MCUXpresso SDK Documentation

Release 25.09.00-pvw1

NXP
Jul 17, 2025

Table of contents

This documentation contains information specific to the evkmimxrt1160 board.

Chapter 1

Middleware

1.1 Boot

1.1.1 MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource

Overview

This repository is a fork of MCUboot (https://github.com/mcu-tools/mcuboot) for MCUXpresso SDK delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation

Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit MCUboot - Documentation to review details on the contents in this sub-repo.

Setup

Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution

Contributions are not currently accepted. If the intended contribution is not related to NXP specific code, consider contributing directly to the upstream MCUboot project. Once this MCUboot fork is synchronized with the upstream project, such contributions will end up here as well. If the intended contribution is a bugfix or improvement for NXP porting layer or for code added or modified by NXP, please open an issue or contact NXP support.

NXP Fork

This fork of MCUboot contains specific modifications and enhancements for NXP MCUXpresso SDK integration.

See *changelog* for details.

1.1.2 MCUboot

This is MCUboot version 2.2.0

MCUboot is a secure bootloader for 32-bits microcontrollers. It defines a common infrastructure for the bootloader and the system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software upgrade.

MCUboot is not dependent on any specific operating system and hardware and relies on hardware porting layers from the operating system it works with. Currently, MCUboot works with the following operating systems and SoCs:

- Zephyr
- Apache Mynewt
- Apache NuttX
- RIOT
- Mbed OS
- Espressif
- Cypress/Infineon

RIOT is supported only as a boot target. We will accept any new port contributed by the community once it is good enough.

MCUboot How-tos

See the following pages for instructions on using MCUboot with different operating systems and SoCs:

- Zephyr
- Apache Mynewt
- Apache NuttX
- RIOT
- Mbed OS
- Espressif
- *Cypress/Infineon*

There are also instructions for the *Simulator*.

Roadmap

The issues being planned and worked on are tracked using GitHub issues. To give your input, visit MCUboot GitHub Issues.

Source files

You can find additional documentation on the bootloader in the source files. For more information, use the following links:

- boot/bootutil - The core of the bootloader itself.
- boot/boot_serial - Support for serial upgrade within the bootloader itself.
- boot/zephyr - Port of the bootloader to Zephyr.
- boot/mynewt - Bootloader application for Apache Mynewt.
- boot/nuttx - Bootloader application and port of MCUboot interfaces for Apache NuttX.
- boot/mbed - Port of the bootloader to Mbed OS.
- boot/espressif - Bootloader application and MCUboot port for Espressif SoCs.
- boot/cypress - Bootloader application and MCUboot port for Cypress/Infineon SoCs.
- imgtool - A tool to securely sign firmware images for booting by MCUboot.
- sim - A bootloader simulator for testing and regression.

Joining the project

Developers are welcome!

Use the following links to join or see more about the project:

- Our developer mailing list
- Our Discord channel Get your invite

1.2 Cloud

1.2.1 AWS IoT

Device Shadow Library

AWS IoT Device Shadow library The AWS IoT Device Shadow library enables you to store and retrieve the current state (the "shadow") of every registered device. The device's shadow is a persistent, virtual representation of your device that you can interact with from AWS IoT Core even if the device is offline. The device state is captured as its "shadow" within a JSON document. The device can send commands over MQTT to get, update and delete its latest state as well as receive notifications over MQTT about changes in its state. Each device's shadow is uniquely identified by the name of the corresponding "thing", a representation of a specific device or logical entity on the AWS Cloud. See Managing Devices with AWS IoT for more information on IoT "thing". More details about AWS IoT Device Shadow can be found in AWS IoT documentation. This library is distributed under the *MIT Open Source License*.

Note: From v1.1.0 release onwards, you can used named shadow, a feature of the AWS IoT Device Shadow service that allows you to create multiple shadows for a single IoT device.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library here.

AWS IoT Device Shadow v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Device Shadow v1.0.2 source code is part of the FreeRTOS 202012.00 LTS release.

AWS IoT Device Shadow Config File The AWS IoT Device Shadow library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *shadow_config_defaults.h*. To provide custom values for the configuration macros, a custom config file named shadow_config.h can be provided by the user application to the library.

By default, a shadow_config.h custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide SHADOW_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Building the Library The *shadowFilePaths.cmake* file contains the information of all source files and the header include path required to build the AWS IoT Device Shadow library.

As mentioned in the *previous section*, either a custom config file (i.e. shadow_config.h) OR the SHADOW_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the AWS IoT Device Shadow library.

For a CMake example of building the AWS IoT Device Shadow library with the shadowFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Checkout CMock Submodule By default, the submodules in this repository are configured with update=none in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like amazon-freertos that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive --test/unit-test/CMock
```

Platform Prerequisites

- For building the library, **CMake 3.13.0** or later and a **C90 compiler**.
- For running unit tests, **Ruby 2.0.0** or later is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build unit tests

1. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described *above*.)
2. Run the *cmake* command: cmake -S test -B build

3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples Please refer to the demos of the AWS IoT Device Shadow library in the following locations for reference examples on POSIX and FreeRTOS platforms:

Platform	Location	Transport Interface Implementation (for coreMQTT stack)
POSIX	AWS IoT Device SDK for Embedded C	POSIX sockets for TCP/IP and OpenSSL for TLS stack
FreeRTOS	FreeRTOS/FreeRTOS	FreeRTOS+TCP for TCP/IP and mbedTLS for TLS stack
FreeRTOS	FreeRTOS AWS Reference Integrations	Based on Secure Sockets Abstraction

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of IoT Device Shadow library may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

Device Defender Library

AWS IoT Device Defender Library The Device Defender library enables you to send device metrics to the AWS IoT Device Defender Service. This library also supports custom metrics, a feature that helps you monitor operational health metrics that are unique to your fleet or use case. For example, you can define a new metric to monitor the memory usage or CPU usage

on your devices. This library has no dependencies on any additional libraries other than the standard C library, and therefore, can be used with any MQTT client library. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone static code analysis using Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

AWS IoT Device Defender v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Device Defender v1.1.0 source code is part of the FreeRTOS 202012.01 LTS release.

AWS IoT Device Defender Client Config File The AWS IoT Device Defender Client Library exposes build configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *defender_config_defaults.h*. To provide custom values for the configuration macros, a config file named defender_config.h can be provided by the application to the library.

By default, a defender_config.h config file is required to build the library. To disable this requirement and build the library with default configuration values, provide DEFENDER_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Thus, the Device Defender client library can be built by either:

- Defining a defender_config.h file in the application, and adding it to the include directories list of the library.

OR

- Defining the DEFENDER_DO_NOT_USE_CUSTOM_CONFIG preprocessor macro for the library build.

Building the Library The *defenderFilePaths.cmake* file contains the information of all source files and the header include paths required to build the Device Defender client library.

As mentioned in the previous section, either a custom config file (i.e. defender_config.h) or DEFENDER_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the Device Defender client library.

For a CMake example of building the Device Defender client library with the defenderFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Platform Prerequisites

- For running unit tests:
  - **C90 compiler** like gcc.
  - **CMake 3.13.0 or later**.
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository.
2. Run the *cmake* command: `cmake -S test -B build -DBUILD_CLONE_SUBMODULES=ON`.
3. Run this command to build the library and unit tests: `make -C build all`.
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples The AWS IoT Embedded C-SDK repository contains a demo showing the use of AWS IoT Device Defender Client Library here on a POSIX platform.

Documentation

Existing documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of the AWS IoT Device Defender library may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

Jobs Library

README

AWS IoT Jobs library The AWS IoT Jobs library helps you notify connected IoT devices of a pending Job. A Job can be used to manage your fleet of devices, update firmware and security certificates on your devices, or perform administrative tasks such as restarting devices and performing diagnostics. It interacts with the AWS IoT Jobs service using MQTT, a lightweight publish-subscribe protocol. This library provides a convenience API to compose and recognize the MQTT topic strings used by the Jobs service. The library is written in C compliant with ISO C90 and MISRA C:2012, and is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard . Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity, and validation of memory safety with the CBMC bounded model checker.

See memory requirements for this library *here*.

AWS IoT Jobs v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Jobs v1.1.0 source code is part of the FreeRTOS 202012.01 LTS release.

Building the Jobs library A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Given an application in a file named example.c, *gcc* can be used like so:

```
gcc -I source/include example.c source/jobs.c -o example
```

gcc can also produce an object file to be linked later:

```
gcc -I source/include -c source/jobs.c
```

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The test/cbmc/proofs directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference example The AWS IoT Device SDK for Embedded C repository contains a demo using the jobs library on a POSIX platform. https://github.com/aws/aws-iot-device-sdk-embedded-C/tree/main/demos/jobs/jobs_demo_mosquitto

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of the AWS IoT Jobs library may differ across repositories.

Generating Documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Building unit tests

Checkout Unity Submodule By default, the submodules in this repository are configured with `update=none` in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories that submodule this repository.

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive --test/unit-test/Unity
```

Platform Prerequisites

- For running unit tests
  - C90 compiler like gcc
  - CMake 3.13.0 or later
  - Ruby 2.0.0 or later is additionally required for the Unity test framework (that we use).
- For running the coverage target, lcov is additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described *above*.)
2. Create build directory: `mkdir build && cd build`
3. Run *cmake* while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

Contributing See *CONTRIBUTING.md* for information on contributing.

Over-the-air Update Library

AWS IoT Over-the-air Update Library The OTA library enables you to manage the notification of a newly available update, download the update, and perform cryptographic verification of the firmware update. Using the library, you can logically separate firmware updates from the application running on your devices. The OTA library can share a network connection with the application, saving memory in resource-constrained devices. In addition, the OTA library lets you define application-specific logic for testing, committing, or rolling back a firmware update. The library supports different application protocols like Message Queuing Telemetry Transport (MQTT) and Hypertext Transfer Protocol (HTTP), and provides various configuration options you can fine tune depending on network type and conditions. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8. This library has also undergone static code analysis from Coverity static analysis.

See memory requirements for this library *here*.

AWS IoT Over-the-air Update Library v3.4.0 source code is part of the FreeRTOS 202210.00 LTS release.

AWS IoT Over-the-air Update Library v3.3.0 source code is part of the FreeRTOS 202012.01 LTS release.

AWS IoT Over-the-air Updates Config File The AWS IoT Over-the-air Updates library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *ota_config_defaults.h*. To provide custom values for the configuration macros, a custom config file named ota_config.h can be provided by the user application to the library.

By default, a ota_config.h custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide OTA_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Building the Library The *otaFilePaths.cmake* file contains the information of all source files and the header include paths required to build the AWS IoT Over-the-air Updates library.

As mentioned in the previous section, either a custom config file (i.e. ota_config.h) OR the OTA_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the AWS IoT Over-the-air Updates library.

For a CMake example of building the AWS IoT Over-the-air Updates library with the otaFilePaths.cmake file, refer to the coverity_analysis library target in the *test/CMakeLists.txt* file.

Building Unit Tests

Checkout CMock Submodule By default, the submodules in this repository are configured with update=none in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like AWS IoT Device SDK for Embedded C that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

Platform Prerequisites

- For building the library, **CMake 3.13.0** or later and a **C90 compiler**.
- For running unit tests, **Ruby 2.0.0** or later is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build unit tests

1. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described *above*.)
2. Run the *cmake* command: cmake -S test -B build

3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

Migration Guide

How to migrate from v2.0.0 (Release Candidate) to v3.4.0 The following table lists equivalent API function signatures in v2.0.0 (Release Candidate) and v3.4.0 declared in *ota.h*

v2.0.0 (Release Candidate)	v3.4.0	Notes
`OtaState_t OTA_Shutdown( uint32_t ticksToWait );`	`OtaState_t OTA_Shutdown( uint32_t ticksToWait, uint8_t unsubscribeFlag );`	`unsubscribeFlag` indicates if unsubscribe operations should be performed from the job topics when shutdown is called. Set this as 1 to unsubscribe, 0 otherwise.

How to migrate from version 1.0.0 to version 3.4.0 for OTA applications Refer to OTA Migration document for the summary of updates to the API. Migration document for OTA PAL also provides a summary of updates required for upgrading the OTA-PAL to work with v3.4.0 of the library.

Porting In order to support AWS IoT Over-the-air Updates on your device, it is necessary to provide the following components:

1. Port for the OTA Portable Abstraction Layer (PAL).
2. OS Interface
3. MQTT Interface

For enabling data transfer over HTTP dataplane the following component should also be provided:

1. HTTP Interface

NOTE When using OTA over HTTP dataplane, MQTT is required for control plane operations and should also be provided.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples Please refer to the demos of the AWS IoT Over-the-air Updates library in the following location for reference examples on POSIX and FreeRTOS:

Platform	Location
POSIX	AWS IoT Device SDK for Embedded C
FreeRTOS	FreeRTOS/FreeRTOS
FreeRTOS	FreeRTOS AWS Reference Integrations

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of coreMQTT may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

1.3 Connectivity

1.3.1 lwIP

This is the NXP fork of the lwIP networking stack.

- For details about changes and additions made by NXP, see CHANGELOG.
- For details about the NXP porting layer, see *The NXP lwIP Port*.
- For usage and API of lwIP, use official documentation at http://www.nongnu.org/lwip/.

The NXP lwIP Port

Below is description of possible settings of the port layer and an overview of a few helper functions.

The best place for redefinition of any mentioned macro is lwipopts.h.

The declaration of every mentioned function is in ethernetif.h. Please check the doxygen comments of those functions before.

Link state Physical link state (up/down) and its speed and duplex must be read out from PHY over MDIO bus. Especially link information is useful for lwIP stack so it can for example send DHCP discovery immediately when a link becomes up.

To simplify this port layer offers a function ethernetif_probe_link() which reads those data from PHY and forwards them into lwIP stack.

In almost all examples this function is called every ETH_LINK_POLLING_INTERVAL_MS (1500ms) by a function probe_link_cyclic().

By setting ETH_LINK_POLLING_INTERVAL_MS to 0 polling will be disabled. On FreeRTOS, probe_link_cyclic() will be then called on an interrupt generated by PHY. GPIO port and pin for

the interrupt line must be set in the ethernetifConfig struct passed to ethernetif_init(). On bare metal interrupts are not supported right now.

Rx task To improve the reaction time of the app, reception of packets is done in a dedicated task. The rx task stack size can be set by ETH_RX_TASK_STACK_SIZE macro, its priority by ETH_RX_TASK_PRIO.

If you want to save memory you can set reception to be done in an interrupt by setting ETH_DO_RX_IN_SEPARATE_TASK macro to 0.

Disabling Rx interrupt when out of buffers If ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS is set to 1, then when the port gets out of Rx buffers, Rx enet interrupt will be disabled for a particular controller. Everytime Rx buffer is freed, Rx interrupt will be enabled.

This prevents your app from never getting out of Rx interrupt when the network is flooded with traffic.

ETH_DISABLE_RX_INT_WHEN_OUT_OF_BUFFERS is by default turned on, on FreeRTOS and off on bare metal.

Limit the number of packets read out from the driver at once on bare metal. You may define macro ETH_MAX_RX_PKTS_AT_ONCE to limit the number of received packets read out from the driver at once.

In case of heavy Rx traffic, lowering this number improves the realtime behaviour of an app. Increasing improves Rx throughput.

Setting it to value < 1 or not defining means “no limit”.

Helper functions If your application needs to wait for the link to become up you can use one of the following functions:

- ethernetif_wait_linkup()- Blocks until the link on the passed netif is not up.
- ethernetif_wait_linkup_array() - Blocks until the link on at least one netif from the passed list of netifs becomes up.

If your app needs to wait for the IPv4 address on a particular netif to become different than “ANY” address (255.255.255.255) function ethernetif_wait_ipv4_valid() does this.

1.4 File System

1.4.1 FatFs

MCUXpresso SDK : mcuxsdk-middleware-fatfs

Overview This repository is for FatFs middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit FatFs - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution Contributions are not currently accepted. Guidelines to contribute will be posted in the future.

Repo Specific Content This is MCUXpresso SDK fork of FatFs (FAT file system created by ChaN). Official documentation is available at http://elm-chan.org/fsw/ff/

MCUXpresso version is extending original content by following hardware specific porting layers:

- mmc_disk
- nand_disk
- ram_disk
- sd_disk
- sdspi_disk
- usb_disk

Changelog FatFs

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog

[R0.15_rev0]

- Upgraded to version 0.15
- Applied patches from http://elm-chan.org/fsw/ff/patches.html

[R0.14b_rev1]

- Applied patches from http://elm-chan.org/fsw/ff/patches.html

[R0.14b_rev0]

- Upgraded to version 0.14b

[R0.14a_rev0]

- Upgraded to version 0.14a
- Applied patch ff14a_p1.diff and ff14a_p2.diff

[R0.14_rev0]

- Upgraded to version 0.14
- Applied patch ff14_p1.diff and ff14_p2.diff

[R0.13c_rev0]

- Upgraded to version 0.13c
- Applied patches ff_13c_p1.diff,ff_13c_p2.diff, ff_13c_p3.diff and ff_13c_p4.diff.

[R0.13b_rev0]

- Upgraded to version 0.13b

[R0.13a_rev0]

- Upgraded to version 0.13a. Added patch ff_13a_p1.diff.

[R0.12c_rev1]

- Add NAND disk support.

[R0.12c_rev0]

- Upgraded to version 0.12c and applied patches ff_12c_p1.diff and ff_12c_p2.diff.

[R0.12b_rev0]

- Upgraded to version 0.12b.

[R0.11a]

- Added glue functions for low-level drivers (SDHC, SDSPI, RAM, MMC). Modified diskio.c.
- Added RTOS wrappers to make FatFs thread safe. Modified syscall.c.
- Renamed ffconf.h to ffconf_template.h. Each application should contain its own ffconf.h.
- Included ffconf.h into diskio.c to enable the selection of physical disk from ffconf.h by macro definition.
- Conditional compilation of physical disk interfaces in diskio.c.

1.5 Motor Control

1.5.1 FreeMASTER

Communication Driver User Guide

Introduction

What is FreeMASTER? FreeMASTER is a PC-based application developed by NXP for NXP customers. It is a versatile tool usable as a real-time monitor, visualization tool, and a graphical control panel of embedded applications based on the NXP processing units.

This document describes the embedded-side software driver which implements an interface between the application and the host PC. The interface covers the following communication:

- **Serial** UART communication either over plain RS232 interface or more typically over a USB-to-Serial either external or built in a debugger probe.

- **USB** direct connection to target microcontroller
- **CAN bus**
- **TCP/IP network** wired or WiFi
- **Segger J-Link RTT**
- **JTAG** debug port communication
- ...and all of the above also using a **Zephyr** generic drivers.

The driver also supports so-called "packet-driven BDM" interface which enables a protocol-based communication over a debugging port. The BDM stands for Background Debugging Module and its physical implementation is different on each platform. Some platforms leverage a semi-standard JTAG interface, other platforms provide a custom implementation called BDM. Regardless of the name, this debugging interface enables non-intrusive access to the memory space while the target CPU is running. For basic memory read and write operations, there is no communication driver required on the target when communicating with the host PC. Use this driver to get more advanced FreeMASTER protocol features over the BDM interface. The driver must be configured for the packet-driven BDM mode, in which the host PC uses the debugging interface to write serial command frames directly to the target memory buffer. The same method is then used to read response frames from that memory buffer.

Similar to "packet-driven BDM", the FreeMASTER also supports a communication over [J-Link RTT]((https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/) interface defined by SEGGER Microcontroller GmbH for ARM CortexM-based microcontrollers. This method also uses JTAG physical interface and enables high-speed real time communication to run over the same channel as used for application debugging.

Driver version 3 This document describes version 3 of the FreeMASTER Communication Driver. This version features the implementation of the new Serial Protocol, which significantly extends the features and security of its predecessor. The new protocol internal number is v4 and its specification is available in the documentation accompanying the driver code.

Driver V3 is deployed to modern 32-bit MCU platforms first, so the portfolio of supported platforms is smaller than for the previous V2 versions. It is recommended to keep using the V2 driver for legacy platforms, such as S08, S12, ColdFire, or Power Architecture. Reach out to FreeMASTER community or to the local NXP representative with requests for more information or to port the V3 driver to legacy MCU devices.

Thanks to a layered approach, the new driver simplifies the porting of the driver to new UART, CAN or networking communication interfaces significantly. Users are encouraged to port the driver to more NXP MCU platforms and contribute the code back to NXP for integration into future releases. Existing code and low-level driver layers may be used as an example when porting to new targets.

Note: Using the FreeMASTER tool and FreeMASTER Communication Driver is only allowed in systems based on NXP microcontroller or microprocessor unit. Use with non-NXP MCU platforms is **not permitted** by the license terms.

Target platforms The driver implementation uses the following abstraction mechanisms which simplify driver porting and supporting new communication modules:

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified

user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.

- **Transport Communication Layer** - The Serial, CAN, Networking, PD-BDM, and other methods of transport logic are implemented as a driver layer called FMSTR_TRANSPORT with a uniform API. A support of the Network transport also extends single-client modes of operation which are native for Serial, USB and CAN by a concept of multiple client sessions.
- **Low-level Communication Driver** - Each type of transport further defines a low-level API used to access the physical communication module. For example, the Serial transport defines a character-oriented API implemented by different serial communication modules like UART, LPUART, USART, and also USB-CDC. Similarly, the CAN transport defines a message-oriented API implemented by the FlexCAN or MCAN modules. Moreover, there are multiple different implementations for the same kind of communication peripherals. The difference between the implementation is in the way the low-level hardware registers are accessed. The *mcuxsdk* folder contains implementations which use MCUXpresso SDK drivers. These drivers should be used in applications based on the NXP MCUXpresso SDK. The "ampsdk" drivers target automotive-specific MCUs and their respective SDKs. The "dreg" implementations use a plain C-language access to hardware register addresses which makes it a universal and the most portable solution. In this case, users are encouraged to add more drivers for other communication modules or other respective SDKs and contribute the code back to NXP for integration.

  The low-level drivers defined for the Networking transport enable datagram-oriented UDP and stream TCP communication. This implementation is demonstrated using the lwIP software stack but shall be portable to other TCP/IP stacks. It may sound surprisingly, but also the Segger J-Link RTT communication driver is linked to the Networking transport (RTT is stream oriented communication handled similarly to TCP).

Replacing existing drivers For all supported platforms, the driver described in this document replaces the V2 implementation and also older driver implementations that were available separately for individual platforms (PC Master SCI drivers).

Clocks, pins, and peripheral initialization The FreeMASTER communication driver is only responsible for runtime processing of the communication and must be integrated with an user application code to function properly. The user application code is responsible for general initialization of clock sources, pin multiplexers, and peripheral registers related to the communication speed. Such initialization should be done before calling the FMSTR_Init function.

It is recommended to develop the user application using one of the Software Development Kits (SDKs) available from third parties or directly from NXP, such as MCUXpresso SDK, MCUXpresso IDE, and related tools. This approach simplifies the general configuration process significantly.

MCUXpresso SDK The MCUXpresso SDK is a software package provided by NXP which contains the device initialization code, linker files, and software drivers with example applications for the NXP family of MCUs. The MCUXpresso Config Tools may be used to generate the clock-setup and pin-multiplexer setup code suitable for the selected processor.

The MCUXpresso SDK also contains this FreeMASTER communication driver as a "middleware" component which may be downloaded along with the example applications from https://mcuxpresso.nxp.com/en/welcome.

MCUXpresso SDK on GitHub The FreeMASTER communication driver is also released as one of the middleware components of the MCUXpresso SDK on the GitHub. This release enables direct integration of the FreeMASTER source code Git repository into a target applications including Zephyr applications.

Related links:

- The official FreeMASTER middleware repository.
- Online version of this document

FreeMASTER in Zephyr The FreeMASTER middleware repository can be used with MCUXpresso SDK as well as a Zephyr module. Zephyr-specific samples which include examples of Kconfig and Device Tree configurations for Serial, USB and Network communications are available in separate repository. West manifest in this sample repository fetches the full Zephyr package including the FreeMASTER middleware repository used as a Zephyr module.

Example applications

MCUX SDK Example applications There are several example applications available for each supported MCU platform.

- **fmstr_uart** demonstrates a plain serial transmission, typically connecting to a computer's physical or virtual COM port. The typical transmission speed is 115200 bps.
- **fmstr_can** demonstrates CAN bus communication. This requires a suitable CAN interface connected to the computer and interconnected with the target MCU using a properly terminated CAN bus. The typical transmission speed is 500 kbps. A FreeMASTER-over-CAN communication plug-in must be used.
- **fmstr_usb_cdc** uses an on-chip USB controller to implement a CDC communication class. It is connected directly to a computer's USB port and creates a virtual COM port device. The typical transmission speed is above 1 Mbps.
- **fmstr_net** demonstrates the Network communication over UDP or TCP protocol. Existing examples use lwIP stack to implement the communication, but in general, it shall be possible to use any other TCP/IP stack to achieve the same functionality.
- **fmstr_wifi** is the fmstr_net application modified to use a WiFi network interface instead of a wired Ethernet connection.
- **fmstr_rtt** demonstrates the communication over SEGGER J-Link RTT interface. Both fmstr_net and fmstr_rtt examples require the FreeMASTER TCP/UDP communication plug-in to be used on the PC host side.
- **fmstr_eonce** uses the real-time data unit on the JTAG EOnCE module of the 56F800E family to implement pseudo-serial communication over the JTAG port. The typical transmission speed is around 10 kbps. This communication requires FreeMASTER JTAG/EOnCE communication plug-in.
- **fmstr_pdbdm** uses JTAG or BDM debugging interface to access the target RAM directly while the CPU is running. Note that such approach can be used with any MCU application, even without any special driver code. The computer reads from and writes into the RAM directly without CPU intervention. The Packet-Driven BDM (PD-BDM) communication uses the same memory access to exchange command and response frames. With PD-BDM, the FreeMASTER tool is able to go beyond basic memory read/write operations and accesses also advanced features like Recorder, TSA, or Pipes. The typical transmission speed is around 10 kbps. A PD-BDM communication plug-in must be used in FreeMASTER and configured properly for the selected debugging interface. Note that this communication cannot be used while a debugging interface is used by a debugger session.
- **fmstr_any** is a special example application which demonstrates how the NXP MCUXpresso Config Tools can be used to configure pins, clocks, peripherals, interrupts, and even the FreeMASTER "middleware" driver features in a graphical and user friendly way. The user can switch between the Serial, CAN, and other ways of communication and generate the required initialization code automatically.

Zephyr sample spplications Zephyr sample applications demonstrate Kconfig and Device Tree configuration which configure the FreeMASTER middleware module for a selected communication option (Serial, CAN, Network or RTT).

Refer to *readme.md* files in each sample directory for description of configuration options required to implement FreeMASTER connectivity.

Description

This section shows how to add the FreeMASTER Communication Driver into application and how to configure the connection to the FreeMASTER visualization tool.

Features The FreeMASTER driver implements the FreeMASTER protocol V4 and provides the following features which may be accessed using the FreeMASTER visualization tool:

- Read/write access to any memory location on the target.
- Optional password protection of the read, read/write, and read/write/flash access levels.
- Atomic bit manipulation on the target memory (bit-wise write access).
- Optimal size-aligned access to memory which is also suitable to access the peripheral register space.
- Oscilloscope access—real-time access to target variables. The sample rate may be limited by the communication speed.
- Recorder— access to the fast transient recorder running on the board as a part of the FreeMASTER driver. The sample rate is only limited by the MCU CPU speed. The length of the data recorded depends on the amount of available memory.
- Multiple instances of Oscilloscopes and Recorders without the limitation of maximum number of variables.
- Application commands—high-level message delivery from the PC to the application.
- TSA tables—describing the data types, variables, files, or hyperlinks exported by the target application. The TSA newly supports also non-memory mapped resources like external EEPROM or SD Card files.
- Pipes—enabling the buffered stream-oriented data exchange for a general-purpose terminal-like communication, diagnostic data streaming, or other data exchange.

The FreeMASTER driver features:

- Full FreeMASTER protocol V4 implementation with a new V4 style of CRC used.
- Layered approach supporting Serial, CAN, Network, PD-BDM, and other transports.
- Layered low-level Serial transport driver architecture enabling to select UART, LPUART, USART, and other physical implementations of serial interfaces, including USB-CDC.
- Layered low-level CAN transport driver architecture enabling to select FlexCAN, msCAN, MCAN, and other physical implementations of the CAN interface.
- Layered low-level Networking transport enabling to select TCP, UDP or J-Link RTT communication.
- TSA support to write-protect memory regions or individual variables and to deny the access to the unsafe memory.
- The pipe callback handlers are invoked whenever new data is available for reading from the pipe.

- Two Serial Single-Wire modes of operation are enabled. The “external” mode has the RX and TX shorted on-board. The “true” single-wire mode interconnects internally when the MCU or UART modules support it.

The following sections briefly describe all FreeMASTER features implemented by the driver. See the PC-based FreeMASTER User Manual for more details on how to use the features to monitor, tune, or control an embedded application.

Board Detection The FreeMASTER protocol V4 defines the standard set of configuration values which the host PC tool reads to identify the target and to access other target resources properly. The configuration includes the following parameters:

- Version of the driver and the version of the protocol implemented.
- MTU as the Maximum size of the Transmission Unit (for example; communication buffer size).
- Application name, description, and version strings.
- Application build date and time as a string.
- Target processor byte ordering (little/big endian).
- Protection level that requires password authentication.
- Number of the Recorder and Oscilloscope instances.
- RAM Base Address for optimized memory access commands.

Memory Read This basic feature enables the host PC to read any data memory location by specifying the address and size of the required memory area. The device response frame must be shorter than the MTU to fit into the outgoing communication buffer. To read a device memory of any size, the host uses the information retrieved during the Board Detection and splits the large-block request to multiple partial requests.

The driver uses size-aligned operations to read the target memory (for example; uses proper read-word instruction when an address is aligned to 4 bytes).

Memory Write Similarly to the Memory Read operation, the Memory Write feature enables to write to any RAM memory location on the target device. A single write command frame must be shorter than the MTU to fit into the target communication buffer. Larger requests must be split into smaller ones.

The driver uses size-aligned operations to write to the target memory (for example; uses proper write-word instruction when an address is aligned to 4 bytes).

Masked Memory Write To implement the write access to a single bit or a group of bits of target variables, the Masked Memory Write feature is available in the FreeMASTER protocol and it is supported by the driver using the Read-Modify-Write approach.

Be careful when writing to bit fields of volatile variables that are also modified in an application interrupt. The interrupt may be serviced in the middle of a read-modify-write operation and it may cause data corruption.

Oscilloscope The protocol and driver enables any number of variables to be read at once with a single request from the host. This feature is called Oscilloscope and the FreeMASTER tool uses it to display a real-time graph of variable values.

The driver can be configured to support any number of Oscilloscope instances and enable simultaneously running graphs to be displayed on the host computer screen.

Recorder The protocol enables the host to select target variables whose values are then periodically recorded into a dedicated on-board memory buffer. After such data sampling stops (either on a host request or by evaluating a threshold-crossing condition), the data buffer is downloaded to the host and displayed as a graph. The data sampling rate is not limited by the speed of the communication line, so it enables displaying the variable transitions in a very high resolution.

The driver can be configured to support multiple Recorder instances and enable multiple recorder graphs to be displayed on the host screen. Having multiple recorders also enables setting the recording point differently for each instance. For example; one instance may be recording data in a general timer interrupt while another instance may record at a specific control algorithm time in the PWM interrupt.

TSA With the TSA feature, data types and variables can be described directly in the application source code. Such information is later provided to the FreeMASTER tool which may use it instead of reading symbol data from the application ELF executable file.

The information is encoded as so-called TSA tables which become direct part of the application code. The TSA tables contain descriptors of variables that shall be visible to the host tool. The descriptors can describe the memory areas by specifying the address and size of the memory block or more conveniently using the C variable names directly. Different set of TSA descriptors can be used to encode information about the structure types, unions, enumerations, or arrays.

The driver also supports special types of TSA table entries to describe user resources like external EEPROM and SD Card files, memory-mapped files, virtual directories, web URL hyperlinks, and constant enumerations.

TSA Safety When the TSA is enabled in the application, the TSA Safety can be enabled and validate the memory accesses directly by the embedded-side driver. When the TSA Safety is turned on, any memory request received from the host is validated and accepted only if it belongs to a TSA-described object. The TSA entries can be declared as Read-Write or Read-Only so that the driver can actively deny the write access to the Read-Only objects.

Application commands The Application Commands are high-level messages that can be delivered from the PC Host to the embedded application for further processing. The embedded application can either poll the status, or be called back when a new Application Command arrives to be processed. After the embedded application acknowledges that the command is handled, the host receives the Result Code and reads the other return data from memory. Both the Application Commands and the Result Codes are specific to a given application and it is user's responsibility to define them. The FreeMASTER protocol and the FreeMASTER driver only implement the delivery channel and a set of API calls to enable the Application Command processing in general.

Pipes The Pipes enable buffered and stream-oriented data exchange between the PC Host and the target application. Any pipe can be written to and read from at both ends (either on the PC or the MCU). The data transmission is acknowledged using the special FreeMASTER protocol commands. It is guaranteed that the data bytes are delivered from the writer to the reader in a proper order and without losses.

Serial single-wire operation The MCU Serial Communication Driver natively supports normal dual-wire operation. Because the protocol is half-duplex only, the driver can also operate in two single-wire modes:

- "External" single-wire operation where the Receiver and Transmitter pins are shorted on the board. This mode is supported by default in the MCU driver because the Receiver and Transmitter units are enabled or disabled whenever needed. It is also easy to extend this operation for the RS485 communication.

- "True" single-wire mode which uses only a single pin and the direction switching is made by the UART module. This mode of operation must be enabled by defining the FMSTR_SERIAL_SINGLEWIRE configuration option.

Multi-session support With networking interface it is possible for multiple clients to access the target MCU simultaneously. Reading and writing of target memory is processed atomically so there is no risk of data corruption. The state-full resources such as Recorders or Oscilloscopes are locked to a client session upon first use and access is denied to other clients until lock is released..

Zephyr-specific

Dedicated communication task FreeMASTER communication may run isolated in a dedicated task. The task automates the FMSTR_Init and FMSTR_Poll calls together with periodic activities enabling the FreeMASTER UI to fetch information about tasks and CPU utilization. The task can be started automatically or manually, and it must be assigned a priority to be able to react on interrupts and other communication events. Refer to Zephyr FreeMASTER sample applications which all use this communication task.

Zephyr shell and logging over FreeMASTER pipe FreeMASTER implements a shell backend which may use FreeMASTER pipe as a I/O terminal and logging output. Refer to Zephyr FreeMASTER sample applications which all use this feature.

Automatic TSA tables TSA tables can be declared as "automatic" in Zephyr which make them automatically registered in the table list. This may be very useful when there are many TSA tables or when the tables are defined in different (often unrelated) libraries linked together. In this case user does not need to build a list of all tables manually.

Driver files The driver source files can be found in a top-level src folder, further divided into the sub-folders:

- ***src/platforms*** platform-specific folder—one folder exists for each supported processor platform (for example; 32-bit Little Endian platform). Each such folder contains a platform header file with data types and a code which implements the potentially platform-specific operations, such as aligned memory access.
- ***src/common*** folder—contains the common driver source files shared by the driver for all supported platforms. All the *.c* files must be added to the project, compiled, and linked together with the application.
  - *freemaster.h* - master driver header file, which declares the common data types, macros, and prototypes of the FreeMASTER driver API functions.
  - *freemaster_cfg.h.example* - this file can serve as an example of the FreeMASTER driver configuration file. Save this file into a project source code folder and rename it to *freemaster_cfg.h*. The FreeMASTER driver code includes this file to get the project-specific configuration options and to optimize the compilation of the driver.
  - *freemaster_defcfg.h* - defines the default values for each FreeMASTER configuration option if the option is not set in the *freemaster_cfg.h* file.
  - *freemaster_protocol.h* - defines the FreeMASTER protocol constants used internally by the driver.
  - *freemaster_protocol.c* - implements the FreeMASTER protocol decoder and handles the basic Get Configuration Value, Memory Read, and Memory Write commands.

  - *freemaster_rec.c* - handles the Recorder-specific commands and implements the Recorder sampling and triggering routines. When the Recorder is disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_scope.c* - handles the Oscilloscope-specific commands. If the Oscilloscope is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_pipes.c* - implements the Pipes functionality when the Pipes feature is enabled.
  - *freemaster_appcmd.c* - handles the communication commands used to deliver and execute the Application Commands within the context of the embedded application. When the Application Commands are disabled by the FreeMASTER driver configuration file, this file only compiles to empty API functions.
  - *freemaster_tsa.c* - handles the commands specific to the TSA feature. This feature enables the FreeMASTER host tool to obtain the TSA memory descriptors declared in the embedded application. If the TSA is disabled by the FreeMASTER driver configuration file, this file compiles as void.
  - *freemaster_tsa.h* - contains the declaration of the macros used to define the TSA memory descriptors. This file is indirectly included into the user application code (via *freemaster.h*).
  - *freemaster_sha.c* - implements the SHA-1 hash code used in the password authentication algorithm.
  - *freemaster_private.h* - contains the declarations of functions and data types used internally in the driver. It also contains the C pre-processor statements to perform the compile-time verification of the user configuration provided in the *freemaster_cfg.h* file.
  - *freemaster_serial.c* - implements the serial protocol logic including the CRC, FIFO queuing, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a character-oriented API exported by the specific low-level driver.
  - *freemaster_serial.h* - defines the low-level character-oriented Serial API.
  - *freemaster_can.c* - implements the CAN protocol logic including the CAN message preparation, signalling using the first data byte in the CAN frame, and other communication-related operations. This code calls the functions of the low-level communication driver indirectly via a message-oriented API exported by the specific low-level driver.
  - *freemaster_can.h* - defines the low-level message-oriented CAN API.
  - *freemaster_net.c* - implements the Network protocol transport logic including multiple session management code.
  - *freemaster_net.h* - definitions related to the Network transport.
  - *freemaster_pdbdm.c* - implements the packet-driven BDM communication buffer and other communication-related operations.
  - *freemaster_utils.c* - aligned memory copy routines, circular buffer management and other utility functions
  - *freemaster_utils.h* - definitions related to utility code.

- ***src/drivers/[sdk]/serial*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_serial_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the UART, LPUART, USART, and other kinds of Serial communication modules.

- ***src/drivers/[sdk]/can*** - contains the code related to the serial communication implemented using one of the supported SDK frameworks.
  - *freemaster_XXX.c* and *.h* - implement low-level access to the communication peripheral registers. Different files exist for the FlexCAN, msCAN, MCAN, and other kinds of CAN communication modules.
- ***src/drivers/[sdk]/network*** - contains low-level code adapting the FreeMASTER Network transport to an underlying TCP/IP or RTT stack.
  - *freemaster_net_lwip_tcp.c* and *_udp.c* - default networking implementation of TCP and UDP transports using lwIP stack.
  - *freemaster_net_segger_rtt.c* - implementation of network transport using Segger J-Link RTT interface

Driver configuration The driver is configured using a single header file (*freemaster_cfg.h*). Create this file and save it together with other project source files before compiling the driver code. All FreeMASTER driver source files include the *freemaster_cfg.h* file and use the macros defined here for the conditional and parameterized compilation. The C compiler must locate the configuration file when compiling the driver files. Typically, it can be achieved by putting this file into a folder where the other project-specific included files are stored.

As a starting point to create the configuration file, get the *freemaster_cfg.h.example* file, rename it to *freemaster_cfg.h*, and save it into the project area.

Note: It is NOT recommended to leave the *freemaster_cfg.h* file in the FreeMASTER driver source code folder. The configuration file must be placed at a project-specific location, so that it does not affect the other applications that use the same driver.

Configurable items This section describes the configuration options which can be defined in *freemaster_cfg.h*.

Interrupt modes

```
#define FMSTR_LONG_INTR   [0|1]
#define FMSTR_SHORT_INTR  [0|1]
#define FMSTR_POLL_DRIVEN [0|1]
```

Value Type boolean (0 or 1)

Description Exactly one of the three macros must be defined to non-zero. The others must be defined to zero or left undefined. The non-zero-defined constant selects the interrupt mode of the driver. See *Driver interrupt modes*.

- FMSTR_LONG_INTR — long interrupt mode
- FMSTR_SHORT_INTR — short interrupt mode
- FMSTR_POLL_DRIVEN — poll-driven mode

Note: Some options may not be supported by all communication interfaces. For example, the FMSTR_SHORT_INTR option is not supported by the USB_CDC interface.

