party/mbedtls/configs` and set the file name to the variable `MTK_MBEDTLS_CONFIG_FILE` in project's `feature.mk` makefile such as `<sdk_root>/project/<chip>/apps/iot_sdk/GCC/feature.mk`.
- Define `MBEDTLS_DEBUG_C` in the configuration file to enable debugging and error handling.
- You can switch the feature options in the configuration file and define some of the values or parameters. The configuration file is well documented and you can refer to the comments in `<sdk_root>/middleware/third_party/mbedtls/include/mbedtls/config.h`.

4.5. Limitations

Due to the limitation of ARM Cortex-M4 with floating point computational power, the TLS handshake at server side takes more than 10 seconds with RSA 2048-bit public key length. Therefore, the SDK disables the server option in the suggested configuration file.

4.6. Developer notes

1) In order to keep mbed TLS thread safe, it is important to keep a few things in mind.
2) Most functions use an explicit context. As long as the context is not shared among the threads, the application is thread safe. However, sometimes a context is shared indirectly. For example, an SSL context can point to an RSA context (the private key).
3) The rule of thumb is that a context should only be used or accessed by a single thread at a time, unless:
   a) The function is documented explicitly that it is thread safe to access the shared context, or
   b) You have applied an explicit locking mechanism, such as a mutex, to protect a critical section through a wrapper function.
4) `MBEDTLS_SSL_MAX_CONTENT_LEN` is assigned with (6*1024) bytes in the example configuration files under `<sdk_root>/middleware/third_party/mbedtls/configs/config-mtk-*.h`, to reduce the heap usage. The default value of `MBEDTLS_SSL_MAX_CONTENT_LEN` is 16384 bytes. This value is defined in the specification. Modifying this value to reduce the required buffer may lead to interoperability problems. You can safely reduce this to a smaller size such as two kilobytes, if…
   a) Both client and server sides support the `max_fragment_length` SSL extension (allowing reduction to less than 1k bytes for the buffer allocation).

b) You control both sides of the connection or know the maximum size that will be sent in a single SSL/TLS frame.

5) Client verifies the server's identity with the received certificate during the handshake operation. Typically the received server certificate is composed of three-level hierarchy. If the length is larger, the TLS client will require more heap space for verification process.

6) Complete the following procedure to configure the trusted root certificates.

a) Since the SDK doesn't support File system in Airoha IoT Development Platform, you can only keep the trusted certificate authentications (CAs) in memory and parse them by calling the `mbedtls_x509_crt_parse()` function.

b) If there are several trusted CAs, invoke the `mbedtls_x509_crt_parse()` function several times to set them to the trusted CA chain one by one.

c) Set authorization mode to `NONE`, `OPTIONAL` or `REQUIRED`, by calling the `mbedtls_ssl_conf_authmode()` function. If `REQUIRED` option is selected, the handshake operation will fail once the certificate verification fails.

d) If the function `mbedtls_ssl_set_hostname()` is called, the client compares the hostname with the common name of the server certificate in the certification verification process.

5. XML: Mini-XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format, which is both human and machine-readable. It is defined by the W3C's XML 1.0 specification and by several other related specifications, all of which are free open standards.

Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services.

5.1. Features

The Mini-XML module is a small XML library that enables parsing the XML and XML-like data in the application without requiring large non-standard libraries. It supports reading of UTF-8 and UTF-16 and writing of UTF-8 encoded XML strings. Data is stored in a linked-list tree structure, preserving the XML data hierarchy and arbitrary element names, attributes and attribute values are supported with no preset limits, just available memory.

5.2. Memory usage

The XML library of the SDK has two build types, full and basic with optimized and reduced memory footprint.

The XML module provides several compilation options to disable unused sub-modules. Overall, the ROM size of most basic version is 13052 bytes. It is 21940 bytes if all of the sub-modules are enabled, see section 5.4, “Customization”.

5.3. Examples

1) File: `<sdk_root>/project/<chip>/apps/xml/src/main.c.`
2) Application Name: XML sample application.
3) Application Overview: The sample application is a reference to parse a typical XML file by means of this XML library. The developers can refer to this simple application and reuse the functions in their applications.

