



MCUXpresso SDK Documentation

Release 25.09.00-pvw1



NXP
Jul 17, 2025



Table of contents

1	Middleware	3
1.1	Boot	3
1.1.1	MCUXpresso SDK : mcuxsdk-middleware-mcuboot_opensource	3
1.1.2	MCUboot	4
1.2	Cloud	5
1.2.1	AWS IoT	5
1.3	Connectivity	14
1.3.1	lwIP	14
1.4	File System	15
1.4.1	FatFs	15
1.5	Motor Control	17
1.5.1	FreeMASTER	17
1.6	MultiCore	54
1.6.1	Multicore SDK	54
1.7	Wireless	149
1.7.1	NXP Wireless Framework and Stacks	149
2	RTOS	201
2.1	FreeRTOS	201
2.1.1	FreeRTOS kernel	201
2.1.2	FreeRTOS drivers	207
2.1.3	backoffalgorithm	207
2.1.4	corehttp	210
2.1.5	corejson	212
2.1.6	coremqtt	215
2.1.7	coremqtt-agent	218
2.1.8	corepkcs11	222
2.1.9	freertos-plus-tcp	225

This documentation contains information specific to the mc56f80000evk 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 use 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 tick- sToWait);	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 applications 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_QUEUE_SIZE

```
#define FMSTR_COMM_QUEUE_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_QUEUE_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_QUEUE_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-language 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_UINTn	Unsigned integer type of size <i>n</i> bits (n=8,16,32,64)
FMSTR_TSA_SINTn	Signed integer type of size <i>n</i> bits (n=8,16,32,64)
FMSTR_TSA_FRACn	Fractional number of size <i>n</i> bits (n=16,32,64).
FMSTR_TSA_FRAC_Q(<i>m,n</i>)	Signed fractional number in general Q form (m+n+1 total bits)
FMSTR_TSA_FRAC_UQ(<i>m,n</i>)	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)
FM-STR_TSA_USERTYPE(<i>name</i>)	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
<i>FM-STR_ADDR</i> 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.	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.
<i>FM-STR_SIZE</i> It is required that this type is unsigned and at least 16 bits wide integer.	Data type used to hold the memory block size.
<i>FM-STR_BOOL</i> This type is used only in zero/non-zero conditions in the driver code.	Data type used as a general boolean type.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command code.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to create the Application Command data buffer.
<i>FM-STR_APPCM</i> Generally, this is an unsigned 8-bit value.	Data type used to hold the Application Command result code.

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.

<i>FM-STR_TSA_TII</i>	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 <i>FM-STR_SIZE</i> .
<i>FM-STR_TSA_TS</i>	Data type used to hold a memory block size, as used in the TSA descriptors. By default, this is defined as <i>FM-STR_SIZE</i> .

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

<i>FM-STR_HPIPE</i>	Pipe handle that identifies the open-pipe object. Generally, this is a pointer to a void type.
<i>FM-STR_PIPE_PC</i>	Integer type required to hold at least 7 bits of data. Generally, this is an unsigned 8-bit or 16-bit type.
<i>FM-STR_PIPE_SI</i>	Integer type required to hold at least 16 bits of data. This is used to store the data buffer sizes.
<i>FM-STR_PPIPEF</i>	Pointer to the pipe handler function. See FM-STR_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.

<i>FMSTR_U8</i>	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.	
<i>FMSTR_U16</i>	Unsigned 16-bit integer.
<i>FMSTR_U32</i>	Unsigned 32-bit integer.
<i>FMSTR_S8</i>	Signed 8-bit integer.
<i>FMSTR_S16</i>	Signed 16-bit integer.
<i>FMSTR_S32</i>	Signed 32-bit integer.
<i>FMSTR_FLOAT</i>	4-byte standard IEEE floating-point type.
<i>FMSTR_FLAGS</i>	Data type forming a union with a structure of flag bit-fields.
<i>FMSTR_SIZE8</i>	Data type holding a general size value, at least 8 bits wide.
<i>FMSTR_INDEX</i>	General for-loop index. Must be signed, at least 16 bits wide.
<i>FMSTR_BCHR</i>	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.	
<i>FMSTR_BPTR</i>	A pointer to the communication buffer (an array of <i>FMSTR_BCHR</i>).

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 MultiCore

1.6.1 Multicore SDK

Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

Multicore SDK (MCSDK) Release Notes

Overview These are the release notes for the NXP Multicore Software Development Kit (MCSDK) version 25.06.00.