Protocol transport

```
#define FMSTR_TRANSPORT [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER source code. Specify one of existing instances to make use of the protocol transport.

Description Use one of the pre-defined constants, as implemented by the FreeMASTER code. The current driver supports the following transports:

- **FMSTR_SERIAL** - serial communication protocol
- **FMSTR_CAN** - using CAN communication
- **FMSTR_PDBDM** - using packet-driven BDM communication
- **FMSTR_NET** - network communication using TCP or UDP protocol

Serial transport This section describes configuration parameters used when serial transport is used:

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

FMSTR_SERIAL_DRV Select what low-level driver interface will be used when implementing the Serial communication.

```
#define FMSTR_SERIAL_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing serial driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/serial* implementation):

- **FMSTR_SERIAL_MCUX_UART** - UART driver
- **FMSTR_SERIAL_MCUX_LPUART** - LPUART driver
- **FMSTR_SERIAL_MCUX_USART** - USART driver
- **FMSTR_SERIAL_MCUX_MINIUSART** - miniUSART driver
- **FMSTR_SERIAL_MCUX_QSCI** - DSC QSCI driver
- **FMSTR_SERIAL_MCUX_USB** - USB/CDC class driver (also see code in the */support/mcuxsdk_usb* folder)
- **FMSTR_SERIAL_56F800E_EONCE** - DSC JTAG EOnCE driver

Other SDKs or BSPs may define custom low-level driver interface structure which may be used as FMSTR_SERIAL_DRV. For example:

- **FMSTR_SERIAL_DREG_UART** - demonstrates the low-level interface implemented without the MCUXpresso SDK and using direct access to peripheral registers.

FMSTR_SERIAL_BASE

```
#define FMSTR_SERIAL_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the UART, LPUART, USART, or other serial peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetSerialBaseAddress() to select the peripheral module.

FMSTR_COMM_BUFFER_SIZE

```
#define FMSTR_COMM_BUFFER_SIZE [number]
```

Value Type 0 or a value in range 32...255

Description Specify the size of the communication buffer to be allocated by the driver. Default value, which suits all driver features, is used when this option is defined as 0.

FMSTR_COMM_RQUEUE_SIZE

```
#define FMSTR_COMM_RQUEUE_SIZE [number]
```

Value Type Value in range 0...255

Description Specify the size of the FIFO receiver queue used to quickly receive and store characters in the FMSTR_SHORT_INTR interrupt mode.
The default value is 32 B.

FMSTR_SERIAL_SINGLEWIRE

```
#define FMSTR_SERIAL_SINGLEWIRE [0|1]
```

Value Type Boolean 0 or 1.

Description Set to non-zero to enable the “True” single-wire mode which uses a single MCU pin to communicate. The low-level driver enables the pin direction switching when the MCU peripheral supports it.

CAN Bus transport This section describes configuration parameters used when CAN transport is used:

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

FMSTR_CAN_DRV Select what low-level driver interface will be used when implementing the CAN communication.

```
#define FMSTR_CAN_DRV [identifier]
```

Value Type Driver identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing CAN driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/can implementation*):

- **FMSTR_CAN_MCUX_FLEXCAN** - FlexCAN driver
- **FMSTR_CAN_MCUX_MCAN** - MCAN driver
- **FMSTR_CAN_MCUX_MSCAN** - msCAN driver
- **FMSTR_CAN_MCUX_DSCFLEXCAN** - DSC FlexCAN driver
- **FMSTR_CAN_MCUX_DSCMSCAN** - DSC msCAN driver

Other SDKs or BSPs may define the custom low-level driver interface structure which may be used as FMSTR_CAN_DRV.

FMSTR_CAN_BASE

```
#define FMSTR_CAN_BASE [address|symbol]
```

Value Type Optional address value (numeric or symbolic)

Description Specify the base address of the FlexCAN, msCAN, or other CAN peripheral module to be used for the communication. This value is not defined by default. User application should call FMSTR_SetCanBaseAddress() to select the peripheral module.

FMSTR_CAN_CMDID

```
#define FMSTR_CAN_CMDID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for FreeMASTER commands (direction from PC Host tool to target application). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Default value is 0x7AA.

FMSTR_CAN_RSPID

```
#define FMSTR_CAN_RSPID [number]
```

Value Type CAN identifier (11-bit or 29-bit number)

Description CAN message identifier used for responding messages (direction from target application to PC Host tool). When declaring 29-bit identifier, combine the numeric value with FMSTR_CAN_EXTID bit. Note that both *CMDID* and *RSPID* values may be the same. Default value is 0x7AA.

FMSTR_FLEXCAN_TXMB

```
#define FMSTR_FLEXCAN_TXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame transmission. Default value is 0.

FMSTR_FLEXCAN_RXMB

```
#define FMSTR_FLEXCAN_RXMB [number]
```

Value Type Number in range of 0..N where N is number of CAN message-buffers supported by HW module.

Description Only used when the FlexCAN low-level driver is used. Define the FlexCAN message buffer for CAN frame reception. Note that the FreeMASTER driver may also operate with a common message buffer used by both TX and RX directions. Default value is 1.

Network transport This section describes configuration parameters used when Network transport is used:

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

```
#define FMSTR_NET_DRV [identifier]
```

Value Type Identifiers are structure instance names defined in FreeMASTER drivers code. Specify one of existing NET driver instances.

Description When using MCUXpresso SDK, use one of the following constants (see */drivers/mcuxsdk/network implementation*):

- **FMSTR_NET_LWIP_TCP** - TCP communication using lwIP stack
- **FMSTR_NET_LWIP_UDP** - UDP communication using lwIP stack
- **FMSTR_NET_SEGGER_RTT** - Communication using SEGGER J-Link RTT interface

Other SDKs or BSPs may define the custom networking interface which may be used as FMSTR_CAN_DRV.

Add another row below:

FMSTR_NET_PORT

```
#define FMSTR_NET_PORT [number]
```

Value Type TCP or UDP port number (short integer)

Description Specifies the server port number used by TCP or UDP protocols.

FMSTR_NET_BLOCKING_TIMEOUT

```
#define FMSTR_NET_BLOCKING_TIMEOUT [number]
```

Value Type Timeout as number of milliseconds

Description This value specifies a timeout in milliseconds for which the network socket operations may block the execution inside *FMSTR_Poll*. This may be set high (e.g. 250) when a dedicated RTOS task is used to handle FreeMASTER protocol polling. Set to a lower value when the polling task is also responsible for other operations. Set to 0 to attempt to use non-blocking socket operations.

FMSTR_NET_AUTODISCOVERY

```
#define FMSTR_NET_AUTODISCOVERY [0|1]
```

Value Type Boolean 0 or 1.

Description This option enables the FreeMASTER driver to use a separate UDP socket to broadcast auto-discovery messages to network. This helps the FreeMASTER tool to discover the target device address, port and protocol options.

Debugging options

FMSTR_DISABLE

```
#define FMSTR_DISABLE [0|1]
```

Value Type boolean (0 or 1)

Description Define as non-zero to disable all FreeMASTER features, exclude the driver code from build, and compile all its API functions empty. This may be useful to remove FreeMASTER without modifying any application source code. Default value is 0 (false).

FMSTR_DEBUG_TX

```
#define FMSTR_DEBUG_TX [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to enable the driver to periodically transmit test frames out on the selected communication interface (SCI or CAN). With the debug transmission enabled, it is simpler to detect problems in the baudrate or other communication configuration settings.

The test frames are transmitted until the first valid command frame is received from the PC Host tool. The test frame is a valid error status frame, as defined by the protocol format. On the serial line, the test frame consists of three printable characters (+©W) which are easy to capture using the serial terminal tools.

This feature requires the FMSTR_Poll() function to be called periodically. Default value is 0 (false).

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

```
#define FMSTR_USE_READMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Read command and enable FreeMASTER to have read access to memory and variables. The access can be further restricted by using a TSA feature.
Default value is 1 (true).

FMSTR_USE_WRITEMEM

```
#define FMSTR_USE_WRITEMEM [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Memory Write command.
The default value is 1 (true).

Oscilloscope options

FMSTR_USE_SCOPE

```
#define FMSTR_USE_SCOPE [number]
```

Value Type Integer number.

Description Number of Oscilloscope instances to be supported. Set to 0 to disable the Oscilloscope feature.
Default value is 0.

FMSTR_MAX_SCOPE_VARS

```
#define FMSTR_MAX_SCOPE_VARS [number]
```

Value Type Integer number larger than 2.

Description Number of variables to be supported by each Oscilloscope instance.
Default value is 8.

Recorder options

FMSTR_USE_RECORDER

```
#define FMSTR_USE_RECORDER [number]
```

Value Type Integer number.

Description Number of Recorder instances to be supported. Set to 0 to disable the Recorder feature.
Default value is 0.

FMSTR_REC_BUFF_SIZE

```
#define FMSTR_REC_BUFF_SIZE [number]
```

Value Type Integer number larger than 2.

Description Defines the size of the memory buffer used by the Recorder instance #0.
Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_TIMEBASE

```
#define FMSTR_REC_TIMEBASE [time specification]
```

Value Type Number (nanoseconds time).

Description Defines the base sampling rate in nanoseconds (sampling speed) Recorder instance #0.

Use one of the following macros:

- FMSTR_REC_BASE_SECONDS(x)
- FMSTR_REC_BASE_MILLISEC(x)
- FMSTR_REC_BASE_MICROSEC(x)
- FMSTR_REC_BASE_NANOSEC(x)

Default: not defined, user shall call 'FMSTR_RecorderCreate()" API function to specify this parameter in run time.

FMSTR_REC_FLOAT_TRIG

```
#define FMSTR_REC_FLOAT_TRIG [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the floating-point triggering. Be aware that floating-point triggering may grow the code size by linking the floating-point standard library.

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

```
#define FMSTR_USE_APPCMD [0|1]
```

Value Type Boolean 0 or 1.

Description Define as non-zero to implement the Application Commands feature.
Default value is 0 (false).

FMSTR_APPCMD_BUFF_SIZE

```
#define FMSTR_APPCMD_BUFF_SIZE [size]
```

Value Type Numeric buffer size in range 1..255

Description The size of the Application Command data buffer allocated by the driver. The buffer stores the (optional) parameters of the Application Command which waits to be processed.

FMSTR_MAX_APPCMD_CALLS

```
#define FMSTR_MAX_APPCMD_CALLS [number]
```

Value Type Number in range 0..255

Description The number of different Application Commands that can be assigned a callback handler function using FMSTR_RegisterAppCmdCall(). Default value is 0.

TSA options

FMSTR_USE_TSA

```
#define FMSTR_USE_TSA [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER TSA feature to be used. With this option enabled, the TSA tables defined in the applications are made available to the FreeMASTER host tool.
Default value is 0 (false).

FMSTR_USE_TSA_SAFETY

```
#define FMSTR_USE_TSA_SAFETY [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the memory access validation in the FreeMASTER driver. With this option, the host tool is not able to access the memory which is not described by at least one TSA descriptor. Also a write access is denied for objects defined as read-only in TSA tables.
Default value is 0 (false).

FMSTR_USE_TSA_INROM

```
#define FMSTR_USE_TSA_INROM [0|1]
```

Value Type Boolean 0 or 1.

Description Declare all TSA descriptors as *const*, which enables the linker to put the data into the flash memory. The actual result depends on linker settings or the linker commands used in the project.
Default value is 0 (false).

FMSTR_USE_TSA_DYNAMIC

```
#define FMSTR_USE_TSA_DYNAMIC [0|1]
```

Value Type Boolean 0 or 1.

Description Enable runtime-defined TSA entries to be added to the TSA table by the FMSTR_SetUpTsaBuff() and FMSTR_TsaAddVar() functions.
Default value is 0 (false).

Pipes options

FMSTR_USE_PIPES

```
#define FMSTR_USE_PIPES [0|1]
```

Value Type Boolean 0 or 1.

Description Enable the FreeMASTER Pipes feature to be used.
Default value is 0 (false).

FMSTR_MAX_PIPES_COUNT

```
#define FMSTR_MAX_PIPES_COUNT [number]
```

Value Type Number in range 1..63.

Description The number of simultaneous pipe connections to support.
The default value is 1.

Driver interrupt modes To implement the communication, the FreeMASTER driver handles the Serial or CAN module's receive and transmit requests. Use the *freemaster_cfg.h* configuration file to select whether the driver processes the communication automatically in the interrupt service routine handler or if it only polls the status of the module (typically during the application idle time).

This section describes each of the interrupt mode in more details.

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

In this mode, both the communication and the FreeMASTER protocol decoding is done in the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, or other interrupt service routine. Because the protocol execution may be a lengthy task (especially with the TSA-Safety enabled) it is recommended to use this mode only if the interrupt prioritization scheme is possible in the application and the FreeMASTER interrupt is assigned to a lower (the lowest) priority.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from that handler.

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

In this mode, the communication processing time is split between the interrupt routine and the main application loop or task. The raw communication is handled by the *FMSTR_SerialIsr, FMSTR_CanIsr*, or other interrupt service routine, while the protocol decoding and execution is handled by the *FMSTR_Poll* routine. Call *FMSTR_Poll* during the idle time in the application main loop.

The interrupt processing in this mode is relatively fast and deterministic. Upon a serial-receive event, the received character is only placed into a FIFO-like queue and it is not further processed. Upon a CAN receive event, the received frame is stored into a receive buffer. When transmitting, the characters are fetched from the prepared transmit buffer.

In this mode, the application code must register its own interrupt handler for all interrupt vectors related to the selected communication interface and call the *FMSTR_SerialIsr* or *FMSTR_CanIsr* functions from that handler.

When the serial interface is used as the serial communication interface, ensure that the *FMSTR_Poll* function is called at least once per *N* character time periods. *N* is the length of the FreeMASTER FIFO queue (*FMSTR_COMM_RQUEUE_SIZE*) and the character time is the time needed to transmit or receive a single byte over the SCI line.

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

In this mode, both the communication and the FreeMASTER protocol decoding are done in the *FMSTR_Poll* routine. No interrupts are needed and the *FMSTR_SerialIsr*, *FMSTR_CanIsr*, and similar handlers compile to an empty code.

When using this mode, ensure that the *FMSTR_Poll* function is called by the application at least once per the serial "character time" which is the time needed to transmit or receive a single character.

In the latter two modes (*FMSTR_SHORT_INTR* and *FMSTR_POLL_DRIVEN*), the protocol handling takes place in the FMSTR_Poll routine. An application interrupt can occur in the middle of the Read Memory or Write Memory commands' execution and corrupt the variable being accessed by the FreeMASTER driver. In these two modes, some issues or glitches may occur when using FreeMASTER to visualize or monitor volatile variables modified in interrupt servicing code.

The same issue may appear even in the full interrupt mode (FMSTR_LONG_INTR), if volatile variables are modified in the interrupt code with a priority higher than the priority of the communication interrupt.

Data types Simple portability was one of the main requirements when writing the FreeMASTER driver. This is why the driver code uses the privately-declared data types and the vast majority of the platform-dependent code is separated in the platform-dependent source files. The data types used in the driver API are all defined in the platform-specific header file.

To prevent name conflicts with the symbols used in the application, all data types, macros, and functions have the FMSTR_ prefix. The only global variables used in the driver are the transport and low-level API structures exported from the driver-implementation layer to upper layers. Other than that, all private variables are declared as static and named using the fmstr_ prefix.

Communication interface initialization The FreeMASTER driver does not perform neither the initialization nor the configuration of the peripheral module that it uses to communicate. It is the application startup code responsibility to configure the communication module before the FreeMASTER driver is initialized by the FMSTR_Init call.

When the Serial communication module is used as the FreeMASTER communication interface, configure the UART receive and transmit pins, the serial communication baud rate, parity (no-parity), the character length (eight bits), and the number of stop bits (one) before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected serial peripheral module. Call the FMSTR_SerialIsr function from the application handler.

When a CAN module is used as the FreeMASTER communication interface, configure the CAN receive and transmit pins and the CAN module bit rate before initializing the FreeMASTER driver. For either the long or the short interrupt modes of the driver (see *Driver interrupt modes*), configure the interrupt controller and register an application-specific interrupt handler for all interrupt sources related to the selected CAN peripheral module. Call the FMSTR_CanIsr function from the application handler.

Note: It is not necessary to enable or unmask the serial nor the CAN interrupts before initializing the FreeMASTER driver. The driver enables or disables the interrupts and communication lines, as required during runtime.

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set

up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

In applications where FMSTR_Recorder is called periodically with a constant period, specify the period in the Recorder configuration structure before calling FMSTR_RecorderCreate. This setting enables the PC Host FreeMASTER tool to display the X-axis of the Recorder graph properly scaled for the time domain.

Driver usage Start using or evaluating FreeMASTER by opening some of the example applications available in the driver setup package.

Follow these steps to enable the basic FreeMASTER connectivity in the application:

- Make sure that all **.c* files of the FreeMASTER driver from the *src/common/platforms/[your_platform]* folder are a part of the project. See *Driver files* for more details.
- Configure the FreeMASTER driver by creating or editing the *freemaster_cfg.h* file and by saving it into the application project directory. See *Driver configuration* for more details.
- Include the *freemaster.h* file into any application source file that makes the FreeMASTER API calls.
- Initialize the Serial or CAN modules. Set the baud rate, parity, and other parameters of the communication. Do not enable the communication interrupts in the interrupt mask registers.
- For the FMSTR_LONG_INTR and FMSTR_SHORT_INTR modes, install the application-specific interrupt routine and call the FMSTR_SerialIsr or FMSTR_CanIsr functions from this handler.
- Call the FMSTR_Init function early on in the application initialization code.
- Call the FMSTR_RecorderCreate functions for each Recorder instance to enable the Recorder feature.
- In the main application loop, call the FMSTR_Poll API function periodically when the application is idle.
- For the FMSTR_SHORT_INTR and FMSTR_LONG_INTR modes, enable the interrupts globally so that the interrupts can be handled by the CPU.

Communication troubleshooting The most common problem that causes communication issues is a wrong baud rate setting or a wrong pin multiplexer setting of the target MCU. When a communication between the PC Host running FreeMASTER and the target MCU cannot be established, try enabling the FMSTR_DEBUG_TX option in the *freemaster_cfg.h* file and call the FMSTR_Poll function periodically in the main application task loop.

With this feature enabled, the FreeMASTER driver periodically transmits a test frame through the Serial or CAN lines. Use a logic analyzer or an oscilloscope to monitor the signals at the communication pins of the CPU device to examine whether the bit rate and signal polarity are configured properly.

Driver API

This section describes the driver Application Programmers' Interface (API) needed to initialize and use the FreeMASTER serial communication driver.

Control API There are three key functions to initialize and use the driver.

FMSTR_Init

Prototype

```
FMSTR_BOOL FMSTR_Init(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description This function initializes the internal variables of the FreeMASTER driver and enables the communication interface. This function does not change the configuration of the selected communication module. The hardware module must be initialized before the *FMSTR_Init* function is called.

A call to this function must occur before calling any other FreeMASTER driver API functions.

FMSTR_Poll

Prototype

```
void FMSTR_Poll(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_protocol.c*

Description In the poll-driven or short interrupt modes, this function handles the protocol decoding and execution (see *Driver interrupt modes*). In the poll-driven mode, this function also handles the communication interface with the PC. Typically, the *FMSTR_Poll* function is called during the "idle" time in the main application task loop.

To prevent the receive data overflow (loss) on a serial interface, make sure that the FMSTR_Poll function is called at least once per the time calculated as:

*N * Tchar*

where:

- *N* is equal to the length of the receive FIFO queue (configured by the FMSTR_COMM_RQUEUE_SIZE macro). *N* is 1 for the poll-driven mode.
- *Tchar* is the character time, which is the time needed to transmit or receive a single byte over the SCI line.

Note: In the long interrupt mode, this function typically compiles as an empty function and can still be called. It is worthwhile to call this function regardless of the interrupt mode used in the application. This approach enables a convenient switching between the different interrupt modes only by changing the configuration macros in the *freemaster_cfg.h* file.

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

```
void FMSTR_SerialIsr(void);
void FMSTR_CanIsr(void);
```

- Declaration: *freemaster.h*
- Implementation: *hw-specific low-level driver C file*

Description This function contains the interrupt-processing code of the FreeMASTER driver. In long or short interrupt modes (see *Driver interrupt modes*), this function must be called from the application interrupt service routine registered for the communication interrupt vector. On platforms where the communication module uses multiple interrupt vectors, the application should register a handler for all vectors and call this function at each interrupt.

Note: In a poll-driven mode, this function is compiled as an empty function and does not have to be used.

Recorder API

FMSTR_RecorderCreate

Prototype

```
FMSTR_BOOL FMSTR_RecorderCreate(FMSTR_INDEX recIndex, FMSTR_REC_BUFF* buffCfg);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function registers a recorder instance and enables it to be used by the PC Host tool. Call this function for all recorder instances from 0 to the maximum number defined by the FMSTR_USE_RECORDER configuration option (minus one). An exception to this requirement is the recorder of instance *0* which may be automatically configured by FMSTR_Init when the *freemaster_cfg.h* configuration file defines the *FMSTR_REC_BUFF_SIZE* and *FMSTR_REC_TIMEBASE* options.

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

```
void FMSTR_Recorder(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function takes a sample of the variables being recorded using the FreeMASTER Recorder instance *recIndex*. If the selected Recorder is not active when the *FMSTR_Recorder* function is being called, the function returns immediately. When the Recorder is active, the values of the variables being recorded are copied into the recorder buffer and the trigger conditions are evaluated.

If a trigger condition is satisfied, the Recorder enters the post-trigger mode, where it counts down the follow-up samples (number of *FMSTR_Recorder* function calls) and de-activates the Recorder when the required post-trigger samples are finished.

The *FMSTR_Recorder* function is typically called in the timer or PWM interrupt service routines. This function can also be called in the application main loop (for testing purposes).

FMSTR_RecorderTrigger

Prototype

```
void FMSTR_RecorderTrigger(FMSTR_INDEX recIndex);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_rec.c*

Description This function forces the Recorder trigger condition to happen, which causes the Recorder to be automatically deactivated after the post-trigger samples are sampled. Use this function in the application code for programmatic control over the Recorder triggering. This can be useful when a more complex triggering conditions need to be used.

Fast Recorder API The Fast Recorder feature is not available in the FreeMASTER driver version 3. This feature was heavily dependent on the target platform and it was only available for the 56F8xxxx DSCs.

TSA Tables When the TSA is enabled in the FreeMASTER driver configuration file (by setting the FMSTR_USE_TSA macro to a non-zero value), it defines the so-called TSA tables in the application. This section describes the macros that must to be used to define the TSA tables.

There can be any number of TSA tables spread across the application source files. There must be always exactly one TSA Table List defined, which informs the FreeMASTER driver about the active TSA tables.

When there is at least one TSA table and one TSA Table List defined in the application, the TSA information automatically appears in the FreeMASTER symbols list. The symbols can then be used to create FreeMASTER variables for visualization or control.

TSA table definition The TSA table describes the static or global variables together with their address, size, type, and access-protection information. If the TSA-described variables are of a structure type, the TSA table may also describe this type and provide an access to the individual structure members of the variable.

The TSA table definition begins with the FMSTR_TSA_TABLE_BEGIN macro with a *table_id* identifying the table. The *table_id* shall be a valid C-langiage symbol.

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

After this opening macro, the TSA descriptors are placed using these macros:

```
/* Adding variable descriptors */
FMSTR_TSA_RW_VAR(name, type)  /* read/write variable entry */
FMSTR_TSA_RO_VAR(name, type)  /* read-only variable entry */

/* Description of complex data types */
FMSTR_TSA_STRUCT(struct_name) /* structure or union type entry */
```

(continues on next page)

(continued from previous page)

```
FMSTR_TSA_MEMBER(struct_name, member_name, type)  /* structure member entry */

/* Memory blocks */
FMSTR_TSA_RW_MEM(name, type, address, size) /* read/write memory block */
FMSTR_TSA_RO_MEM(name, type, address, size) /* read-only memory block */
```

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

- *name* — variable name. The variable must be defined before the TSA descriptor references it.
- *type* — variable or member type. Only one of the pre-defined type constants may be used (see below).
- *struct_name* — structure type name. The type must be defined (typedef) before the TSA descriptor references it.
- *member_name* — structure member name.

Note: The structure member descriptors (FMSTR_TSA_MEMBER) must immediately follow the parent structure descriptor (FMSTR_TSA_STRUCT) in the table.

Note: To write-protect the variables in the FreeMASTER driver (FMSTR_TSA_RO_VAR), enable the TSA-Safety feature in the configuration file.

TSA variable types The table lists *type* identifiers which can be used in TSA descriptors:

Constant	Description
FMSTR_TSA_UINT*n*	Unsigned integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_SINT*n*	Signed integer type of size *n* bits (n=8,16,32,64)
FMSTR_TSA_FRAC*n*	Fractional number of size *n* bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(*m,n*)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(*m,n*)	Unsigned fractional number in general UQ form (m+n total bits)
FMSTR_TSA_FLOAT	4-byte standard IEEE floating-point type
FMSTR_TSA_DOUBLE	8-byte standard IEEE floating-point type
FMSTR_TSA_POINTER	Generic pointer type defined (platform-specific 16 or 32 bit)
FMSTR_TSA_USERTYPE(*name*)	Structure or union type declared with FMSTR_TSA_STRUCT record

TSA table list There shall be exactly one TSA Table List in the application. The list contains one entry for each TSA table defined anywhere in the application.

The TSA Table List begins with the FMSTR_TSA_TABLE_LIST_BEGIN macro and continues with the TSA table entries for each table.

```
FMSTR_TSA_TABLE_LIST_BEGIN()

FMSTR_TSA_TABLE(table_id)
FMSTR_TSA_TABLE(table_id2)
FMSTR_TSA_TABLE(table_id3)
...
```

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

TSA Active Content entries FreeMASTER v2.0 and higher supports TSA Active Content, enabling the TSA tables to describe the memory-mapped files, virtual directories, and URL hyperlinks. FreeMASTER can access such objects similarly to accessing the files and folders on the local hard drive.

With this set of TSA entries, the FreeMASTER pages can be embedded directly into the target MCU flash and accessed by FreeMASTER directly over the communication line. The HTML-coded pages rendered inside the FreeMASTER window can access the TSA Active Content resources using a special URL referencing the *fmstr:* protocol.

This example provides an overview of the supported TSA Active Content entries:

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

/* Directory entry applies to all subsequent MEMFILE entries */
FMSTR_TSA_DIRECTORY("/text_files")     /* entering a new virtual directory */

/* The readme.txt file will be accessible at the fmstr://text_files/readme.txt URL */
FMSTR_TSA_MEMFILE("readme.txt", readme_txt, sizeof(readme_txt)) /* memory-mapped file */

/* Files can also be specified with a full path so the DIRECTORY entry does not apply */
FMSTR_TSA_MEMFILE("/index.htm", index, sizeof(index))          /* memory-mapped file */
FMSTR_TSA_MEMFILE("/prj/demo.pmp", demo_pmp, sizeof(demo_pmp)) /* memory-mapped file */

/* Hyperlinks can point to a local MEMFILE object or to the Internet */
FMSTR_TSA_HREF("Board's Built-in Welcome Page", "/index.htm")
FMSTR_TSA_HREF("FreeMASTER Home Page", "http://www.nxp.com/freemaster")

/* Project file links simplify opening the projects from any URLs */
FMSTR_TSA_PROJECT("Demonstration Project (embedded)", "/prj/demo.pmp")
FMSTR_TSA_PROJECT("Full Project (online)", "http://mycompany.com/prj/demo.pmp")

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

```
FMSTR_BOOL FMSTR_SetUpTsaBuff(FMSTR_ADDR buffAddr, FMSTR_SIZE buffSize);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *buffAddr* [in] - address of the memory buffer for the dynamic TSA table
- *buffSize* [in] - size of the memory buffer which determines the maximum number of TSA entries to be added in the runtime

Description This function must be used to assign the RAM memory buffer to the TSA subsystem when FMSTR_USE_TSA_DYNAMIC is enabled. The memory buffer is then used to store the TSA entries added dynamically to the runtime TSA table using the FMSTR_TsaAddVar function call. The runtime TSA table is processed by the FreeMASTER PC Host tool along with all static tables as soon as the communication port is open.

The size of the memory buffer determines the number of TSA entries that can be added dynamically. Depending on the MCU platform, one TSA entry takes either 8 or 16 bytes.

FMSTR_TsaAddVar

Prototype

```
FMSTR_BOOL FMSTR_TsaAddVar(FMSTR_TSATBL_STRPTR tsaName, FMSTR_TSATBL_STRPTR
↪tsaType,
    FMSTR_TSATBL_VOIDPTR varAddr, FMSTR_SIZE32 varSize,
    FMSTR_SIZE flags);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_tsa.c*

Arguments

- *tsaName* [in] - name of the object
- *tsaType* [in] - name of the object type
- *varAddr* [in] - address of the object
- *varSize* [in] - size of the object
- *flags* [in] - access flags; a combination of these values:
  - *FMSTR_TSA_INFO_RO_VAR* — read-only memory-mapped object (typically a variable)
  - *FMSTR_TSA_INFO_RW_VAR* — read/write memory-mapped object
  - *FMSTR_TSA_INFO_NON_VAR* — other entry, describing structure types, structure members, enumerations, and other types

Description This function can be called only when the dynamic TSA table is enabled by the FMSTR_USE_TSA_DYNAMIC configuration option and when the FMSTR_SetUpTsaBuff function call is made to assign the dynamic TSA table memory. This function adds an entry into the dynamic TSA table. It can be used to register a read-only or read/write memory object or describe an item of the user-defined type.

See *TSA table definition* for more details about the TSA table entries.

Application Commands API

FMSTR_GetAppCmd

Prototype

```
FMSTR_APPCMD_CODE FMSTR_GetAppCmd(void);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Description This function can be used to detect if there is an Application Command waiting to be processed by the application. If no command is pending, this function returns the FMSTR_APPCMDRESULT_NOCMD constant. Otherwise, this function returns the code of the Application Command that must be processed. Use the FMSTR_AppCmdAck call to acknowledge the Application Command after it is processed and to return the appropriate result code to the host.

The FMSTR_GetAppCmd function does not report the commands for which a callback handler function exists. If the FMSTR_GetAppCmd function is called when a callback-registered command is pending (and before it is actually processed by the callback function), this function returns FMSTR_APPCMDRESULT_NOCMD.

FMSTR_GetAppCmdData

Prototype

```
FMSTR_APPCMD_PDATA FMSTR_GetAppCmdData(FMSTR_SIZE* dataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *dataLen* [out] - pointer to the variable that receives the length of the data available in the buffer. It can be NULL when this information is not needed.

Description This function can be used to retrieve the Application Command data when the application determines that an Application Command is pending (see *FMSTR_GetAppCmd*).

There is just a single buffer to hold the Application Command data (the buffer length is FMSTR_APPCMD_BUFF_SIZE bytes). If the data are to be used in the application after the command is processed by the FMSTR_AppCmdAck call, copy the data out to a private buffer.

FMSTR_AppCmdAck

Prototype

```
void FMSTR_AppCmdAck(FMSTR_APPCMD_RESULT resultCode);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultCode* [in] - the result code which is to be returned to FreeMASTER

Description This function is used when the Application Command processing finishes in the application. The resultCode passed to this function is returned back to the host and the driver is re-initialized to expect the next Application Command.

After this function is called and before the next Application Command arrives, the return value of the FMSTR_GetAppCmd function is FMSTR_APPCMDRESULT_NOCMD.

FMSTR_AppCmdSetResponseData

Prototype

```
void FMSTR_AppCmdSetResponseData(FMSTR_ADDR resultDataAddr, FMSTR_SIZE resultDataLen);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *resultDataAddr* [in] - pointer to the data buffer that is to be copied to the Application Command data buffer
- *resultDataLen* [in] - length of the data to be copied. It must not exceed the FMSTR_APPCMD_BUFF_SIZE value.

Description This function can be used before the Application Command processing finishes, when there are data to be returned back to the PC.

The response data buffer is copied into the Application Command data buffer, from where it is accessed when the host requires it. Do not use FMSTR_GetAppCmdData and the data buffer after FMSTR_AppCmdSetResponseData is called.

Note: The current version of FreeMASTER does not support the Application Command response data.

FMSTR_RegisterAppCmdCall

Prototype

```
FMSTR_BOOL FMSTR_RegisterAppCmdCall(FMSTR_APPCMD_CODE appCmdCode, FMSTR_
↪PAPPCMDFUNC callbackFunc);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_appcmd.c*

Arguments

- *appCmdCode* [in] - the Application Command code for which the callback is to be registered
- *callbackFunc* [in] - pointer to the callback function that is to be registered. Use NULL to unregister a callback registered previously with this Application Command.

Return value This function returns a non-zero value when the callback function was successfully registered or unregistered. It can return zero when trying to register a callback function for more than FMSTR_MAX_APPCMD_CALLS different Application Commands.

Description This function can be used to register the given function as a callback handler for the Application Command. The Application Command is identified using single-byte code. The callback function is invoked automatically by the FreeMASTER driver when the protocol decoder obtains a request to get the application command result code.

The prototype of the callback function is

```
FMSTR_APPCMD_RESULT HandlerFunction(FMSTR_APPCMD_CODE nAppcmd,
    FMSTR_APPCMD_PDATA pData, FMSTR_SIZE nDataLen);
```

Where:

- *nAppcmd* -Application Command code
- *pData* —points to the Application Command data received (if any)
- *nDataLen* —information about the Application Command data length

The return value of the callback function is used as the Application Command Result Code and returned to FreeMASTER.

Note: The FMSTR_MAX_APPCMD_CALLS configuration macro defines how many different Application Commands may be handled by a callback function. When FMSTR_MAX_APPCMD_CALLS is undefined or defined as zero, the FMSTR_RegisterAppCmdCall function always fails.

Pipes API

FMSTR_PipeOpen

Prototype

```
FMSTR_HPIPE FMSTR_PipeOpen(FMSTR_PIPE_PORT pipePort, FMSTR_PPIPEFUNC pipeCallback,
↪
    FMSTR_ADDR pipeRxBuff, FMSTR_PIPE_SIZE pipeRxSize,
    FMSTR_ADDR pipeTxBuff, FMSTR_PIPE_SIZE pipeTxSize,
    FMSTR_U8 type, const FMSTR_CHAR *name);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipePort* [in] - port number that identifies the pipe for the client
- *pipeCallback* [in] - pointer to the callback function that is called whenever a pipe data status changes
- *pipeRxBuff* [in] - address of the receive memory buffer
- *pipeRxSize* [in] - size of the receive memory buffer
- *pipeTxBuff* [in] - address of the transmit memory buffer
- *pipeTxSize* [in] - size of the transmit memory buffer
- *type* [in] - a combination of FMSTR_PIPE_MODE_xxx and FMSTR_PIPE_SIZE_xxx constants describing primary pipe data format and usage. This type helps FreeMASTER decide how to access the pipe by default. Optional, use 0 when undetermined.
- *name* [in] - user name of the pipe port. This name is visible to the FreeMASTER user when creating the graphical pipe interface.

Description This function initializes a new pipe and makes it ready to accept or send the data to the PC Host client. The receive memory buffer is used to store the received data before they are read out by the FMSTR_PipeRead call. When this buffer gets full, the PC Host client denies the data transmission into this pipe until there is enough free space again. The transmit memory buffer is used to store the data transmitted by the application to the PC Host client using the FMSTR_PipeWrite call. The transmit buffer can get full when the PC Host is disconnected or when it is slow in receiving and reading out the pipe data.

The function returns the pipe handle which must be stored and used in the subsequent calls to manage the pipe object.