5.4. Customization

The XML module provides compile options to enable or disable sub-features. Table 8 shows the details.

Table 8. Options and footprint of XML

Compilation option	Function	ROM size (bytes)
MXML_SUPPORT_ENTITY	Convert the reserved characters in XML to corresponding character entities.	4584
MXML_SUPPORT_GET_FUNCTIONS	Get the attribute and content of an element	892
MXML_SUPPORT_SET_FUNCTIONS	Set the attribute and content of an element.	1040
MXML_SUPPORT_INDEX	Create the index with all elements, the elements are sorted by element name or attribute value.	1560
MXML_SUPPORT_SEARCH	Find the element by name.	812

5.5. Limitations

For the platforms that don’t support file system, such as LinkIt 7687 development board, the file system related API sets of `mxml*Fd()` and `mxml*File()` are not accessible, and the XML objects can only be loaded and saved by `mxml*String()` APIs.

5.6. Developer notes

None.

6. File System: FatFs

FatFs is a generic FAT file system that manages the access to storage devices for small embedded systems. FatFs is written in compliance with ANSI C (C89) and is completely separated from the disk I/O layer.

6.1. Features

To facilitate the data (files or directories) management in the storage device, FatFs provides the following features:

- Windows OS compatible FAT file system.
- Platform independent, easy to port to any target platform.
- Small footprint for code and work area (file system objects, file objects, etc.).
- Switches character encoding (ANSI/OEM and UTF-16 for R0.12b and R0.14b, UTF-8 and UTF-32 only for R0.14b) for the file name on the API.
- RTOS support for multi-tasking.
- Multiple sector size support up to 4kB.
- Read-only, optional API, I/O buffer and other features.

More information on the FatFs and its features can be found here.

6.2. Memory usage

The memory usage varies depending on the configuration options, as described in section 6.4, “Customization”. The FatFs module provides several compilation options to disable unused sub-modules. Overall, according to the configuration in the SDK release, the ROM size of the current release is 12367 bytes, the RAM size is 25 bytes. The memory size may be different for different versions of the SDK.

6.3. Examples

The source code and API documentation of the FatFs can be found at official website.

6.4. Customization

Users can customize the configuration for the FatFs in `ffconf.h` header file. The configurable parameters include the volume to support, the sector size, etc.

More details can be found in the online documentation.

6.5. Limitations

- The list of feature limitations for the FatFs is provided below.
- Multiple sector size support up to 4kB.
- FAT sub-types - FAT12, FAT16 and FAT32.
- Number of open files - unlimited depending on the available memory.
- Number of volumes - up to 10.

- File size - up to 4GB minus 1 byte. (FAT specifications.)
- Volume size - up to 2TB at 512 bytes per sector. (FAT specifications.)
- Cluster size - up to 64kB at 512 bytes/sector. (FAT specs.)
- Sector size - 512, 1024, 2048 and 4096 bytes. (FAT specs.)

6.6. Developer notes

FatFs integrated in current SDK version only supports file system on the SD (Secure Digital Memory Card) or eMMC (Embedded Multi Media Card) without flash.

- FatFs is not enabled by default in order to provide the user with the flexibility to select whether to use this file system or a different file systems.
- Due to changes in API and feature options of configuration files from R0.12b to R0.14b, in order to be compatible with existing customers, the current SDK supports multiple versions (R0.12b and R0.14b).
- R0.14b can support two partitions on storage disk with a capacity less than 500KB, while R0.12b requires at least 1MB. In addition, R0.14b adds API support for UTF-8 and UTF-32.
- The feature option prefix of R0.14b is “FF_”, while R0.12b is “_”.
- Developers can add Fatfs module.mk to the Makefile to import FatFs support, and specify FATFS_VERSION in the Makefile to select different versions. The currently supported versions are R0.12b and R0.14b.
- During compilation, developers can use the __FATFS_VERSION__ macro to confirm the actual FatFs version used to write the compatible code. If __FATFS_VERSION__ is 201609L, then the current version is R0.12b and if __FATFS_VERSION__ is 202104L, then the current version is R0.14b.