This software package contains components for efficient work with multicore devices as well as for the multiprocessor communication.

What is new

- eRPC [CHANGELOG](#)
- RPMsg-Lite [CHANGELOG](#)
- MCMgr [CHANGELOG](#)
- Supported evaluation boards (multicore examples):
 - LPCXpresso55S69
 - FRDM-K32L3A6
 - MIMXRT1170-EVKB
 - MIMXRT1160-EVK
 - MIMXRT1180-EVK
 - MCX-N5XX-EVK
 - MCX-N9XX-EVK
 - FRDM-MCXN947
 - MIMXRT700-EVK
 - KW47-EVK
 - KW47-LOC
 - FRDM-MCXW72
 - MCX-W72-EVK
- Supported evaluation boards (multiprocessor examples):
 - LPCXpresso55S36
 - FRDM-K22F
 - FRDM-K32L2B
 - MIMXRT685-EVK
 - MIMXRT1170-EVKB
 - MIMXRT1180
 - FRDM-MCXN236
 - FRDM-MCXC242
 - FRDM-MCXC444
 - MCX-N9XX-EVK
 - FRDM-MCXN947
 - MIMXRT700-EVK

Development tools The Multicore SDK (MCSDK) was compiled and tested with development tools referred in: [Development tools](#)

Release contents This table describes the release contents. Not all MCUXpresso SDK packages contain the whole set of these components.

Deliverable	Location
Multicore SDK location	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call component	<MCSDK_dir>/erpc/
Multicore Manager component	<MCSDK_dir>/mcmgr/
RPMMsg-Lite	<MCSDK_dir>/rpmsg_lite/
Multicore demo applications	<MCUXpressoSDK_install_dir>/examples/multicore_examples/
Multiprocessor demo applications	<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/

Multicore SDK release overview Together, the Multicore SDK (MCSDK) and the MCUXpresso SDK (SDK) form a framework for the development of software for NXP multicore devices. The MCSDK release consists of the following elementary software components for multicore:

- Embedded Remote Procedure Call (eRPC)
- Multicore Manager (MCMGR) - included just in SDK for multicore devices
- Remote Processor Messaging - Lite (RPMMsg-Lite) - included just in SDK for multicore devices

The MCSDK is also accompanied with documentation and several multicore and multiprocessor demo applications.