The callback function (if specified) is called whenever new data are received through the pipe and available for reading. This callback is also called when the data waiting in the transmit buffer are successfully pushed to the PC Host and the transmit buffer free space increases. The prototype of the callback function provided by the user application must be as follows. The *PipeHandler* name is only a placeholder and must be defined by the application.

```
void PipeHandler(FMSTR_HPIPE pipeHandle);
```

FMSTR_PipeClose

Prototype

```
void FMSTR_PipeClose(FMSTR_HPIPE pipeHandle);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call

Description This function de-initializes the pipe object. No data can be received or sent on the pipe after this call.

FMSTR_PipeWrite

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeWrite(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE writeGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data to be written
- *pipeDataLen* [in] - length of the data to be written
- *writeGranularity* [in] - size of the minimum unit of data which is to be written

Description This function puts the user-specified data into the pipe's transmit memory buffer and schedules it for transmission. This function returns the number of bytes that were successfully written into the buffer. This number may be smaller than the number of the requested bytes if there is not enough free space in the transmit buffer.

The *writeGranularity* argument can be used to split the data into smaller chunks, each of the size given by the *writeGranularity* value. The FMSTR_PipeWrite function writes as many data chunks as possible into the transmit buffer and does not attempt to write an incomplete chunk.

This feature can prove to be useful to avoid the intermediate caching when writing an array of integer values or other multi-byte data items. When making the nGranularity value equal to the nLength value, all data are considered as one chunk which is either written successfully as a whole or not at all. The nGranularity value of 0 or 1 disables the data-chunk approach.

FMSTR_PipeRead

Prototype

```
FMSTR_PIPE_SIZE FMSTR_PipeRead(FMSTR_HPIPE pipeHandle, FMSTR_ADDR pipeData,
    FMSTR_PIPE_SIZE pipeDataLen, FMSTR_PIPE_SIZE readGranularity);
```

- Declaration: *freemaster.h*
- Implementation: *freemaster_pipes.c*

Arguments

- *pipeHandle* [in] - pipe handle returned from the FMSTR_PipeOpen function call
- *pipeData* [in] - address of the data buffer to be filled with the received data
- *pipeDataLen* [in] - length of the data to be read
- *readGranularity* [in] - size of the minimum unit of data which is to be read

Description This function copies the data received from the pipe from its receive buffer to the user buffer for further processing. The function returns the number of bytes that were successfully copied to the buffer. This number may be smaller than the number of the requested bytes if there is not enough data bytes available in the receive buffer.

The readGranularity argument can be used to copy the data in larger chunks in the same way as described in the FMSTR_PipeWrite function.

API data types This section describes the data types used in the FreeMASTER driver. The information provided here can be useful when modifying or porting the FreeMASTER Communication Driver to new NXP platforms.

Note: The licensing conditions prohibit use of FreeMASTER and the FreeMASTER Communication Driver with non-NXP MPU or MCU products.

Public common types The table below describes the public data types used in the FreeMASTER driver API calls. The data types are declared in the *freemaster.h* header file.

Type name	Description
FMSTR_ADDR	Data type used to hold the memory address. On most platforms, this is normally a C-pointer, but it may also be a pure integer type.
For example, this type is defined as long integer on the 56F8xxx platform where the 24-bit addresses must be supported, but the C-pointer may be only 16 bits wide in some compiler configurations.	
FMSTR_SIZE	Data type used to hold the memory block size.
It is required that this type is unsigned and at least 16 bits wide integer.	
FMSTR_BOOL	Data type used as a general boolean type.
This type is used only in zero/non-zero conditions in the driver code.	
FMSTR_APPCM	Data type used to hold the Application Command code.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to create the Application Command data buffer.
Generally, this is an unsigned 8-bit value.	
FMSTR_APPCM	Data type used to hold the Application Command result code.
Generally, this is an unsigned 8-bit value.	

Public TSA types The table describes the TSA-specific public data types. These types are declared in the *freemaster_tsa.h* header file, which is included in the user application indirectly by the *freemaster.h* file.

FMSTR_TSA_TII	Data type used to hold a descriptor index in the TSA table or a table index in the list of TSA tables.
By default, this is defined as FMSTR_SIZE.	
FMSTR_TSA_TS	Data type used to hold a memory block size, as used in the TSA descriptors.
By default, this is defined as FMSTR_SIZE.	

Public Pipes types The table describes the data types used by the FreeMASTER Pipes API:

FMSTR_HPIPE	Pipe handle that identifies the open-pipe object.
Generally, this is a pointer to a void type.	
FMSTR_PIPE_P(	Integer type required to hold at least 7 bits of data.
Generally, this is an unsigned 8-bit or 16-bit type.	
FMSTR_PIPE_SI	Integer type required to hold at least 16 bits of data.
This is used to store the data buffer sizes.	
FMSTR_PPIPEF	Pointer to the pipe handler function.
See *FMSTR_PipeOpen* for more details.	

Internal types The table describes the data types used internally by the FreeMASTER driver. The data types are declared in the platform-specific header file and they are not available in the application code.

FMSTR_U8	The smallest memory entity. On the vast majority of platforms, this is an unsigned 8-bit integer. On the 56F8xx DSP platform, this is defined as an unsigned 16-bit integer.
FMSTR_U16	Unsigned 16-bit integer.
FMSTR_U32	Unsigned 32-bit integer.
FMSTR_S8	Signed 8-bit integer.
FMSTR_S16	Signed 16-bit integer.
FMSTR_S32	Signed 32-bit integer.
FMSTR_FLOAT	4-byte standard IEEE floating-point type.
FMSTR_FLAGS	Data type forming a union with a structure of flag bit-fields.
FMSTR_SIZE8	Data type holding a general size value, at least 8 bits wide.
FMSTR_INDEX	General for-loop index. Must be signed, at least 16 bits wide.
FMSTR_BCHR	A single character in the communication buffer. Typically, this is an 8-bit unsigned integer, except for the DSP platforms where it is a 16-bit integer.
FMSTR_BPTR	A pointer to the communication buffer (an array of FMSTR_BCHR).

Document references

Links

- This document online: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/middleware/freemaster/doc/index.html

- FreeMASTER tool home: www.nxp.com/freemaster
- FreeMASTER community area: community.nxp.com/community/freemaster
- FreeMASTER GitHub code repo: https://github.com/nxp-mcuxpresso/mcux-freemaster
- MCUXpresso SDK home: www.nxp.com/mcuxpresso
- MCUXpresso SDK builder: mcuxpresso.nxp.com/en

Documents

- *FreeMASTER Usage Serial Driver Implementation* (document AN4752)
- *Integrating FreeMASTER Time Debugging Tool With CodeWarrior For Microcontrollers v10.X Project* (document AN4771)
- *Flash Driver Library For MC56F847xx And MC56F827xx DSC Family* (document AN4860)

Revision history This Table summarizes the changes done to this document since the initial release.

Revision	Date	Description
1.0	03/2006	Limited initial release
2.0	09/2007	Updated for FreeMASTER version. New Freescale document template used.
2.1	12/2007	Added description of the new Fast Recorder feature and its API.
2.2	04/2010	Added support for MPC56xx platform, Added new API for use CAN interface.
2.3	04/2011	Added support for Kxx Kinetis platform and MQX operating system.
2.4	06/2011	Serial driver update, adds support for USB CDC interface.
2.5	08/2011	Added Packet Driven BDM interface.
2.7	12/2013	Added FLEXCAN32 interface, byte access and isr callback configuration option.
2.8	06/2014	Removed obsolete license text, see the software package content for up-to-date license.
2.9	03/2015	Update for driver version 1.8.2 and 1.9: FreeMASTER Pipes, TSA Active Content, LIN Transport Layer support, DEBUG-TX communication troubleshooting, Kinetis SDK support.
3.0	08/2016	Update for driver version 2.0: Added support for MPC56xx, MPC57xx, KEAxx and S32Kxx platforms. New NXP document template as well as new license agreement used. added MCAN interface. Folders structure at the installation destination was rearranged.
4.0	04/2019	Update for driver released as part of FreeMASTER v3.0 and MCUXpresso SDK 2.6. Updated to match new V4 serial communication protocol and new configuration options. This version of the document removes substantial portion of outdated information related to S08, S12, ColdFire, Power and other legacy platforms.
4.1	04/2020	Minor update for FreeMASTER driver included in MCUXpresso SDK 2.8.
4.2	09/2020	Added example applications description and information about the MCUXpresso Config Tools. Fixed the pipe-related API description.
4.3	10/2024	Added description of Network and Segger J-Link RTT interface configuration. Accompanying the MCUXpresso SDK version 24.12.00.
4.4	04/2025	Added Zephyr-specific information. Accompanying the MCUXpresso SDK version 25.06.00.

1.6 Multimedia

1.6.1 Audio Voice

Audio Voice Components

MCUXpresso SDK : audio-voice-components

Overview This repository is for MCUXpresso SDK audio-voice-components middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

Visit Audio Voice Components - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution Contributions are not currently accepted. Guidelines to contribute will be posted in the future.

Overview This repository allows users to add additional functionality to the Maestro Audio framework. This structure is designed for integration with Maestro and is not intended for standalone use. For information on the use of individual components, please refer to the Maestro programmer's guide.

This repository acts as Zephyr module, to be able to use these libraries in Zephyr build system.

Content

- *asrc* - Libraries and public files of Asynchronous Sample Rate Converter, version 1.0.0
- *ssrc* - Libraries and public files of Synchronous Sample Rate Converter, version 1.0.0
- *opus* - Source files of Opus decoder and encoder, version 1.3.1
- *opusfile* - Source files for Opus streams in the Ogg container, version 0.12
- *ogg* - Source files of Ogg container, version 1.3.5
- *decoders* - Libraries and public files of following audio decoders:
  - *aac* - AAC decoder, version 1.0.0
  - *flac* - FLAC decoder, version 1.0.0
  - *mp3* - MP3 decoder, version 1.0.0
  - *wav* - WAV decoder, version 1.0.0
- zephyr/ - Files allowing usage of the libraries in Zephyr build

Following table contains information about libraries and source files availability:

Asynchronous Sample Rate Converter The Asynchronous Sample Rate Converter (ASRC) software module compensates the drift between two mono audio signals. This is not a frequency converter and so the nominal signal frequency is the same before and after the ASRC. More details about ASRC are available in the User Guide, which is located in asrc\doc\.

Synchronous Sample Rate Converter The Synchronous Sample Rate Converter (SSRC) software module converts an audio signal (mono or stereo) with a certain sampling frequency to an audio signal with another sampling frequency. More details about SSRC are available in the *User Guide*.

Opus For Opus decoder and encoder documentation please see following link: opus.

Opus File The Opus File provides a API for decoding and basic manipulation of Opus streams in Ogg container and depends on *Opus* and *Ogg* libraries. For Opus File documentation please see following link: opusfile.

Ogg Container For Ogg container documentation please see following link: ogg.

Decoders Each decoder contains libraries for supported processor and toolchain (see table above), corresponding Public API file and documentation folder.

AAC For decoder features please see *aacdec*, for API Usage please see *aacd_ug*.

FLAC For decoder features please see *flacdec*, for API Usage please see *flacd_ug*.

MP3 For decoder features please see *mp3dec*, for API Usage please see *mp3d_ug*.

WAV For decoder features please see *wavdec*, for API Usage please see *wavd_ug*.

Zephyr build To add library into the Zephyr build, add CONFIG_NXP_AUDIO_VOICE_COMPONENTS_* for specific libraries into your prj.conf. For all configuration options, see *zephyr/Kconfig*.

List of supported libraries in Zephyr:

- Decoders:
  - AAC
  - FLAC
  - MP3
  - FLAC
  - OPUS
- Encoders
  - OPUS

AAC decoder

AAC decoder features

- The AAC decoder implementation supports the following:
- Supported profile : AAC-LC
- Sampling rate : 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, 48 kHz
- Channel : stereo and mono
- Bits per samples : 16 bit
- Container format : (MPEG-2 Style)AAC transport format - ADTS and ADIF.

Specification and reference

Performance

Memory information The memory usage of the decoder in bytes is:

- Code/flash = 26332 + 19264 = 45596
- Data/RAM = 26832

Section	Size
.text	26332
.ro & .const	19264
.bss	26832

CPU usage

- CPU core clock in MHz: 20.97.

Track type	Duration of track in second	Frame size in bytes	Performance MIPS of codec (in MHz)
48 kHz, stereo	38 s	4096	12.2 MHz

API Usage of AAC Decoder

Overview

- This section describes the integration steps to call AAC decoder APIs by the application code. During each step, the used data structures and functions are explained. All CCI public APIs are defined in aac_cci.h header file. This file is located at \decoders\aac.

Configuration

Build Options AAC Decoder library is built with the following defined/enabled macros.

- There is no macro or define used to build the AAC decoder.

Buffer Allocation

- The AAC decoder does not perform dynamic memory allocation. The application calls the function AACDecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder, then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- AACDecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which is used by the decoder to read or seek the input stream.

Decoding

- AACDecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- AACDecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions are assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API AAC Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

FLAC decoder

FLAC decoder features

- The FLAC decoder implementation support the following:
- Sampling rate: 8 kHz, 11.05 kHz, 12 kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- Channel : stereo and mono
- Bits per samples : 16 bits

Specification and reference

Official website

- FLAC lossless audio codec is at https://xiph.org/flac.

Inbound licensing

- For licensing information please refer to FLAC's official website: https://xiph.org/flac/license.html.

Performance

Memory information The memory usage of the decoder in bytes is:

- Code/flash = 15744 + 2080 = 17824
- Data/RAM = 27936

Section	Size
.text	15744
.ro & .const	2080
.bss	27936

CPU usage

- Output frame size: 16384 bytes.
- CPU core clock in MHz: 20.97.

Track type	Duration of track in second	Performance MIPS of codec (in MHz)
48 kHz, stereo	76 s	30.7 MHz
32 kHz, stereo	76 s	20.3 MHz
8 kHz, stereo	37 s	5.34 MHz

Following test cases are performed:

- Audio format listening test
- Audio quality test

For all above test cases, test tracks are played through the end without any distortion, glitching, hanging, or crashing.

API Usage of FLAC Decoder

Overview

- This section describes the integration steps to call FLAC decoder APIs by the application code. During each step the used data structures and functions are explained. All cci public APIs are defined in flac_cci.h header file. This file is located at \decoders\flac\include.

Configuration

Build Options

- SUPPORT_16_BITS_ONLY :- This macro is used to enable 16bits per sample flac decoder.
- *ASM* :- This macro is used to enable ARM assembly macros for 24bits per sample flac decoder.

Buffer Allocation

- The FLAC decoder does not perform dynamic memory allocation. The application calls the function FLACDecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder and then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- FLACDecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which will be used by decoder to read or to seek the input stream.

Decoding

- FLACDecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- FLACDecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions will be assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API FLAC Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

MP3 decoder

MP3 decoder features

- MP3 decoder supports mpeg-1, mpeg-2, mpeg-2.5.
- All MP3 features supported , including joint stereo, mid-side stereo, intensity stereo, and dual channel.
- Supported sampling rate: 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz and 48 kHz.
- Supported channel: stereo and mono
- Supported bits per samples: 16 bit
- Supported bit rate: 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 256, 320, 384, 416, and 448.

Performance

Memory information The memory usage of the decoder (data obtained from IAR compiler) in bytes is:

- Code/flash = 26884 + 18372 = 45256
- RAM = 16200

Section	Size
.text	26884
.ro & .const	18372
.bss	16200

CPU usage The performance of the decoder was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Track type	Duration of track in second	Frame size in bytes	Performance MIPS of codec (in MHz)
320 Kbps, 44.1 kHz, stereo	358 s	2304	~24 MHz
192 Kbps, 48 kHz, stereo	10 s	2304	~18 MHz

API Usage of MP3 Decoder

Overview

- This section describes the integration steps to call MP3 decoder APIs by the application code. During each step the used data structures and functions are explained. All cci public APIs are defined in mp3_cci.h header file. This file is located at \decoders\mp3.

Configuration

Build Options MP3 Decoder library is built with the following defined/enabled macros.

- There is no macro or define used to build the MP3 decoder.

Buffer Allocation

- The MP3 decoder does not perform dynamic memory allocation. The application calls the function MP3DecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder and then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- MP3DecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which will be used by decoder to read or to seek the input stream.

Decoding

- MP3DecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- MP3DecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions will be assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API MP3 Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

WAV decoder

WAV decoder features

- The WAV decoder implementation support the following:
- Sampling rate: 8 kHz, 11.025kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- Channel: stereo and mono
- PCM format with 8/16/24 bits per sample.

Performance

Memory information The memory usage of the decoder in bytes is:

- Code/flash = 6260 + 342 = 6602
- Data/RAM = 16 + 20696 = 20712

Section	Size
.text	6260
.ro & .const	342
.bss	20696
.data	16

CPU usage The performance of the decoder was measured using the decoder standalone unit test.

- CPU core clock in MHz: 20.97 MHz.

Track type	Duration of track in second	Frame size in bytes	Performance MIPS of codec (in MHz)
48 kHz, stereo, PCM	12 s	4096	9.68 MHz

Following test cases were performed:

- Audio format listening test
- Audio quality test

For all above test cases, test tracks are played through the end without any distortion, glitching, hanging, or crashing.

API Usage of WAV Decoder

Overview

- This section describes the integration steps to call MP3 decoder APIs by the application code. During each step the used data structures and functions are explained. All cci public APIs are defined in wav_cci.h header file. This file is located at \decoders\wav.

Configuration

Build Options WAV Decoder library is built with the following defined/enabled macros.

- There is no macro or define used to build the WAV decoder.

Buffer Allocation

- The WAV decoder does not perform dynamic memory allocation. The application calls the function WAVDecoderGetMemorySize() to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.
- The application first gets the required memory size for the decoder and then allocates memory for the decoder structures. Structures contain Main Decoder parameters and decoder information parameters.
- This function populates the required memory for the decoder and returns the required memory size in bytes.

Initialization

- WAVDecoderInit() function must be called before decode API. This API allocates the memory to decoder main structure and also initializes the decoder main structure parameters.
- It also registers the call back functions to the decoder, which will be used by decoder to read or to seek the input stream.

Decoding

- WAVDecoderDecode() function is main decoding API of the decoder. This API decodes the encoded input stream and fills the PCM output samples into decoder output PCM buffer.
- This API gives the information about the number of samples produced by the decoder and also provides the pointer to the decoder output PCM samples buffer.

Seeking

- WAVDecoderSeek() function calculates the actual frame boundary align offset from the unalign seek offset and returns the actual seek offset. It also resets the decoder internal states and variables.

Callback Usage All the callback functions will be assigned to the respective pointers before the codec initialization is called. Callback APIs are described below.

Read Callback API WAV Decoder read call back API reads the bytes from the input stream and fills them into decoder internal bit stream buffer. It returns the number of bytes read from the input stream.

Seek Callback API This call back API is for the seek operation.

Get File Position Callback API This call back API gives the current file position.

Synchronous Sample Rate Converter

Introduction The Synchronous Sample Rate Converter (SSRC) software module converts a mono or stereo audio signal with a certain sampling frequency to an audio signal with a different sampling frequency. The sample rate converter works synchronously, meaning that input and output sampling rates are exactly known for a mutual clock reference.

To accomplish a professional sampling conversion quality and minimal system footprint, the SRC SW module contains highly optimized components.

The SSRC module supports the following features.

- Multiple instances of the sample rate converter can run at the same time.
- Supported sampling frequencies: 32 kHz, 44.1 kHz, and 48 kHz plus the halves and the quarters of these three sample rates. The input and output sample rates are freely selectable out of the supported sampling rates
- Selectable Mono/Stereo Input/Output.
- Selectable quality level: high quality/ very high quality.

Acronyms *Table 1* lists the acronyms used in this document.

Acror	Description
Fs	Sampling Frequency
Fs-LOW(	Lowest sample rate used for the conversion **Note:** Input sample rate for up sampling and the output sample rate for down sampling
FsIN	Input sample rate
FsOU	Output sample rate
MIPS	Million Instructions Per Second
SSRC	Synchronous sample rate converter
THD+	Total Harmonic Distortion plus Noise **Note:** The THD+N is defined as the total power of the unwanted signal divided by the power of the wanted signal. The wanted signal is defined as a full scale, 1 kHz sine wave.

Parent topic:*Introduction*

Performance figures The Total Harmonic Distortion Plus Noise (THD+N) of the converted signals is below - 76 (high-quality mode) and - 85 (very high-quality mode) for signal frequencies below 0.45*FsLOW (=90 % of the Nyquist range of the lowest sample clock)

Table 1 and *Table 2* give the THD+N performance (FsIN on the vertical axis and FsOUT on the horizontal axis) for the two supported quality levels. The numbers in the tables give the worst-case THD+N measured for signal frequencies below 0.45*FsLOW. For each conversion ratio, 100 THD+N measurements were executed with signal frequencies linearly spread over the complete Nyquist range.

FsIN/ FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	-92.1	-79.7	-80.1	-80.1	-79.6	-80.2	-79.4	-79.1	-79.2
11025	-79	-92.9	-80	-79.9	-80.2	-79.8	-79.9	-79.5	-78.9
12000	-79	-79.2	-92.7	-80.1	-79.8	-80.3	-79.8	-79.8	-79.5
16000	-81.7	-78.8	-80.2	-93	-78.3	-77.7	-78.3	-78.3	-77.9
22050	-77.5	-81.8	-78.2	-79	-93	-79.9	-79.8	-80.3	-79.9
24000	-77.4	-77.9	-81.2	-79.1	-79.2	-92.5	-80.1	-79.8	-79.9
32000	-81	-77.5	-78.9	-81.2	-78.7	-80.1	-92.9	-79.7	-79.2
44100	-79.1	-81.2	-76.7	-77.8	-82	-78.2	-79.1	-93	-79.7
48000	-78.7	-78.8	-81.1	-77.6	-77.9	-81.8	-79.1	-79.3	-93

FsIN/ FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	-92.1	-86.6	-88.6	-91.5	-86.4	-89	-89.7	-89.3	-89.3
11025	-89.1	-92.9	-86.3	-86.3	-91.6	-86.3	-86.5	-89.7	-89.3
12000	-91.4	-88.4	-92.7	-89.6	-86.6	-91.5	-86.8	-86.6	-89.7
16000	-93.1	-88.4	-90.4	-93	-86.6	-88.8	-91.5	-86.5	-89.4
22050	-90.7	-93.5	-89.7	-89.3	-93	-86.5	-86.3	-91.5	-86.6
24000	-93.8	-90.5	-93.5	-91.7	-88.4	-92.5	-89.7	-86.6	-91.5
32000	-93.8	-91	-91.2	-93.3	-88.4	-90.5	-92.9	-86.7	-89
44100	-93.7	-93.6	-91.5	-90.6	-93.8	-89.8	-89.3	-93	-86.5
48000	-94.1	-92.6	-94	-94	-90.1	-93.7	-91.8	-88.4	-93

Parent topic:*Introduction*

Resource usage This section lists the memory and processing requirements for the SSRC module.

Memory requirements The following are the memory requirements for the SSRC module.

Memory item	Size in bytes
Instance memory (persistent)	548
Scratch memory (non-persistent)	15.536 *1*
Program memory for Arm9E and XScale	14k
Program memory for Arm7	15k

Parent topic: Resource usage

1 Worst case number for I/O buffers of 40 ms. If smaller I/O buffers are used, this number is smaller. The required scratch memory is roughly equal to 2 times the buffer size on the highest sample rate.

Processing requirements The following tables give the MIPS performance of the SSRC module. The cycles are measured with zero wait state memory and for I/O buffers of 40 ms.

Note: The user processing 32-bit processing must refer to the very high-quality MIPS results.

On Arm7 and Arm9

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.13	4.77	5.17	1.84	6.75	7.33	3.55	9.1	9.89
11025	5.42	0.18	5.58	6.84	2.53	7.75	9.71	4.89	10.31
12000	5.85	6.39	0.2	7.01	8.97	2.76	9.89	12.94	5.32
16000	1.69	7.74	7.99	0.26	9.54	10.33	3.68	13.5	14.65
22050	7.2	2.33	10.09	10.83	0.36	11.17	13.67	5.07	15.49
24000	7.79	8.33	2.53	11.7	12.78	0.39	14.03	17.94	5.51
32000	3.12	10.32	10.58	3.38	15.48	15.98	0.52	19.08	20.66
44100	9.96	4.3	13.65	14.4	4.65	20.18	21.67	0.72	22.34
48000	10.8	11.34	4.68	15.58	16.67	5.06	23.4	25.56	0.78

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.07	7.71	8.24	2.28	10.5	11.28	4.41	13.44	14.48
11025	8.19	0.1	8.96	11.04	3.14	12	15.09	6.08	15.2
12000	8.76	9.52	0.1	11.3	14.48	3.41	15.36	20.07	6.61
16000	2.14	11.73	12.01	0.14	15.41	16.48	4.55	21	22.56
22050	10.78	2.94	15.39	16.38	0.19	17.92	22.08	6.27	24
24000	11.57	12.34	3.2	17.51	19.04	0.21	22.61	28.97	6.83
32000	4.19	15.48	15.77	4.27	23.46	24.01	0.28	30.83	32.96
44100	14.78	5.77	20.56	21.56	5.89	30.77	32.75	0.38	35.83
48000	15.92	16.7	6.28	23.15	24.69	6.41	35.02	38.08	0.42

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.13	13.61	14.52	4.43	19.03	20.43	8.8	25.06	26.99
11025	14.85	0.18	15.91	19.47	6.1	21.82	27.35	12.13	28.38
12000	15.84	17.36	0.2	19.97	25.4	6.64	27.85	36.26	13.21
16000	4.25	21.24	21.79	0.26	27.22	29.03	8.86	38.07	40.85
22050	20.02	5.85	27.72	29.7	0.36	31.81	38.94	12.2	43.63
24000	21.45	22.98	6.37	31.68	34.71	0.39	39.94	50.8	13.28
32000	8.39	28.74	29.29	8.5	42.48	43.58	0.52	54.43	58.07
44100	28.11	11.57	38.05	40.03	11.71	55.43	59.4	0.72	63.62
48000	30.19	31.71	12.59	42.9	45.96	12.74	63.36	69.42	0.78

Parent topic: Processing requirements

On Arm9e and XScale

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.03	1.14	1.25	0.54	1.95	2.14	1.04	3.85	4.23
11025	1.31	0.05	1.36	1.62	0.75	2.23	2.78	1.44	4.38
12000	1.43	1.57	0.05	1.68	2.13	0.82	2.84	3.72	1.57
16000	0.5	1.86	1.93	0.07	2.27	2.5	1.09	3.9	4.29
22050	2.19	0.69	2.42	2.61	0.1	2.72	3.24	1.5	4.46
24000	2.4	2.52	0.75	2.86	3.15	0.1	3.35	4.25	1.63
32000	0.92	3.12	3.18	1.01	3.72	3.86	0.14	4.55	4.99
44100	4.28	1.27	4.15	4.37	1.39	4.83	5.23	0.19	5.43
48000	4.7	4.9	1.39	4.8	5.03	1.51	5.72	6.3	0.21

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.06	1.87	2.02	1.07	3.09	3.36	2.07	6.09	6.63
11025	2.27	0.09	2.25	2.66	1.47	3.56	4.4	2.85	7.01
12000	2.45	2.76	0.09	2.75	3.43	1.6	4.5	5.83	3.1
16000	0.99	3.23	3.36	0.13	3.73	4.05	2.14	6.17	6.72
22050	3.69	1.36	4.14	4.55	0.17	4.51	5.31	2.95	7.13
24000	4.01	4.28	1.48	4.9	5.51	0.19	5.51	6.85	3.21
32000	1.83	5.26	5.39	1.98	6.46	6.71	0.25	7.47	8.09
44100	7.22	2.52	6.94	7.38	2.72	8.27	9.1	0.35	9.02
48000	7.85	8.33	2.74	8.02	8.57	2.97	9.81	11.03	0.38

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.03	1.21	1.33	0.61	2.08	2.29	1.17	4.1	4.51
11025	1.47	0.05	1.44	1.72	0.84	2.38	2.97	1.61	4.66
12000	1.62	1.76	0.05	1.78	2.26	0.91	3.03	3.98	1.75
16000	0.55	2.1	2.17	0.07	2.42	2.65	1.22	4.16	4.57
22050	2.49	0.76	2.73	2.95	0.1	2.88	3.45	1.68	4.75
24000	2.75	2.86	0.83	3.23	3.52	0.1	3.56	4.53	1.83
32000	1	3.56	3.63	1.11	4.2	4.34	0.14	4.84	5.3
44100	4.86	1.38	4.74	4.98	1.53	5.46	5.89	0.19	5.75
48000	5.38	5.55	1.5	5.5	5.71	1.66	6.47	7.05	0.21

FsIN / FsOUT	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	0.06	2.11	2.29	1.2	3.55	3.86	2.31	6.99	7.61
11025	2.62	0.09	2.52	3.01	1.66	4.07	5.07	3.19	8
12000	2.85	3.15	0.09	3.11	3.9	1.81	5.17	6.75	3.47
16000	1.09	3.73	3.85	0.13	4.22	4.57	2.41	7.1	7.72
22050	4.32	1.5	4.79	5.23	0.17	5.05	6.02	3.32	8.15
24000	4.74	4.99	1.64	5.69	6.3	0.19	6.22	7.8	3.61
32000	1.98	6.18	6.3	2.18	7.45	7.71	0.25	8.44	9.14
44100	8.43	2.72	8.18	8.64	3.01	9.59	10.47	0.35	10.1
48000	9.26	9.66	2.97	9.49	9.97	3.27	11.39	12.59	0.38

Parent topic: Processing requirements

On Cortex-A8 for worst case of 48000 Hz to 44100 Hz

Mode	MIPs
Mono at High Quality	3.13
Stereo at High Quality	3.61
Mono at Very High Quality	4.13
Stereo at Very High Quality	6.52

Parent topic: Processing requirements

Parent topic: Resource usage

Parent topic:*Introduction*

Application programmers interface (API) This section describes the application programming interface (API) libraries of the SSRC module.

Type definitions This section describes the type definitions of the SSRC module.

Types for allocation of instance and scratch memory The instance memory is the memory that contains the state of one instance of the SSRC module. Multiple instances of the SSRC module can exist, each with it is own instance memory. S memory is the memory that is only used temporarily by the process function of the SSRC module. This memory can be used as scratch memory by any other function running in the same thread as the SSRC module. Different threads cannot share the scratch memories.

The application must allocate both the instance and the scratch memory. The SSRC module does not allocate memory.

There is a data type available for both the instance and the scratch memory, namely SSRC_Instance_t and SSRC_Scratch_t. The instance type is defined as structures of the correct size in the SSRC header file. Both the instance and the scratch memory must be 4 bytes aligned.

Parent topic: Type definitions

LVM_Fs_en **Definition:**

```
typedef enum
{
    LVM_FS_8000    = 0,
    LVM_FS_11025   = 1,
    LVM_FS_12000   = 2,
    LVM_FS_16000   = 3,
    LVM_FS_22050   = 4,
    LVM_FS_24000   = 5,
    LVM_FS_32000   = 6,
    LVM_FS_44100   = 7,
    LVM_FS_48000   = 8
} LVM_Fs_en;
```

Description:

Used to pass the input and the output sample rate to the SSRC.

Parent topic: Type definitions

LVM_Format_en **Definition:**

```
typedef enum
{
    LVM_STEREO          = 0,
    LVM_MONOINSTEREO    = 1,
    LVM_MONO            = 2
} LVM_Format_en;
```

Description:

The LVM_Format_en enumerated type is used to set the value of the SSRC data format.

The SSRC supports input data in two formats Mono and Stereo. For an input buffer of NumSamples = N (meaning N sample pairs for Stereo and MonoInStereo or N samples for Mono), the format of data in the buffer is as listed in *Table 1*:

Sample Number	Stereo	MonoInStereo	Mono
0	Left(0)	Mono(0)	Mono(0)
1	Right(0)	Mono(0)	Mono(1)
2	Left(1)	Mono(1)	Mono(2)
3	Right(1)	Mono(1)	Mono(3)
4	Left(2)	Mono(2)	Mono(4)
“	“	“	“
“	“	“	“
N-2	Left(N/2-1)	Mono(N/2-1)	Mono(N-2)
N-1	Right(N/2-1)	Mono(N/2-1)	Mono(N-1)
N	Left(N/2)	Mono(N/2)	Not Used
N+1	Right(N/2)	Mono(N/2)	Not Used
N+2	Left(N/2+1)	Mono(N/2+1)	Not Used
N+3	Right(N/2+1)	Mono(N/2+1)	Not Used
“	“	“	Not Used
“	“	“	Not Used
2*N-2	Left(N-1)	Mono(N-1)	Not Used

Parent topic: Type definitions

SSRC_Quality_en **Definition:**

```
typedef enum
{
    SSRC_QUALITY_HIGH           = 0,
    SSRC_QUALITY_VERY_HIGH      = 1,
    SSRC_QUALITY_DUMMY          = LVM_MAXENUM
} SSRC_Quality_en;
```

Description:

Used to select the quality level of the SSRC. For details, see Performance figures. Selecting the highest-quality level, comes with a cost in the SSRC processing requirements. Therefore, it should only be done for critical applications.

Parent topic: Type definitions

Instance parameters **Definition:**

```
typedef struct
{
    SSRC_Quality_en     Quality;
    LVM_Fs_en           SSRC_Fs_In;
    LVM_Fs_en           SSRC_Fs_Out;
    LVM_Format_en       SSRC_NrOfChannels;
    short               NrSamplesIn;
    short               NrSamplesOut;
} SSRC_Params_t;
```

Description:

Used to pass the SSRC instance parameters to the SSRC module. It is a structure that contains the members for input sample rate, output sample rate, the number of channels, and the number of samples on the input and output audio stream.

Parent topic: Type definitions

Nr of samples mode **Definition:**

```
typedef enum
{
    SSRC_NR_SAMPLES_DEFAULT     = 0,
    SSRC_NR_SAMPLES_MIN         = 1,
    SSRC_NR_SAMPLES_DUMMY       = LVM_MAXENUM
} SSRC_NR_SAMPLES_MODE_en;
```

Description:

The SSRC_NR_SAMPLES_MODE_en enumerated type specifies the two different modes that can be used to retrieve the number of samples using the SSRC_GetNrSamples function.

Parent topic: Type definitions

Function return status **Definition:**

```
typedef enum
{
    SSRC_OK                     = 0,
    SSRC_INVALID_FS             = 1,
    SSRC_INVALID_NR_CHANNELS    = 2,
    SSRC_NULL_POINTER           = 3,
    SSRC_WRONG_NR_SAMPLES       = 4,
    SSRC_ALLINGMENT_ERROR       = 5,
    SSRC_INVALID_MODE           = 6,
    SSRC_INVALID_VALUE          = 7,
    SSRC_ALLINGMENT_ERROR       = 8,
    LVXXX_RETURNSTATUS_DUMMY = LVM_MAXENUM
} SSRC_ReturnStatus_en;
```

Description:

The SSRC_ReturnStatus_en enumerated type specifies the different error codes returned by the API functions. For the exact meaning, see the individual function descriptions.

Parent topic: Type definitions

Parent topic:*Application programmers interface (API)*

Functions This section lists all the API functions of the SSRC module and explains their parameters.

SSRC_GetNrSamples **Prototype:**

```
SSRC_ReturnStatus_en SSRC_GetNrSamples
  (SSRC_NR_SAMPLES_MODE_en  Mode,
   SSRC_Params_t*           pSSRC_Params );
```

Description:

This function retrieves the number of samples or sample pairs for stereo used as an input and as an output of the SSRC module.

Nam	Type	Description
Mod	SSRC_N	There are two modes: - SSRC_NR_SAMPLES_DEFAULT: In this mode, the function returns the number of samples for 40 ms blocks - SSRC_NR_SAMPLES_MIN: the function returns the minimal number of samples supported for this conversion ratio. The SSRC_Init function accepts each integer multiple of this ratio. Formula: $\text{blocksize (ms)} = 1/\text{gcd(Fs\_In,Fs\_Out)}$
pSSF	SSRC_Pa	Pointer to the instance parameters. The application fills in the values of the input sample rate, the output sample rate, and the number of channels. Based on this input, the SSRC_GetNrSamples fills in the values for the number of samples for the input and the output audio stream.

Returns:

SSRC_OK	When the function call succeeds.
SSRC_INVALID_FS	When the requested input or output sampling rates are invalid.
SSRC_INVALID_NR_CHANN	When the channel format is not equal to LVM_MONO or LVM_STEREO.
SSRC_NULL_POINTER	When pSSRC_Params is a NULL pointer.
SSRC_INVALID_MODE	When mode is not a valid setting.

Note: The SSRC_GetNrSamples function returns the values from the following tables. Instead of calling the SSRC_GetNrSamples function, use the values from these tables directly.

Sample rate	Nr of samples
8000	320
11025	441
12000	480
16000	640
22050	882
24000	960
32000	1280
44100	1764
48000	1920

In/Out	8000	11025	12000	16000	22050	24000	32000	44100	48000
8000	11	320441	23	12	160441	13	14	80441	16
11025	441320	11	147160	441640	12	147320	4411280	14	147640
12000	32	160147	11	34	80147	12	38	40147	14
16000	21	640441	43	11	320441	23	12	160441	13
22050	441160	21	14780	441320	11	147160	441640	12	147320
24000	31	320147	21	32	160147	11	34	80147	12
32000	41	1280441	83	21	640441	43	11	320441	23
44100	44180	41	14740	441160	21	14780	441320	11	147160
48000	61	640147	41	31	320147	21	32	160147	11

Parent topic: Functions

SSRC_GetScratchSize **Prototype:**

```
SSRC_ReturnStatus_en SSRC_GetScratchSize
  (SSRC_Params_t*     pSSRC_Params,
   LVM_INT32*         pScratchSize );
```

Description:

This function retrieves the scratch size for a given conversion ratio and for given buffer sizes at the input and at the output.

Name	Type	Description
pSSRC_Par	SSRC_Param	Pointer to the instance parameters. All members should have a valid value.
pScratch-Size	LVM_INT32*	Pointer to the scratch size. The SSRC_GetScratchSize function fills in the correct value (in bytes).

|

Returns:

SSRC_OK	When the function call succeeds.
SSRC_INVALID_FS	When the requested input or output sampling rates are invalid.
SSRC_INVALID_NR_CHANN	When the channel format is not equal to LVM_MONO or LVM_STEREO.
SSRC_NULL_POINTER	When pSSRC_Params or pScratchSize is a NULL pointer.
SSRC_WRONG_NR_SAMPLI	When the number of samples on the input or on the output are incorrect.

Parent topic: Functions

SSRC_Init **Prototype:**

```
SSRC_ReturnStatus_en SSRC_Init
  (SSRC_Instance_t* pSSRC_Instance,
   SSRC_Scratch_t*  pSSRC_Scratch,
   SSRC_Params_t*   pSSRC_Params,
   LVM_INT16**      ppInputInScratch,
   LVM_INT16**      ppOutputInScratch);
```

Description:

The SSRC_Init function initializes an instance of the SSRC module.

Name	Type	Description
pSSRC_	SSRC_	Pointer to the instance of the SSRC. This application must allocate the memory before calling the SSRC_Init function.
pSSRC_	SSRC_	Pointer to the scratch memory. The pointer is saved inside the instance and is used by the SSRC_Process function. The application must allocate the scratch memory before calling the SSRC_Init function.
pSSRC_	SSRC_	Pointer to the instance parameters.
ppInputInScratch	LVM_I	The SSRC module can be called with the input samples located in scratch. This pointer points to a location that holds the pointer to the location in the scratch memory that can be used to store the input samples. For example, to save memory.
ppOutputInScratch	LVM_I	The SSRC module can store the output samples in the scratch memory. This pointer points to a location that holds the pointer to the location in the scratch memory that can be used to store the output samples. For example, to save memory.

Returns:

SSRC_OK	When the function call succeeds.
SSRC_INVALID_FS	When the requested input or output sampling rates are invalid.
SSRC_INVALID_NR_CHANN	When the channel format is not equal to LVM_MONO or LVM_STEREO.
SSRC_NULL_POINTER	When pSSRC_Params or pScratchSize is a NULL pointer.
SSRC_WRONG_NR_SAMPLI	When the number of samples on the input or on the output are incorrect.
SSRC_ALIGNMENT_ERROR	When the instance memory or the scratch memory is not 4 bytes aligned.

Parent topic: Functions

SSRC_SetGains **Prototype:**

```
SSRC_ReturnStatus_en SSRC_SetGains
  (SSRC_Instance_t* pSSRC_Instance,
   LVM_Mode_en      bHeadroomGainEnabled,
   LVM_Mode_en      bOutputGainEnabled,
   LVM_INT16        OutputGain);
```

Description:

This function sets headroom gain and the post gain of the SSRC. The SSRC_SetGains function is an optional function that should be used only in rare cases. Preferably, use the default settings.

Name	Type	Description
pSSRC	SSRC_	Pointer to the instance of the SSRC.
bHeadroomGainEnabled	LVM_	Parameter to enable or disable the headroom gain of the SSRC. The default value is LVM_MODE_ON. LVM_MODE_OFF can be used if it can be guaranteed that the input level is below - 6 in all cases (the default headroom is -6 dB).
bOutputGainEnabled	LVM_	Parameter to enable or disable the output gain. The default value is LVM_MODE_ON.
OutputGain	LVM_	The value of the output gain. The output gain is a linear gain value. 0x7FFF is equal to +6 dB and 0x0000 corresponds to -inf dB. By default, a 3 dB gain is applied (OutputGain = 23197), resulting in an overall gain of -3 dB (-6 dB headroom +3 dB output gain). Unit Q format Data Range Default value Linear gain Q1.14 [0;32767] 23197

Returns:

SSRC_OK	When the function call succeeds
SSRC_NULL_POINT	When pSSRC_Instance is a NULL pointer
SSRC_INVALID_MO	Wrong value used for the bHeadroomGainEnabled or the OutputGainEnabled parameters.
SSRC_INVALID_VAI	When OutputGain is out of the range [0;32767].

Parent topic: Functions

SSRC_Process **Prototype:**

```
SSRC_ReturnStatus_en SSRC_Process
  (SSRC_Instance_t* pSSRC_Instance,
   LVM_INT16*       pSSRC_AudioIn,
   LVM_INT16*       pSSRC_AudioOut);
```

Description:

Process function for the SSRC module. The function takes pointers as input and output audio buffers.

The sample format used for the input and output buffers is 16-bit little-endian. Stereo buffers are interleaved (L1, R1, L2, R2, and so on), mono buffers are deinterleaved (L1, L2, and so on).

Name	Type	Description
pSSRC_Instance	SSRC_Instance_t*	Pointer to the instance of the SSRC.
pSSRC_AudioIn	LVM_INT16*	Pointer to the input samples.
pSSRC_AudioOut	LVM_INT16*	Pointer to the output samples.

Returns:

SSRC_OK	When the function call succeeds.
SSRC_NULL_POINTEI	When one of pSSRC_Instance, pSSRC_AudioIn, or pSSRC_AudioOut is NULL.

Parent topic: Functions

SSRC_Process_D32 Prototype:

```
SSRC_ReturnStatus_en SSRC_Process_D32
  (SSRC_Instance_t* pSSRC_Instance,
   LVM_INT32*       pSSRC_AudioIn,
   LVM_INT32*       pSSRC_AudioOut);
```

Description:

Process function for the SSRC module. The function takes pointers as input and output audio buffers.

The sample format used for the input and output buffers is 32-bit little-endian. Stereo buffers are interleaved (L1, R1, L2, R2, and so on), mono buffers are deinterleaved (L1, L2, and so on).

Name	Type	Description
pSSRC_Instance	SSRC_Instance_t*	Pointer to the instance of the SSRC.
pSSRC_AudioIn	LVM_INT32*	Pointer to the input samples.
pSSRC_AudioOut	LVM_INT32*	Pointer to the output samples.

Returns:

|SSRC_OK|When the function call succeeds.| |SSRC_NULL_POINTER|When one of pSSRC_Instance, pSSRC_AudioIn, or pSSRC_AudioOut is NULL.|

Parent topic: Functions

Parent topic:*Application programmers interface (API)*

Dynamic function usage

This chapter explains how and when the SSRC functions are or can be used.

Define the number of samples to be used on input and output

Call the function SSRC_GetNrSamples. Each integer multiple of the returned number of samples can be used.

Parent topic: Dynamic function usage

Allocate scratch memory

To calculate the required size of the scratch memory, call the SSRC_GetScratchSize function. Allocate memory for the returned size.

Parent topic: Dynamic function usage

Initialize the SSRC instance

Call the SSRC_Init function.

Parent topic: Dynamic function usage

Process samples The SSRC_Process function can now be called any number of times.

Parent topic: Dynamic function usage

Destroy the SSRC instance When the processing is completed, the allocated memory for the instance and the scratch can be freed.

Parent topic: Dynamic function usage

Parent topic:*Application programmers interface (API)*

Reentrancy None of the SSRC functions are re-entrant.

Parent topic:*Application programmers interface (API)*

Additional user information This section provides information on the Attenuation of the signal and Notes on integration.

Attenuation of the signal When a fully saturated or clipped input is applied to an SRC module, the aliases after the sample rate conversion, although sufficiently suppressed, can still result in a clipped output. To prevent clipped output, the output of the SSRC module is by default attenuated with 3 dB. Although not advised, this gain value can be changed using the SSRC_SetGains function.

Parent topic:*Additional user information*

Notes on integration Although the sample rate converter module works with audio signals on different sampling rates, it is a synchronous module. The module takes a block of input samples, consumes the input completely, and produces a full buffer with output samples. As a result, the SSRC only accepts a limited number of input and output block sizes. To flush last, incomplete, block of an audio stream, the block is padded with zeros until it is full before the SSRC processes it.

Parent topic:*Additional user information*

Example application The source code of the example application can be found in the .\EX_APP\APP_FileIO\SRC directory of the release package. The .\EX_APP\APP_FileIO\MAKE directory contains a make file that can be used to build the example application. When building the application, an executable is generated in the .\EX_APP\APP_FileIO\EXE directory.

The example application takes as command-line input parameters:

1. The path toward the input PCM file. It assumes raw 16 bit signed little-endian put. Stereo input samples should be interleaved (L1, L2 R1, R2,...), mono samples should be deinterleaved (L1, L2, and so on).
2. The path toward the output PCM file.
3. The input sample rate.
4. The output sample rate.
5. The channel format (mono or stereo).

Integration test A correct integration of the SSRC module can be verified in two ways.

- Bit accurate test
- THD+N measurement

Bit accurate test The TestFiles directory of the release package contains a test input (sampled at 44,100 Hz) and several expected output files (sample rates from 8000 Hz to 48,000 Hz). If the same test input file is applied to the SRC after integration in the target platform, the output is bit accurate with the expected output file that matches the output-sample rate

Parent topic:*Integration test*

THD+N measurement Produce a swept sine and feed it through the SSRC module. Do a THD+N measurement on the obtained output signal. The THD+N of the converted signals should be below - 77 in the interval [0 - 0.45] FsLOW.

Parent topic:*Integration test*

Maestro Audio Framework

MCUXpresso SDK : Maestro

Overview This repository is for MCUXpresso SDK maestro middleware delivery and it contains the components officially provided in NXP MCUXpresso SDK. This repository is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository (mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Documentation Overall details can be reviewed here: MCUXpresso SDK Online Documentation

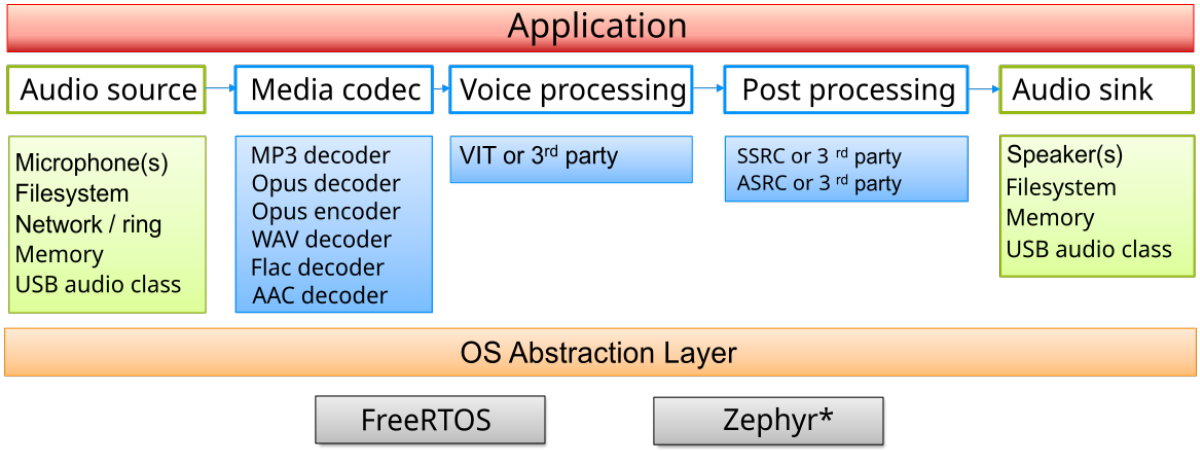
Visit Maestro - Documentation to review details on the contents in this sub-repo.

Setup Instructions on how to install the MCUXpresso SDK provided from GitHub via west manifest Getting Started with SDK - Detailed Installation Instructions

Contribution We welcome and encourage the community to submit patches directly to the Maestro project placed on github. Contributing can be managed via pull-requests.

Introduction Maestro audio framework intends to enable chaining of basic audio processing blocks, called *elements*. These blocks then form stream processing objects, called *pipeline*. This pipeline can be used for multiple audio processing use cases.

The processing blocks can include (but are not limited to) different audio sources (for example file or microphone), decoders or encoders, filters or effects, and audio sinks. Framework overview is depicted in the following picture:



*not all elements and libraries are supported in Zephyr port. For more information, see *Maestro on Zephyr*

The Maestro audio framework is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license. It is running on RTOS (Zephyr or FreeRTOS), abstracted by OSA layer.

For detailed description of the audio Maestro framework, please refer to the *programmer's guide*.

To see what is new, see *changelog*.

Maestro on Zephyr Getting started guide and further information for Maestro on Zephyr may be found *here*.

Maestro on FreeRTOS Maestro on FreeRTOS is supported in NXP's SDK. To get started, see mcuxsdk doc.

Supported examples The current version of the Maestro audio framework supports several optional *features*, some of which are used in these examples:

- *maestro_playback*
- *maestro_record*
- *maestro_usb_mic*
- *maestro_usb_speaker*
- *maestro_sync*

The examples can be found in the **audio_examples** folder of the desired board. The demo applications are based on FreeRTOS and use multiple tasks to form the application functionality.

Example applications overview To set up the audio framework properly, it is necessary to create a streamer with streamer_create API. It is also essential to set up the desired hardware peripherals using the functions described in streamer_pcm.h. The Maestro example projects consist of several files regarding the audio framework. The initial file is main.c with code to create multiple tasks. For features including SD card (in the maestro_playback examples, reading a file from SD card is supported and in maestro_record writing to SD card is currently supported) the APP_SDCARD_Task is created. The command prompt and connected functionalities are handled by APP_Shell_Task.

One of the most important parts of the configuration is the streamer_pcm.c where the initialization of the hardware peripherals, input and output buffer management can be found. For further information please see also streamer_pcm.h

In the Maestro USB examples (maestro_usb_mic and maestro_usb_speaker), the USB configuration is located in the usb_device_descriptor.c, audio_microphone.c and audio_speaker.c files. For further information please see also usb_device_descriptor.h, audio_microphone.h and audio_speaker.h.

In order to be able to get the messages from the audio framework, it is necessary to create a thread for receiving the messages from the streamer, which is usually called a Message Task. The message thread is placed in the app_streamer.c file, reads the streamer message queue, and reacts to the following messages:

- STREAM_MSG_ERROR - stops the streamer and exits the message thread
- STREAM_MSG_EOS - stops the streamer and exits the message thread
- STREAM_MSG_UPDATE_DURATION - prints info about the stream duration
- STREAM_MSG_UPDATE_POSITION - prints info about current stream position
- STREAM_MSG_CLOSE_TASK - exits the message thread

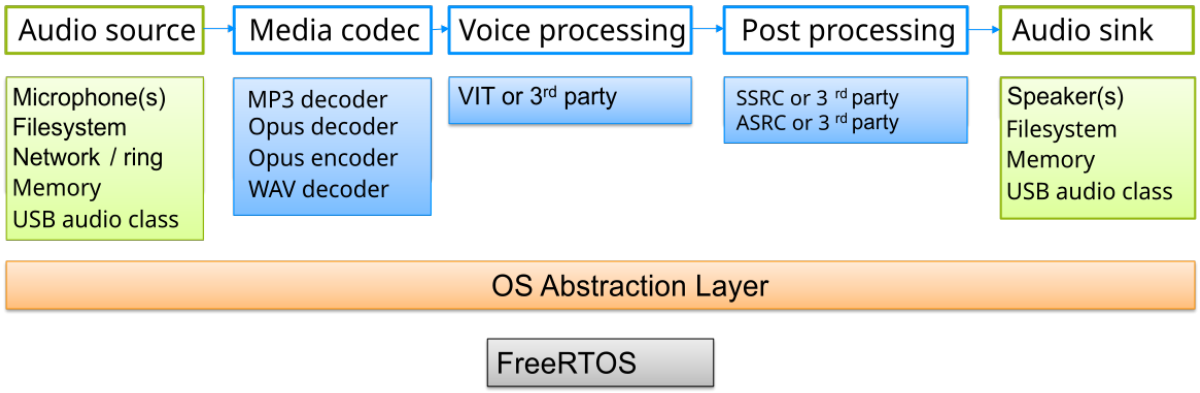
File structure

Folder	Description
src	Maestro audio framework sources
src/inc	Maestro include files
src/core	Maestro core sources
src/cci	Common decoder interface sources
src/cei	Common encoder interface sources
src/elements	Maestro elements sources
src/devices	External audio devices implementation (audio source & audio sink elements)
src/utils	Helper utilities utilized by Maestro
docs	Generated documentation
doxygen	Documentation sources
components	Glue for audio libraries, so they can be used in elements
tests	Maestro tests
zephyr/	Zephyr related files
zephyr/samples/	Zephyr samples
zephyr/tests/	Zephyr tests
zephyr/audioTracks/	Audio tracks for testing
zephyr/wrappers/	Zephyr NXP SDK Wrappers
zephyr/doc/	Zephyr documentation configuration for Sphinx
zephyr/scripts/	Zephyr helper scripts, mostly for testing

Maestro Audio Framework Programmer's Guide

Introduction Maestro audio framework provides instruments for playback and capture of different audio streams. In order to do that the framework uses API for creating various audio and voice pipelines with the support of media and track information. This document describes the framework in its detail, and the usage of API for pipeline creation using different elements. The framework needs an operating system in order to create different tasks for audio processing and communication with the application.

Architecture overview A high-level block diagram of the streamer used in Maestro is shown below. An element is the most important class of objects in the streamer (see streamer_element.c). A chain of elements will be created and linked together when a *pipeline* is created. Data flows through this chain of elements in form of data buffers. An element has one specific function, which can be the reading of data from a file, decoding of this data, or outputting this data to a sink device. By chaining together several such elements, a pipeline is created that can do a specific task, for example, the playback.



Pipeline

The pipeline is created within the streamer_create API using the streamer_create_pipeline call. In the example applications provided in the MCUXpresso SDK the pipeline is created in the app_streamer.c file. In order to create a pipeline user needs to provide a PipelineElements structure consisting of array of element indexes ElementIndex and the number of elements in the pipeline. Then the pipeline is built automatically and user can specify the properties of the elements using the streamer_set_property API. All the element properties can be found in the streamer_element_properties.h file.

The streamer can handle up to two pipelines within a single task. The first pipeline with index 0 can be created using the streamer_create function as described above. Then the streamer_create_pipeline function should be used to create the second pipeline (pipeline with index 1). An example creation can be found in the app_streamer.c file in the *maestro_sync_example*. Both pipelines are processed sequentially, so after the first pipeline is processed, the second pipeline is processed.

After the pipeline is sucessfully created, all elements and entire pipeline are in STATE_NULL state. A user can start the streamer by setting the pipeline state to STATE_PLAYING using the streamer_set_state function. The pipeline can also be paused or stopped using the same function. Use the STATE_PAUSED to pause and use STATE_NULL to stop. The function changes the state of each element that is in the pipeline in turn, and after all the elements have obtained the desired state, the state of entire pipeline is changed.

Elements The current version of the Maestro framework supports several types of elements (StreamElementType). In each pipeline should be used one source element (elements with the _SRC suffix) and one sink element (elements with the _SINK suffix). A decoder, encoder or audio_proc element can be connected between these two elements. The audio_proc element can be used more than once within the same pipeline.

Each element type (StreamElementType) has several functions that are determined by a unique element index (ElementIndex). These indexes are used to create a pipeline, and each element index can only be used once in the same pipeline. The type_lookup_table shows which StreamElementType supports which ElementIndex.

Each element index (ElementIndex) has its own properties and a list of these properties can be found in the streamer_element_properties.h file. These properties are divided into groups and each group is identified by a property mask (e.g. for speaker it is PROP_SPEAKER_MASK). Then the property_lookup_table in the streamer_msg.c file determines which property group relates to

which element index (ElementIndex). When an element is created and added to the pipeline, its properties are set to their defalut values. Default values can be seen in the initialization function of a particular element. The initialization functions are specified in the element_list array in the streamer_element.c file (e.g. for the audio_proc element it is the audio_proc_init_element function). The user can get the value of the property using the streamer_get_property function or change its value using the streamer_set_property function.

The source code of the elements can be found in the middleware\audio_voice\maestro\src\elements\ folder.

Add a new element type The user can add a new element type (StreamElementType) to the Maestro audio framework. For this, the following steps need to be done.

- Add a new element type to the StreamElementType enum type in the streamer_api.h.
- Create a new *.c and *.h files for the new element type in the middleware\audio_voice\maestro\src\elements\ folder. All necessary structures and functions (functions for src pads, sink pads and element itself) needs to be defined in these files. Inspiration can be found in other elements.
- Link the initialization function to the element type in the element_list array in the streamer_element.c file. To do this, a new definition that enables the element needs to be created (e.g. there is a STREAMER_ENABLE_AUDIO_PROC definition for the audio_proc element).
- Associate the newly created element type with an element index (ElementIndex) by adding a new pair to the type_lookup_table in the streamer.c file.
- If the user wants to use the newly created element in an application, the definiton that enables the element must be defined at the project level.

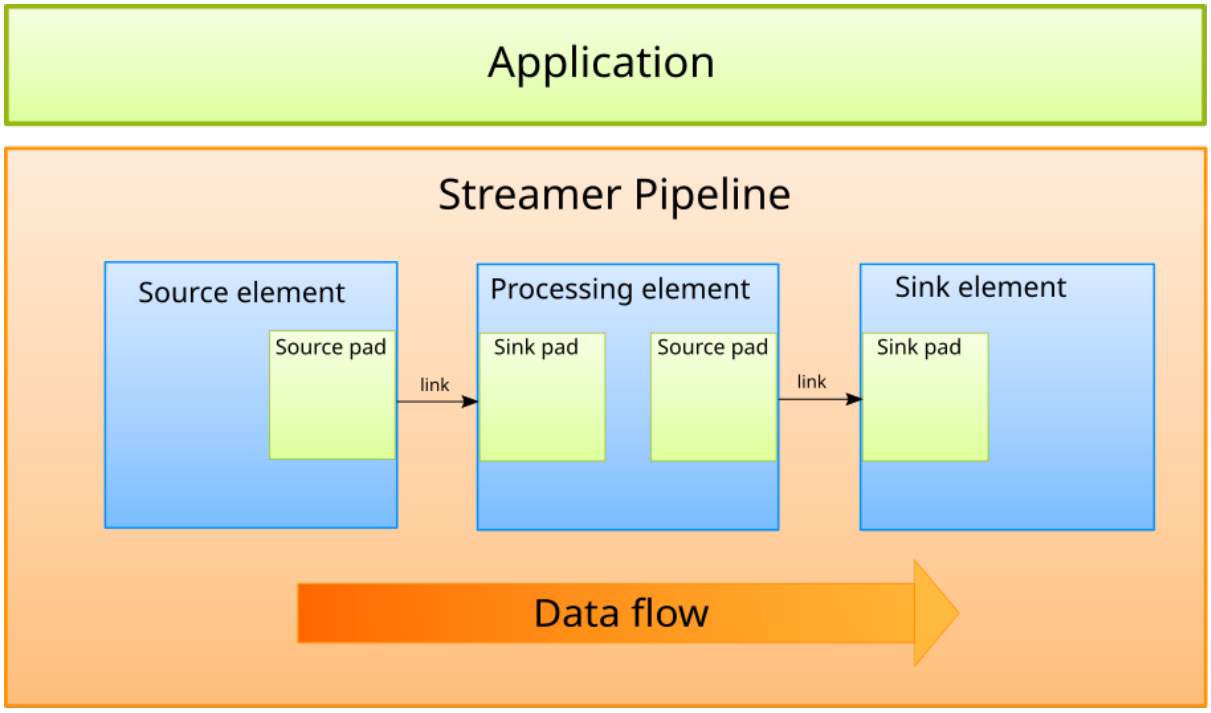
Mostly the user doesn't need to create a new element type, but just create an element index.

Add a new element index To create a new element index in the Maestro audio framework, follow these steps:

- Add a new element index to the ElementIndex enum type in the streamer_api.h.
- Create the required properties for the newly created element index in the streamer_element_properties.h file.
- Associate the newly created property group with newly created element index by adding a new pair to the property_lookup_table in the streamer_msg.c file.
- Associate the newly created element index with an element type (StreamElementType) by adding a new pair to the type_lookup_table in the streamer.c file.
- Add support for the created properties to functions of the associated element type. These functions are defined in files that correspond to a particular element type. The files are located in the middleware\audio_voice\maestro\src\elements\ folder.

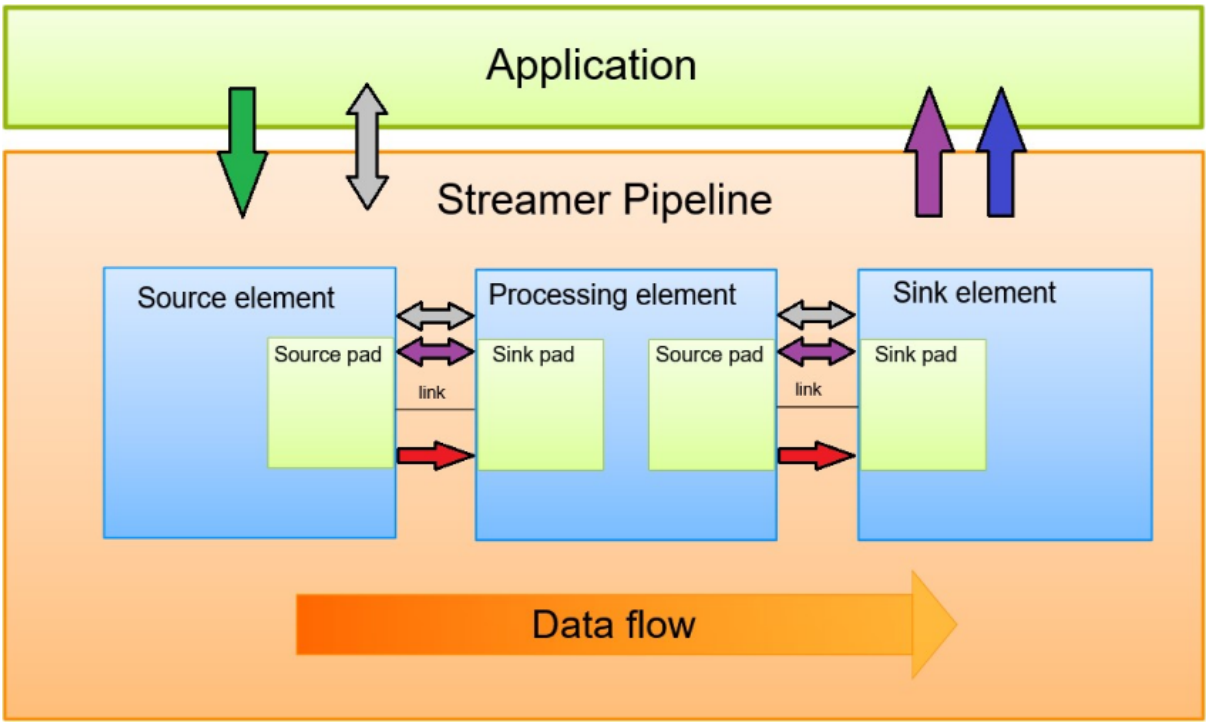
It is important to know that each element type (StreamElementType) can be associated with more than one element index (ElementIndex), but each element index (ElementIndex) can be associated with only one element type (StreamElementType).

Pads Pads are elements' inputs and outputs. A pad can be viewed as a "plug" or "port" on an element where links may be made with other elements, and through which data can flow to or from those elements. Data flows out of an element through a source pad, and elements accept incoming data through a sink pad. Source and sink elements have only source and sink pads, respectively. For detailed information about pads, please see the API reference from pad.c.



Internal communication The streamer (the core of the framework) provides several mechanisms for communication and data exchange between the application, a pipeline, and pipeline elements:

- Buffers are objects for passing streaming data between elements in the pipeline. Buffers always travel from sources to sinks (downstream).
- Messages are objects sent from the application to the streamer task to construct, configure, and control a streamer pipeline.
- Callbacks are used to transmit information such as errors, tags, state changes, etc. from the pipeline and elements to the application.
- Events are objects sent between elements. Events can travel upstream and downstream. Events may also be sent to the application
- Queries allow applications to request information such as duration or current playback position from the pipeline. Elements can also use queries to request information from their peer elements (such as the file size or duration). They can be used both ways within a pipeline, but upstream queries are more common



Decoders and encoders Maestro framework uses a common codec interface for decoding purposes and a common encoder interface for encoding. Those interfaces encapsulate the usage of specific codecs. Reference codecs are available in audio-voice-components repository which should be in \middleware\audio_voice\components\ folder.

Common codec interface The Common Codec Interface is the intended interface for all used **decoders**. The framework will integrate a CCI decoder element into the streamer to interface with all decoders.

Using the CCI to interface with Metadata

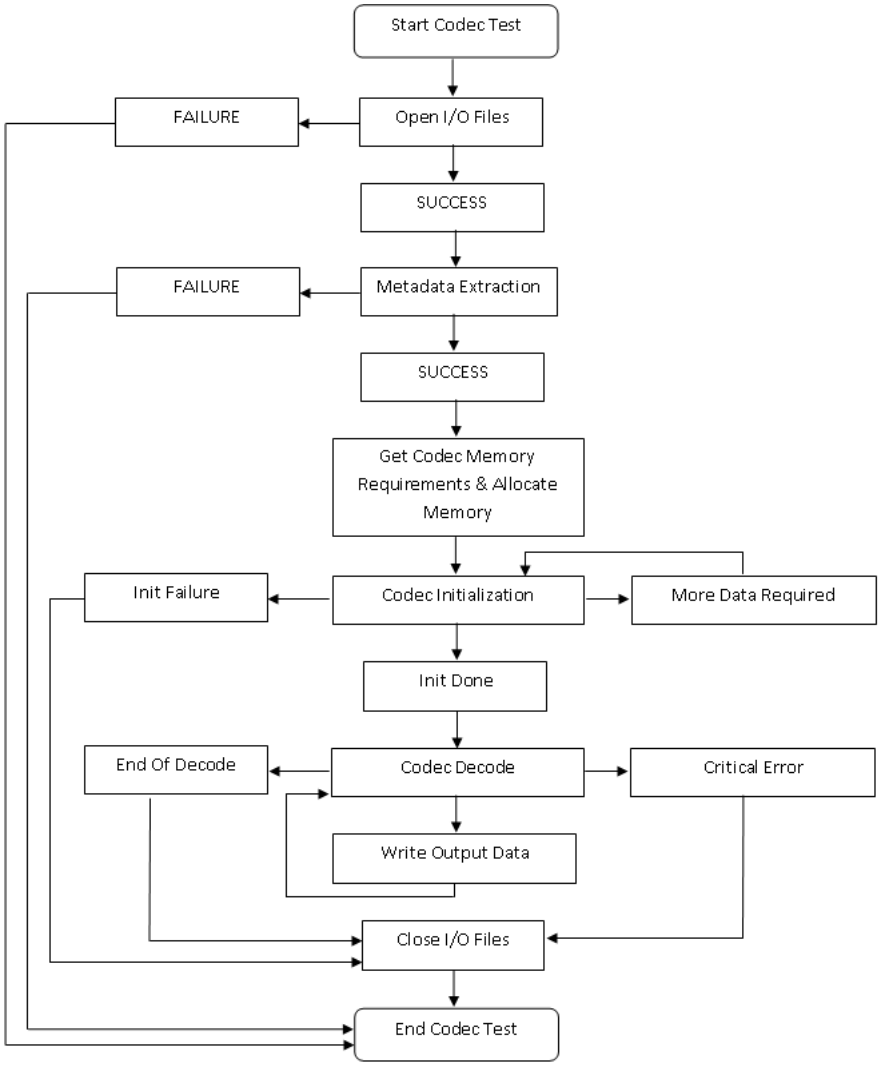
- cci_extract_meta_data must be called before any other Codec Interface APIs. This API extracts the metadata information of the codec and fills this information in the file_meta_data_t structure. The file_meta_data_t structure must be allocated by the application.
- This function first extracts the input file extension and based on that it calls the specific codec's metadata extraction function. If it finds an invalid extension or unsupported extension then it returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.
- If this API finds the valid metadata then it returns with META_DATA_FOUND code. If this API does not find any metadata information then it returns with META_DATA_NOT_FOUND code. It also returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.

Using the CCI to interface with Decoders

- codec_get_mem_info gets the memory requirement based on the specific decoder stream type. It returns the size in bytes of the specific codec. The user of the decoders must allocate memory of this size and this memory is used by the initialization API. The user or application must pass this allocated memory pointer to the init API.

- codec_init must be called before the codec's decode API. This API calls the codec-specific initialization function based on the codec stream type. This API allocates the memory to the codec main structure and also initializes the codec main structure parameters. It also registers the call back functions to the codec which will be used by the codec to read or seek the input stream.
- codec_decode is the main decoding API of the codec. This API calls the codec-specific decoding function based on the codec stream type. This API decodes the input raw stream and fills the PCM output samples into codec output PCM buffer. This API gives the information about the number of samples produced by the codec and also gives the pointer of the codec output PCM samples buffer.
- codec_get_pcm_samples must be called after the codec's decode API. This API calls the codec specific Get PCM Sample API based on the codec stream type. This API gets the PCM samples from the codec in constant block size and fills them into the output PCM buffer. It returns the number of samples get from the codec and also gives the pointer of the output PCM buffer.
- codec_reset calls the codec specific reset API base on stream type and resets the codec.
- codec_seek accepts the seek bytes offset converted from the time by application. This API calls the decoder's internal seek API to calculate the actual seek offset which frame boundary aligns. This API returns the actual seek offset.

The basic sequence to use a decoder with the CCI is shown below:



Adding new decoders to the CCI This section explains how to integrate a new decoder in the Common Codec Interface. The CCI assumes the decoder library to be used is in the \middleware\audio_voice\audiocomponents\decoders*decoder*\libs\ folder of the maestro framework. The CCI is just a wrapper around a specific implementation. The decoder is expected to be extended as needed to meet the APIs described above.

- Register Decoder Top level APIs in Common Codec Interface
  - Place the decoder lib in libs folder.
  - Add prototypes of the decoder top level APIs in codec_interface.h file (located at maestro\src\cci\inc\ folder).
  - In codec_interface.c file (located at maestro\src\cci\src\), add top level Decoder APIs in decoder function table.
  - Pseudo code for this is as described below.

```
const codec_interface_function_table_t g_codec_function_table[STREAM_TYPE_COUNT] = {
#ifdef VORBIS_CODEC
    {
        &VORBISDecoderGetMemorySize,
        &VORBISDecoderInit,
        &VORBISDecoderDecode,
        NULL,
        NULL,
        &VORBISDecoderSeek,
        &VORBISDecoderGetIOFrameSize,
    },
#else
    {
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
    }
#endif
};
```

- Enable or Disable Decoder
  - Define VORBIS_CODEC macro in audio_cfg.h file.
  - Comment this macro if you want to disable VORBIS Decoder otherwise keep it defined in order to enable the decoder.
- Add Extract Metadata API for the decoder
  - Add extract metadata API source file for the decoder at streamer/cci/metadata/src/vorbis folder.
  - Add this code in extract metadata lib project space.
  - Build the extract metadata lib and copy that lib to libs folder.
  - Add the desired stream type into ccidec_extract_meta_data API (in codecextractmetadata.c file) to call VORBIS Decoder extract metadata API.
- Add stream type of the new decoder in the stream type enum audio_stream_type_t in codec_interface_public_api.h
  - Stream type of the decoder in stream type enum and decoder APIs in decoder function table must be in the same sequence.

Common encoder interface Please see the following section about the *cei*.

Maestro performance

Memory information The memory usage of the framework components using reference codecs (data obtained from GNU ARM compiler) in bytes is:

text	data	bss	component
48790	2752	4	aac decoder
4348	16400	212	asrc
15512	0	4	flac decoder
76462	16	5013	maestro
34211	0	4	mp3 decoder
211974	0	0	opus
65446	0	4	ssrc
5850	16	12	wav decoder

Maestro framework uses dynamic allocation of audio buffers. The total amount of memory allocated for the pipeline depends on the following parameters:

- Number of elements in the pipeline
- Element types
- Audio stream properties
  - Sampling rate
  - Bit width
  - Channel number
  - Frame size

CPU usage The performance of the pipeline was measured using the real hardware platform (RT1060).

- CPU core clock in MHz: 600.

Pipeline type	Performance MIPS of pipeline (in MHz)
audio source -> audio sink	~10.26 MHz
audio source -> file sink	~9.84 MHz
file source (8-channel PCM) -> audio sink	~16.5 MHz

For performance details about the supported codecs please see audio-voice-components repository documentation.

CEI encoder The Maestro streamer contains an element adapting an extensible set of audio encoders in the form of functions conforming to the CEI (Common Encoder Interface). This element enables the user to choose and configure a suitable encoder at runtime.

Header files CEI itself and the CEI encoders are using following header files, in which you may be interested:

- cei.h - contains types used by the element itself and an encoder implementing the CEI
- cei_enctypes.h - contains a list of possible encoders and types used for interfacing with a CEI encoder
- cei_table.h - contains a table of functions implementing integrated CEI encoders

Instantiating the element This element's index is ELEMENT_ENCODER_INDEX and its type is TYPE_ELEMENT_ENCODER, as defined in streamer_api.h. It has one source pad (data input) and one sink pad (data output). It is initialized like any other element, meaning that it is instantiated and inserted into the pipeline using the create_element, add_element_pipeline and link_elements functions. Inversely, for destroying the element, the unlink_elements, remove_element_pipeline and destroy_element are used. This element alone does not depend on any additional software layers other than these required by the Maestro streamer itself, so no pre-initialization before this element instantiation is necessary.

Element properties Use Maestro streamer property API (streamer_set_property and streamer_get_property) for setting or getting these. The constants are defined in streamer_element_properties.h.

- PROP_ENCODER_CHUNK_SIZE
  - **Synopsis:** Determines the length of a chunk pulled from the sibling of the source pad and essentially influences the size of allocated buffers. If the actual amount of data pulled is smaller, the rest is zero-filled.
  - **Type:** unsigned 32-bit integer
  - **Default value:** 1920
  - **Constraints:**
    - Must be bigger than zero, otherwise STREAM_ERR_INVALID_ARGS is returned.
    - Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS is returned.
- PROP_ENCODER_TYPE
  - **Synopsis:** Determines the exact encoder (CEI implementation) to be used.
  - **Type:** CeiEncoderType (cei_enctypes.h)
  - **Default value:** CEIENC_LAST
  - **Constraints:**
    - Must not be equal to CEIENC_LAST, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    - Selected encoder must be implemented, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    - Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
  - **Behaviour influenced:** The encoder element process function will return FLOW_ERROR if this property isn't set.
- PROP_ENCODER_CONFIG
  - **Synopsis:** Determines encoder-specific configuration (application, bitrate, ...).
  - **Type:** Pointer to the encoder-specific configuration structure.

  - **Default value:** Determined by the encoder.
  - **Constraints**:
    * The encoder has to be configurable. If it is not, STREAM_ERR_ERR_GENERAL will be returned on any access.
    * The structure has to conform to the encoder requirements. If the encoder returns an error code, STREAM_ERR_GENERAL will be returned.

- PROP_ENCODER_BITSTREAMINFO
  - **Synopsis:** Specifies information about the incoming bitstream (sample rate, sample depth, ...).
  - **Type:** Pointer to CeiBitstreamInfo (cei_enctypes.h).
  - **Default value:**

```
(CeiBitstreamInfo) {
    .sample_rate = 0,
    .num_channels = 0,
    .endian = AF_LITTLE_ENDIAN,
    .sign = TRUE,
    .sample_size = 0,
    .interleaved = TRUE
}
```

  - **Constraints:**
    * Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
    * As of now, only bitstreams containing 16-bit interleaved (if 2 or more channels will be encoded) samples are supported. If anything else was set to the sample_size and interleaved members, STREAM_ERR_INVALID_ARGS will be returned.
  - **Behaviour influenced:**
    * Given the characteristics of some elements available, different packets of data (header and payload, referred to as "chunk" above) may be pulled by this element. Each packet can contain a different header, which may or may not contain useful information about the bitstream. If a packet with the AudioPacketHeader (todofile.h) is pulled at first and any other iteration of the streamer pipeline, the bitstream parameters configured by this property are implicitly available and are not expected to be specified by the user. Other packet header types (such as RawPacketHeader) don't contain any bitstream parameters and require the user to specify the parameters manually using this property. Failure to do so will result in the element's process function returning FLOW_ERROR. Same situation will occur if a packet with the AudioPacketHeader is received and its contents differ from the already acquired bitstream parameters.
    * As of now, CEI is defined to work with 16-bit signed little-endian (s16le) samples, which are interleaved if the bitstream contains more than one channels. This element handles endianness and unsigned to signed conversion.

CEI definition - implementing your own encoder The CEI defines following function pointer types:

- CeiFnGetMemorySize: Returns number of bytes required for encoder state for a given number of channels.
- CeiFnEncoderInit: Initialize an encoder for a given sample rate and channel count.
- CeiFnEncoderGetConfig: Copy current or default configuration to a given structure pointer.

- CeiFnEncoderSetConfig: Configure the encoder from a given structure pointer.
- CeiFnEncode: Encode a given buffer to a given output buffer.

Detailed descriptions of function behaviour, parameters and expected return values are available as docblocks in the cei.h file.

Each encoder is implemented as a set of pointers pointing to functions conforming to these types, grouped in the CeiEncoderFunctions structure. Specifying the CeiEncoderGetConfig fnGetConfig and CeiFnEncoderSetConfig fnSetConfig members is optional, as an encoder does not have to be configurable. If so desired, specify NULL. Implementation of the remaining functions is mandatory, however. If at least one of these functions isn't implemented and NULL is specified instead, the encoder will be considered as not implemented.

To register an implemented encoder with the element, add a new entry to the CeiEncoderType enum and add the CeiEncoderFunctions struct value to the table CeiEncoderFunctions ceiEncTable[] located in the cei_table.h header file. Note and match the order of items in that table, as a CeiEncoderType value is used as an index. Same goes for the size_t ceiEncConfigSizeTable[]. If configuration is not applicable, specify 0 at the appropriate index. If configuration is applicable, describe the configuration structure in the cei_enctypes.h header file and add its size to that table.

Maestro playback example

Table of content

Overview The Maestro playback example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console and the audio files are read from the SD card.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback of encoded files from an SD card with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-channel** - The mode demonstrates playback of raw PCM files from an SD card with 2 or 8 channels, 96kHz sample rate and 32 bit width. The decoders and synchronous sample rate converter are not supported in this mode. The Multi-channel mode is only supported on selected platforms, see the table below. The *Example configuration* section contains information on how to enable it.

As shown in the table below, the application is supported on several development boards and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Decoder:
  - **AAC:**
    - The reference decoder is supported only in the MCUXpresso IDE and ARMGCC.
  - **FLAC:**
    - *LPCXpresso55s69* - When playing FLAC audio files with too small frame size (block size), the audio output may be distorted because the board is not fast enough.
  - **OPUS:**
    - *LPCXpresso55s69* - The decoder is disabled due to insufficient memory may be distorted because the board is not fast enough.
- Sample rate converter:
  - **SSRC:**
    - *LPCXpresso55s69* - When a memory allocation ERROR occurs, it is necessary to disable the SSRC element due to insufficient memory.

Known issues:

- Decoder:
  - **MP3:**
    - The reference decoder has issues with some of the files. One of the channels can be sometimes distorted or missing parts of the signal.
  - **OPUS:**
    - The decoder doesn't support all the combinations of frame sizes and sample rates. The application might crash when playing an unspupported file.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- SD card with supported audio files
- Personal computer
- Optional:
  - Audio expansion board AUD-EXP-42448 (REV B)

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
  1. Please remove below resistors if on board wifi chip is not DNP:
     - R228, R229, R232, R234
  2. Please make sure R136 is weld for GPIO card detect.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board.
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
5. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**********************************
Maestro audio playback demo start
**********************************

[APP_Main_Task] started

Copyright  2022  NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console (*If multi-channel playback mode is enabled, the description is slightly different*):

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"file": Perform audio file decode and playback

  USAGE: file [stop|pause|volume|seek|play|list|info]
```

(continues on next page)

(continued from previous page)

```
stop              Stops actual playback.
pause             Pause actual track or resume if already paused.
volume <volume>   Set volume. The volume can be set from 0 to 100.
seek <seek_time>  Seek currently paused track. Seek time is absolute time in milliseconds.
play <filename>   Select audio track to play.
list              List audio files available on mounted SD card.
info              Prints playback info.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**
  - Add the MULTICHANNEL_EXAMPLE symbol to preprocessor defines on project level.
  - Connect AUD-EXP-42448 (see the point below).
- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
    5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.

Functionality The file play <filename> command calls the STREAMER_file_Create or STREAMER_PCM_Create function from the app_streamer.c file depending on the selected mode.

- When the *Standard* mode is enabled, the command calls the STREAMER_file_Create function that creates a pipeline with the following elements:
  - ELEMENT_FILE_SRC_INDEX
  - ELEMENT_DECODER_INDEX
  - ELEMENT_SRC_INDEX (If SSRC_PROC is defined)
  - ELEMENT_SPEAKER_INDEX
- When the *Multi-channel* mode is enabled, the command calls STREAMER_PCM_Create function, which creates a pipeline with the following elements:
  - ELEMENT_FILE_SRC_INDEX (PCM format only)
  - ELEMENT_SPEAKER_INDEX
  - *Note:*
    * If the input file is an 8 channel PCM file, output to all 8 channels is available. The properties of the PCM file are set in the app_streamer.c file using file source properties sent to the streamer:
      - PROP_FILESRC_SET_SAMPLE_RATE - default value is 96000 [Hz]
      - PROP_FILESRC_SET_NUM_CHANNELS - default value is 8
      - PROP_FILESRC_SET_BIT_WIDTH - default value is 32

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

EXT_PROCESS_DESC_T ssrc_proc = {SSRC_Proc_Init, SSRC_Proc_Execute, SSRC_Proc_Deinit,␣
↪&get_app_data()->proc_args};

prop.prop = PROP_SRC_PROC_FUNCPTR;
prop.val  = (uintptr_t)&ssrc_proc;

if (streamer_set_property(streamer, 0, prop, true) != 0)
{
    return -1;
}

prop.prop = PROP_AUDIOSINK_SET_VOLUME;
prop.val  = volume;
streamer_set_property(streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 3 different states:

- Idle
- Running
- Paused

In each state, each command can have a different behavior. For more information, see *Commands in detail* section.

Commands in detail The applicatin is controlled by commands from the shell interface and the available commands for the selected mode can be displayed using the help command. Commands are processed in the cmd.c file.

- *help, version*
- *file stop*
- *file pause*
- *file volume <volume>*
- *file seek <seek_time>*
- *file play <filename>*
- *file list*
- *file info*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
```

```
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> D[Write help or version]:::function
    B((Running)):::state --> D
    C((Paused)):::state --> D
    D-->E((No state
    change)):::state
```

file stop

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> B
    C((Running)):::state -->E((Idle)):::state
    D((Paused)):::state -->E
```

file pause

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> B
    C((Running)):::state -->E((Paused)):::state
    D((Paused)):::state -->F((Running)):::state
```

file volume <volume>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> M[Error: Play a track first]:::error
    C((Running)):::state --> G{Volume
    parameter
```