7. CMSIS

The CMSIS is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces. The CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals, real-time operating systems and middleware components simplifying software re-use, reducing the learning curve for microcontroller developers and reducing the time to market for new devices.

The CMSIS is intended to enable the combination of software components from multiple middleware vendors.

7.1. Features

The CMSIS components are:

- CMSIS-CORE – Implements basic run-time APIs for a Cortex-M device and provides convenient access to the processor core and the device peripherals.
- CMSIS-DSP – This library provides a suite of common signal processing functions to apply on Cortex-M processor based devices.

Airoha IoT development platform only includes CMSIS-CORE and CMSIS-DSP.

7.2. Memory usage

CMSIS-CORE APIs are implemented in header files, so the memory usage is added to the source file. The library size of the CMSIS-DSP is up to 5.4MB and the actual memory usage depends on which and how many APIs are in use.

7.3. Examples

CMSIS API examples can be found at official website.

7.4. Customization

No customization or configuration is required to use CMSIS-CORE and CMSIS-DSP, apply them directly in your implementation.

7.5. Limitations

- None.

7.6. Developer notes

Airoha IoT Development Platform only includes CMSIS-CORE and CMSIS-DSP.

- We strongly recommend that you visit the official ARM website to find more detailed information about CMSIS.

8. LZMA: LZMA Decoder

The LZMA is an algorithm performing lossless data compression. The LZMA SDK provides a high compression ratio and fast decompression. The Airoha IoT SDK only uses the LZMA decoder algorithm.

8.1. Features

The LZMA decoder provides functions to decompress the compressed data encoded by LZMA encoder.
LZMA decoder has the following features:

- Small memory requirements: 8 to 32kB + Dictionary Size.
- Small code size: 2 to 8kB, depending on speed optimizations.

More information on the LZMA and its features can be found here.

8.2. Memory usage

The memory usage is shown in Table 9.

Table 9. LZMA decoder memory usage

ROM (bytes)	RAM, static analysis (bytes)
6338	40*1024 (suggested decoding buffer)

8.3. Examples

LZMA decoder module is used to decode FOTA package file in bootloader only. Because Cortex-M4 binary size is usually too large to have enough buffer for the whole file data, there is a wrapper function `lzma_decode2flash()` provided in `<sdk_dir>/middleware/third_parity/lzma_decoder/inc/lzma_decoder_interface.h` header file to decode data block by block, then call HAL flash API to write the output data to the specified address on NOR flash.

More information on the LZMA decoder interface can be found in the bootloader module's source file (`<sdk_dir>/driver/board/<chip>/bootloader/core/src/bl_fota.c`).

8.4. Customization

None.

8.5. Limitations

- None.

8.6. Developer notes

LZMA decoder integrated in current version of the SDK only supports bootloader decoding.

9. Micro-ecc

A small and fast ECDH and ECDSA implementation for 8-bit, 32-bit, and 64-bit processors.

9.1. Features

- Resistant to known side-channel attacks.
- Written in C, with optional GCC inline assembly for AVR, ARM, and Thumb platforms.
- Support for 8-bit, 32-bit, and 64-bit architectures.
- Small code size.
- No dynamic memory allocation.
- Support for five standard curves: secp160r1, secp192r1, secp224r1, secp256r1, and secp256k1.
- BSD 2-clause license.

9.2. Memory usage

None

9.3. Examples

None

9.4. Customization

None

9.5. Limitations

None

9.6. Developer notes

None

10. OPUS OGG Codec

Audio Codec library to support to OGG file and OPUS file

10.1. Features

- Support encoder and decoder.
- Only support 16K sampling rate
- Support 16K~256K bitrate
- Frame size : 20ms
- Support Speech/music
- Only support mono.