Demo applications The multicore demo applications demonstrate the usage of the MCSDK software components on supported multicore development boards.

The following multicore demo applications are located together with other MCUXpresso SDK examples in

the <MCUXpressoSDK_install_dir>/examples/multicore_examples subdirectories.

- erpc_matrix_multiply_mu
- erpc_matrix_multiply_mu_rtos
- erpc_matrix_multiply_rpmsg
- erpc_matrix_multiply_rpmsg_rtos
- erpc_two_way_rpc_rpmsg_rtos
- freertos_message_buffers
- hello_world
- multicore_manager
- rpmsg_lite_pingpong
- rpmsg_lite_pingpong_rtos
- rpmsg_lite_pingpong_tzm

The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.

The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in

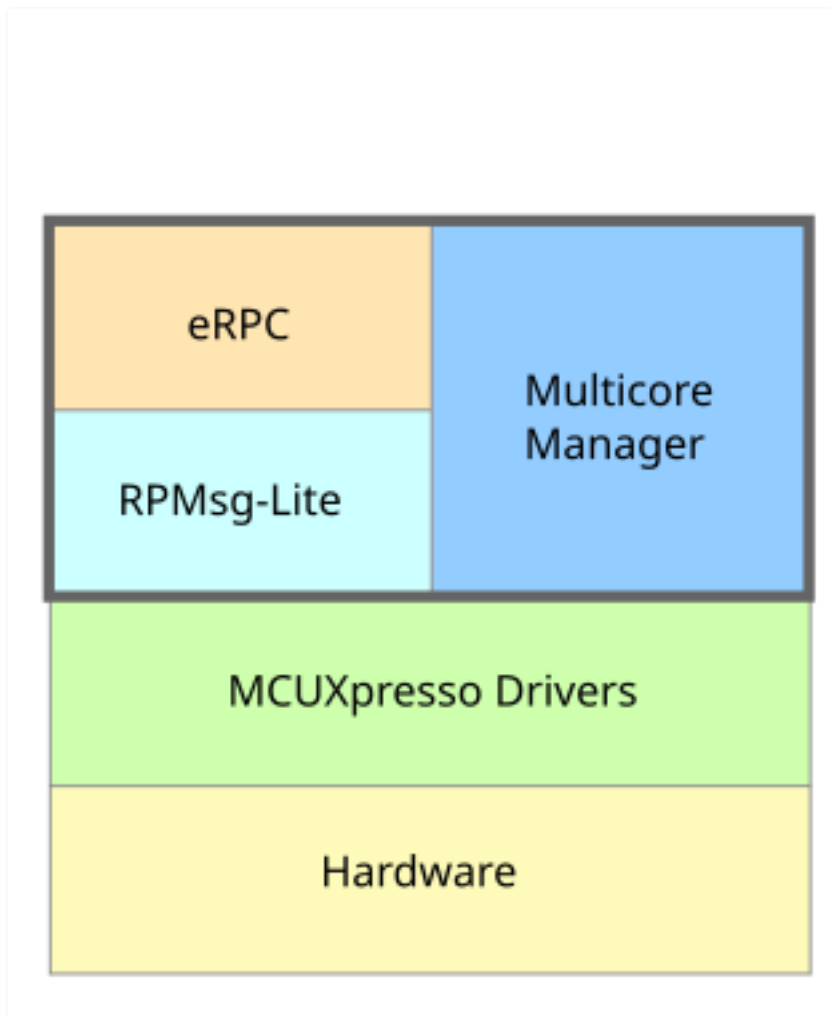
the <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples subdirectories.

- erpc_client_matrix_multiply_spi
- erpc_server_matrix_multiply_spi
- erpc_client_matrix_multiply_uart
- erpc_server_matrix_multiply_uart
- erpc_server_dac_adc
- erpc_remote_control

Getting Started with Multicore SDK (MCSDK)

Overview Multicore Software Development Kit (MCSDK) is a Software Development Kit that provides comprehensive software support for NXP dual/multicore devices. The MCSDK is combined with the MCUXpresso SDK to make the software framework for easy development of multicore applications.

The following figure highlights the layers and main software components of the MCSDK.

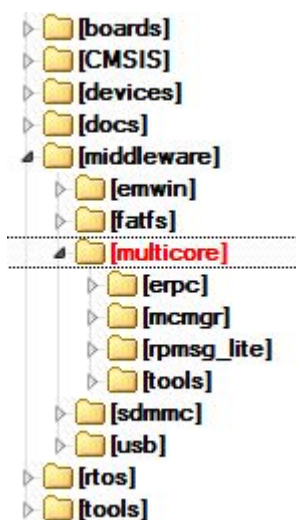


All the MCSDK-related files are located in `<MCUXpressoSDK_install_dir>/middleware/multicore` folder.

For supported toolchain versions, see the *Multicore SDK v25.06.00 Release Notes* (document MCS-DKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

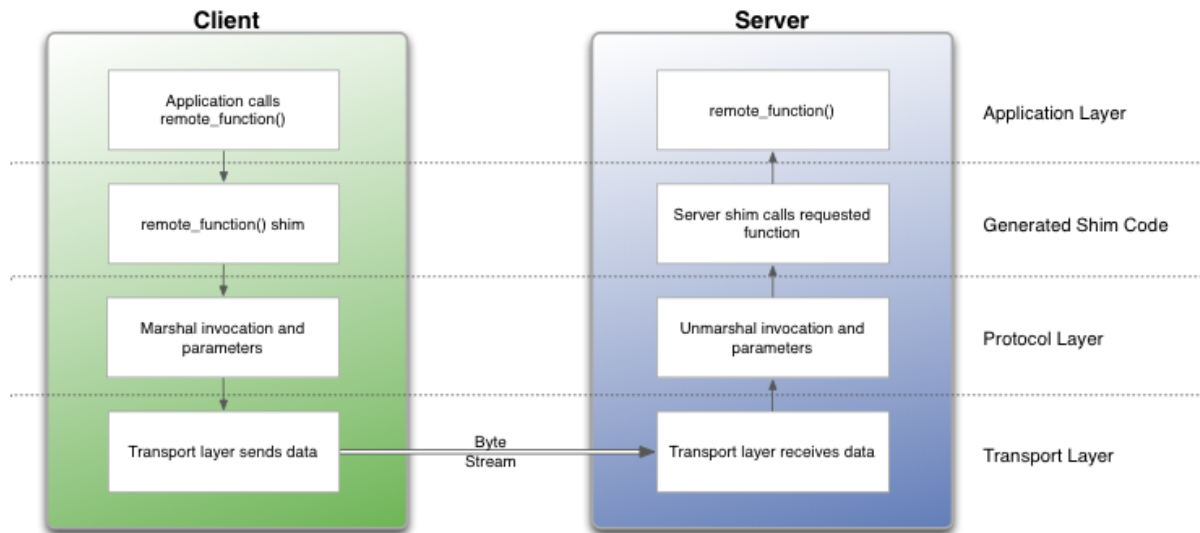
Multicore SDK (MCSDK) components The MCSDK consists of the following software components:

- **Embedded Remote Procedure Call (eRPC):** This component is a combination of a library and code generator tool that implements a transparent function call interface to remote services (running on a different core).