```
empty?}:::condition
D((Paused)):::state --> G
G -- Yes -->H[Error: Enter volume parameter]:::error
G -- No -->I{Volume
in range?}:::condition
I -- No -->J[Error: invalid value]:::error
I -- Yes -->K[Set volume]:::function
J --> L((No state
change)):::state
K --> L
H--> L
```

file seek <seek_time> The seek argument is only supported in the Standard mode.

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> E[Error: First select
    an audio track to play]:::error
    E-->B
    C((Running)):::state --> F[Error: First
    pause the track]:::error
    F --> C
    D((Paused)):::state --> G{Seek
    parameter
    empty?}:::condition
    G --No --> H{AAC file?}:::condition
    G --Yes --> I[Error: Enter
    a seek time value]:::error
    I-->N((Paused)):::state;
    H --Yes -->J[Error: The AAC decoder
    does not support
    the seek command]:::error
    J-->N
    H --No -->K{Seek
    parameter
    positive?}:::condition
    K --No -->L[Error: The seek
    time must be
    a positive value]:::error
    L-->N
    K --Yes -->M[Seek the file]:::function
    M-->N
```

file play <filename>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    C((Running)):::state --> Z[Error: First stop
    current track]:::error
```

```
D((Paused)):::state --> Z
B((Idle)):::state --> E{SD Card
inserted?}:::condition
E -- No -->F[Error: Insert SD
card]:::error
E -- Yes -->G{File
name
empty?}:::condition
G -- Yes -->H[Error: Enter
file name]:::error
G -- No -->I{File exists?}:::condition
I -- No -->O[Error: File
doesn't exist]:::error
I -- Yes -->J{Supported
format?}:::condition
J -- Yes -->K[Play the track]:::function
J -- No -->L[Error: Unsupported
file]:::error
K -->M((Running)):::state
L --> W((No state
change)):::state
O --> W
H --> W
F --> W
Z --> W
```

file list

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> G{SD Card
    inserted?}:::condition
    C((Running)):::state --> G
    D((Paused)):::state --> G
    G -- Yes -->H[List supported files]:::function
    G -- No -->I[Error: Insert SD card]:::error
    I --> J((No state
    change)):::state
    H --> J
```

file info

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state -->E[Write file info]:::function
    C((Running)):::state -->E
    D((Paused)):::state -->E
    E --> F((No state
        change)):::state
```

Processing Time Typical streamer pipeline execution times and their individual elements for the EVKC-MIMXRT1060 development board are presented in the following tables. The time spent on output buffers is not included in the traversal measurements. However, file reading time is accounted for. In the case of the WAV codec, the audio file was accessed in every pipeline run. Therefore, during each run, the file was read from the SD card. However, for the MP3 codec, where data must be processed in complete MP3 frames, the file was not read in every run. Instead, it was read periodically only when the codec buffer did not contain a complete frame of data.

For further details, please refer to the *Processing Time* document.

WAV	streamer	file_src	codec	SSRC_proc	speaker
48kHz	1.1 ms	850 µs	150 µs	70 µs	40 µs
44kHz	1.75 ms	850 µs	180 µs	670 µs	40 µs

MP3	streamer	file_src	codec	SSRC_proc	speaker
48 kHz with file read	2.9 ms	2.3 µs	450 µs	60 µs	50 µs
48 kHz without file read	0.5 ms	x	400 µs	40 µs	40 µs
44 kHz with file read	3.2 ms	2.3 µs	440 µs	400 µs	50 µs
44 kHz without file read	0.9 ms	x	440 µs	390 µs	40 µs

Maestro record example

Table of content

Overview The Maestro record example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Loopback** - The application demonstrates a loopback from the microphone to the speaker without any audio processing. Mono, stereo or multichannel mode can be used, depending on the hardware, see *table* below.
- **File recording** - The application takes audio samples from the microphone inputs and stores them to an SD card as an PCM file. The PCM file has following parameters:

  - Mono and stereo : 2 channels, 16kHz, 16bit width
  - Multi-channel (AUD-EXP-42448): 6 channels, 16kHz, 32bit width
- **Voice control** - The application takes audio samples from the microphone input and uses the VIT library to recognize wake words and voice commands. If a wake word or a voice command is recognized, the application write it to the serial terminal.
- **Encoding** - The application takes PCM samples from memory and sends them to the Opus encoder. The encoded data is stored in memory and compared to a reference. The result of the comparison is finally written into the serial terminal.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Addition labraries
  - **VIT:**
    * The VIT is supported only in the MCUXpresso IDE and ARMGCC.
    * *LPCXpresso55s69* - The VIT is disabled by default due to insufficient memory. To enable it, see the *Example configuration* section.
    * *EVK-MCXN5XX* - Some VIT models can't fit into memory. In order to free some space it is necessary to disable SD card handling and opus encoder. To disable it, see the *Example configuration* section.
  - **VoiceSeeker:**
    * The VoiceSeeker is supported only in the MCUXpresso IDE and ARMGCC.
- Encoder
  - **OPUS:**
    * *LPCXpresso55s69* - The encoder is not supported due to insufficient memory.
- The File recording mode is not supported on *RW612BGA* development board due to missing SD card slot.

Known issues:

- *EVKB-MIMXRT1170* - After several tens of runs (the number of runs is not deterministic), the development board restarts because a power-up sequence is detected on the RESET pin (due to a voltage drop).

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- Personal computer
- Optional:
  - SD card for file output

  - Audio expansion board AUD-EXP-42448 (REV B)
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKB-MIMXRT1170:*
  1. Please remove below resistors if on board wifi chip is not DNP:
     - R228, R229, R232, R234
  2. Please make sure R136 is weld for GPIO card detect.
- *EVK-MCXN5XX:*
  - Short: JP7 2-3, JP8 2-3, JP10 2-3, JP11 2-3
- *RW612BGA:*
  - Connect: JP50; Disconnect JP9, JP11

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
5. *LPCXpresso55s69:*
   - Insert source of sound to audio Line-In connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
******************************
Maestro audio record demo start
******************************

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"record_mic": Record MIC audio and perform one (or more) of following actions:
- playback on codec
- perform VoiceSeeker processing
- perform voice recognition (VIT)
- store samples to a file.

USAGE: record_mic [audio|file|<file_name>|vit] 20 [<language>]
The number defines length of recording in seconds.

Please see the project defined symbols for the languages supported.
Then specify one of: en/cn/de/es/fr/it/ja/ko/pt/tr as the language parameter.
For voice recognition say supported WakeWord and in 3s frame supported command.
Please note that this VIT demo is near-field and uses 1 on-board microphone.

NOTES: This command returns to shell after the recording is finished.
       To store samples to a file, the "file" option can be used to create a file
       with a predefined name, or any file name (without whitespaces) can be specified
       instead of the "file" option.

"opus_encode": Initializes the streamer with the Opus memory-to-memory pipeline and
encodes a hardcoded buffer.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. There are several options how to configure the example settings, depending on the environment. For configuration using west and Kconfig, please follow the instructions here. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
    5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.
    6. Enable VoiceSeeker, see point bellow.
  - *Note:*
    * The audio stream is as follows:
      - Stereo INPUT 1 (J12) -> LINE 1&2 OUTPUT (J6)
      - Stereo INPUT 2 (J15) -> LINE 3&4 OUTPUT (J7)
      - MIC1 & MIC2 (P1, P2) -> LINE 5&6 OUTPUT (J8)
      - Insert the headphones into the different line outputs to hear the inputs.

        - To use the Stereo INPUT 1, 2, connect an audio source LINE IN jack.

- **Enable VoiceSeeker:**
  - On some development boards the VoiceSeeker is enabled by default, see the *table* above.
  - If more than one channel is used and VIT is enabled, the VoiceSeeker that combines multiple channels into one must be used, as VIT can only work with mono signal.
  - Using MCUXPresso IDE:
    * It is necessary to add VOICE_SEEKER_PROC symbol to preprocessor defines on project level:
      - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
  - Using Kconfig:
    * Enable the VoiceSeeker in the guiconfig using MCUX_PRJSEG_middleware.audio_voice.components.voice_seeker
- **Enable VIT:**
  - *LPCXpresso55s69 and MCX-N5XX:*
    * In MCUXPresso IDE (SDK package):
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level.
      2. Add VIT_PROC symbol to preprocessor defines on project level:
         - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
    * In armgcc in SDK package:
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
      2. Remove OPUS_ENCODE=1 and STREAMER_ENABLE_ENCODER preprocessor defines in flags.cmake file.
      3. Add VIT_PROC symbol to preprocessor defines in flags.cmake file.
      4. Remove sdmmc_config.c,.h files from CMakeLists.txt file.
    * In Kconfig:
      1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
      2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd and MCUX_COMPONENT_middleware.sdmmc.host.usdhc
      3. Make sure sdmmc_config files (.c, .h) is excluded from project build
         - remove mcux_add_source function that adds the sources in reconfig.cmake in maestro_record/cm33_core0 folder
      4. Disable fatfs MCUX_COMPONENT_middleware.fatfs and MCUX_COMPONENT_middleware.fatfs.sd
      5. Disable file utils MCUX_COMPONENT_middleware.audio_voice.maestro.file_utils.enable
      6. Make sure Opus encoder is disabled MCUX_COMPONENT_middleware.audio_voice.maestro.element.encoder.opus.enable
      7. Make sure VIT_PROC symbol is defined

        - remove mcux_remove_macro function that removes the VIT_PROC preprocessor definition in reconfig.cmake in maestro_record folder

    8. Make sure VIT processing is enabled MCUX_PRJSEG_middleware.audio_voice.components.vit

- **VIT model generation:**
  - For custom VIT model generation (defining own wake words and voice commands) please use https://vit.nxp.com/
- **Disable SD card handling:**
  - In MCUXPresso IDE:
    - Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level:
      - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
  - In armgcc in SDK package:
    - Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
  - In Kconfig:
    1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
    2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd

Functionality The record_mic or opus_encode command calls the STREAMER_mic_Create or STREAMER_opusmem2mem_Create function from the app_streamer.c file depending on the selected mode.

- When the *Loopback* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements:
  - ELEMENT_MICROPHONE_INDEX
  - ELEMENT_SPEAKER_INDEX
- When the *File recording* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_FILE_SINK_INDEX
- When the *Voice control* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_VOICESEEKER_INDEX (If VOICE_SEEKER_PROC is defined) - ELEMENT_VIT_INDEX
- When the Encoding mode is selected, the command calls the STREAMER_opusmem2mem_Create function that creates a pipeline with the following elements: - ELEMENT_MEM_SRC_INDEX - ELEMENT_ENCODER_INDEX - ELEMENT_MEM_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
    ELEMENT_PROPERTY_T prop;

    prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
    prop.val  = DEMO_MIC_CHANNEL_NUM;
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_MICROPHONE_SET_BITS_PER_SAMPLE;
    prop.val  = DEMO_AUDIO_BIT_WIDTH;
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
    prop.val  = DEMO_MIC_FRAME_SIZE;
    streamer_set_property(handle->streamer, 0, prop, true);

    prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
    prop.val  = DEMO_AUDIO_SAMPLE_RATE;
    streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *record_mic audio <time>*
- *record_mic file <time>*
- *record_mic <file_name> <time>*
- *record_mic vit <time> <language>*
- *opus_encode*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
```

```
B((Running)):::state --> C
C --> E((No state
change)):::state
```

record_mic audio <time>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> D{time
    > 0 ?}:::condition
    D -- Yes --> F[recording]:::function
    D -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    F --> C((Running)):::state
    C -->G{time
    expired?}:::condition
    G -- No --> C
    G -- Yes --> B
```

record_mic file <time>/record_mic <file_name> <time>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> D{SD card
    inserted?}:::condition
    C -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    D -- Yes --> G{Custom
    file name?}:::condition
    G -- Yes --> H[Create custom
    file name]:::function
    G -- No --> I[Create default
    file name]:::function
    H --> J[Recording]:::function
    I --> J
    J --> K((Running)):::state
    K -->L{time
    expired?}:::condition
    L -- No --> K
    L -- Yes --> B
    D -- No --> F[Error: Insert SD
    card first]:::error
    F --> B
```

record_mic vit <time> <language>

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> E{Selected
    language?}:::condition
    C -- No --> D[Error: Record length
    must be greater than 0]:::error
    D --> B
    E -- Yes --> G{Supported
    language?}:::condition
    E -- No --> F[Error: Language
    not selected]:::error
    F -->B
    G -- Yes -->I[Recording with
    voice recognition]:::function
    G -- No -->H[Error: Language not supported]:::error
    H --> B
    I --> J((Running)):::state
    J -->K{time
    expired?}:::condition
    K -- No --> J
    K -- Yes --> B
```

opus_encode

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    B((Idle)):::state -->C[Encode file]:::function
    C -->D[Check result]:::function
    D -->B
```

Processing Time Typical execution times of the streamer pipeline for the EVKC-MIMXRT1060 development board are detailed in the following table. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines were considered. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone, voice seeker, and VIT blocks.

For further details of execution times on individual elements, please refer to the *Processing Time* document.

	streamer
microphone -> speaker 1 channel	40 µs
microphone -> speaker 2 channels	115 µs
microphone -> VIT	7.4 ms
microphone -> voice seeker -> VIT	9.9 ms

Maestro sync example

Table of content

Overview The Maestro sync example demonstrates the use of synchronous pipelines (Tx and Rx in this case) processing in a single streamer task on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The feature is useful for testing the latency of the pipeline or implementing algorithms requiring reference signals such as echo cancellation. The VoiceSeeker library available in this example is not featuring AEC (Acoustic Echo Cancellation), but NXP is offering it in the premium version of the library. More information about the premium version can be found at VoiceSeeker. page. The demo uses two pipelines running synchronously in a single streamer task:

1. Playback (Tx) pipeline:
    - Playback of audio data in PCM format stored in flash memory to the audio Line-Out connector (speaker).
2. Recording (Rx) pipline:
    - Record audio data using a microphone.
    - VoiceSeeker processing.
    - Wake words + voice commands recognition.
    - Save the VoiceSeeker output to the voiceseeker_output.pcm file on the SD card.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- Addition labraries
  - **VIT:**
    * The VIT is supported only in the MCUXpresso IDE.
  - **VoiceSeeker:**
    * The VoiceSeeker is supported only in the MCUXpresso IDE.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- Micro USB cable
- Speaker with 3.5 mm stereo jack
- Personal computer
- Optional:
  - SD card for file output

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

- *EVKC-MIMXRT1060:*
  1. Please make sure resistors below are removed to be able to use SD-Card.
     - R368, R347, R349, R365, R363
  2. Please Make sure J99 is installed.

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the speaker into the Line-Out connector (headphone jack) on the development board.
5. *Optional:* Insert an SD card into the SD card slot to record to the VoiceSeeker output.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
****************************
Maestro audio sync demo start
****************************

Copyright 2022 NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

>> [APP_SDCARD_Task] SD card drive mounted
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"start [nosdcard]": Starts a streamer task.
            - Initializes the streamer with the Memory->Speaker pipeline and with
              the Microphone->VoiceSeeker->VIT->SDcard pipeline.
            - Runs repeatedly until stop command.
    nosdcard - Doesn't use SD card to store data.

"stop": Stops a running streamer:

"debug [on|off]": Starts / stops debugging.
              - Starts / stops saving VoiceSeeker input data (reference and microphone data)
                to SDRAM.
              - After the stop command, this data is transferred to the SD card.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable the premium version of VoiceSeeker:**
  - The premium version of the VoiceSeeker library with AEC is API compatible with this example.
  - To get the premium version, please visit VoiceSeeker page.
  - The following steps are required to run this example with the VoiceSeeker&AEC library.
    - Link the voiceseeker.a library instead of voiceseeker_no_aec.a.
    - Set the RDSP_ENABLE_AEC definition to 1U in the voiceseeker.h file
- **VIT model generation:**
  - For custom VIT model generation (defining own wake words and voice commands) please use https://vit.nxp.com/

Functionality The start <nosdcard> command calls the STREAMER_Create function from the app_streamer.c file that creates pipelines with the following elements:

- Playback pipeline:

  - ELEMENT_MEM_SRC_INDEX
  - ELEMENT_SPEAKER_INDEX
- Record pipeline:
  - ELEMENT_MICROPHONE_INDEX
  - ELEMENT_VOICESEEKER_INDEX
  - ELEMENT_VIT_PROC_INDEX
  - ELEMENT_FILE_SINK_INDEX (If the nosdcard argument is not used)

Processing itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

MEMSRC_SET_BUFFER_T buf;
buf.location = (int8_t *)TESTAUDIO_DATA;
buf.size     = TESTAUDIO_LEN;

prop.prop = PROP_MEMSRC_SET_BUFF;
prop.val  = (uintptr_t)&buf;
if (STREAM_OK != streamer_set_property(handle->streamer, 0, prop, true))
{
    return kStatus_Fail;
}

prop.prop = PROP_MEMSRC_SET_MEM_TYPE;
prop.val  = AUDIO_DATA;
if (STREAM_OK != streamer_set_property(handle->streamer, 0, prop, true))
{
    return kStatus_Fail;
}

prop.prop = PROP_MEMSRC_SET_SAMPLE_RATE;
prop.val  = DEMO_SAMPLE_RATE;
if (STREAM_OK != streamer_set_property(handle->streamer, 0, prop, true))
{
    return kStatus_Fail;
}
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *start [nosdcard]*
- *stop*
- *debug [on|off]*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
    C --> E((No state
    change)):::state
```

start [nosdcard]

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> B{nosdcard
    parameter?}:::condition
    B -- Yes --> CH[Playing to Line-out and
    recording]:::function
    CH --> L((Running)):::state
    B -- No --> C{Is SD card
    inserted?}:::condition
    C -- Yes --> E[Playing to Line-out and
    recording to SD card]:::function
    E --> F((Running)):::state
    F --> G{Debugging
    is enabled?}:::condition
    G -- No --> F
    G -- Yes --> H[Save reference and
    microphone data to SDRAM]:::function
    H --> F
    C -- No --> D[Error: Insert SD
card first]:::error
    D --> A
    J((Running)):::state --> K[Error: The streamer task is
    already running]:::error
    K --> J
```

stop

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> A
    B((Running)):::state --> C{Is debugging
    enabled?}:::condition
    C --Yes -->E[Copy reference and
microphone data to
the SD card]:::function
    E --> G((Idle)):::state
    C -- No --> G
```

debug [on|off]

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> B[Error: First, start
    the streamer task]:::error
    C((Running)):::state --> D{Any
    parameter?}:::condition
    D -- Yes --> F{Started with
    nosdcard
    parameter?}:::condition
    F -- No --> H[Set debugging]:::function
    H --> C
    F --Yes --> G[Error: Debugging cannot be used]:::error
    G --> C
    D -- No --> E[Error: Use the parameter
    either on or off]:::error
    E --> C
```

Maestro USB microphone example

Table of content

Overview The Maestro USB microphone example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the microphone inputs and sends them to the USB host via the USB bus. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- *Note:*
  1. When connected to MacBook, change the PCM format from (0x02,0x00,) to (0x01,0x00,) in the g_config_descriptor[CONFIG_DESC_SIZE] in the usb_descriptor.c file. Otherwise, it can't be enumerated and noise is present when recording with the QuickTime player because the sampling frequency and bit resolution do not match.
  2. When device functionality is changed, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.
  3. If you're having audio problems on Windows 10 for recorder, please disable signal enhancement as the following if it is enabled and have a try again.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit

- No flow control

3. Download the program to the target board.
4. *LPCXpresso55s69:*
   - Insert source of sound to Audio Line-In connector (headphone jack) on the development board.
5. Connect the second micro USB cable between the PC host and the USB port on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**************************************************
Maestro audio USB microphone solutions demo start
**************************************************

Copyright 2022 NXP
[APP_Shell_Task] start

>> usb_mic -1

Starting maestro usb microphone application
The application will run until the board restarts
[STREAMER] Message Task started
Starting recording
[STREAMER] start usb microphone
Set Cur Volume : 1f00
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_mic": Record MIC audio and playback to the USB port as an audio 2.0
        microphone device.

  USAGE: usb_mic <seconds>
    <seconds> Time in seconds how long the application should run.
              When you enter a negative number the application will
              run until the board restarts.
  EXAMPLE:  The application will run for 20 seconds: usb_mic 20
```

Details of commands can be found *here*.

Example configuration The example only supports one mode and do not support any additional libraries, so the example can't be configured by user.

Functionality The usb_mic command calls the STREAMER_mic_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_USB_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_mic <seconds>*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
```

```
C --> E((No state
change)):::state
```

usb_mic <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{seconds
    == 0?}:::condition
    C -- No --> E{seconds
    < 0?}:::condition
    C -- Yes --> D[Error: Incorrect
    command parameter]:::error
    D -->B
    E -- Yes --> G[recording]:::function
    G --> H((Running)):::state
    H --> H
    E -- No --> F[recording]:::function
    F --> I((Running)):::state
    I --> J{seconds
    expired?}:::condition
    J -- No -->I
    J -- Yes --> B
```

Maestro USB speaker example

Table of content

Overview The Maestro USB speaker example demonstrates audio processing on the ARM cortex core utilizing the Maestro Audio Framework library.

The application is controlled by commands from a shell interface using serial console.

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the USB host and sends them to the audio Line-Out port. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

Depending on target platform or development board there are different modes and features of the demo supported.

- **Standard** - The mode demonstrates playback with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-Channel** - In this mode the device is enumerated as a UAC 5.1. This mode is disabled by default. See the *Example configuration* section to see how to enable the mode.
  - When playing an 5.1 audio file, the example sends only the front-left and front-right channels to the audio Line-Out port (the other channels are ignored), since this example only supports on-board codecs with stereo audio output.

As shown in the table below, the application is supported on several development boards, and each development board may have certain limitations, some development boards may also require hardware modifications or allow to use of an audio expansion board. Therefore, please check the supported features and *Hardware modifications* or *Example configuration* sections before running the demo.

Limitations:

- *Note:*
  - If the USB device audio speaker example uses an ISO IN feedback endpoint, please attach the device to a host like PC which supports feedback function. Otherwise, there might be attachment issue or other problems.

Known issues:

- No known issues.

More information about supported features can be found on the *Supported features* page.

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- Headphones with 3.5 mm stereo jack

Hardware modifications Some development boards need some hardware modifications to run the application. If the development board is not listed here, its default setting is required.

Preparation

1. Connect the first micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Connect the second micro USB cable between the PC host and the USB port on the development board.

5. Insert the headphones into Line-Out connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

Running the demo When the example runs successfully, you should see similar output on the serial terminal as below:

```
**********************************************
Maestro audio USB speaker solutions demo start
**********************************************

Copyright 2022 NXP
[APP_Shell_Task] start

>> usb_speaker -1

Starting maestro usb speaker application
The application will run until the board restarts
[STREAMER] Message Task started
Starting playing
[STREAMER] start usb speaker
Set Cur Volume : fbd5
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_speaker": Play data from the USB port as an audio 2.0
               speaker device.

 USAGE:      usb_speaker <seconds>
  <seconds>  Time in seconds how long the application should run.
             When you enter a negative number the application will
             run until the board restarts.
  EXAMPLE:   The application will run for 20 seconds: usb_speaker 20
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Enable Multi-channel mode:**
  - The feature can be enabled by set the USB_AUDIO_CHANNEL5_1 macro to 1U in the usb_device_descriptor.h file.
  - *Note:* When device functionality is changed, such as UAC 5.1, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.

Functionality The Usb_speaker command calls the STREAMER_speaker_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_USB_SRC_INDEX - ELEMENT_SPEAKER_INDEX

Playback itself can be started with the STREAMER_Start function.

Each of the elements has several properties that can be accessed using the streamer_get_property or streamer_set_property function. These properties allow a user to change the values of the appropriate elements. The list of properties can be found in streamer_element_properties.h. See the example of setting property value in the following piece of code from the app_streamer.c file:

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_USB_SRC_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_NUM_CHANNELS;
prop.val  = 2;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_FRAME_MS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

Some of the predefined values can be found in the streamer_api.h.

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_speaker <seconds>*

Legend for diagrams:

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((State)):::state
    B{Condition}:::condition
    C[Error message]:::error
    D[Process function]:::function
```

help, version

```
flowchart TD
    classDef function fill:#69CA00
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

    A((Idle)):::state --> C[Write help or version]:::function
    B((Running)):::state --> C
```

```
C --> E((No state
change)):::state
```

usb_speaker <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{Duration
    == 0?}:::condition
    C -- No --> E{Duration
    < 0?}:::condition
    C -- Yes --> D[Error: Incorrect
    command parameter]:::error
    D -->B
    E -- Yes --> G[playing]:::function
    G --> H((Running)):::state
    H --> H
    E -- No --> F[playing]:::function
    F --> I((Running)):::state
    I --> J{Duration
    expired?}:::condition
    J -- No -->I
    J -- Yes --> B
```

Supported features The current version of the audio framework supports several optional features. These can be limited to some MCU cores or development boards variants. More information about support can be found on the specific example page:

- *maestro_playback*
- *maestro_record*
- *maestro_usb_mic*
- *maestro_usb_speaker*
- *maestro_sync*

Some features are delivered as prebuilt library and the binaries can be found in the \middleware\audio_voice\components*component*\libs folder. The source code of some features can be found in the \middleware\audio_voice\maestro\src folder.

Decoders Supported decoders and its options are:

Decoder	Sample rates [kHz]	Number of channels	Bit depth
AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
FLAC	8, 11.025, 12, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	16
MP3	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
OPUS	8, 16, 24, 48	1, 2 (mono/stereo)	16
WAV	8, 11.025, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	8, 16, 24

For more details about the reference decoders please see audio-voice-components repository documentation \middleware\audio_voice\components\.

Encoders

- **OPUS encoder** - The current verion of the audio framework only supports a OPUS encoder. For more details about the encoder please see the following link.

Sample rate converters

- **SSRC** - Synchronous sample rate converter. More details about SSRC are available in the User Guide, which is located in middleware\audio_voice\components\ssrc\doc\.
- **ASRC** - Asynchronous sample rate converter is not used in our examples, but it is part of the maestro middleware and can be enabled. To enable ASRC, the maestro_framework_asrc and CMSIS_DSP_Library_Source components must be added to the project. Furthermore, it is necessary to switch from Redlib to Newlib (semihost) library and add a platform definition to the project (e.g. for RT1170: PLATFORM_RT1170_CORTEXM7). Supported platforms can be found in the PL_platformTypes.h file. More details about ASRC are available in the User Guide, which is located in middleware\audio_voice\components\asrc\doc\.

Additional libraries

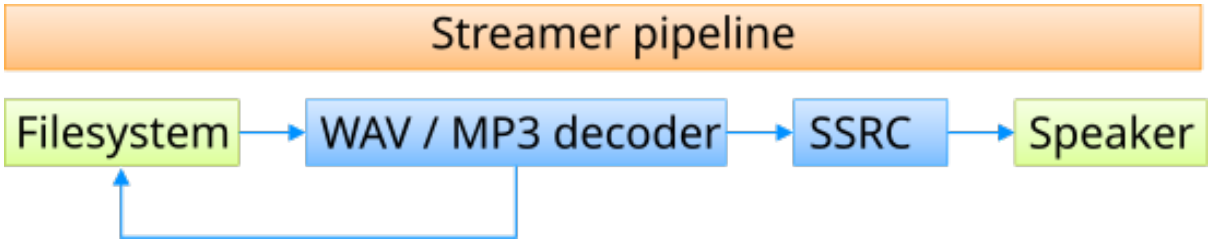
- **VIT** - Voice Intelligent Technology (VIT) Wake Word and Voice Command Engines provide free, ready to use voice UI enablement for developers. It enables customer-defined wake words and commands using free online tools. More details about VIT are available in the VIT package, which is located in middleware\audio_voice\components\vit\{platform}\Doc\(depending on the platform) or via following link.
- **VoiceSeeker** - VoiceSeeker is a multi-microphone voice control audio front-end signal processing solution. More details about VoiceSeeker are available in the VoiceSeeker package, which is located in middleware\audio_voice\components\voice_seeker\{platform}\Doc\ (depending on the platform) or via following link.

Processing Time

Table of content

The individual time measurements were conducted using a logic analyzer by monitoring changes in the GPIO port levels on the EVKC-MIMXRT1060 development board. These measurements were executed for each individual pipeline run, capturing the timing at each corresponding element, and, when relevant, the interconnections between these elements.

Maestro playback example For the Maestro playback example the following reference audio file was used: test_48khz_16bit_2ch.wav. In this example, the pipeline depicted in the diagram was considered. Media codecs WAV and MP3 were taken into account. To compare the times spent on the SSRC block, sampling rates for both codecs were selected: 44.1 kHz and 48 kHz.



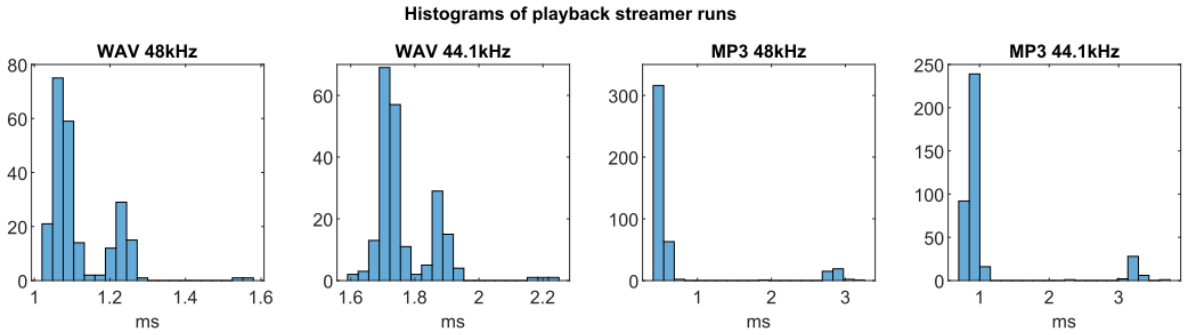
The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines(): streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.

In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read during every run. Rather, it was accessed periodically, triggered when the codec buffer lacked a complete MP3 frame of data. The total time spent on codec processing varies significantly depending on the type and implementation of the codec. For certain types of codecs, like FLAC, there may be multiple file accesses during a single pipeline run. The provided values are specific to the reference implementation. For details about the codecs please see see audio-voice-components documentation middleware\audio_voice\components\.

The duration of the streamer pipeline illustrates that with a sampling frequency of 48 kHz, there is no resampling occurring at the SSRC element. Consequently, the overall pipeline time is lower than in the case of 44.1 kHz audio, where resampling takes place.

To enhance comprehension of the system's behavior, histograms of the pipeline run times and its elements are included. The greater time variance with the MP3 codec is precisely due to the absence of file reads in every run. In clusters with shorter times, there are no file accesses, while in clusters with longer times, file reads occur. This indicates that the majority of runs do not involve file access.

	WAV 48 kHz	WAV 44 kHz	MP3 48 kHz file read	MP3 48 kHz w/o file read	MP3 44 kHz file read	MP3 44 kHz w/o file read
mean	1.11 ms	1.76 ms	2.87 ms	0.51 ms	3.22 ms	0.89 ms
min	1.03 ms	1.60 ms	2.74 ms	0.41 ms	2.33 ms	0.74 ms
max	1.29 ms	2.23 ms	3.24 ms	1.83 ms	3.73 ms	1.12 ms



Time on each element In the tables and histograms below, the timings for individual elements and their connections are provided. Given that the file reading function was invoked during the codec's operation, the tables for individual elements display the total time on the codec element, the time on the codec element before the file read, and the time on the codec element after the file read. The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - time on audio sink without ouput buffers

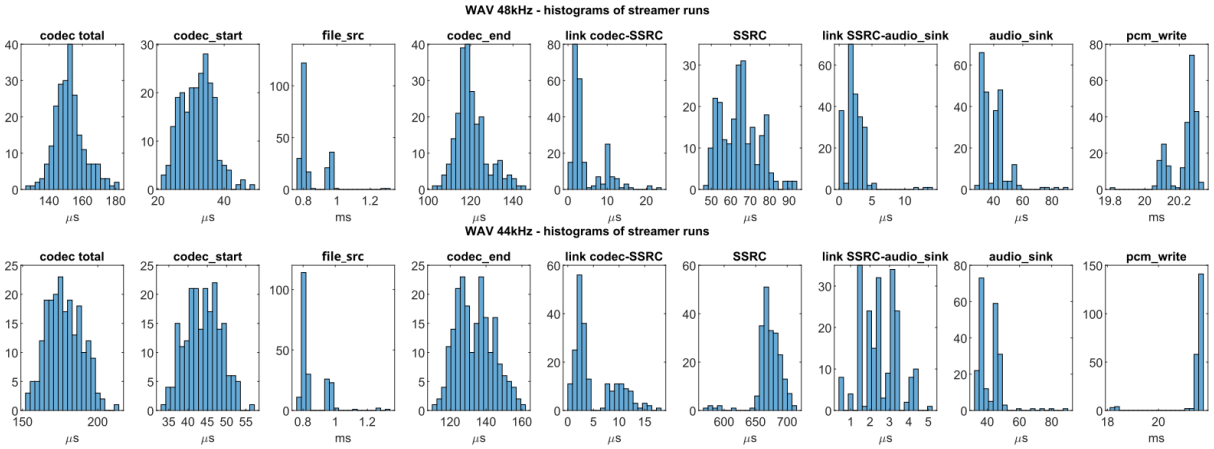
- **pcm_write** - time on output buffers
- **link** - time on element links

The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **codec** - decoder_sink_pad_process_handler():decoder_pads.c
- **file_src** - filesrc_read():file_src_rtos.c
- **SSRC_proc** - SSRC_Proc_Execute():ssrc_proc.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

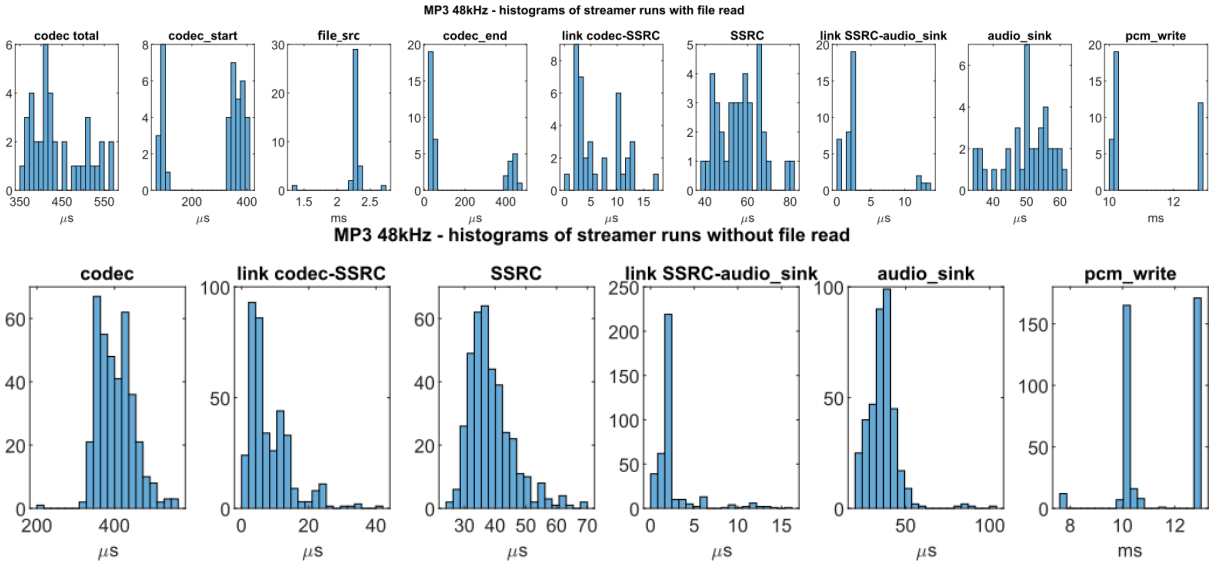
WAV 48kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.119 ms	152 µs	31 µs	0.843 ms	120 µs	5 µs	64 µs	2 µs	40 µs	20.228 ms
min	1.026 ms	125 µs	21 µs	0.773 ms	104 µs	<1 µs	47 µs	<1 µs	30 µs	19.805 ms
max	1.290 ms	193 µs	49 µs	1.311 ms	144 µs	23 µs	93 µs	14 µs	91 µs	20.324 ms

WAV 44kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.765 ms	178 µs	44 µs	0.853 ms	134 µs	5 µs	671 µs	3 µs	42 µs	21.472 ms
min	1.604 ms	145 µs	33 µs	0.770 ms	112 µs	<1 µs	574 µs	<1 µs	33 µs	18.163 ms
max	2.233 ms	218 µs	57 µs	1.335 ms	161 µs	18 µs	715 µs	5 µs	89 µs	21.746 ms



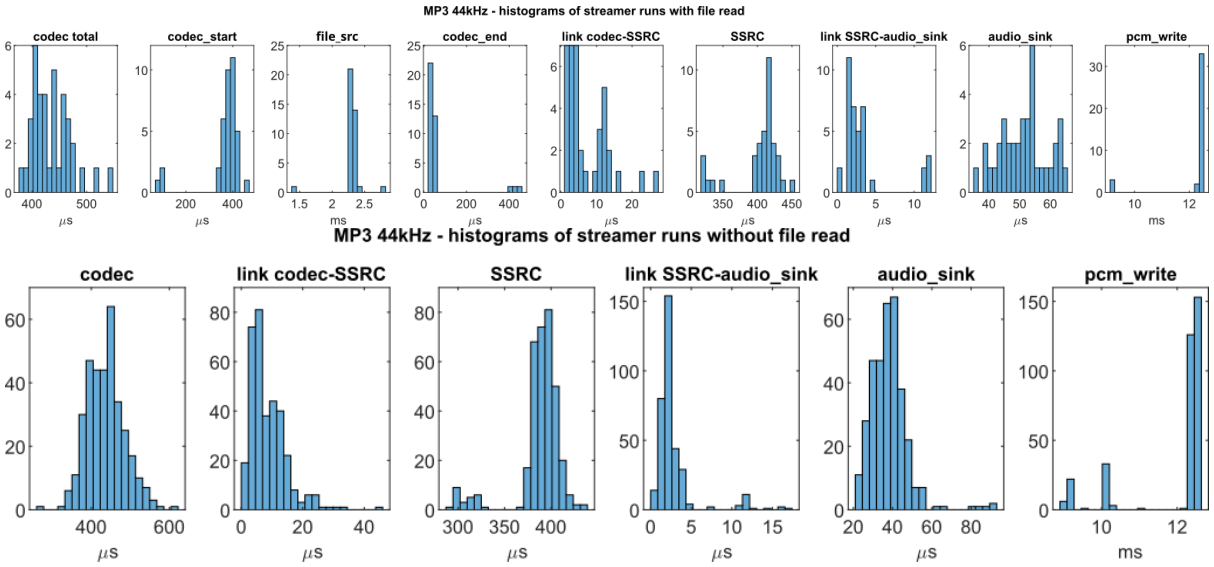
MP3 48 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	2.871 ms	441 µs	279 µs	2.271 ms	162 µs	6 µs	56 µs	3 µs	50 µs	11.019 ms
min	2.739 ms	353 µs	74 µs	1.353 ms	26 µs	<1 µs	40 µs	<1 µs	34 µs	10.091 ms
max	3.244 ms	570 µs	409 µs	2.728 ms	467 µs	18 µs	80 µs	14 µs	62 µs	12.910 ms

MP3 48 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.508 ms	403 µs	x	x	x	8 µs	39 µs	3 µs	36 µs	11.326 ms
min	0.407 ms	208 µs	x	x	x	<1 µs	25 µs	<1 µs	21 µs	7.715 ms
max	1.834 ms	563 µs	x	x	x	41 µs	69 µs	16 µs	104 µs	12.941 ms

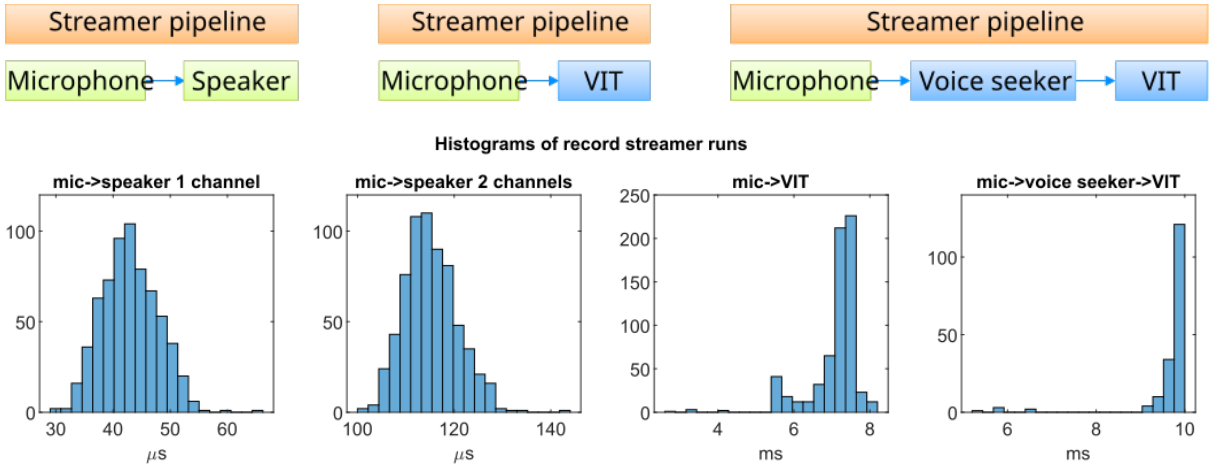


MP3 44 kHz w/ file read	strear	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_l	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	3.217 ms	436 µs	367 µs	2.300 ms	66 µs	7 µs	403 µs	3 µs	51 µs	12.188 ms
min	2.329 ms	383 µs	73 µs	1.411 ms	26 µs	2 µs	318 µs	<1 µs	35 µs	9.119 ms
max	3.726 ms	547 µs	464 µs	2.801 ms	441 µs	27 µs	454 µs	12 µs	65 µs	12.529 ms

MP3 44 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.891 ms	437 µs	x	x	x	9 µs	388 µs	3 µs	38 µs	11.934 ms
min	0.738 ms	268 µs	x	x	x	<1 µs	290 µs	<1 µs	22 µs	8.964 ms
max	1.115 ms	620 µs	x	x	x	45 µs	438 µs	17 µs	92 µs	12.624 ms



Maestro record example Typical execution times of the streamer pipeline and its individual elements for the EVKC-MIMXRT1060 development board are detailed in the following tables. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines are depicted in the figure below. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone, voice seeker, and VIT blocks. The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines():streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.