10.2. Memory usage

Table 10. OPUS OGG Codec usage

	ROM (bytes)	RAM, static analysis (bytes)
OPUS OGG	136366	12000

10.3. Examples

- File: `<sdk_root>/project/<chip>/apps/iot_sdk_demo/opus_proc.c.`
- Feature Name: ./build.sh iot_sdk_demo -f=feature_ogg_opus.mk
- Feature Overview: refer to iot_sdk_demo/readme.txt

10.4. Customization

None

10.5. Limitations

Not support multi thread.

10.6. Developer notes

None

11. nanopb

Nanopb is an ANSI-C library for encoding and decoding messages in Google's Protocol Buffers format with minimal requirements for RAM and code space. It is primarily suitable for 32-bit microcontrollers.

11.1. Features

- Pure C runtime
- Allows specifying maximum size for strings and arrays, so that they can be allocated statically.
- No malloc needed: everything can be allocated statically or on the stack. Optional malloc support available.
- You can use either encoder or decoder alone to cut the code size in half.
- Support for most protobuf features, including: all data types, nested submessages, default values, repeated and optional fields, oneofs, packed arrays, extension fields.
- Callback mechanism for handling messages larger than can fit in available RAM.
- Extensive set of tests.

11.2. Memory usage

- Small code size (5–10 kB depending on processor and compilation options, plus any message definitions)
- Small ram usage (typically ~300 bytes stack, plus any message structs)

11.3. Examples

None

11.4. Customization

None

11.5. Limitations

- Some speed has been sacrificed for code size.
- Encoding is focused on writing to streams. For memory buffers only it could be made more efficient.
- The deprecated Protocol Buffers feature called "groups" is not supported.
- Fields in the generated structs are ordered by the tag number, instead of the natural ordering in .proto file.
- Unknown fields are not preserved when decoding and re-encoding a message.
- Reflection (runtime introspection) is not supported. E.g. you can't request a field by giving its name in a string.
- Numeric arrays are always encoded as packed, even if not marked as packed in .proto.
- Cyclic references between messages are supported only in callback and malloc mode.

- Nanopb doesn't have a stable ABI (application binary interface) between versions, so using it as a shared library (.so / .dll) requires extra care.

11.6. Developer notes

None

12. mSBC: mSBC Encoder

Modified version of the SBC codec (hereafter called mSBC) is mandatory if Wide Band Speech is supported in Bluetooth HANDS-FREE Profile. The original SBC codec is specified in A2DP (Advanced Audio Distribution Profile). The changes to the A2DP SBC are limited to the frame header syntax and semantics. All other parts of the SBC definition remain un-modified.

The Airoha IoT SDK only uses the mSBC encoder of open source software.

12.1. Features

- Available bitrate is 63 kb/s only
- Sampling rate is 16 kHz only
- Frame size is 15 ms only
- Support for speech and music

12.2. Memory usage

For the mSBC encoder library, ROM size is 2.3 Kbytes.

12.3. Examples

None

12.4. Customization

None

12.5. Limitations

- None

12.6. Developer notes

None

13. Gsensor-key

A module used to realize single tap and double tap by Gsensor.

13.1. Features

- Support single tap and double tap
- BSD 3-clause license

13.2. Memory usage

None

13.3. Examples

None

13.4. Customization

None

13.5. Limitations

None

13.6. Developer notes

None

14. Pre_libloader

A module that allows dynamically loading and unloading code to execute on an Xtensa processor without the aid of an operating system.

14.1. Features

- A mechanism which dynamically loading and unloading code to execute.
- Gives code and data memory sizes about loadable library.
- Relocated the dynamical library when loading other memory region.
- Loadable lib is a standalone program.

14.2. Memory usage

For the pre_libloader driver library, ROM size is 18863 bytes.

14.3. Examples

The source code and API documentation of the pre_libloader can be found.

14.4. Customization

None

14.5. Limitations

The dynamical library need add extra the –fpic and -mlongcalls option than static library when build library.a.

The failure to use a static library as a dynamic library carries unexpected risks.

14.6. Developer notes

If you need to use the function of pre_libloader, please follow the operation flow of DSP API reference document.