- **Multicore Manager (MCMGR):** This library maintains information about all cores and starts up secondary/auxiliary cores.
- **Remote Processor Messaging - Lite (RPMsg-Lite):** Inter-Processor Communication library.



Embedded Remote Procedure Call (eRPC) The Embedded Remote Procedure Call (eRPC) is the RPC system created by NXP. The RPC is a mechanism used to invoke a software routine on a remote system via a simple local function call.

When a remote function is called by the client, the function's parameters and an identifier for the called routine are marshaled (or serialized) into a stream of bytes. This byte stream is transported to the server through a communications channel (IPC, TPC/IP, UART, and so on). The server unmarshals the parameters, determines which function was invoked, and calls it. If the function returns a value, it is marshaled and sent back to the client.



RPC implementations typically use a combination of a tool (erpcgen) and IDL (interface definition language) file to generate source code to handle the details of marshaling a function's parameters and building the data stream.

Main eRPC features:

- Scalable from BareMetal to Linux OS - configurable memory and threading policies.
- Focus on embedded systems - intrinsic support for C, modular, and lightweight implementation.
- Abstracted transport interface - RPMsg is the primary transport for multicore, UART, or SPI-based solutions can be used for multichip.

The eRPC library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc` folder. For detailed information about the eRPC, see the documentation available in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems.

The main MCMGR features:

- Maintains information about all cores in system.
- Secondary/auxiliary cores startup and shutdown.
- Remote core monitoring and event handling.

The MCMGR library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr` folder. For detailed information about the MCMGR library, see the documentation available in the `<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/doc` folder.

Remote Processor Messaging Lite (RPMsg-Lite) RPMsg-Lite is a lightweight implementation of the RPMsg protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system. Compared to the legacy OpenAMP implementation, RPMsg-Lite offers a code size reduction, API simplification, and improved modularity.

The main RPMsg protocol features:

- Shared memory interprocessor communication.
- Virtio-based messaging bus.
- Application-defined messages sent between endpoints.

- Portable to different environments/platforms.
- Available in upstream Linux OS.

The RPMsg-Lite library is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg-lite` folder. For detailed information about the RPMsg-Lite, see the RPMsg-Lite User's Guide located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/doc` folder.

MCSDK demo applications Multicore and multiprocessor example applications are stored together with other MCUXpresso SDK examples, in the dedicated multicore subfolder.

Location		Folder
Multicore projects	example	<code><MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/</code>
Multiprocessor projects	example	<code><MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/<application_name>/</code>

See the *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) and *Getting Started with MCUXpresso SDK for XXX Derivatives* documents for more information about the MCUXpresso SDK example folder structure and the location of individual files that form the example application projects. These documents also contain information about building, running, and debugging multicore demo applications in individual supported IDEs. Each example application also contains a readme file that describes the operation of the example and required setup steps.

Inter-Processor Communication (IPC) levels The MCSDK provides several mechanisms for Inter-Processor Communication (IPC). Particular ways and levels of IPC are described in this chapter.

IPC using low-level drivers

The NXP multicore SoCs are equipped with peripheral modules dedicated for data exchange between individual cores. They deal with the Mailbox peripheral for LPC parts and the Messaging Unit (MU) peripheral for Kinetis and i.MX parts. The common attribute of both modules is the ability to provide a means of IPC, allowing multiple CPUs to share resources and communicate with each other in a simple manner.

The most lightweight method of IPC uses the MCUXpresso SDK low-level drivers for these peripherals. Using the Mailbox/MU driver API functions, it is possible to pass a value from core to core via the dedicated registers (could be a scalar or a pointer to shared memory) and also to trigger inter-core interrupts for notifications.