The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers and without time reading from the microphone
- **audio_src_start** - time on audio src before reading from the microphone
- **audio_src_end** - time on audio src after reading from the microphone
- **pcm_read** - reading from the microphone
- **voiceseeker** - time on voice seeker element
- **vit** - time on VIT element
- **audio_sink** - time on audio sink without ouput buffers
- **pcm_write** - time on output buffers
- **link** - time on element links

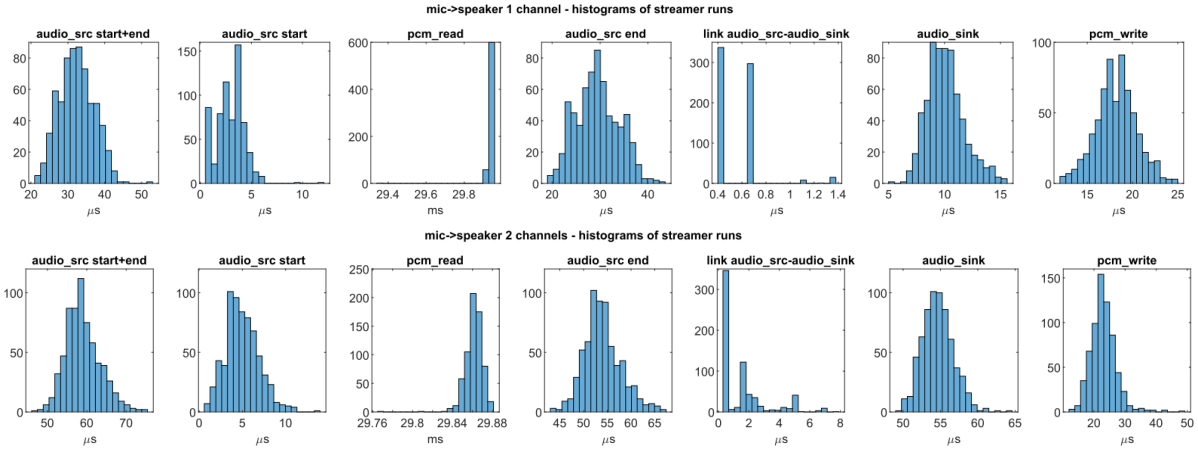
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **audio_src** - audiosrc_src_process():audio_src.c
- **pcm_read** - streamer_pcm_read():streamer_pcm.c
- **voiceseeker** - audio_proc_sink_pad_chain_handler():audio_proc.c
- **vit** - vitsink_sink_pad_chain_handler():vit_sink.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

Pipeline Microphone -> Speaker

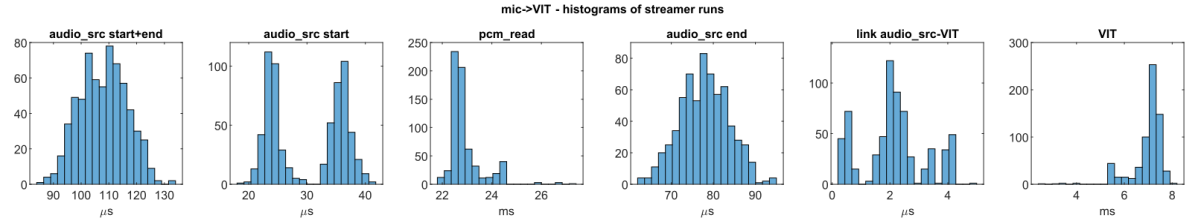
microphone -> speaker mono	stream	audio_src_sta	pcm_re	audio_src_er	link audio_src-audio_sink	audio_sink	pcm_write
mean	43 µs	3 µs	29.938 ms	29 µs	<1 µs	10 µs	18 µs
min	26 µs	<1 µs	29.350 ms	19 µs	<1 µs	5 µs	12 µs
max	72 µs	12 µs	29.957 ms	44 µs	1 µs	15 µs	25 µs

microphone -> speaker stereo	stream	audio_src_sta	pcm_re	audio_src_er	link audio_src-audio_sink	audio_sink	pcm_write
mean	115 µs	5 µs	29.861 ms	54 µs	2 µs	55 µs	23 µs
min	94 µs	<1 µs	29.768 ms	43 µs	<1 µs	50 µs	12 µs
max	154 µs	14 µs	29.880 ms	67 µs	8 µs	65 µs	49 µs



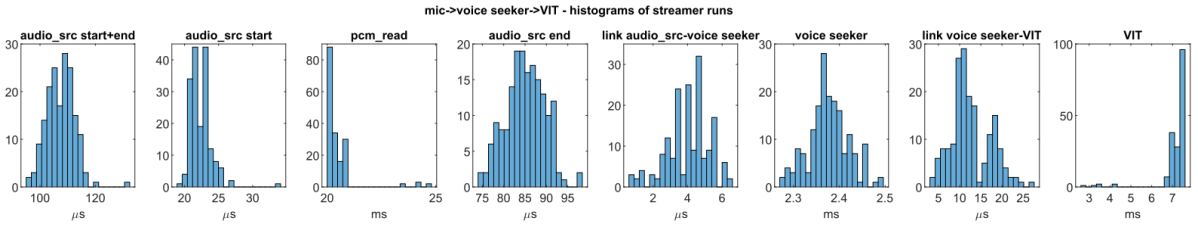
Pipeline Microphone -> VIT

microphone -> VIT	streamer	audio_src_start	pcm_read	audio_src_end	link audio_src-vit	vit
mean	7.380 ms	30 µs	22.624 ms	78 µs	2 µs	7.261 ms
min	2.641 ms	10 µs	2.2265 ms	58 µs	<1 µs	2.559 ms
max	7.780 ms	42 µs	2.7341 ms	94 µs	5 µs	7.624 ms



Pipeline Microphone -> Voice seeker -> VIT

microphone -> voice seeker -> VIT	stream	audio_src_s	pcm_r	audio_src_(	link audio_src-voiceseeker	voice-seeker	link voiceseeke vit	vit
mean	9.916 ms	22 µs	20.084 ms	84 µs	4 µs	2.386 ms	13 µs	7.407 ms
min	4.983 ms	19 µs	19.738 ms	72 µs	<1 µs	2.228 ms	2 µs	2.662 ms
max	10.423 ms	34 µs	24.777 ms	100 µs	7 µs	2.522 ms	31 µs	7.729 ms



Maestro on Zephyr

- Based on and tested with Zephyr version, given by tag v4.0.0
- Tested with Zephyr SDK version 16.4
- To see the pre-built documentation, see: README.html. Also see the *documentation section*.

Maestro sample for recording data from microphone to RAM

Description This sample records data from microphone (alias dmic0 in devicetree) and stores them to a buffer in RAM.

Currently one PDM channel with fixed 16 kHz sample rate and 16 bit sample width is supported.

For configuration options, see Kconfig and prj.conf.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro voice detection sample using VIT

Description Records data from microphone (alias dmic0 in devicetree) and detects voice commands from selected language model. Detected commands are printed via UART.

Language model may be changed via Kconfig using CONFIG_MAESTRO_EXAMPLE_VIT_LANGUAGE selection. For other configuration options, see example's Kconfig and prj.conf.

This project requires an NXP board supported by the VIT library.

The example has to be modified if a new board needs to be added. Please create an issue in that case.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - List of voice commands the model can detect (printed immediately after start)
  - <Specific voice command> if voice command was detected
  - Demo result: FAIL otherwise

Dependencies

- VIT library: https://www.nxp.com/design/design-center/software/embedded-software/voice-intelligent-technology-wake-word-and-voice-command-engines:VOICE-INTELLIGENT-TECHNOLOGY

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro decoder sample

Description Tests and demonstrates decoder functionality in Maestro pipeline.

Supported decoders:

- MP3
- WAV
- AAC
- FLAC
- OPUS with OGG envelop
- (RAW OPUS - TBD)

Data Input:

- Prepared encoded audio data (part of Maestro repository, folder `zephyr/audioTracks`)
- Prepared decoded audio data (RAW PCM format, part of Maestro repository, folder `zephyr/audioTracks`)

Function:

1. Loads encoded data into source buffer stored in RAM
2. Decodes audio data using selected decoder and stores data in RAM
3. Compares prepared data with decoded data to check if its the same
4. Prints `Demo result: OK` or `Demo result: FAIL` via UART

User Input/Output

- Input:

  None

- Output:

  UART Output

  - `Demo result: OK` if everything went OK
  - `Demo result: FAIL` otherwise

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Decoder libraries

Configuration

- See prj.conf for user input sections
  - Selecting decoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_DECODER_SELECTED in prj.conf file. When no decoder is selected, default one (WAV) is used instead.
  - System settings should be modified (stack size, heap size) based on selected decoder and system capabilities/requirements in prj.conf.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms Currently tested for:

- RD_RW612_BGA - Working decoders: FLAC, WAV, OPUS OGG

Maestro encoder sample

Description Tests and demonstrates encoder functionality in Maestro pipeline.

Supported encoders:

- OPUS with OGG envelop - TBD
- RAW OPUS - TBD

Input:

- Prepared decoded audio data (RAW PCM format, part of Maestro repository)
- Prepared encoded audio data (part of Maestro repository)

Function:

1. Loads RAW data into source buffer stored in RAM
2. Encodes audio data using selected encoder and stores data in RAM
3. Compares prepared data with decoded data if same
4. Prints Demo result: OK or Demo result: FAIL via UART

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Encoder libraries

User Input/Output Input:

- None

Output:

- UART Output
  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Configuration

- See prj.conf for user input sections
  - Selecting encoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_ENCODER_SELECTED in prj.conf file. When no encoder is selected, default one (OPUS) is used instead.
  - System settings should be modified (stack size, heap size) based on selected encoder and system capabilities/requirements in prj.conf file.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms Currently tested for:

- RD_RW612_BGA - Working encoders: None.

Maestro mem2mem sample

Description Tests basic memory to memory pipeline.

Function:

1. Moves generated data with fixed size of 256B from memory source to memory sink.
2. Compares copied data to check if they're the same.
3. Returns Demo result: OK or Demo result: FAIL via UART.

Maestro environment setup Follow these steps to set up a Maestro development environment on your machine.

1. If you haven't already, please follow this guide to set up a Zephyr development environment and its dependencies first:
   - Cmake
   - Python
   - Devicetree compiler
   - West
   - Zephyr SDK bundle

2. Get Maestro. You can pick either of the options listed below. If you need help deciding which option is the best fit for your needs, please see the *FAQ*.

   - Freestanding Maestro - This option pulls in only Maestro's necessary dependencies.

     Run:

     ```
     1. west init -m <maestro repository url> --mr <revision> --mf west-freestanding.yml
     ↪<foldername>
     2. cd <foldername>
     3. west update
     ```

   - Maestro as a Zephyr module

     To include Maestro into Zephyr, update Zephyr's west.yml file:

     ```
     projects:
     name: maestro
     url: <maestro repository url>
     revision: <revision with Zephyr support>
     path: modules/audio/maestro
     import: west.yml
     ```

     Then run west update maestro command.

Build and run Maestro example These steps will guide you through building and running Maestro samples. You can use either the command line utilizing Zephyr's powerful west tool or you can use VS Code's GUI. Detailed steps for both options are listed below.

Using command line See Zephyr's Building, Flashing and Debugging guide if you aren't familiar with it yet.

1. To **build** a project, run:

   ```
   west build -b <board> -d <output build directory> <path to example> -p
   ```

   For example, this compiles VIT example for rd_rw612_bga board:

   ```
   1. cd maestro/zephyr
   2. west build -b rd_rw612_bga -d build samples/vit -p
   ```

2. To **run** a project, run:

   ```
   west flash -d <directory>
   ```

   e.g.:

   ```
   west flash -d build
   ```

3. To **debug** a project, run:

   ```
   west debug -d <directory>
   ```

   e.g.:

   ```
   west debug -d build
   ```

Using MCUXpresso for VS Code For this you have to have NXP's MCUXpresso for VS Code extension installed.

1. Import your topdir as a repository to MCUXPresso for VS Code:

- Open the MCUXpresso Extension. In the *Quickstart Panel* click *Import Repository*.
- In the displayed menu click *LOCAL* tab and select the folder location of your *topdir*.
- Click *Import*.
- The repository is successfully added to the *Installed Repositories* view once the import is successful.

2. To import any project from the imported repository:
   - In the *Quickstart Panel* click *Import Example from Repository*.
   - For **Repository** select *your imported* repository.
   - For **Zephyr SDK** the installed Zephyr SDK is selected automatically. If not, select one.
   - For **Board** select your board (*make sure* you've selected the *correct* revision).
   - For **Template** select the folder path to your project.
   - Click the *Create* button.
3. Build the project by clicking the *Build Selected* icon (displayed on hover) in the extension's *Projects* view. After the build, the debug console window displays the memory usage (or compiler errors if any).
4. Debug the project by clicking the *Debug* (play) icon (displayed on hover) in the extension's *Projects* view.
5. The execution will pause. To continue execution click *Continue* on the debug options.
6. In the *SERIAL MONITOR* tab of your console panel, the application prints the Zephyr boot banner during startup and then prints the test results.

Folder structure

```
maestro/
    ...
    zephyr/              All Zephyr related files
      samples/           Sample examples
      tests/             Tests
      audioTracks/       Audio tracks for testing
      doc/               Documentation configuration for Sphinx
      wrappers/          NXP SDK Wrappers
      scripts/           Helper scripts, mostly for testing
      module.yml         Defines module name, Cmake and Kconfig locations
      CMakeList.txt      Defines module's build process
      Kconfig            Defines module's configuration
      osa/               Deprecated. OSA port for Zephyr
      ...
```

Supported elements and libraries Here is the list of all features currently supported in Maestro on Zephyr. Our goal is to support all features in Maestro on Zephyr that are already supported in Maestro on NXP's SDK and to extend them further.

Supported elements:

- Memory source
- Memory sink
- Audio source
- Audio sink
- Process sink

- Decoder
- Encoder

Supported decoders:

- WAV
- MP3
- FLAC
- OPUS OGG
- AAC

Supported encoders:

- OPUS RAW

Supported libraries:

- VIT

Examples support All included examples use UART as output. Examples are located in zephyr/tests and zephyr/samples directories.

List of included examples:

- *Maestro sample for recording data from microphone to RAM*
- *Maestro voice detection sample using VIT*
- *Maestro encoder sample*
- *Maestro decoder sample*
- *Maestro mem2mem sample*

Examples support for specific boards:

Example	RDRW612BGA	LPCx-presso55s69	MIMXRT1060EVKE	MIMXRT1170EVKB
Record	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
VIT	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
Encoder	In progress: OPUS RAW	TO BE TESTED	TO BE TESTED	TO BE TESTED
Decoder	YES - WAV, FLAC, OPUS OGG	TO BE TESTED	TO BE TESTED	TO BE TESTED
Mem2mem	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED

Creating your own example There are two ways to create your own example - you can either one of the included examples as a reference or you can create your own example from scratch by hand.

When creating your own example from scratch, set CONFIG_MAESTRO_AUDIO_FRAMEWORK=y in your prj.conf file. Then you can start enabling specific elements by setting CONFIG_MAESTRO_ELEMENT_<NAME>_ENABLE=y.

However, the recommended way to edit config options is to open gui-config (or menuconfig) by calling west build -t guiconfig. Then you can use the graphical interface to interactively turn on/off the features you need.

Documentation Please note, Maestro documentation is under reconstruction. It is currently mixing several tools and formats.

To see the pre-generated Maestro Zephyr documentation, see zephyr/doc/doc/README.html

To generate the Zephyr documentation, go under zephyr/doc folder and execute make html. Sphinx version sphinx-build 8.1.3 must be installed. Open doc/doc/html/README.hml afterwards.

To see Maestro core documentation, go to the Maestro top directory and see README.md.

FAQ

1. Should I choose the freestanding version of Maestro or should integrate it into my west instead?
    - Freestanding version of Maestro pulls in all the dependencies it needs including Zephyr itself.
    - Integrating it as a module is easier if you already have your Zephyr environment set up.

Maestro Audio Framework changelog

2.0.0 (newest)

- Added Zephyr port, see *Zephyr README*.
    - Possible to use standalone version, pulling its own Zephyr and dependencies
    - Possible to import it as a module in your Zephyr project
- Changed build system - newly uses Kconfig and Cmake
- Supports NXP MCUXSDK (previously 2.x)
- Changed folder structure and names to improve readability (description may be found in *README*)
- Removed audio libraries and placed into audio-voice-components repository
- Added libraries are pulled into the build via Kconfig and Cmake
- Changed Maestro library core - minor changes

1.8.0

- New platforms support: MCX-N5XX-EVK, FRDMMCXN236 and RD-RW612-BGA
- Fixed compilation warnings
- Documentation improvements and updates
    - Added section with processing time information
    - Added application state diagrams
- Various updates and fixes

1.7.0

- Removed EAP support for future SDK releases
- Created new API for audio_sink and audio_src to support USB source, sink
- ASRC library integrated
- License changed to BSD 3-Clause
- Improved pipeline creation API
- Fixed compilation warnings in Opus
- Various other improvements and bug fixes

1.6.0

- Up to 2 parallel pipelines supported
- Synchronous Sample Rate Converter support Added
- Various improvements and bug fixes

1.5.0

- Enabled switching from 2 to 4 channel output during processing
- PadReturn type has been replaced by FlowReturn
- Support of AAC, WAV, FLAC decoders
- Renamed eap element to audio_proc element
- Added audio_proc to VIT pipeline to support VoiceSeeker
- Minor bug fixes

1.4.0

- Use Opusfile lib for Ogg Opus decoder
- Refactor code, fix issues found in unit tests
- Various bug fixes

1.3.0

- Make Maestro framework open source (except mp3 and wav decoder)
- Refactor code, remove unused parts, add comments

1.2.0

- Unified buffering in audio source, audio sink
- Various improvements and bug fixes

1.0_rev0

- Initial version of framework with support for Cortex-M7 platforms

1.7 Wireless

1.7.1 NXP Wireless Framework and Stacks

Wi-Fi, Bluetooth, 802.15.4

Application notes

- Link AN12918-Wi-Fi-Tx-Power-Table-and-Channel-Scan-Management-for-i.MX-RT-SDK.pdf
- Link TN00066-WFA-Derivative-Certification-Process.pdf

User manuals

- Link UM11441-Getting-Started-with-NXP-based-Wireless-Modules-and-i.MX-RT-Platforms.pdf
- UM11442-NXP-Wi-Fi-and-Bluetooth-Demo-Applications-for-i.MX-RT-Platforms.pdf
- Link UM11443-NXP-Wi-Fi-and-Bluetooth-Debug-Feature-Configuration-Guide-for-i.MX-RT-Platforms.pdf
- Link UM11567-WFA-Certification-Guide-for-NXP-based-Wireless-Modules-on-i.MX-RT-Platform-Running-RTOS.pdf

Release notes

Wireless SoC features and release notes for FreeRTOS

About this document This document provides information about the supported features, release versions, fixed and/or known issues, performance of the Wi-Fi, Bluetooth/802.15.4 radios, including the coexistence.

The SDK release version 25.06.00 has been tested for the wireless SoCs listed in Supported products.

Supported products

- 88W8987
- IW416
- IW611[1]
- IW612[2]
- AW611[3]
- RW610
- RW612

Parent topic:*About this document*

[1] The support of IW611 is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.
[2] The support of IW612 is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.
[3] AW611 module support is available only in i.MX RT1180 EVKA

Features

Wi-Fi radio

Client mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) TX and RX sup
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) 4k TX and RX su
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC) 1x1 20 MHz and 40 MHz
802.11n - High throughput	HT Beamformee (explicit)
802.11ac - Very high throughput	2.4 GHz band supported channel bandwidth: 20MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 80 MHz
802.11ac - Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac - Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9) - 1x1
802.11ac - Very high throughput	MU-MIMO Beamformee (Explicit and Implicit)
802.11ac - Very high throughput	RTS/CTS with BW signaling
802.11ac - Very high throughput	Operation mode notification
802.11ac - Very high throughput	Backward compatibility with non-VHT devices
802.11ac - Very high throughput	TX VHT MCS rate adaptation
802.11ac - Very high throughput	Low density parity check (LDPC)
802.11ax - High efficiency	2.4 GHz band supported channel bandwidth: 20MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 20 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 40 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidths: 80 MHz
802.11ax - High efficiency	OFDMA (UL/DL, 106 RU)
802.11ax - High efficiency	OFDMA (UL/DL, 484 RU)
802.11ax - High efficiency	1024 QAM
802.11ax - High efficiency	Target wake time (TWT)
802.11ax - High efficiency	256 QAM modulation – MCS8 and MCS9
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 2.4 GHz
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 5 GHz
802.11ax - High efficiency	DCM
802.11ax - High efficiency	DCM
802.11ax - High efficiency	ER (extended range)
802.11ax - High efficiency	SU Beamforming
802.11ax - High efficiency	OMI (operating mode indication)
802.11a/b/g features	802.11b/g data rates up to 54 Mbit/s
802.11a/b/g features	802.11a data rates up to 54 Mbit/s
802.11a/b/g features	TX rate adaptation (BG)
802.11a/b/g features	Fragmentation/defragmentation
802.11a/b/g features	ERP protection, slot time, preamble

Table 1 – continued from p

Features	Sub features
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Opensource WPA Supplicant Support
802.11i security	WPA2-PSK AES | WPA Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | WPA
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | W
802.11i security	Wi-Fi Enhanced Open - OWE (Opportunistic Wireless Encry
802.11i security	802.1x EAP Authentication Methods | WPA Supplicant
802.11i security	WPA2-Enterprise Mixed Mode | WPA Supplicant
802.11i security	WPA3-Enterprise (Suite-B) |National Security Algorithm (CS
802.11i security	802.11w - PMF (Protected Management Frames) | WPA Supp
802.11i security	Embedded Supplicant Support
802.11i security	WPA2-PSK AES | Embedded Supplicant
802.11i security	WPA+WPA2 PSK Mixed Mode | Embedded Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Embe
802.11i security	802.11w - PMF (Protected Management Frames) | Embedde
802.11i security	Wi-Fi Roaming
802.11i security	WPA3 Enterprise
Power save mode	Deep sleep
Power save mode	IEEE power save
Power save mode	Host sleep/WoWLAN (inband)
Power save mode	Host sleep/WoWLAN (outband)
Power save mode	U-APSD
802.11w - PMF (protected management frames)	PMF require and capable
802.11w - PMF (protected management frames)	Unicast management frames - Encryption/decryption - using
802.11w - PMF (protected management frames)	Broadcast management frames - Encryption/decryption - us
802.11w - PMF (protected management frames)	SA query request/response
802.11w - PMF (protected management frames)	PMF support using embedded supplicant
DPP functionality	Wi-Fi easy connect[^3]
General features	Embedded supplicant
General features	Host sleep packet filtering
General features	Host-based supplicant
General features	Embedded MLME
General features	EDMAC - EU adaptivity support (ETSI certification)
General features	External coexistence
General features	IPv6 NS offload
General features	FIPS
General features	TKIP[^1]
General features	RF test mode
General features	802.11k
General features	802.11v
General features	DFS radar detection in peripheral mode (follow AP)[^5]
General features	Embedded roaming based on RSSI threshold beacon loss
General features	ARP offload
General features	Cloud keep alive
General features	UNII-4 channel support
General features	ClockSync using TSF
General features	Auto reconnect
General features	CSI (channel state information)
General features	Independent reset (in-band)[^3]
General features	Independent reset (out-band)[^3]
General features	Wi-Fi agile multiband
General features	Network co-processor (NCP) mode
General features	802.11mc - WLS (Wi-Fi location service)

Table 1 – continued from p

Features	Sub features
General features	802.11az

Parent topic: Wi-Fi radio

[^1] As per Wi-Fi specification, connecting in TKIP security in non 802.11n mode is allowed.

[^2] Support available in host-base supplicant.

[^3] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

[^4] Read more about NCP feature in *References*. **[^5]** To enable the feature, CONFIG_ECSA = 1 must be defined in wifi_config.h (does not apply to RW610 and RW612).

AP mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) Rx support
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) -4k RX support
802.11n - High throughput	Max client support (up to 8 devices)
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC)
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 80MHz
802.11ac – Very high throughput	Short/long guard interval (400ns/800ns)
802.11ac – Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac – Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9)
802.11ac – Very high throughput	Single user- Aggregated MAC protocol data unit (SU-AMPDU
802.11ac – Very high throughput	RTS/CTS with BW signaling
802.11ac – Very high throughput	Backward compatibility with non-VHT devices
802.11ac – Very high throughput	TX VHT MCS rate adaptation
802.11ac – Very high throughput	MU-MIMO Beamformee (explicit and implicit)
802.11ac – Very high throughput	Operation mode notification
802.11ax – High efficiency	2.4 GHz band operation (20 MHz channel bandwidth)
802.11ax – High efficiency	2.4 GHz band operation (40 MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (20MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (40MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (80 MHz channel bandwidth)
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e -QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Hostapd Support
802.11i security	WPA2-PSK AES | hostapd
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Hosta
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | H

Table 2 – continued from previ

Features	Sub features
802.11i security	Wi-Fi Enhanced Open - OWE (Opportunistic Wireless Encry
802.11i security	802.1x EAP Authentication Methods | Hostapd
802.11i security	WPA2-Enterprise Mixed Mode | Hostapd
802.11i security	WPA3-Enterprise (Suite-B) |National Security Algorithm (CS
802.11i security	802.11w - PMF (Protected Management Frames) | Hostapd
802.11i security	Embedded Authenticator
802.11i security	WPA2-PSK AES | Embedded Supplicant
802.11i security	WPA+WPA2 PSK Mixed Mode | Embedded Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Embe
802.11i security	802.11w - PMF (Protected Management Frames) | Embedde
802.11y	Extended channel switch announcement (ECSA)
802.11w - protected management frames (PMF)	PMF require and capable
802.11w - protected management frames (PMF)	Unicast management frames -Encryption/decryption - using
802.11w - protected management frames (PMF)	Broadcast management frames -encryption/decryption - usi
802.11w - protected management frames (PMF)	SA query request/response
General features	Embedded authenticator
General features	Embedded MLME
General features	EU adaptivity support
General features	Automatic channel selection (ACS)
General features	External coexistence (software interface)
General features	Independent reset (in-band)[^1]
General features	Network co-processor (NCP) mode[^2]
General features	Vendor specific IE (custom IE)
General features	Hidden SSID (broadcast SSID disabled)
General features	MAC address filter
General features	Multiple external STA support

Parent topic: Wi-Fi radio

[^1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory. **[^2]** Read more about NCP feature in *References*.

AP-STA mode

Features	Sub features	88W8	IW41	IW611/IV	RW610/R	IW61	AW611
Simultaneous AP-STA operation (same channel)	AP-STA functionality	Y	Y	Y	Y	Y	Y
SAD	Software antenna diversity[^1]	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[^1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

Parent topic:*Features*

Wi-Fi Generic features

Features	Sub features	88W898	IW41(	IW611/IW(	RW610/RW(	IW61(	AW611
Generi(	Firmware download (parallel)[^1]	Y	Y	Y	N	N	Y
Generi(	Secure boot	N	N	Y	Y	Y	Y
Generi(	Kconfig memory optimizer	Y	Y	Y	Y	Y	Y
Generi(	Firmware Compression[^2]	N	Y	N	N	N	N
Generi(	u-AP intra-BSS	Y	N	Y	Y	Y	Y
Generi(	Net Monitor Mode	N	N	N	Y	Y	N
Generi(	Net Monitor Mode with packet transmission	N	N	N	Y	Y	N
Generi(	In-Channel Net Monitor mode	N	N	N	N	N	N

Parent topic: Wi-Fi radio

[^1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature. **[^2]** The feature is used to compress the Wi-Fi Bluetooth firmware and optimize the flashing of the host

Wi-Fi direct/P2P

Features	Sub features	88W8987	IW416[/	IW611/IW61	RW610/RW61	IW610[/	AW611[^3]
P2P basic functionality[^1]	P2P Auto GO	Y	Y	Y	Y	Y	Y
P2P basic functionality[^1]	P2P GO	Y	Y	Y	Y	Y	Y
P2P basic functionality[^1]	P2P GC	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[^1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature. **[^2]** This is an experimental software release for this feature for IW416. **[^3]** Contact your support representative to use this feature for.

Bluetooth radio

Bluetooth classic

Feature	Sub feature	88W8	IW4'	IW611/	RW610/	IW6'	AW611
General features	Bluetooth Class 1.5 and Class 2 support	Y	Y	Y	N	N	Y
General features	Scatternet support	Y	Y	Y	N	N	Y
General features	Maximum of seven simultaneous ACL connections – Central links	Y	Y	Y	N	N	Y
General features	Automatic packet type selection	Y	Y	Y	N	N	Y
General features	Bluetooth - 2.1 to 5.0 specification support	Y	Y	Y	N	N	Y
General features	Low power sniff	Y	Y	Y	N	N	Y
General features	Deep sleep using out-of-band	Y	Y	N	N	N	N
General features	Wake on Bluetooth (SoC to host)	Y	Y	Y	N	N	Y
General features	Independent reset (in-band)[^1]	Y	Y	Y	Y	N	Y
General features	Independent reset (out-band)[^1]	Y	Y	N	N	N	N
General features	Firmware download (parallel)[^1]	Y	Y	N	N	N	N
General features	RF test mode	Y	Y	Y	N	N	Y
Bluetooth packet type supported	ACL (DM1, DH1, DM3, DH3, DM5, DH5, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	SCO (HV1, HV3)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	eSCO (EV3, EV4, EV5, 2EV3, 3EV3, 2EV5, 3EV5)	Y	Y	Y	N	N	Y
Bluetooth profiles supported	A2DP source/sink	Y	Y	Y	N	N	Y
Bluetooth profiles supported	AVRCP target/controller	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HFP Dev/AG	Y	Y	Y	N	N	Y
Bluetooth profiles supported	OPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	SPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HID target/device	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM NBS central/peripheral	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM WBS central/peripheral	Y	Y	Y	N	N	Y

Parent topic: Bluetooth radio

[^1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

Bluetooth LE

Features	Sub features
Generic features	Maximum 16 Bluetooth LE connections (central role)
Generic features	Deep sleep using out-of-band
Generic features	Wake on Bluetooth LE (SoC to Host)
Generic features	RF Test mode
Bluetooth profile support	Bluetooth LE GATT
Bluetooth profile support	Bluetooth LE HID over GATT
Bluetooth profile support	Bluetooth LE GAP
Bluetooth LE 4.0 support	Low Energy physical layer
Bluetooth LE 4.0 support	Low Energy link layer
Bluetooth LE 4.0 support	Enhancements to HCI for Low Energy
Bluetooth LE 4.0 support	Low energy direct test mode
Bluetooth 4.1 support	Low duty cycle directed advertising
Bluetooth 4.1 support	Bluetooth LE dual mode topology
Bluetooth 4.1 support	Bluetooth LE privacy v1.1
Bluetooth 4.1 support	Bluetooth LE link layer topology
Bluetooth 4.2 support	Bluetooth LE secure connection
Bluetooth 4.2 support	Bluetooth LE link layer privacy v1.2
Bluetooth 4.2 support	Bluetooth LE data length extension
Bluetooth 4.2 support	Link layer extended scanner filter policies
Bluetooth 5.0 support	Bluetooth LE 2 Mbps support
Bluetooth 5.0 support	High duty cycle directed advertising
Bluetooth 5.0 support	Low Energy advertising extension
Bluetooth 5.0 support	Low Energy long range
Bluetooth 5.0 support	Low Energy periodic advertisement
Bluetooth 5.2 support	Low Energy power control
Bluetooth LE audio support[^1] [^2]	Isochronous channel
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS source
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS sink
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIG Validation
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio framed mode
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio unframed mode
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio sequential packing
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio: Mono and Stereo
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS encrypted audio
Bluetooth LE audio support[^1] [^2]	Broadcast LE Audio BIS unencrypted audio
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS source
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS sink
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIG validation
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS synchronization
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio framed mode
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio unframed mode
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio sequential packing
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio: mono and stereo
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS encrypted audio
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio CIS unencrypted audio
Bluetooth LE audio support[^1] [^2]	Unicast LE Audio TX/RX and bidirectional traffic

Table 3 – continued from prev

Features	Sub features
Bluetooth LE audio support[^1] [^2]	ISO interval for LE Audio: 7.5ms 10ms 20ms 30ms
Bluetooth LE audio support[^1] [^2]	Sampling frequency for LE Audio: 8kHz 16kHz 24kHz, 32kH
Bluetooth LE audio support[^1] [^2]	LE Audio Auracast use cases: Auracast streaming 2 BISes
Bluetooth LE audio support[^1] [^2]	LE Audio Unicast use cases: Unicast streaming 2 CISes
Bluetooth LE audio support[^1] [^2]	LE Audio Unicast Use cases: Unicast streaming 4 CISes
Bluetooth LE audio support[^1] [^2]	A2DP + Auracast/Unicast Bridge use cases – CIS/BIS
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth + Bluetooth LE coexistence

Note: Details of the tested Bluetooth LE Audio use cases:

- Number of streams:
  - 1-CIG | upto 4-CIS with 1 LE ACL (for 4-CIS: execute only mono UCs, SDU Int: 10ms)
  - 1-CIG | upto 4-CIS with 4 separate LE ACL (for 4-CIS: SDU Size= Max 100 Oct, PHY=2M, RTN=1, SDU Int: 10ms only) (execute only mono UCs for 4-CIS)
  - 1-BIG | upto 4-BIS (for 4-BIS: execute only mono UCs, SDU Int: 10ms only)
- PHY: 2M and 1M
- Audio mode: mono (for 1 to 4 streams) and stereo (for 1 stream)
- Packing: sequential and interleaved
- Bit rate: maximum 96kbps
  - For 1-CIG with upto 3-CIS: maximum bit rate 96kbps
  - For 1-CIG with 4-CIS: maximum bit rate 80kbps
  - For 1-BIG with 4-BIS: maximum bit rate 80kbps
  - For 2-CIG cases: maximum bit rate 80kbps
- Mode: unframed mode
- 48_5 and 48_6 mono and stereo configurations are not supported.

Details of the tested Bluetooth coexistence (Bluetooth + Bluetooth LE Audio) use cases:

- Bluetooth + Bluetooth LE Audio
- A2DP + Bluetooth LE Audio bridging support
- A2DP sink link (central) -> LEA 2-CIS (SDU Int: 10ms only | A2DP only with SBC Codec | PHY: 2M)

Parent topic: Bluetooth radio

[^1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[^2] LE audio feature is supported for standalone scenarios only and not for BR/EDR and Wi-Fi coexistence scenarios such as LE audio + BR/EDR link or LE audio + Wi-Fi link. From the

perspective of NXP Edgefast Bluetooth host stack, LE audio feature can be disabled by the CONFIG_BT_AUDIO macro without impact on any other features. LE audio feature can be tested by the user, using their own supported host stack.

Parent topic:*Features*

802.15.4 radio

Features	Sub features	IW612	IW610	RW612
General features	Spinel over SPI	Y	N	N
General features	OpenThread RCP Mode implementing Thread1.3	Y	N	N
General features	802.15.4-2015 MAC/PHY as required by Thread 1.3	Y	Y	Y
General features	OpenThread Border Router (OTBR) v1.1	Y	Y	Y
General features	Direct/indirect transmission with/without ACK	Y	Y	Y
General features	802.15.4 CSL parent feature implementation	Y	Y	Y
General features	Enhanced Frame Pending	Y	Y	Y
General features	Enhanced keep alive	Y	Y	Y
General features	Router	Y	Y	Y
General features	Leader	Y	Y	Y
General features	Router Eligible End Device (REED)	Y	Y	Y
General features	End Device (FED, MED)	Y	Y	Y
Zigbee features	Coordinator	N	N	Y
Zigbee features	Router	N	N	Y
Zigbee features	End Device (RX ON)	N	N	Y
Zigbee features	R23	N	N	Y
Zigbee features	OTA Client	N	N	Y
Zigbee features	OTA server	N	N	Y
Matter features	Matter over Wi-Fi	Y	N	N
Matter features	Matter over Thread	Y	N	Y

Parent topic:*Features*

Coexistence

Wi-Fi and Bluetooth/802.15.4 coexistence

Features	Sub features	IW6´	IW6´	RW612
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	STA + Bluetooth	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Mobile AP + Bluetooth	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + Wi-Fi	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth LE[^2]	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth + Bluetooth LE	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Single antenna configuration	Y	Y	Y
BCA_TDM separate antenna[^1] (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	External Coexistence PTA	N	Y	Y

Parent topic: Coexistence

[^1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[^2] The narrow-band radio can be configured to support Bluetooth LE, 802.15.4, and to time-slice between Bluetooth LE and 802.15.4.

Parent topic:*Features*

Feature enable and memory impact

Features	Macros to enable the feature	Memory impact
CSI	CONFIG_CSI	Flash - 60K, RAM - 4K
DPP	CONFIG_WPA_SUPP_DPP	Flash - 240K, RAM - 12K
Independent reset	CONFIG_WIFI_IND_DNLDCONFIG_WIFI_IND_RESET	Minimal
Parallel firmware download Wi-Fi	CONFIG_WIFI_IND_DNLD	Minimal
Parallel firmware download Bluetooth	CONFIG_BT_IND_DNLD	Minimal
WPA3 enterprise	CONFIG_WPA_SUPP_CRYPTO_ENTERPRISE [Macros specific to EAP-methods included] CONFIG_EAP_TLS CONFIG_EAP_PEAP CONFIG_EAP_TTLS CONFIG_EAP_FAST CONFIG_EAP_SIM CONFIG_EAP_AKA CONFIG_EAP_AKA_PRIME	Flash - 165K, RAM - 18K
WPA2 enterprise	CONFIG_WPA_SUPP_CRYPTO_ENTERPRISE [Macros specific to EAP-methods included] CONFIG_EAP_TLS CONFIG_EAP_PEAP CONFIG_EAP_TTLS CONFIG_EAP_FAST CONFIG_EAP_SIM CONFIG_EAP_AKA CONFIG_EAP_AKA_PRIME	Flash - 165K, RAM - 18K
Host sleep	CONFIG_HOST_SLEEP	Minimal
WMM	CONFIG_WMM[^1]	Flash - 10K, RAM - 57K
802.11mc	CONFIG_11MC CONFIG_CSI CONFIG_WLS_CSI_PROC[^2] CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
802.11az	CONFIG_11MC CONFIG_CSI[2] CONFIG_WLS_CSI_PROC[^2] CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
Non-blocking firmware download mechanism	CONFIG_FW_DNLD_ASYNC	—
Antenna diversity	CONFIG_WLAN_CALDATA_2ANT_DIVERSITY	-
P2P	CONFIG_WPA_SUPP_P2P	-

Note:

- For Wi-Fi, the macros are set with the value "**0**" by default in the file wifi_config_default.h located in <SDK_PATH>/middleware/wifi_nxp/incl/ directory.