For details about individual driver API functions, see the MCUXpresso SDK API Reference Manual of the specific multicore device. The MCUXpresso SDK is accompanied with the RPMsg-Lite documentation that shows how to use this API in multicore applications.

Messaging mechanism

On top of Mailbox/MU drivers, a messaging system can be implemented, allowing messages to send between multiple endpoints created on each of the CPUs. The RPMsg-Lite library of the MCSDK provides this ability and serves as the preferred MCUXpresso SDK messaging library. It implements ring buffers in shared memory for messages exchange without the need of a locking mechanism.

The RPMsg-Lite provides the abstraction layer and can be easily ported to different multicore platforms and environments (Operating Systems). The advantages of such a messaging system are ease of use (there is no need to study behavior of the used underlying hardware) and smooth application code portability between platforms due to unified messaging API.

However, this costs several kB of code and data memory. The MCUXpresso SDK is accompanied by the RPMsg-Lite documentation and several multicore examples. You can also obtain the latest RPMsg-Lite code from the GitHub account github.com/nxp-mcuxpresso/rpmsg-lite.

Remote procedure calls

To facilitate the IPC even more and to allow the remote functions invocation, the remote procedure call mechanism can be implemented. The eRPC of the MCSDK serves for these purposes and allows the ability to invoke a software routine on a remote system via a simple local function call. Utilizing different transport layers, it is possible to communicate between individual cores of multicore SoCs (via RPMsg-Lite) or between separate processors (via SPI, UART, or TCP/IP). The eRPC is mostly applicable to the MPU parts with enough of memory resources like i.MX parts.

The eRPC library allows you to export existing C functions without having to change their prototypes (in most cases). It is accompanied by the code generator tool that generates the shim code for serialization and invocation based on the IDL file with definitions of data types and remote interfaces (API).

If the communicating peer is running as a Linux OS user-space application, the generated code can be either in C/C++ or Python.

Using the eRPC simplifies the access to services implemented on individual cores. This way, the following types of applications running on dedicated cores can be easily interfaced:

- Communication stacks (USB, Thread, Bluetooth Low Energy, Zigbee)
- Sensor aggregation/fusion applications
- Encryption algorithms
- Virtual peripherals

The eRPC is publicly available from the following GitHub account: github.com/EmbeddedRPC/erpc. Also, the MCUXpresso SDK is accompanied by the eRPC code and several multicore and multiprocessor eRPC examples.

The mentioned IPC levels demonstrate the scalability of the Multicore SDK library. Based on application needs, different IPC techniques can be used. It depends on the complexity, required speed, memory resources, system design, and so on. The MCSDK brings users the possibility for quick and easy development of multicore and multiprocessor applications.

Changelog Multicore 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](#).

[25.06.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.14.0
 - eRPC generator (erpcgen) v1.14.0
 - Multicore Manager (MCMgr) v5.0.0
 - RPMsg-Lite v5.2.0

[25.03.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0

- eRPC generator (erpcgen) v1.13.0
- Multicore Manager (MCMgr) v4.1.7
- RPPMsg-Lite v5.1.4

[24.12.00]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.6
 - RPPMsg-Lite v5.1.3

[2.16.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.13.0
 - eRPC generator (erpcgen) v1.13.0
 - Multicore Manager (MCMgr) v4.1.5
 - RPPMsg-Lite v5.1.2

[2.15.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.12.0
 - eRPC generator (erpcgen) v1.12.0
 - Multicore Manager (MCMgr) v4.1.5
 - RPPMsg-Lite v5.1.1

[2.14.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.11.0
 - eRPC generator (erpcgen) v1.11.0
 - Multicore Manager (MCMgr) v4.1.4
 - RPPMsg-Lite v5.1.0

[2.13.0_imxrt1180a0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.10.0
 - eRPC generator (erpcgen) v1.10.0
 - Multicore Manager (MCMgr) v4.1.3
 - RPPMsg-Lite v5.0.0

[2.13.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.10.0
 - eRPC generator (erpcgen) v1.10.0
 - Multicore Manager (MCMgr) v4.1.3
 - RPSMsg-Lite v5.0.0

[2.12.0_imx93]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.1
 - eRPC generator (erpcgen) v1.9.1
 - Multicore Manager (MCMgr) v4.1.2
 - RPSMsg-Lite v4.0.1

[2.12.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.1
 - eRPC generator (erpcgen) v1.9.1
 - Multicore Manager (MCMgr) v4.1.2
 - RPSMsg-Lite v4.0.0

[2.11.1]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.0
 - eRPC generator (erpcgen) v1.9.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.2.1

[2.11.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.9.0
 - eRPC generator (erpcgen) v1.9.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.2.0

[2.10.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.8.1
 - eRPC generator (erpcgen) v1.8.1
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.1.2

[2.9.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.8.0
 - eRPC generator (erpcgen) v1.8.0
 - Multicore Manager (MCMgr) v4.1.1
 - RPSMsg-Lite v3.1.1

[2.8.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.4
 - eRPC generator (erpcgen) v1.7.4
 - Multicore Manager (MCMgr) v4.1.0
 - RPSMsg-Lite v3.1.0

[2.7.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.3
 - eRPC generator (erpcgen) v1.7.3
 - Multicore Manager (MCMgr) v4.1.0
 - RPSMsg-Lite v3.0.0

[2.6.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.2
 - eRPC generator (erpcgen) v1.7.2
 - Multicore Manager (MCMgr) v4.0.3
 - RPSMsg-Lite v2.2.0

[2.5.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.1
 - eRPC generator (erpcgen) v1.7.1
 - Multicore Manager (MCMgr) v4.0.2
 - RPSMsg-Lite v2.0.2

[2.4.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.7.0
 - eRPC generator (erpcgen) v1.7.0
 - Multicore Manager (MCMgr) v4.0.1
 - RPSMsg-Lite v2.0.1

[2.3.1]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.6.0
 - eRPC generator (erpcgen) v1.6.0
 - Multicore Manager (MCMgr) v4.0.0
 - RPSMsg-Lite v1.2.0

[2.3.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.5.0
 - eRPC generator (erpcgen) v1.5.0
 - Multicore Manager (MCMgr) v3.0.0
 - RPSMsg-Lite v1.2.0

[2.2.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.4.0
 - eRPC generator (erpcgen) v1.4.0
 - Multicore Manager (MCMgr) v2.0.1
 - RPSMsg-Lite v1.1.0

[2.1.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.3.0
 - eRPC generator (erpcgen) v1.3.0

[2.0.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.2.0
 - eRPC generator (erpcgen) v1.2.0
 - Multicore Manager (MCMgr) v2.0.0
 - RPMMsg-Lite v1.0.0

[1.1.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.1.0
 - Multicore Manager (MCMgr) v1.1.0
 - Open-AMP / RPMMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev01

[1.0.0]

- Multicore SDK component versions:
 - embedded Remote Procedure Call (eRPC) v1.0.0
 - Multicore Manager (MCMgr) v1.0.0
 - Open-AMP / RPMMsg based on SHA1 ID 44b5f3c0a6458f3cf80 rev00

Multicore SDK Components

RPMSG-Lite

MCUXpresso SDK : mcuxsdk-middleware-rpmsg-lite

Overview This repository is for MCUXpresso SDK RPMSG-Lite middleware delivery and it contains RPMSG-Lite component 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](#) for the complete delivery of MCUXpresso SDK to be able to build and run RPMSG-Lite examples that are based on mcux-sdk-middleware-rpmsg-lite component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [RPMSG-Lite - 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 rpmsg-lite project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

RPMSG-Lite This documentation describes the RPMsg-Lite component, which is a lightweight implementation of the Remote Processor Messaging (RPMsg) protocol. The RPMsg protocol defines a standardized binary interface used to communicate between multiple cores in a heterogeneous multicore system.

Compared to the RPMsg implementation of the Open Asymmetric Multi Processing (OpenAMP) framework (<https://github.com/OpenAMP/open-amp>), the RPMsg-Lite offers a code size reduction, API simplification, and improved modularity. On smaller Cortex-M0+ based systems, it is recommended to use RPMsg-Lite.

The RPMsg-Lite is an open-source component developed by NXP Semiconductors and released under the BSD-compatible license.

For Further documentation, please look at doxygen documentation at: <https://nxp-mcuxpresso.github.io/rpmsg-lite/>

Motivation to create RPMsg-Lite There are multiple reasons why RPMsg-Lite was developed. One reason is the need for the small footprint of the RPMsg protocol-compatible communication component, another reason is the simplification of extensive API of OpenAMP RPMsg implementation.