  To enable the features, set the value of the macros to "***1****" in the file wifi_config.h located in*<SDK_Wi-Fi_Example_PATH>/ *directory****.***

- Bluetooth

  To enable the features, set the value of the macros to "**1**" in the file app_bluetooth_config.h located in <SDK_Bluetooth_Example_PATH>/ directory.

Kconfig memory optimizer The MCUXpresso SDK provides options to reduce the host memory usage with build-time configuration parameters referred to as Kconfig memory optimizer. The configuration parameters are used to reduce the use of the flash memory and SRAM.

This section explains how to enable the host memory saving configurations within the Wi-Fi drivers of NXP wireless devices.

Memory impact of i.MX RT1060 EVKC + IW416 module (Murata 1XK):

- Maximum Flash usage: 889 KB
- Maximum SRAM usage: 418.77 KB

To reduce the use of the flash and SRAM, change the settings of the Kconfig macros listed in table in the file wifi_config.h located in <path-to-SDK_Wi-Fi_Example> directory.

Kconfig macros	Feature disabled
CONFIG_WIFI_SL	CONFIG_ROAMING CONFIG_11R
CONFIG_WIFI_SL	CONFIG_CLOUD_KEEP_ALIVE CONFIG_WIFI_EU_CRYPTO CONFIG_TX_AMPDU_PROT_MODE CONFIG_WNM_PS CONFIG_TURBO_MODE CONFIG_AUTO_RECONNECT CONFIG_DRIVER_OWE CONFIG_OWE CONFIG_WIFI_FORCE_RTS CONFIG_WIFI_FRAG_THRESHOLD CONFIG_COMBO_SCAN CONFIG_SCAN_CHANNEL_GAP
CONFIG_WIFI_SL	CONFIG_UAP_STA_MAC_ADDR_FILTER CONFIG_WIFI_MAX_CLIENTS_CNT
CONFIG_FREERTC	If the macro is enabled, the heap memory usage is reduced by 10 KB (from 70 KB to 60 KB).
CONFIG_LWIP_LC	Curtails LWIP stack parameters, reduces data throughput, disables data net-stats
Non-blocking firmware download mechanism	CONFIG_FW_DNLD_ASYNC

Parent topic:*Feature enable and memory impact*

[^1] The macro is not used for IW416.

[^2] Prerequisite macros for 802.11mc and 802.11az features

88W8987 release notes

Package information

- SDK version: 25.06.00

Parent topic:*88W8987 release notes*

Version information

- Wireless SoC: 88W8987
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p151.7
  - 16 - Major revision
  - 91 - Feature pack
  - 21 - Release version
  - p151.7 - Patch number

Parent topic:*88W8987 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS.
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*88W8987 release notes*

Wi-Fi and Bluetooth certification The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | 802.11ac
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification QDID: refer to *4*.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic:*88W8987 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- External Access Point: ASUS AX88U
- DUT: W8987 Murata (Module: **1ZM M.2**) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- External Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

Parent topic: Wi-Fi throughput

STA throughput External APs: ASUS AX88U

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	46	51	60	60
WPA2-AES	45	42	60	54
WPA3-SAE	46	41	60	54

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	62	83	121	124
WPA2-AES	61	82	120	126
WPA3-SAE	60	82	120	126

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	52	60	64
WPA2-AES	43	52	61	64
WPA3-SAE	43	52	60	65

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	64	87	126	125
WPA2-AES	63	85	125	120
WPA3-SAE	63	80	125	123

STA mode throughput - AC Mode | 5 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	48	60	73	78
WPA2-AES	47	60	73	77
WPA3-SAE	47	60	73	77

STA mode throughput - AC Mode | 5 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	68	96	161	157
WPA2-AES	69	92	160	155
WPA3-SAE	70	94	160	155

STA mode throughput - AC Mode | 5 GHz Band | 80 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	78	99	125	203
WPA2-AES	78	98	126	197
WPA3-SAE	82	98	125	197

Parent topic: Wi-Fi throughput

Mobile AP throughput External client: Apple Macbook Air

Mobile AP Mode Throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	51	56	62
WPA2-AES	42	50	54	61
WPA3-SAE	40	50	65	62

Mobile AP Mode Throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	66	81	107	121
WPA2-AES	65	80	107	120
WPA3-SAE	65	80	108	120

Mobile AP Mode Throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	52	60	61
WPA2-AES	44	51	60	61
WPA3-SAE	44	51	60	61

Mobile AP Mode Throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	70	89	126	103
WPA2-AES	70	87	124	102
WPA3-SAE	70	88	125	103

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	60	73	76
WPA2-AES	48	59	73	76
WPA3-SAE	48	60	73	76

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	106	161	102
WPA2-AES	77	104	160	102
WPA3-SAE	77	104	160	111

Mobile AP Mode Throughput - AC Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	93	114	125	206
WPA2-AES	92	111	125	191
WPA3-SAE	92	111	125	173

Parent topic: Wi-Fi throughput

Parent topic:*88W8987 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic:*88W8987 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode-Associated event is received even when connecting using wrong password WFA APUT Low iperf TCP/UDP Tx throughput with Realtek station

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	In wrong password scenario, After updating new password the phone is not able to connect with DUTAP

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p142.5 {#firmware_version_from_16_91_21_p133_to_16_91_21_p142_5}

Component	Description
Wi-Fi	Fails to encrypt and decrypt data with ccmp 128 and 256 using CLI crypto commands.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p142.5 to 16.91.21.p149.2 {#firmware_version_from_16_91_21_p142_5_to_16_91_21_p149_2}

Component	Description
Wi-Fi	DUTSTA does not associate to hidden SSID beaconing in DFS channel.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.2 to 16.92.21.p151.7 {#firmware_version_from_16_91_21_p149_2_to_16_92_21_p151_7}

Component	Description
Wi-Fi	Getting low TCP/UDP TP in DUT-AP 11ac-vht80 mode after hard-reset or wlan-reset.

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*88W8987 release notes*

Known issues

Component	Description
-	NA

Parent topic:*88W8987 release notes*

IW416 release notes

Package information

- SDK version: 25.06.00

Parent topic:*IW416 release notes*

Version information

- Wireless SoC: IW416
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p151.7
  - 16 - Major revision

  - 92 - Feature pack
  - 21 - Release version
  - p151.7 - Patch number

Parent topic:*IW416 release notes*

Host platform

- All i.MX RT platforms running FreeRTOS.
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 Support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*IW416 release notes*

Wi-Fi and Bluetooth certification The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic:*IW416 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW416 Murata (Module: 1XK M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

Parent topic: Wi-Fi throughput

STA throughput External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	38	59	58
WPA2-AES	38	36	57	58
WPA3-SAE	41	35	57	57

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	58	58	93	91
WPA2-AES	56	49	94	74
WPA3-SAE	54	57	94	73

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	45	61	58
WPA2-AES	40	43	61	57
WPA3-SAE	40	44	61	57

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	60	66	94	100
WPA2-AES	58	61	94	98
WPA3-SAE	59	61	94	98

Parent topic: Wi-Fi throughput

Mobile AP throughput External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	43	59	57
WPA2-AES	40	42	59	57
WPA3-SAE	39	42	59	57

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	62	74	121	118
WPA2-AES	60	64	116	91
WPA3-SAE	60	65	116	91

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	50	42	45	62
WPA2-AES	42	45	53	62
WPA3-SAE	42	62	53	45

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	66	76	126	96
WPA2-AES	63	68	121	95
WPA3-SAE	63	67	121	95

Parent topic: Wi-Fi throughput

Parent topic:*IW416 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic:*IW416 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p133.2 {#firmware_version_from_16_91_21_p133_to_16_91_21_p133_2}

Component	Description
Wi-Fi	DUT STA getting rebooted after 15~20 iterations of 11R-Command based roaming0xa4 command timeout after several hours of stress test

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133.2 to 16.91.21.p142.5 {#firmware_version_from_16_91_21_p133_2_to_16_91_21_p142_5}

Component	Description
Wi-Fi	DUT fails to reconnect after the configured auto-reconnect time interval.
Coex	During HFP call, TX side noise is observed with coex CLI

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p142.5 to 16.91.21.p149.4 {#firmware_version_from_16_91_21_p142_5_to_16_91_21_p149_4}

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p149.4 to 16.92.21.p151.7 {#firmware_version_from_16_91_21_p149_4_to_16_92_21_p151_7}

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*IW416 release notes*

Known issues

Component	Description
Coex	Wi-Fi connection in 2.4GHz is not stable, observed deauthentication within 10sec.

Parent topic:*IW416 release notes*

IW611/IW612 release notes **Note:** The IW611/IW612 support is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.

Package information

- SDK version: 25.06.00

Parent topic:*IW611/IW612 release notes*

Version information

- Wireless SoC: IW611/IW612
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.3.p25.11
  - 18 - Major revision
  - 99 - Feature pack
  - 3 - Release version
  - p25.11 - Patch number

Parent topic:*IW611/IW612 release notes*

Host platform

- i.MX RT1170 EVKB and i.MX RT1060 EVKC Platforms running FreeRTOS
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
  - 802.15.4 over SPI (IW612 only)
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*IW611/IW612 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic:*IW611/IW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW612 Murata (Module: 2EL M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

iPerf host configuration and impact on throughput {#iperf_host_configuration_and_impact_on_throughpu
To get the highest throughput, the throughput values shown in STA throughput and Mobile AP throughput are measured with the maximum values of the default host configuration macros. STA and AP throughput captured with the minimum values of the host configuration macros shows the throughput numbers obtained when using the minimum values of the host configuration macros. The macro values are defined in *lwipopts.h* file.

The table below lists the minimum and maximum values of the host configuration macros.

Values of the host configuration macros

Parameter	Maximum value	Minimum value
TCPIP_MBOX_SIZE	96	32
DEFAULT_RAW_RECVMBOX_SIZE	32	12
DEFAULT_UDP_RECVMBOX_SIZE	64	12
DEFAULT_TCP_RECVMBOX_SIZE	64	12
TCP_MSS	1460	536
TCP_SND_BUF	24 * TCP_MSS	2 * TCP_MSS
MEM_SIZE	319160	41,080
TCP_WND	15 * TCP_MSS	10 * TCP_MSS
MEMP_NUM_PBUF	20	10
MEMP_NUM_TCP_SEG	96	12
MEMP_NUM_TCPIP_MSG_INPKT	80	16
MEMP_NUM_TCPIP_MSG_API	80	8
MEMP_NUM_NETBUF	32	16

STA and AP throughput captured with the minimum values of the host configuration macros {#sta_and_ap_throughput_captured_with_the_minimum_values_of_the_host_configuration_macr

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	7	18	111	124
WPA2-AES	7	18	110	124
WPA3-SAE	6	18	110	124

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	2	19	93	127
WPA2-AES	2	19	105	126
WPA3-SAE	2	19	104	132

Parent topic:iPerf host configuration and impact on throughput

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	47	45	60
WPA2-AES	43	46	48	59
WPA3-SAE	47	49	63	63

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	68	82	131	131
WPA2-AES	72	82	130	129
WPA3-SAE	68	81	129	130

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	45	51	63	65
WPA2-AES	45	51	63	65
WPA3-SAE	45	51	63	65

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	67	83	129	134
WPA2-AES	66	83	129	133
WPA3-SAE	65	83	129	133

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	53	72	71
WPA2-AES	48	52	73	70
WPA3-SAE	52	56	75	75

STA mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	88	172	172
WPA2-AES	75	92	171	169
WPA3-SAE	77	92	172	171

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	50	58	76	78
WPA2-AES	49	57	76	77
WPA3-SAE	49	57	76	77

STA mode throughput - VHT Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	93	175	177
WPA2-AES	74	93	174	174
WPA3-SAE	73	93	173	175

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	94	221	196
WPA2-AES	87	95	219	194
WPA3-SAE	89	95	219	195

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	60	60	98	115
WPA2-AES	62	61	94	113
WPA3-SAE	61	59	97	108

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	71	215	190
WPA2-AES	77	72	212	187
WPA3-SAE	76	72	152	189

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	63	65	127	128
WPA2-AES	63	67	125	128
WPA3-SAE	63	67	125	126

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	79	64	212	199
WPA2-AES	78	68	218	199
WPA3-SAE	79	68	217	198

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	70	219	192
WPA2-AES	87	72	219	193
WPA3-SAE	91	72	220	194

Parent topic: Wi-Fi throughput

Mobile AP throughput External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	52	61	62
WPA2-AES	40	51	61	62
WPA3-SAE	40	51	61	62

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	63	83	116	130
WPA2-AES	67	82	115	131
WPA3-SAE	60	81	115	132

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	49	60	62
WPA2-AES	39	49	60	62
WPA3-SAE	41	51	63	62

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	66	89	128	133
WPA2-AES	64	87	128	133
WPA3-SAE	62	86	128	133

Mobile AP mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	60	74	75
WPA2-AES	43	59	74	75
WPA3-SAE	43	59	74	75

Mobile AP mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	64	53	112	75
WPA2-AES	65	51	111	75
WPA3-SAE	62	51	112	75

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	45	60	76	76
WPA2-AES	44	59	76	76
WPA3-SAE	44	59	76	76

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	73	100	155	178
WPA2-AES	72	99	152	176
WPA3-SAE	72	99	152	176

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	80	86	211	189
WPA2-AES	84	87	223	188
WPA3-SAE	83	94	224	192

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	53	66	85	123
WPA2-AES	52	65	83	122
WPA3-SAE	52	65	83	120

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	75	102	124	166
WPA2-AES	74	100	121	148
WPA3-SAE	73	101	121	154

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	54	68	84	124
WPA2-AES	53	66	83	122
WPA3-SAE	54	66	83	123

Mobile AP mode throughput - HE Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	76	107	155	193
WPA2-AES	75	105	152	192
WPA3-SAE	76	106	151	191

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	86	118	220	187
WPA2-AES	86	119	221	185
WPA3-SAE	86	116	220	188

Parent topic: Wi-Fi throughput

Parent topic:*IW611/IW612 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic:*IW611/IW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p7.19 to 18.99.2.p49.9

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p49.9 to 18.99.2.p155

Component	Description
Bluetooth	Audio lost occurs due to periodic adv sync lost, during 2 BIS 44.1kHz unencrypted streams with 1M PHY configuration.BIS sync loss may occur in long audio streaming sessions.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p155 to 18.99.2.p66.30

Component	Description
Wi-Fi	802.11R Fast BSS roaming works only with hostapd and does not work with standard APs (supporting 11R)
Bluetooth	DUT is not able to sustain a connection with the remote device that does extended advertisement with coded PHY configuration. When 2 CIS streams are active, after the first device disconnects followed by the second device disconnecting, the second peripheral device hangs.Audio Play/Pause does not work in BIS case.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.2.p66.30 to 18.99.3.p10.5

Component	Description
Wi-Fi	STAUT not sending Neighbor Advertisement packet after receiving Neighbor Solicitation packet from Ex-AP.Antenna selection time exceeds configured evaluation time
Bluetooth	When DUT works as CIS source and CIS Offset is 612us, high packet drops observed which affects the audio streaming.For BIS Source Use Cases, Periodic Interval and ISO Interval should be multiple of each other value.In 1-CIS and 2-CIS, Continuous Audio Glitches are observed with 96 kbps bit rate.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9 {#firmware_version_18_99_3_p10_5_to_18_99_3_p17_9}

Component	Description
Wi-Fi	After performing independent reset (out-of-band mode), the STAUT fails to connect to the external AP via `wlan-connect` command, observed command timeout `0x107` error.
Bluetooth	Audio glitches observed with Google Pixel 7 Pro streaming audio after CIS is established with DUT.During Call Gateway (CG) / Call Terminal (CT) Use Case, the firmware periodically sends NULL PDU, which results in frequent Audio Glitch on both CG and CT sides.Heavy audio glitches observed with CIS SRC Google Pixel 7 ProContinuous audio glitches observed in 1 CIS and 2 CIS for 48_3 and 48_4 config.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p17.9 to 18.99.3.p21.154 {#firmware_version_18_99_3_p17_9_to_18_99_3_p21_154}

Component	Description
Wi-Fi	STAUT fail to ping AP backend machine when connected with DFS channel and DUTSTA went in bad state.
Bluetooth	CIS Sink frequently fails to acknowledge CIS Source TX PDU.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p21.154 to 18.99.3.p23.16 {#firmware_version_18_99_3_p21_154_to_18_99_3_p23_16}

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p23.16 to 18.99.3.p25.11 {#firmware_version_18_99_3_p23_16_to_18_99_3_p25_11}

Component	Description
Bluetooth	Packet lost observed in CIS case, which causes audio noise.

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*IW611/IW612 release notes*

Known issues

Component	Description
Bluetootl	Sequential Removal of CIS Handles as per current Controller implementation i.e CIS Disconnection sequence should be in sequence => CIS - 4,3,2,1While 4-CIS streaming, audio glitches observed on all CIS SINK with Samsung Galaxy budsWhile 4-CIS streaming, disconnection with connection timeout observed on first CIS SINK with Samsung Galaxy budsOnly two streams (CIS/BIS) with one channel is supported.

Parent topic:*IW611/IW612 release notes*

RW610/RW612 release notes

Package information

- SDK version: 25.06.00

Parent topic:*RW610/RW612 release notes*

Version information

- Wi-Fi firmware version: 18.99.6.p40
  - rw61x_sb_wifi_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 6 - Release version
  - p40 - Patch number
- Bluetooth LE firmware version: 18.25.6.p40
  - rw61x_sb_ble_a2.bin for A2
  - 18 - Major revision
  - 25 - Feature pack
  - 6 - Release version
  - p40 - Patch number
- 802.15.4 and Bluetooth LE (up to core 4.1) firmware version: 18.34.6.p40
  - rw61x_sb_ble_15d4_combo_a2.bin for A2
  - 18 - Major revision
  - 34 - Feature pack
  - 6 - Release version
  - p40 - Patch number

Parent topic:*RW610/RW612 release notes*

Host platform

- RW610/RW612 platform running FreeRTOS
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*RW610/RW612 release notes*

Wireless certification The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *1*.

Note: This release supports STAUT only certifications.

Parent topic: Wireless certification

Bluetooth LE controller certification QDID: Refer to *4*.

Parent topic: Wireless certification

Thread Thread group: refer to *7*.

Product Name: NXP RW612 Wireless MCU with Integrated Tri-Radio

Thread version: V1.3.0

CID #: 13A109

Parent topic: Wireless certification

Matter RW612 certification: refer to *8*.

Certificate ID: CSA23C36MAT41746-24

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi

RW610 certification: refer to *9*.

Certificate ID: CSA23C43MAT41753-50

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi and Matter over Thread

Parent topic: Wireless certification

Parent topic:*RW610/RW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: RW610/RW612
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli
- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	38	38	62	62
WPA2-AES	37	37	61	63
WPA3-SAE	37	37	60	61

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	39	64	64
WPA2-AES	37	38	62	64
WPA3-SAE	39	38	62	64

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	41	41	75	74
WPA2-AES	41	41	73	74
WPA3-SAE	40	41	72	73

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	42	42	76	76
WPA2-AES	42	41	75	75
WPA3-SAE	42	41	75	74

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	45	97	99
WPA2-AES	43	44	96	98
WPA3-SAE	42	44	97	98

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	47	47	100	103
WPA2-AES	45	46	100	101
WPA3-SAE	47	46	100	101

Parent topic: Wi-Fi throughput

Mobile AP throughput External client: Apple MacBook Air

Mobile AP throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	39	62	62
WPA2-AES	39	39	61	61
WPA3-SAE	38	39	61	61

Mobile AP throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	40	63	63
WPA2-AES	39	39	62	61
WPA3-SAE	39	39	62	61

Mobile AP throughput - VHT Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	43	73	73
WPA2-AES	43	42	72	72
WPA3-SAE	43	42	73	72

Mobile AP throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	44	74	74
WPA2-AES	43	43	74	74
WPA3-SAE	43	43	74	74

Mobile AP throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	48	48	95	96
WPA2-AES	47	47	98	95
WPA3-SAE	47	47	97	95

Mobile AP throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	49	96	97
WPA2-AES	48	48	101	97
WPA3-SAE	48	48	101	97

Parent topic: Wi-Fi throughput

Parent topic:*RW610/RW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p34 to 18.99.6.p40 {#firmware_version_18_99_6_p34_to_18_99_6_p40}

Com-ponent	Description
Zigbee	Zigbee Coordinator and Router are disconnected during BLE connection pairing and bonding with a mobile app for the first time.

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*RW610/RW612 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—
Zigbee	-

Parent topic:*RW610/RW612 release notes*

IW610 release notes

Package information

- SDK version: 25.06.00

Parent topic:*IW610 release notes*

Version information

- Wireless SoC: IW610
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.5.p66
  - 18 - Major revision
  - 99 - Feature pack
  - 5 - Release version
  - p66 - Patch number

Parent topic:*IW610 release notes*

Host platform

- IW610 platform running FreeRTOS
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*IW610 release notes*

Wi-Fi and Bluetooth certification The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification QDID: Refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic:*IW610 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW610
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli
- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	42	63	61
WPA2-AES	40	47	60	62
WPA3-SAE	40	39	60	62

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	42	49	64	64
WPA2-AES	41	48	62	63
WPA3-SAE	41	48	62	63

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	45	54	75	74
WPA2-AES	45	53	73	73
WPA3-SAE	44	53	73	72

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	46	48	77	70
WPA2-AES	45	47	74	68
WPA3-SAE	46	47	74	68

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	51	62	98	98
WPA2-AES	50	60	96	91
WPA3-SAE	51	60	96	91

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	53	60	101	94
WPA2-AES	53	58	99	93
WPA3-SAE	52	58	99	93

Parent topic: Wi-Fi throughput

Mobile AP throughput External client: Apple MacBook Air

Mobile AP throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	39	63	62
WPA2-AES	39	38	60	60
WPA3-SAE	39	38	60	60

Mobile AP throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	39	64	62
WPA2-AES	39	39	61	61
WPA3-SAE	39	38	61	61

Mobile AP throughput - VHT Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	43	74	73
WPA2-AES	42	42	73	71
WPA3-SAE	42	42	74	72

Mobile AP throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	43	75	73
WPA2-AES	43	42	74	72
WPA3-SAE	43	43	74	72

Mobile AP throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	46	46	95	94
WPA2-AES	45	45	96	91
WPA3-SAE	45	45	96	91

Mobile AP throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	47	47	96	94
WPA2-AES	46	46	98	91
WPA3-SAE	46	46	99	91

Parent topic: Wi-Fi throughput

Parent topic:*IW610 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—

Parent topic:*IW610 release notes*

AW611 release notes

Note: The AW611 support is enabled in i.MX RT1180 EVKA.

Package information

- SDK version: 25.06.00

Parent topic:*AW611 release notes*

Version information

- Wireless SoC: AW611
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.3.p25.11
  - 18 - Major revision
  - 99 - Feature pack
  - 3 - Release version
  - p25.11 - Patch number

Parent topic:*AW611 release notes*

Host platform

- i.MX RT1180 EVKA Platform running FreeRTOS
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 Support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*AW611 release notes*

Wi-Fi and Bluetooth certification The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification QDID: Refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic:*AW611 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: AW611 uBlox (Module: U-BLOX_Jody_W5 M.2) with EVK-MIMXRT1180 EVKA platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version: gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*.

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

STA throughput External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	47	45	60
WPA2-AES	43	46	48	59
WPA3-SAE	47	49	63	63

STA mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	68	82	131	131
WPA2-AES	72	82	130	129
WPA3-SAE	68	81	129	130

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	45	51	63	65
WPA2-AES	45	51	63	65
WPA3-SAE	45	51	63	65

STA mode throughput - AN Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	67	83	129	134
WPA2-AES	66	83	129	133
WPA3-SAE	65	83	129	133

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	53	72	71
WPA2-AES	48	52	73	70
WPA3-SAE	52	56	75	75

STA mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	88	172	172
WPA2-AES	75	92	171	169
WPA3-SAE	77	92	172	171

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	50	58	76	78
WPA2-AES	49	57	76	77
WPA3-SAE	49	57	76	77

STA mode throughput - VHT Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	93	175	177
WPA2-AES	74	93	174	174
WPA3-SAE	73	93	173	175

STA mode throughput - VHT Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	94	221	196
WPA2-AES	87	95	219	194
WPA3-SAE	89	95	219	195

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	60	60	98	115
WPA2-AES	62	61	94	113
WPA3-SAE	61	59	97	108

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	77	71	215	190
WPA2-AES	77	72	212	187
WPA3-SAE	76	72	152	189

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	63	65	127	128
WPA2-AES	63	67	125	128
WPA3-SAE	63	67	125	126

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	79	64	212	199
WPA2-AES	78	68	218	199
WPA3-SAE	79	68	217	198

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	88	70	219	192
WPA2-AES	87	72	219	193
WPA3-SAE	91	72	220	194

Parent topic: Wi-Fi throughput

Mobile AP throughput External client: Apple MacBook Air

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	52	61	62
WPA2-AES	40	51	61	62
WPA3-SAE	40	51	61	62

Mobile AP mode throughput - BGN Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	63	83	116	130
WPA2-AES	67	82	115	131
WPA3-SAE	60	81	115	132

Mobile AP mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	49	60	62
WPA2-AES	39	49	60	62
WPA3-SAE	41	51	63	62

Mobile AP mode throughput - AN Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	66	89	128	133
WPA2-AES	64	87	128	133
WPA3-SAE	62	86	128	133

Mobile AP mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	60	74	75
WPA2-AES	43	59	74	75
WPA3-SAE	43	59	74	75

Mobile AP mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	64	53	112	75
WPA2-AES	65	51	111	75
WPA3-SAE	62	51	112	75

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	45	60	76	76
WPA2-AES	44	59	76	76
WPA3-SAE	44	59	76	76

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	73	100	155	178
WPA2-AES	72	99	152	176
WPA3-SAE	72	99	152	176

Mobile AP mode throughput - VHT Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	80	86	211	189
WPA2-AES	84	87	223	188
WPA3-SAE	83	94	224	192

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	53	66	85	123
WPA2-AES	52	65	83	122
WPA3-SAE	52	65	83	120

Mobile AP mode throughput - HE Mode | 2.4 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	75	102	124	166
WPA2-AES	74	100	121	148
WPA3-SAE	73	101	121	154

Mobile AP mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	54	68	84	124
WPA2-AES	53	66	83	122
WPA3-SAE	54	66	83	123

Mobile AP mode throughput - HE Mode | 5 GHz Band | 40 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	76	107	155	193
WPA2-AES	75	105	152	192
WPA3-SAE	76	106	151	191

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	86	118	220	187
WPA2-AES	86	119	221	185
WPA3-SAE	86	116	220	188

Parent topic: Wi-Fi throughput

Parent topic:*AW611 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic:*AW611 release notes*

Bug fixes and/or feature enhancements {#bug_fixes_and_or_feature_enhancements_04}

Firmware version: 18.99.3.p10.5 to 18.99.3.p17.9 {#firmware_version_18_99_3_p10_5_to_18_99_3_p17_9_0}

Component	Description
Wi-Fi	After performing independent reset (out-of-band mode), the STAUT fails to connect to the external AP via `wlan-connect` command, observed command timeout `0x107` error.
Bluetooth	Audio glitches observed with Google Pixel 7 Pro streaming audio after CIS is established with DUT.During Call Gateway (CG) / Call Terminal (CT) Use Case, the firmware periodically sends NULL PDU, which results in frequent Audio Glitch on both CG and CT sides.Heavy audio glitches observed with CIS SRC Google Pixel 7 ProContinuous audio glitches observed in 1 CIS and 2 CIS for 48_3 and 48_4 config.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p17.9 to 18.99.3.p21.154 {#firmware_version_18_99_3_p17_9_to_18_99_3_p21_154_0}

Component	Description
Wi-Fi	STAUT fail to ping AP backend machine when connected with DFS channel and DUTSTA went in bad state.
Bluetooth	CIS Sink frequently fails to acknowledge CIS Source TX PDU.

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*AW611 release notes*

Known issues

Component	Description
Bluetootl	Packet lost would be observed in CIS case which causes audio noise.Sequential Removal of CIS Handles as per current Controller implementation i.e CIS Disconnection sequence should be in sequence => CIS - 4,3,2,1While 4-CIS streaming, audio glitches observed on all CIS SINK with Samsung Galaxy budsWhile 4-CIS streaming, disconnection with connection timeout observed on first CIS SINK with Samsung Galaxy budsOnly two streams (CIS/BIS) with one channel is supported.

Parent topic:*AW611 release notes*

Abbreviations

Abbreviation	Definition
A2DP	Advanced audio distribution profile
AMPDU	Aggregated MAC protocol data unit
AMSDU	Aggregated MAC service data unit
AP	Access point
BW	Bandwidth
CCMP	Counter mode CBC-MAC protocol
CSI	Channel state information
CTS	Clear To Send
DL	Down link
EDCA	Enhanced distributed channel access
ER	Extended range
ERP	Extended rate physical
GATT	Generic attribute profile
HFP	Hands free profile
HID	Human interface device
HT	High throughput
LDPC	Low density parity check
MCS	Modulation and coding scheme
MLME	Mac layer management entity
OMI	Operating mode indication
PMF	Protected management frames
RTS	Request to send

continues on next page

Table 4 – continued from previous page

Abbreviation	Definition
SAE	Simultaneous authentication of equals
STA	Station
TWT	Target wake time
UL	Up link
VHT	Very high throughput
WEP	Wired equivalent private
WFD	Wi-Fi direct
WMM	Wireless multi-media
WPA	Wi-Fi protected access
WPS	Wi-Fi protected setup
WSC	Wi-Fi Simple Configuration

References

1. Application note - AN13681 – Wi-Fi Alliance (WFA) Derivative Certification Process (available in the SDK package)
2. User manual – UM11442 - NXP Wi-Fi and Bluetooth Demo Applications User Guide for i.MX RT Platforms (available in the SDK package)
3. User manual – UM11799 - NXP Wi-Fi and Bluetooth Demo Applications User Guide for RW61x (available in the SDK package)
4. Certification – Bluetooth controller - QDID (*link*)
5. User manual - UM12133 - NXP NCP Application Guide for RW612 with MCU Host
6. Technical note - TN00066 – Wi-Fi Alliance (WFA) Derivative Certification Process (available in the SDK package)
7. Web page – Thread certified products (link)
8. Web page – Connectivity standard alliance (csa) – NXP RW612 Tri-Radio Wireless MCU Development Platform (link)
9. Web page – Connectivity standard alliance (csa) – NXP RW610 Wireless MCU Development Platform (link)

Chapter 2

RTOS

2.1 FreeRTOS

2.1.1 FreeRTOS kernel

Open source RTOS kernel for small devices.

FreeRTOS kernel for MCUXpresso SDK Readme

FreeRTOS kernel for MCUXpresso SDK

Overview The purpose of this document is to describes the FreeRTOS kernel repo integration into the NXP MCUXpresso Software Development Kit: mcuxsdk. MCUXpresso SDK provides a comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on MCUs from NXP. This project involves the FreeRTOS kernel repo fork with:

- cmake and Kconfig support to allow the configuration and build in MCUXpresso SDK ecosystem
- FreeRTOS OS additions, such as FreeRTOS driver wrappers, RTOS ready FatFs file system, and the implementation of FreeRTOS tickless mode

The history of changes in FreeRTOS kernel repo for MCUXpresso SDK are summarized in *CHANGELOG_mcuxsdk.md* file.

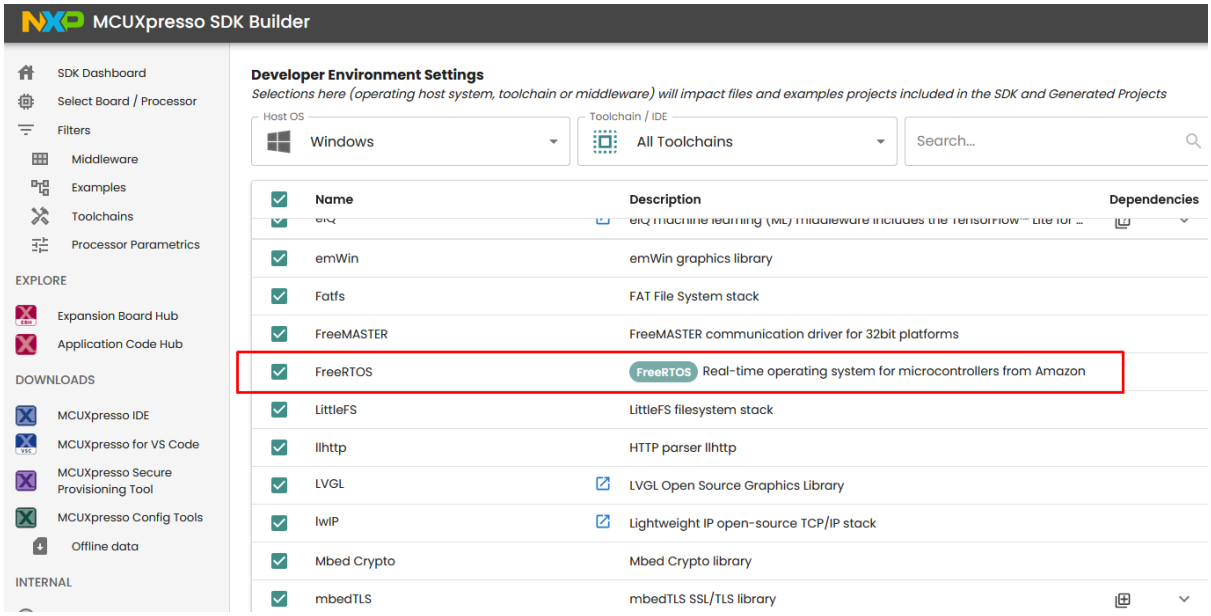
The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

FreeRTOS example applications The FreeRTOS examples are written to demonstrate basic FreeRTOS features and the interaction between peripheral drivers and the RTOS.

List of examples The list of freertos_examples, their description and availability for individual supported MCUXpresso SDK development boards can be obtained here: https://mcuxpresso.nxp.com/mcuxsdk/latest/html/examples/freertos_examples/index.html

Location of examples The FreeRTOS examples are located in mcuxsdk-examples repository, see the freertos_examples folder.

Once using MCUXpresso SDK zip packages created via the MCUXpresso SDK Builder the FreeRTOS kernel library and associated freertos_examples are added into final zip package once FreeRTOS components is selected on the Developer Environment Settings page:



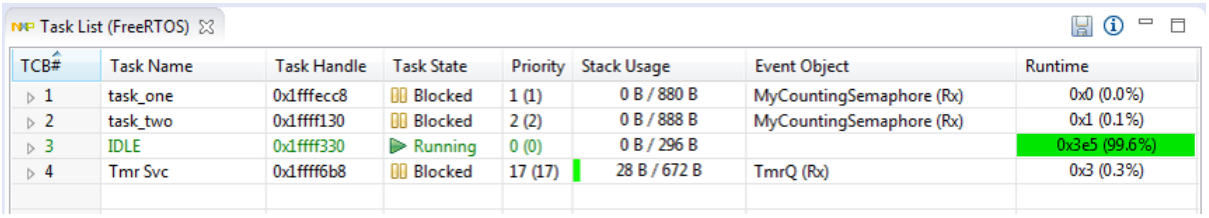
The FreeRTOS examples in MCUXpresso SDK zip packages are located in <MCUXpressoSDK_install_dir>/boards/<board_name>/freertos_examples/ subfolders.

Building a FreeRTOS example application For information how to use the cmake and Kconfig based build and configuration system and how to build freertos_examples visit: MCUXpresso SDK documentation for Build And Configuration MCUXpresso SDK Getting Start Guide

Tip: To list all FreeRTOS example projects and targets that can be built via the west build command, use this west list_project command in mcuxsdk workspace:

```
west list_project -p examples/freertos_examples
```

FreeRTOS aware debugger plugin NXP provides FreeRTOS task aware debugger for GDB. The plugin is compatible with Eclipse-based (MCUXpressoIDE) and is available after the installation.



FreeRTOS kernel for MCUXpresso SDK ChangeLog

Changelog FreeRTOS kernel for MCUXpresso SDK All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[Unreleased]

Added

- Kconfig added CONFIG_FREERTOS_USE_CUSTOM_CONFIG_FRAGMENT config to optionally include custom FreeRTOSConfig fragment
  include file FreeRTOSConfig_frag.h. File must be provided by application.
- Added missing Kconfig option for configUSE_PICOLIBC_TLS.
- Add correct header files to build when configUSE_NEWLIB_REENTRANT and configUSE_PICOLIBC_TLS is selected in config.

[11.1.0_rev0]

- update amazon freertos version

[11.0.1_rev0]

- update amazon freertos version

[10.5.1_rev0]

- update amazon freertos version

[10.4.3_rev1]

- Apply CM33 security fix from 10.4.3-LTS-Patch-2. See rtos\freertos\freertos_kernel\History.txt
- Apply CM33 security fix from 10.4.3-LTS-Patch-1. See rtos\freertos\freertos_kernel\History.txt

[10.4.3_rev0]

- update amazon freertos version.

[10.4.3_rev0]

- update amazon freertos version.

[9.0.0_rev3]

- New features:
  - Tickless idle mode support for Cortex-A7. Add fsl_tickless_epit.c and fsl_tickless_generic.h in portable/IAR/ARM_CA9 folder.
  - Enabled float context saving in IAR for Cortex-A7. Added configUSE_TASK_FPU_SUPPORT macros. Modified port.c and portmacro.h in portable/IAR/ARM_CA9 folder.
- Other changes:
  - Transformed ARM_CM core specific tickless low power support into generic form under freertos/Source/portable/low_power_tickless/.

[9.0.0_rev2]

- New features:
  - Enabled MCUXpresso thread aware debugging. Add freertos_tasks_c_additions.h and configINCLUDE_FREERTOS_TASK_C_ADDITIONS_H and configFREERTOS_MEMORY_SCHEME macros.

[9.0.0_rev1]

- New features:
  - Enabled -flto optimization in GCC by adding **attribute**((used)) for vTaskSwitchContext.
  - Enabled KDS Task Aware Debugger. Apply FreeRTOS patch to enable configRECORD_STACK_HIGH_ADDRESS macro. Modified files are task.c and FreeRTOS.h.

[9.0.0_rev0]

- New features:
  - Example freertos_sem_static.
  - Static allocation support RTOS driver wrappers.
- Other changes:
  - Tickless idle rework. Support for different timers is in separated files (fsl_tickless_systick.c, fsl_tickless_lptmr.c).
  - Removed configuration option configSYSTICK_USE_LOW_POWER_TIMER. Low power timer is now selected by linking of apropriate file fsl_tickless_lptmr.c.
  - Removed configOVERRIDE_DEFAULT_TICK_CONFIGURATION in RVDS port. Use of **attribute**((weak)) is the preferred solution. Not same as _weak!

[8.2.3]

- New features:
  - Tickless idle mode support.
  - Added template application for Kinetis Expert (KEx) tool (template_application).
- Other changes:
  - Folder structure reduction. Keep only Kinetis related parts.

FreeRTOS kernel Readme

MCUXpresso SDK: FreeRTOS kernel This repository is a fork of FreeRTOS kernel (https://github.com/FreeRTOS/FreeRTOS-Kernel)(11.1.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable FreeRTOS kernel repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

For more information about the FreeRTOS kernel repo adoption see README_mcuxsdk.md: FreeRTOS kernel for MCUXpresso SDK Readme document.

Getting started This repository contains FreeRTOS kernel source/header files and kernel ports only. This repository is referenced as a submodule in FreeRTOS/FreeRTOS repository, which contains pre-configured demo application projects under FreeRTOS/Demo directory.

The easiest way to use FreeRTOS is to start with one of the pre-configured demo application projects. That way you will have the correct FreeRTOS source files included, and the correct include paths configured. Once a demo application is building and executing you can remove the demo application files, and start to add in your own application source files. See the FreeRTOS Kernel Quick Start Guide for detailed instructions and other useful links.

Additionally, for FreeRTOS kernel feature information refer to the Developer Documentation, and API Reference.

Also for contributing and creating a Pull Request please refer to *the instructions here*.