RPMsg protocol was not documented, and its only definition was given by the Linux Kernel and legacy OpenAMP implementations. This has changed with [1] which is a standardization protocol allowing multiple different implementations to coexist and still be mutually compatible.

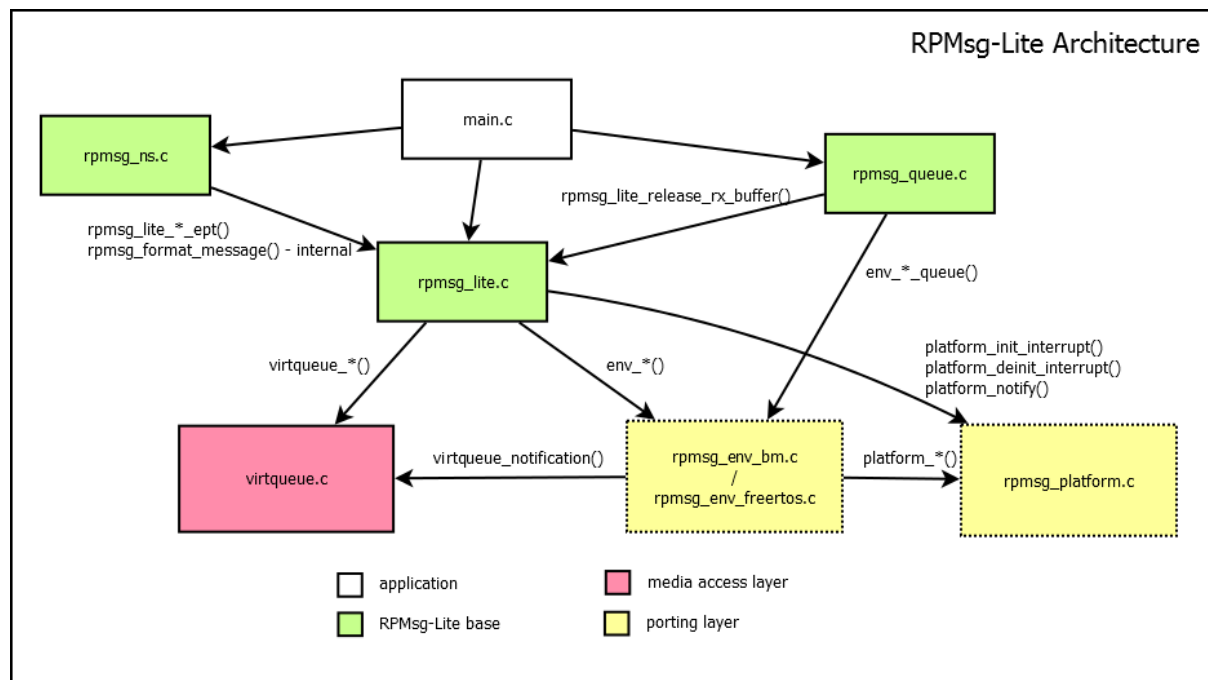
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property introduced by dynamic memory. The following table shows some rough comparison data between the OpenAMP RPMsg implementation and new RPMsg-Lite implementation:

Component / Configuration	Flash [B]	RAM [B]
OpenAMP RPMsg / Release (reference)	5547	456 + dynamic
RPMsg-Lite / Dynamic API, Release	3462	56 + dynamic
Relative Difference [%]	~62.4%	~12.3%
RPMsg-Lite / Static API (no malloc), Release	2926	352
Relative Difference [%]	~52.7%	~77.2%

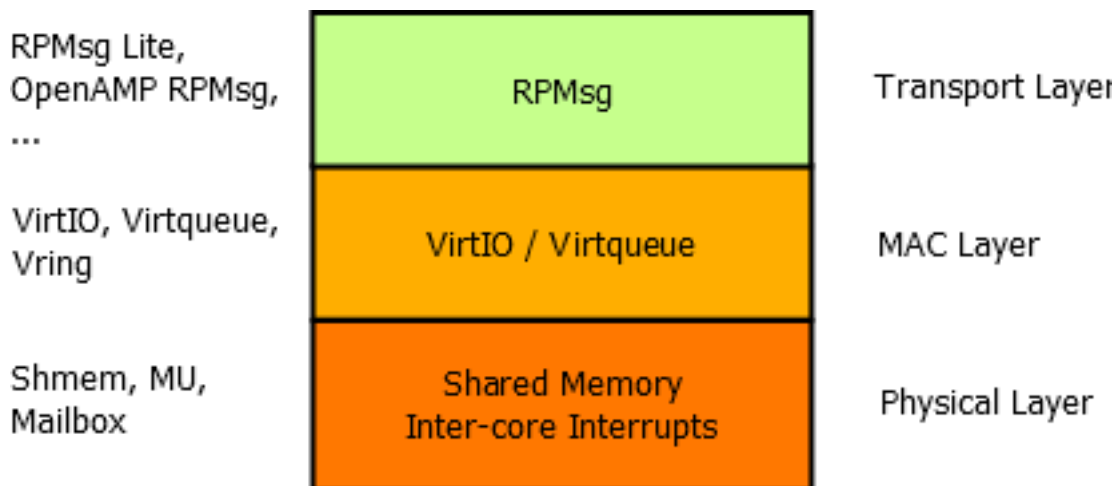
Implementation The implementation of RPMsg-Lite can be divided into three sub-components, from which two are optional. The core component is situated in `rpmsg_lite.c`. Two optional components are used to implement a blocking receive API (in `rpmsg_queue.c`) and dynamic “named” endpoint creation and deletion announcement service (in `rpmsg_ns.c`).