Getting help If you have any questions or need assistance troubleshooting your FreeRTOS project, we have an active community that can help on the FreeRTOS Community Support Forum.

To consume FreeRTOS-Kernel

Consume with CMake If using CMake, it is recommended to use this repository using FetchContent. Add the following into your project's main or a subdirectory's CMakeLists.txt:

- Define the source and version/tag you want to use:

```
FetchContent_Declare( freertos_kernel
  GIT_REPOSITORY https://github.com/FreeRTOS/FreeRTOS-Kernel.git
  GIT_TAG        main #Note: Best practice to use specific git-hash or tagged version
)
```

In case you prefer to add it as a git submodule, do:

```
git submodule add https://github.com/FreeRTOS/FreeRTOS-Kernel.git <path of the submodule>
git submodule update --init
```

- Add a freertos_config library (typically an INTERFACE library) The following assumes the directory structure:
  - include/FreeRTOSConfig.h

```
add_library(freertos_config INTERFACE)

target_include_directories(freertos_config SYSTEM
INTERFACE
    include
)

target_compile_definitions(freertos_config
  INTERFACE
    projCOVERAGE_TEST=0
)
```

In case you installed FreeRTOS-Kernel as a submodule, you will have to add it as a subdirectory:

```
add_subdirectory(${FREERTOS_PATH})
```

- Configure the FreeRTOS-Kernel and make it available
  - this particular example supports a native and cross-compiled build option.

```
set( FREERTOS_HEAP "4" CACHE STRING "" FORCE)
# Select the native compile PORT
set( FREERTOS_PORT "GCC_POSIX" CACHE STRING "" FORCE)
# Select the cross-compile PORT
if (CMAKE_CROSSCOMPILING)
  set(FREERTOS_PORT "GCC_ARM_CA9" CACHE STRING "" FORCE)
endif()

FetchContent_MakeAvailable(freertos_kernel)
```

- In case of cross compilation, you should also add the following to freertos_config:

```
target_compile_definitions(freertos_config INTERFACE ${definitions})
target_compile_options(freertos_config INTERFACE ${options})
```

Consuming stand-alone - Cloning this repository To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Kernel.git
```

Using SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Kernel.git
```

Repository structure

- The root of this repository contains the three files that are common to every port - list.c, queue.c and tasks.c. The kernel is contained within these three files. croutine.c implements the optional co-routine functionality - which is normally only used on very memory limited systems.
- The ./portable directory contains the files that are specific to a particular microcontroller and/or compiler. See the readme file in the ./portable directory for more information.
- The ./include directory contains the real time kernel header files.
- The ./template_configuration directory contains a sample FreeRTOSConfig.h to help jumpstart a new project. See the *FreeRTOSConfig.h* file for instructions.

Code Formatting FreeRTOS files are formatted using the "uncrustify" tool. The configuration file used by uncrustify can be found in the FreeRTOS/CI-CD-GitHub-Actions's uncrustify.cfg file.

Line Endings File checked into the FreeRTOS-Kernel repository use unix-style LF line endings for the best compatibility with git.

For optimal compatibility with Microsoft Windows tools, it is best to enable the git autocrlf feature. You can enable this setting for the current repository using the following command:

```
git config core.autocrlf true
```

Git History Optimizations Some commits in this repository perform large refactors which touch many lines and lead to unwanted behavior when using the git blame command. You can configure git to ignore the list of large refactor commits in this repository with the following command:

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
```

Spelling and Formatting We recommend using Visual Studio Code, commonly referred to as VSCode, when working on the FreeRTOS-Kernel. The FreeRTOS-Kernel also uses cSpell as part of its spelling check. The config file for which can be found at *cspell.config.yaml* There is additionally a cSpell plugin for VSCode that can be used as well. .cSpellWords.txt contains words that are not traditionally found in an English dictionary. It is used by the spellchecker to verify the various jargon, variable names, and other odd words used in the FreeRTOS code base are correct. If your pull request fails to pass the spelling and you believe this is a mistake, then add the word to .cSpellWords.txt. When adding a word please then sort the list, which can be done by running the bash command: `sort -u .cSpellWords.txt -o .cSpellWords.txt` Note that only the FreeRTOS-Kernel Source Files, *include*, *portable/MemMang*, and *portable/Common* files are checked for proper spelling, and formatting at this time.

2.1.2 FreeRTOS drivers

This is set of NXP provided FreeRTOS reentrant bus drivers.

2.1.3 backoffalgorithm

Algorithm for calculating exponential backoff with jitter for network retry attempts.

Readme

MCUXpresso SDK: backoffAlgorithm Library This repository is a fork of backoffAlgorithm library (https://github.com/FreeRTOS/backoffalgorithm)(1.3.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable backoffAlgorithm repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

backoffAlgorithm Library This repository contains the backoffAlgorithm library, a utility library to calculate backoff period using an exponential backoff with jitter algorithm for retrying network operations (like failed network connection with server). This library uses the "Full Jitter" strategy for the exponential backoff with jitter algorithm. More information about the algorithm can be seen in the Exponential Backoff and Jitter AWS blog.

The backoffAlgorithm library is distributed under the *MIT Open Source License*.

Exponential backoff with jitter is typically used when retrying a failed network connection or operation request with the server. An exponential backoff with jitter helps to mitigate failed network operations with servers, that are caused due to network congestion or high request load on the server, by spreading out retry requests across multiple devices attempting network operations. Besides, in an environment with poor connectivity, a client can get disconnected at any time. A backoff strategy helps the client to conserve battery by not repeatedly attempting reconnections when they are unlikely to succeed.

See memory requirements for this library *here*.

backoffAlgorithm v1.3.0 source code is part of the FreeRTOS 202210.00 LTS release.

backoffAlgorithm v1.0.0 source code is part of the FreeRTOS 202012.00 LTS release.

Reference example The example below shows how to use the backoffAlgorithm library on a POSIX platform to retry a DNS resolution query for amazon.com.

```
#include "backoff_algorithm.h"
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>
#include <time.h>

/* The maximum number of retries for the example code. */
#define RETRY_MAX_ATTEMPTS            ( 5U )

/* The maximum back-off delay (in milliseconds) for between retries in the example. */
#define RETRY_MAX_BACKOFF_DELAY_MS    ( 5000U )

/* The base back-off delay (in milliseconds) for retry configuration in the example. */
#define RETRY_BACKOFF_BASE_MS         ( 500U )

int main()
{
    /* Variables used in this example. */
    BackoffAlgorithmStatus_t retryStatus = BackoffAlgorithmSuccess;
    BackoffAlgorithmContext_t retryParams;
    char serverAddress[] = "amazon.com";
    uint16_t nextRetryBackoff = 0;

    int32_t dnsStatus = -1;
    struct addrinfo hints;
    struct addrinfo ** pListHead = NULL;
    struct timespec tp;

    /* Add hints to retrieve only TCP sockets in getaddrinfo. */
    ( void ) memset( &hints, 0, sizeof( hints ) );

    /* Address family of either IPv4 or IPv6. */
    hints.ai_family = AF_UNSPEC;
    /* TCP Socket. */
    hints.ai_socktype = ( int32_t ) SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;

    /* Initialize reconnect attempts and interval. */
    BackoffAlgorithm_InitializeParams( &retryParams,
                                       RETRY_BACKOFF_BASE_MS,
                                       RETRY_MAX_BACKOFF_DELAY_MS,
                                       RETRY_MAX_ATTEMPTS );


    /* Seed the pseudo random number generator used in this example (with call to
     * rand() function provided by ISO C standard library) for use in backoff period
     * calculation when retrying failed DNS resolution. */

    /* Get current time to seed pseudo random number generator. */
    ( void ) clock_gettime( CLOCK_REALTIME, &tp );
    /* Seed pseudo random number generator with seconds. */
    srand( tp.tv_sec );

    do
    {
        /* Perform a DNS lookup on the given host name. */
        dnsStatus = getaddrinfo( serverAddress, NULL, &hints, pListHead );
```

(continues on next page)

(continued from previous page)

```
        /* Retry if DNS resolution query failed. */
        if( dnsStatus != 0 )
        {
            /* Generate a random number and get back-off value (in milliseconds) for the next retry.
             * Note: It is recommended to use a random number generator that is seeded with
             * device-specific entropy source so that backoff calculation across devices is different
             * and possibility of network collision between devices attempting retries can be avoided.
             *
             * For the simplicity of this code example, the pseudo random number generator, rand()
             * function is used. */
            retryStatus = BackoffAlgorithm_GetNextBackoff( &retryParams, rand(), &nextRetryBackoff );

            /* Wait for the calculated backoff period before the next retry attempt of querying DNS.
             * As usleep() takes nanoseconds as the parameter, we multiply the backoff period by 1000. */
            ( void ) usleep( nextRetryBackoff * 1000U );
        }
    } while( ( dnsStatus != 0 ) && ( retryStatus != BackoffAlgorithmRetriesExhausted ) );

    return dnsStatus;
}
```

Building the library A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Additionally, the library uses a header file introduced in ISO C99, stdint.h. For compilers that do not provide this header file, the *source/include* directory contains *stdint.readme*, which can be renamed to stdint.h to build the backoffAlgorithm library.

For instance, if the example above is copied to a file named example.c, *gcc* can be used like so:

```
gcc -I source/include example.c source/backoff_algorithm.c -o example
./example
```

gcc can also produce an output file to be linked:

```
gcc -I source/include -c source/backoff_algorithm.c
```

Building unit tests

Checkout Unity Submodule By default, the submodules in this repository are configured with update=none in *.gitmodules*, to avoid increasing clone time and disk space usage of other repositories (like amazon-freertos that submodules this repository).

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/Unity
```

Platform Prerequisites

- For running unit tests
  - C89 or later compiler like gcc
  - CMake 3.13.0 or later
- For running the coverage target, gcov is additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described *above*.)
2. Create build directory: `mkdir build && cd build`
3. Run *cmake* while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

Contributing See *CONTRIBUTING.md* for information on contributing.

2.1.4 corehttp

C language HTTP client library designed for embedded platforms.

MCUXpresso SDK: coreHTTP Client Library

This repository is a fork of coreHTTP Client library (https://github.com/FreeRTOS/corehttp)(3.0.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreHTTP Client repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

coreHTTP Client Library

This repository contains a C language HTTP client library designed for embedded platforms. It has no dependencies on any additional libraries other than the standard C library, llhttp, and a customer-implemented transport interface. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8. This library has also undergone both static code analysis from Coverity static analysis, and validation of memory safety and data structure invariance through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

coreHTTP v3.0.0 source code is part of the FreeRTOS 202210.00 LTS release.

coreHTTP v2.0.0 source code is part of the FreeRTOS 202012.00 LTS release.

coreHTTP Config File The HTTP client library exposes configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *core_http_config_defaults.h*. To provide custom values for the configuration macros, a custom config file named `core_http_config.h` can be provided by the user application to the library.

By default, a `core_http_config.h` custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide `HTTP_DO_NOT_USE_CUSTOM_CONFIG` as a compile time preprocessor macro.

The HTTP client library can be built by either:

- Defining a core_http_config.h file in the application, and adding it to the include directories for the library build. **OR**
- Defining the HTTP_DO_NOT_USE_CUSTOM_CONFIG preprocessor macro for the library build.

Building the Library The *httpFilePaths.cmake* file contains the information of all source files and header include paths required to build the HTTP client library.

As mentioned in the *previous section*, either a custom config file (i.e. core_http_config.h) OR HTTP_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the HTTP client library.

For a CMake example of building the HTTP library with the httpFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Platform Prerequisites

- For running unit tests, the following are required:
  - **C90 compiler** like gcc
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is required for this repository's CMock test framework.
- For running the coverage target, the following are required:
  - **gcov**
  - **lcov**

Steps to build Unit Tests

1. Go to the root directory of this repository.
2. Run the *cmake* command: cmake -S test -B build -DBUILD_CLONE_SUBMODULES=ON
3. Run this command to build the library and unit tests: make -C build all
4. The generated test executables will be present in build/bin/tests folder.
5. Run cd build && ctest to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The test/cbmc/proofs directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples The AWS IoT Device SDK for Embedded C repository contains demos of using the HTTP client library here on a POSIX platform. These can be used as reference examples for the library API.

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of coreHTTP may differ across repositories.

Generating Documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

2.1.5 corejson

JSON parser.

Readme

MCUXpresso SDK: coreJSON Library This repository is a fork of coreJSON library (https://github.com/FreeRTOS/corejson)(3.2.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreJSON repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

coreJSON Library This repository contains the coreJSON library, a parser that strictly enforces the ECMA-404 JSON standard and is suitable for low memory footprint embedded devices. The coreJSON library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

coreJSON v3.2.0 source code is part of the FreeRTOS 202210.00 LTS release.

coreJSON v3.0.0 source code is part of the FreeRTOS 202012.00 LTS release.

Reference example

```
#include <stdio.h>
#include "core_json.h"

int main()
{
    // Variables used in this example.
    JSONStatus_t result;
    char buffer[] = "{\"foo\":\"abc\",\"bar\":{\"foo\":\"xyz\"}}";
    size_t bufferLength = sizeof( buffer ) - 1;
    char queryKey[] = "bar.foo";
    size_t queryKeyLength = sizeof( queryKey ) - 1;
    char * value;
    size_t valueLength;

    // Calling JSON_Validate() is not necessary if the document is guaranteed to be valid.
    result = JSON_Validate( buffer, bufferLength );

    if( result == JSONSuccess )
    {
        result = JSON_Search( buffer, bufferLength, queryKey, queryKeyLength,
                              &value, &valueLength );
    }

    if( result == JSONSuccess )
    {
        // The pointer "value" will point to a location in the "buffer".
        char save = value[ valueLength ];
        // After saving the character, set it to a null byte for printing.
        value[ valueLength ] = '\0';
        // "Found: bar.foo -> xyz" will be printed.
        printf( "Found: %s -> %s\n", queryKey, value );
        // Restore the original character.
        value[ valueLength ] = save;
    }

    return 0;
}
```

A search may descend through nested objects when the queryKey contains matching key strings joined by a separator, .. In the example above, bar has the value {"foo":"xyz"}. Therefore, a search for query key bar.foo would output xyz.

Building coreJSON A compiler that supports **C90 or later** such as *gcc* is required to build the library.

Additionally, the library uses 2 header files introduced in ISO C99, stdbool.h and stdint.h. For compilers that do not provide this header file, the *source/include* directory contains *stdbool.readme* and *stdint.readme*, which can be renamed to stdbool.h and stdint.h respectively.

For instance, if the example above is copied to a file named example.c, *gcc* can be used like so:

```
gcc -I source/include example.c source/core_json.c -o example
./example
```

gcc can also produce an output file to be linked:

```
gcc -I source/include -c source/core_json.c
```

Documentation

Existing documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C FreeRTOS.org

Note that the latest included version of the coreJSON library may differ across repositories.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Building unit tests

Checkout Unity Submodule By default, the submodules in this repository are configured with `update=none` in *.gitmodules*, to avoid increasing clone time and disk space usage of other repositories (like amazon-freertos that submodules this repository).

To build unit tests, the submodule dependency of Unity is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/Unity
```

Platform Prerequisites

- For running unit tests
  - C90 compiler like gcc
  - CMake 3.13.0 or later
  - Ruby 2.0.0 or later is additionally required for the Unity test framework (that we use).
- For running the coverage target, gcov is additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **Unity** submodule is cloned as described *above*.)
2. Create build directory: `mkdir build && cd build`
3. Run *cmake* while inside build directory: `cmake -S ../test`
4. Run this command to build the library and unit tests: `make all`
5. The generated test executables will be present in `build/bin/tests` folder.
6. Run `ctest` to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The test/cbmc/proofs directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Contributing See *CONTRIBUTING.md* for information on contributing.

2.1.6 coremqtt

MQTT publish/subscribe messaging library.

MCUXpresso SDK: coreMQTT Library

This repository is a fork of coreMQTT library (https://github.com/FreeRTOS/coremqtt)(2.1.1). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreMQTT repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

coreMQTT Client Library

This repository contains the coreMQTT library that has been optimized for a low memory footprint. The coreMQTT library is compliant with the MQTT 3.1.1 standard. It has no dependencies on any additional libraries other than the standard C library, a customer-implemented network transport interface, and *optionally* a user-implemented platform time function. This library is distributed under the *MIT Open Source License*.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

coreMQTT v2.1.1 source code is part of the FreeRTOS 202210.01 LTS release.

MQTT Config File The MQTT client library exposes build configuration macros that are required for building the library. A list of all the configurations and their default values are defined in *core_mqtt_config_defaults.h*. To provide custom values for the configuration macros, a custom config file named core_mqtt_config.h can be provided by the application to the library.

By default, a core_mqtt_config.h custom config is required to build the library. To disable this requirement and build the library with default configuration values, provide MQTT_DO_NOT_USE_CUSTOM_CONFIG as a compile time preprocessor macro.

Thus, the MQTT library can be built by either:

- Defining a core_mqtt_config.h file in the application, and adding it to the include directories list of the library
  OR
- Defining the MQTT_DO_NOT_USE_CUSTOM_CONFIG preprocessor macro for the library build.

Sending metrics to AWS IoT When establishing a connection with AWS IoT, users can optionally report the Operating System, Hardware Platform and MQTT client version information of their device to AWS. This information can help AWS IoT provide faster issue resolution and technical support. If users want to report this information, they can send a specially formatted string (see below) in the username field of the MQTT CONNECT packet.

Format

The format of the username string with metrics is:

```
<Actual_Username>?SDK=<OS_Name>&Version=<OS_Version>&Platform=<Hardware_Platform>&
↪MQTTLib=<MQTT_Library_name>@<MQTT_Library_version>
```

Where

- <Actual_Username> is the actual username used for authentication, if username and password are used for authentication. When username and password based authentication is not used, this is an empty value.
- <OS_Name> is the Operating System the application is running on (e.g. FreeRTOS)
- <OS_Version> is the version number of the Operating System (e.g. V10.4.3)
- <Hardware_Platform> is the Hardware Platform the application is running on (e.g. WinSim)
- <MQTT_Library_name> is the MQTT Client library being used (e.g. coreMQTT)
- <MQTT_Library_version> is the version of the MQTT Client library being used (e.g. 1.0.2)

Example

- Actual_Username = “iotuser”, OS_Name = FreeRTOS, OS_Version = V10.4.3, Hardware_Platform_Name = WinSim, MQTT_Library_Name = coremqtt, MQTT_Library_version = 2.1.1. If username is not used, then “iotuser” can be removed.

```
/* Username string:
 * iotuser?SDK=FreeRTOS&Version=v10.4.3&Platform=WinSim&MQTTLib=coremqtt@2.1.1
 */

#define OS_NAME                    ”FreeRTOS”
#define OS_VERSION                 ”V10.4.3”
#define HARDWARE_PLATFORM_NAME     ”WinSim”
#define MQTT_LIB                   ”coremqtt@2.1.1”

#define USERNAME_STRING            ”iotuser?SDK=” OS_NAME ”&Version=” OS_VERSION ”&
↪Platform=” HARDWARE_PLATFORM_NAME ”&MQTTLib=” MQTT_LIB
#define USERNAME_STRING_LENGTH     ( ( uint16_t ) ( sizeof( USERNAME_STRING ) - 1 ) )

MQTTConnectInfo_t connectInfo;
connectInfo.pUserName = USERNAME_STRING;
connectInfo.userNameLength = USERNAME_STRING_LENGTH;
mqttStatus = MQTT_Connect( pMqttContext, &connectInfo, NULL, CONNACK_RECV_TIMEOUT_MS,␣
↪pSessionPresent );
```

Upgrading to v2.0.0 and above With coreMQTT versions >=v2.0.0, there are breaking changes. Please refer to the *coreMQTT version >=v2.0.0 Migration Guide*.

Building the Library The *mqttFilePaths.cmake* file contains the information of all source files and the header include path required to build the MQTT library.

Additionally, the MQTT library requires two header files that are not part of the ISO C90 standard library, stdbool.h and stdint.h. For compilers that do not provide these header files, the

source/include directory contains the files *stdbool.readme* and *stdint.readme*, which can be renamed to stdbool.h and stdint.h, respectively, to provide the type definitions required by MQTT.

As mentioned in the previous section, either a custom config file (i.e. core_mqtt_config.h) OR MQTT_DO_NOT_USE_CUSTOM_CONFIG macro needs to be provided to build the MQTT library.

For a CMake example of building the MQTT library with the mqttFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Checkout CMock Submodule By default, the submodules in this repository are configured with update=none in *.gitmodules* to avoid increasing clone time and disk space usage of other repositories (like amazon-freertos that submodules this repository).

To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

Platform Prerequisites

- Docker

or the following:

- For running unit tests
  - **C90 compiler** like gcc
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build Unit Tests

1. If using docker, launch the container:
   1. docker build -t coremqtt .
   2. docker run -it -v "$PWD":/workspaces/coreMQTT -w /workspaces/coreMQTT coremqtt
2. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described *above*)
3. Run the *cmake* command: cmake -S test -B build
4. Run this command to build the library and unit tests: make -C build all
5. The generated test executables will be present in build/bin/tests folder.
6. Run cd build && ctest to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The test/cbmc/proofs directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples Please refer to the demos of the MQTT client library in the following locations for reference examples on POSIX and FreeRTOS platforms:

Platform	Location	Transport Interface Implementation
POSIX	AWS IoT Device SDK for Embedded C	POSIX sockets for TCP/IP and OpenSSL for TLS stack
FreeRTOS	FreeRTOS/FreeRTOS	FreeRTOS+TCP for TCP/IP and mbedTLS for TLS stack
FreeRTOS	FreeRTOS AWS Reference Integrations	Based on Secure Sockets Abstraction

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C FreeRTOS.org

Note that the latest included version of coreMQTT may differ across repositories.

Generating Documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Contributing See *CONTRIBUTING.md* for information on contributing.

2.1.7 coremqtt-agent

The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library.

Readme

MCUXpresso SDK: coreMQTT Agent Library This repository is a fork of coreMQTT Agent library (https://github.com/FreeRTOS/coremqtt-agent)(1.2.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable coreMQTT Agent repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

coreMQTT Agent Library The coreMQTT Agent library is a high level API that adds thread safety to the coreMQTT library. The library provides thread safe equivalents to the coreMQTT's APIs, greatly simplifying its use in multi-threaded environments. The coreMQTT Agent library manages the MQTT connection by serializing the access to the coreMQTT library and reducing implementation overhead (e.g., removing the need for the application to repeatedly call to MQTT_ProcessLoop). This allows your multi-threaded applications to share the same MQTT connection, and enables you to design an embedded application without having to worry about coreMQTT thread safety.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity static analysis, and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

Cloning this repository This repo uses Git Submodules to bring in dependent components.

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/coreMQTT-Agent.git --recurse-submodules
```

Using SSH:

```
git clone git@github.com:FreeRTOS/coreMQTT-Agent.git --recurse-submodules
```

If you have downloaded the repo without using the --recurse-submodules argument, you need to run:

```
git submodule update --init --recursive
```

coreMQTT Agent Library Configurations The MQTT Agent library uses the same core_mqtt_config.h configuration file as coreMQTT, with the addition of configuration constants listed at the top of *core_mqtt_agent.h* and *core_mqtt_agent_command_functions.h*. Documentation for these configurations can be found here.

To provide values for these configuration values, they must be either:

- Defined in core_mqtt_config.h used by coreMQTT **OR**
- Passed as compile time preprocessor macros

Porting the coreMQTT Agent Library In order to use the MQTT Agent library on a platform, you need to supply thread safe functions for the agent's *messaging interface*.

Messaging Interface Each of the following functions must be thread safe.

Function Pointer	Description
MQTTAgentMessageSend_t	A function that sends commands (as MQTTAgentCommand_t * pointers) to be received by MQTTAgent_CommandLoop. This can be implemented by pushing to a thread safe queue.
MQTTAgentMessageRecv_t	A function used by MQTTAgent_CommandLoop to receive MQTTAgentCommand_t * pointers that were sent by API functions. This can be implemented by receiving from a thread safe queue.
MQTTAgentCommandGet_t	A function that returns a pointer to an allocated MQTTAgentCommand_t structure, which is used to hold information and arguments for a command to be executed in MQTTAgent_CommandLoop(). If using dynamic memory, this can be implemented using malloc().
MQTTAgentCommandRelease_t	A function called to indicate that a command structure that had been allocated with the MQTTAgentCommandGet_t function pointer will no longer be used by the agent, so it may be freed or marked as not in use. If using dynamic memory, this can be implemented with free().

Reference implementations for the interface functions can be found in the *reference examples* below.

Additional Considerations

Static Memory If only static allocation is used, then the MQTTAgentCommandGet_t and MQTTAgentCommandRelease_t could instead be implemented with a pool of MQTTAgentCommand_t structures, with a queue or semaphore used to control access and provide thread safety. The below *reference examples* use static memory with a command pool.

Subscription Management The MQTT Agent does not track subscriptions for MQTT topics. The receipt of any incoming PUBLISH packet will result in the invocation of a single MQTTAgentIncomingPublishCallback_t callback, which is passed to MQTTAgent_Init() for initialization. If it is desired for different handlers to be invoked for different incoming topics, then the publish callback will have to manage subscriptions and fan out messages. A platform independent subscription manager example is implemented in the *reference examples* below.

Building the Library You can build the MQTT Agent source files that are in the *source* directory, and add *source/include* to your compiler's include path. Additionally, the MQTT Agent library requires the coreMQTT library, whose files follow the same source/ and source/include pattern as the agent library; its build instructions can be found here.

If using CMake, the *mqttAgentFilePaths.cmake* file contains the above information of the source files and the header include path from this repository. The same information is found for coreMQTT from mqttFilePaths.cmake in the *coreMQTT submodule*.

For a CMake example of building the MQTT Agent library with the mqttAgentFilePaths.cmake file, refer to the coverity_analysis library target in *test/CMakeLists.txt* file.

Building Unit Tests

Checkout CMock Submodule To build unit tests, the submodule dependency of CMock is required. Use the following command to clone the submodule:

```
git submodule update --checkout --init --recursive test/unit-test/CMock
```

Unit Test Platform Prerequisites

- For running unit tests
  - **C90 compiler** like gcc
  - **CMake 3.13.0 or later**
  - **Ruby 2.0.0 or later** is additionally required for the CMock test framework (that we use).
- For running the coverage target, **gcov** and **lcov** are additionally required.

Steps to build Unit Tests

1. Go to the root directory of this repository. (Make sure that the **CMock** submodule is cloned as described *above*)
2. Run the *cmake* command: `cmake -S test -B build`
3. Run this command to build the library and unit tests: `make -C build all`
4. The generated test executables will be present in `build/bin/tests` folder.
5. Run `cd build && ctest` to execute all tests and view the test run summary.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples Please refer to the demos of the MQTT Agent library in the following locations for reference examples on FreeRTOS platforms:

Location
coreMQTT Agent Demos FreeRTOS/FreeRTOS

Documentation The MQTT Agent API documentation can be found here.

Generating documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages yourself, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Getting help You can use your Github login to get support from both the FreeRTOS community and directly from the primary FreeRTOS developers on our active support forum. You can find a list of frequently asked questions here.

Contributing See *CONTRIBUTING.md* for information on contributing.

License This library is licensed under the MIT License. See the *LICENSE* file.

2.1.8 corepkcs11

PKCS #11 key management library.

Readme

MCUXpresso SDK: corePKCS11 Library This repository is a fork of PKCS #11 key management library (https://github.com/FreeRTOS/corePKCS11/tree/v3.5.0)(v3.5.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable corepkcs11 repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

corePKCS11 Library PKCS #11 is a standardized and widely used API for manipulating common cryptographic objects. It is important because the functions it specifies allow application software to use, create, modify, and delete cryptographic objects, without ever exposing those objects to the application's memory. For example, FreeRTOS AWS reference integrations use a small subset of the PKCS #11 API to, among other things, access the secret (private) key necessary to create a network connection that is authenticated and secured by the Transport Layer Security (TLS) protocol – without the application ever 'seeing' the key.

The Cryptoki or PKCS #11 standard defines a platform-independent API to manage and use cryptographic tokens. The name, "PKCS #11", is used interchangeably to refer to the API itself and the standard which defines it.

This repository contains a software based mock implementation of the PKCS #11 interface (API) that uses the cryptographic functionality provided by Mbed TLS. Using a software mock enables rapid development and flexibility, but it is expected that the mock be replaced by an implementation specific to your chosen secure key storage in production devices.

Only a subset of the PKCS #11 standard is implemented, with a focus on operations involving asymmetric keys, random number generation, and hashing.

The targeted use cases include certificate and key management for TLS authentication and code-sign signature verification, on small embedded devices.

corePKCS11 is implemented on PKCS #11 v2.4.0, the full PKCS #11 standard can be found on the oasis website.

This library has gone through code quality checks including verification that no function has a GNU Complexity score over 8, and checks against deviations from mandatory rules in the MISRA coding standard. Deviations from the MISRA C:2012 guidelines are documented under *MISRA Deviations*. This library has also undergone both static code analysis from Coverity static analysis and validation of memory safety through the CBMC automated reasoning tool.

See memory requirements for this library *here*.

corePKCS11 v3.5.0 source code is part of the FreeRTOS 202210.00 LTS release.

corePKCS11 v3.0.0 source code is part of the FreeRTOS 202012.00 LTS release.

Purpose Generally vendors for secure cryptoprocessors such as Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element, or any other type of secure hardware enclave, distribute a PKCS #11 implementation with the hardware. The purpose of the corePKCS11 software only mock library is therefore to provide a non hardware specific PKCS #11 implementation that allows for rapid prototyping and development before switching to a cryptoprocessor specific PKCS #11 implementation in production devices.

Since the PKCS #11 interface is defined as part of the PKCS #11 specification replacing this library with another implementation should require little porting effort, as the interface will not change. The system tests distributed in this repository can be leveraged to verify the behavior of a different implementation is similar to corePKCS11.

corePKCS11 Configuration The corePKCS11 library exposes preprocessor macros which must be defined prior to building the library. A list of all the configurations and their default values are defined in the doxygen documentation for this library.

Build Prerequisites

Library Usage For building the library the following are required:

- **A C99 compiler**
- **mbedcrypto** library from mbedtls version 2.x or 3.x.
- **pkcs11 API header(s)** available from OASIS or OpenSC

Optionally, variables from the pkcsFilePaths.cmake file may be referenced if your project uses cmake.

Integration and Unit Tests In order to run the integration and unit test suites the following are dependencies are necessary:

- **C Compiler**
- **CMake 3.13.0 or later**
- **Ruby 2.0.0 or later** required by CMock.
- **Python 3** required for configuring mbedtls.
- **git** required for fetching dependencies.
- **GNU Make** or **Ninja**

The *mbedtls*, *CMock*, and *Unity* libraries are downloaded and built automatically using the cmake FetchContent feature.

Coverage Measurement and Instrumentation The following software is required to run the coverage target:

- Linux, MacOS, or another POSIX-like environment.
- A recent version of **GCC** or **Clang** with support for gcov-like coverage instrumentation.
- **gcov** binary corresponding to your chosen compiler
- **lcov** from the Linux Test Project
- **perl** needed to run the lcov utility.

Coverage builds are validated on recent versions of Ubuntu Linux.

Running the Integration and Unit Tests

1. Navigate to the root directory of this repository in your shell.
2. Run **cmake** to construct a build tree: `cmake -S test -B build`
   - You may specify your preferred build tool by appending `-G'Unix Makefiles'` or `-GNinja` to the command above.
   - You may append `-DUNIT_TESTS=0` or `-DSYSTEM_TESTS=0` to disable Unit Tests or Integration Tests respectively.
3. Build the test binaries: `cmake --build ./build --target all`
4. Run `ctest --test-dir ./build` or `cmake --build ./build --target test` to run the tests without capturing coverage.
5. Run `cmake --build ./build --target coverage` to run the tests and capture coverage data.

CBMC To learn more about CBMC and proofs specifically, review the training material here.

The `test/cbmc/proofs` directory contains CBMC proofs.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.

Reference examples The FreeRTOS-Labs repository contains demos using the PKCS #11 library here using FreeRTOS on the Windows simulator platform. These can be used as reference examples for the library API.

Porting Guide Documentation for porting corePKCS11 to a new platform can be found on the AWS docs web page.

corePKCS11 is not meant to be ported to projects that have a TPM, HSM, or other hardware for offloading crypto-processing. This library is specifically meant to be used for development and prototyping.

Related Example Implementations These projects implement the PKCS #11 interface on real hardware and have similar behavior to corePKCS11. It is preferred to use these, over corePKCS11, as they allow for offloading Cryptography to separate hardware.

- ARM's Platform Security Architecture.
- Microchip's cryptoauthlib.
- Infineon's Optiga Trust X.

Documentation

Existing Documentation For pre-generated documentation, please see the documentation linked in the locations below:

Location
AWS IoT Device SDK for Embedded C
FreeRTOS.org

Note that the latest included version of corePKCS11 may differ across repositories.

Generating Documentation The Doxygen references were created using Doxygen version 1.9.2. To generate the Doxygen pages, please run the following command from the root of this repository:

```
doxygen docs/doxygen/config.doxyfile
```

Security See *CONTRIBUTING* for more information.

License This library is licensed under the MIT-0 License. See the LICENSE file.

2.1.9 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme

MCUXpresso SDK: FreeRTOS-Plus-TCP Library This repository is a fork of FreeRTOS-Plus-TCP library (https://github.com/FreeRTOS/freertos-plus-tcp)(4.0.0). Modifications have been made to adapt to NXP MCUXpresso SDK. CMakeLists.txt and Kconfig added to enable FreeRTOS-Plus-TCP repo sources build in MCUXpresso SDK. It is part of the MCUXpresso SDK overall delivery which is composed of several sub-repositories/projects. Navigate to the top/parent repository mcuxsdk-manifests(https://github.com/nxp-mcuxpresso/mcuxsdk-manifests) for the complete delivery of MCUXpresso SDK.

Introduction This branch contains unified IPv4 and IPv6 functionalities. Refer to the Getting started Guide (found here) for more details.

FreeRTOS-Plus-TCP Library FreeRTOS-Plus-TCP is a lightweight TCP/IP stack for FreeRTOS. It provides a familiar Berkeley sockets interface, making it as simple to use and learn as possible. FreeRTOS-Plus-TCP's features and RAM footprint are fully scalable, making FreeRTOS-Plus-TCP equally applicable to smaller lower throughput microcontrollers as well as larger higher throughput microprocessors.

This library has undergone static code analysis and checks for compliance with the MISRA coding standard. Any deviations from the MISRA C:2012 guidelines are documented under MISRA Deviations. The library is validated for memory safety and data structure invariance through the CBMC automated reasoning tool for the functions that parse data originating from the network. The library is also protocol tested using Maxwell protocol tester for both IPv4 and IPv6.

Getting started The easiest way to use the 4.0.0 version of FreeRTOS-Plus-TCP is to refer the Getting started Guide (found here) Another way is to start with the pre-configured demo application project (found in this directory). That way you will have the correct FreeRTOS source files included, and the correct include paths configured. Once a demo application is building and executing you can remove the demo application files, and start to add in your own application source files. See the FreeRTOS Kernel Quick Start Guide for detailed instructions and other useful links.

Additionally, for FreeRTOS-Plus-TCP source code organization refer to the Documentation, and API Reference.

Getting help If you have any questions or need assistance troubleshooting your FreeRTOS project, we have an active community that can help on the FreeRTOS Community Support Forum. Please also refer to FAQ for frequently asked questions.

Also see the Submitting a bugs/feature request section of CONTRIBUTING.md for more details.

Note: All the remaining sections are generic and applies to all the versions from V3.0.0 onwards.

Upgrading to V3.0.0 and V3.1.0 In version 3.0.0 or 3.1.0, the folder structure of FreeRTOS-Plus-TCP has changed and the files have been broken down into smaller logically separated modules. This change makes the code more modular and conducive to unit-tests. FreeRTOS-Plus-TCP V3.0.0 improves the robustness, security, and modularity of the library. Version 3.0.0 adds comprehensive unit test coverage for all lines and branches of code and has undergone protocol testing, and penetration testing by AWS Security to reduce the exposure to security vulnerabilities. Additionally, the source files have been moved to a source directory. This change requires modification of any existing project(s) to include the modified source files and directories. There are examples on how to use the new files and directory structure. For an example based on the Xilinx Zynq-7000, use the code in this branch and follow these instructions to build and run the demo.

FreeRTOS-Plus-TCP V3.1.0 source code(.c .h) is part of the FreeRTOS 202210.00 LTS release.

Generating pre V3.0.0 folder structure for backward compatibility: If you wish to continue using a version earlier than V3.0.0 i.e. continue to use your existing source code organization, a script is provided to generate the folder structure similar to this.

Note: After running the script, while the .c files will have same names as the pre V3.0.0 source, the files in the include directory will have different names and the number of files will differ as well. This should, however, not pose any problems to most projects as projects generally include all files in a given directory.

Running the script to generate pre V3.0.0 folder structure: For running the script, you will need Python version > 3.7. You can download/install it from here.

Once python is downloaded and installed, you can verify the version from your terminal/command window by typing python --version.

To run the script, you should switch to the FreeRTOS-Plus-TCP directory that was created using the *Cloning this repository* step above. And then run python <Path/to/the/script>/GenerateOriginalFiles.py.

To consume FreeRTOS+TCP

Consume with CMake If using CMake, it is recommended to use this repository using FetchContent. Add the following into your project's main or a subdirectory's CMakeLists.txt:

- Define the source and version/tag you want to use:

```
FetchContent_Declare( freertos_plus_tcp
  GIT_REPOSITORY https://github.com/FreeRTOS/FreeRTOS-Plus-TCP.git
  GIT_TAG        master #Note: Best practice to use specific git-hash or tagged version
  GIT_SUBMODULES "" # Don't grab any submodules since not latest
)
```

- Configure the FreeRTOS-Kernel and make it available
  - this particular example supports a native and cross-compiled build option.

```
set( FREERTOS_PLUS_FAT_DEV_SUPPORT OFF CACHE BOOL "" FORCE)
# Select the native compile PORT
set( FREERTOS_PLUS_FAT_PORT "POSIX" CACHE STRING "" FORCE)
# Select the cross-compile PORT
if (CMAKE_CROSSCOMPILING)
  # Eg. Zynq 2019_3 version of port
  set(FREERTOS_PLUS_FAT_PORT "ZYNQ_2019_3" CACHE STRING "" FORCE)
endif()

FetchContent_MakeAvailable(freertos_plus_tcp)
```

Consuming stand-alone This repository uses Git Submodules to bring in dependent components.

Note: If you download the ZIP file provided by GitHub UI, you will not get the contents of the submodules. (The ZIP file is also not a valid Git repository)

To clone using HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Plus-TCP.git ./FreeRTOS-Plus-TCP
cd ./FreeRTOS-Plus-TCP
git submodule update --checkout --init --recursive tools/CMock test/FreeRTOS-Kernel
```

Using SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Plus-TCP.git ./FreeRTOS-Plus-TCP
cd ./FreeRTOS-Plus-TCP
git submodule update --checkout --init --recursive tools/CMock test/FreeRTOS-Kernel
```

Porting The porting guide is available on this page.

Repository structure This repository contains the FreeRTOS-Plus-TCP repository and a number of supplementary libraries for testing/PR Checks. Below is the breakdown of what each directory contains:

- tools
  - This directory contains the tools and related files (CMock/uncrustify) required to run tests/checks on the TCP source code.
- tests
  - This directory contains all the tests (unit tests and CBMC) and the dependencies (FreeRTOS-Kernel/Litani-port) the tests require.
- source/portable
  - This directory contains the portable files required to compile the FreeRTOS-Plus-TCP source code for different hardware/compilers.
- source/include
  - The include directory has all the 'core' header files of FreeRTOS-Plus-TCP source.
- source
  - This directory contains all the [.c] source files.

Note At this time it is recommended to use BufferAllocation_2.c in which case it is essential to use the heap_4.c memory allocation scheme. See memory management.

Kernel sources The FreeRTOS Kernel Source is in FreeRTOS/FreeRTOS-Kernel repository, and it is consumed by testing/PR checks as a submodule in this repository.

The version of the FreeRTOS Kernel Source in use could be accessed at ./test/FreeRTOS-Kernel directory.

CBMC The test/cbmc/proofs directory contains CBMC proofs.

To learn more about CBMC and proofs specifically, review the training material here.

In order to run these proofs you will need to install CBMC and other tools by following the instructions here.