The actual “media access” layer is implemented in `virtqueue.c`, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as `vring` or `virtqueue`.

The porting layer is split into two sub-layers: the environment layer and the platform layer. The first sublayer is to be implemented separately for each environment. (The bare metal environment already exists and is implemented in `rpmsg_env_bm.c`, and the FreeRTOS environment is implemented in `rpmsg_env_freertos.c` etc.) Only the source file, which matches the used environment, is included in the target application project. The second sublayer is implemented in `rpmsg_platform.c` and defines low-level functions for interrupt enabling, disabling, and triggering mainly. The situation is described in the following figure:



RPMsg-Lite core sub-component This subcomponent implements a blocking send API and callback-based receive API. The RPMsg protocol is part of the transport layer. This is realized by using so-called endpoints. Each endpoint can be assigned a different receive callback function. However, it is important to notice that the callback is executed in an interrupt environment in current design. Therefore, certain actions like memory allocation are discouraged to execute in the callback. The following figure shows the role of RPMsg in an ISO/OSI-like layered model:



Queue sub-component (optional) This subcomponent is optional and requires implementation of the `env_*_queue()` functions in the environment porting layer. It uses a blocking receive API, which is common in RTOS-environments. It supports both copy and nocopy blocking receive functions.

Name Service sub-component (optional) This subcomponent is a minimum implementation of the name service which is present in the Linux Kernel implementation of RPMsg. It allows the communicating node both to send announcements about “named” endpoint (in other words, channel) creation or deletion and to receive these announcement taking any user-defined action

in an application callback. The endpoint address used to receive name service announcements is arbitrarily fixed to be 53 (0x35).

Usage The application should put the `/rpmsg_lite/lib/include` directory to the include path and in the application, include either the `rpmsg_lite.h` header file, or optionally also include the `rpmsg_queue.h` and/or `rpmsg_ns.h` files. Both porting sublayers should be provided for you by NXP, but if you plan to use your own RTOS, all you need to do is to implement your own environment layer (in other words, `rpmsg_env_myrtos.c`) and to include it in the project build.

The initialization of the stack is done by calling the `rpmsg_lite_master_init()` on the master side and the `rpmsg_lite_remote_init()` on the remote side. This initialization function must be called prior to any RPMsg-Lite API call. After the init, it is wise to create a communication endpoint, otherwise communication is not possible. This can be done by calling the `rpmsg_lite_create_ept()` function. It optionally accepts a last argument, where an internal context of the endpoint is created, just in case the `RL_USE_STATIC_API` option is set to 1. If not, the stack internally calls `env_alloc()` to allocate dynamic memory for it. In case a callback-based receiving is to be used, an ISR-callback is registered to each new endpoint with user-defined callback data pointer. If a blocking receive is desired (in case of RTOS environment), the `rpmsg_queue_create()` function must be called before calling `rpmsg_lite_create_ept()`. The queue handle is passed to the endpoint creation function as a callback data argument and the callback function is set to `rpmsg_queue_rx_cb()`. Then, it is possible to use `rpmsg_queue_receive()` function to listen on a queue object for incoming messages. The `rpmsg_lite_send()` function is used to send messages to the other side.

The RPMsg-Lite also implements no-copy mechanisms for both sending and receiving operations. These methods require specifics that have to be considered when used in an application.

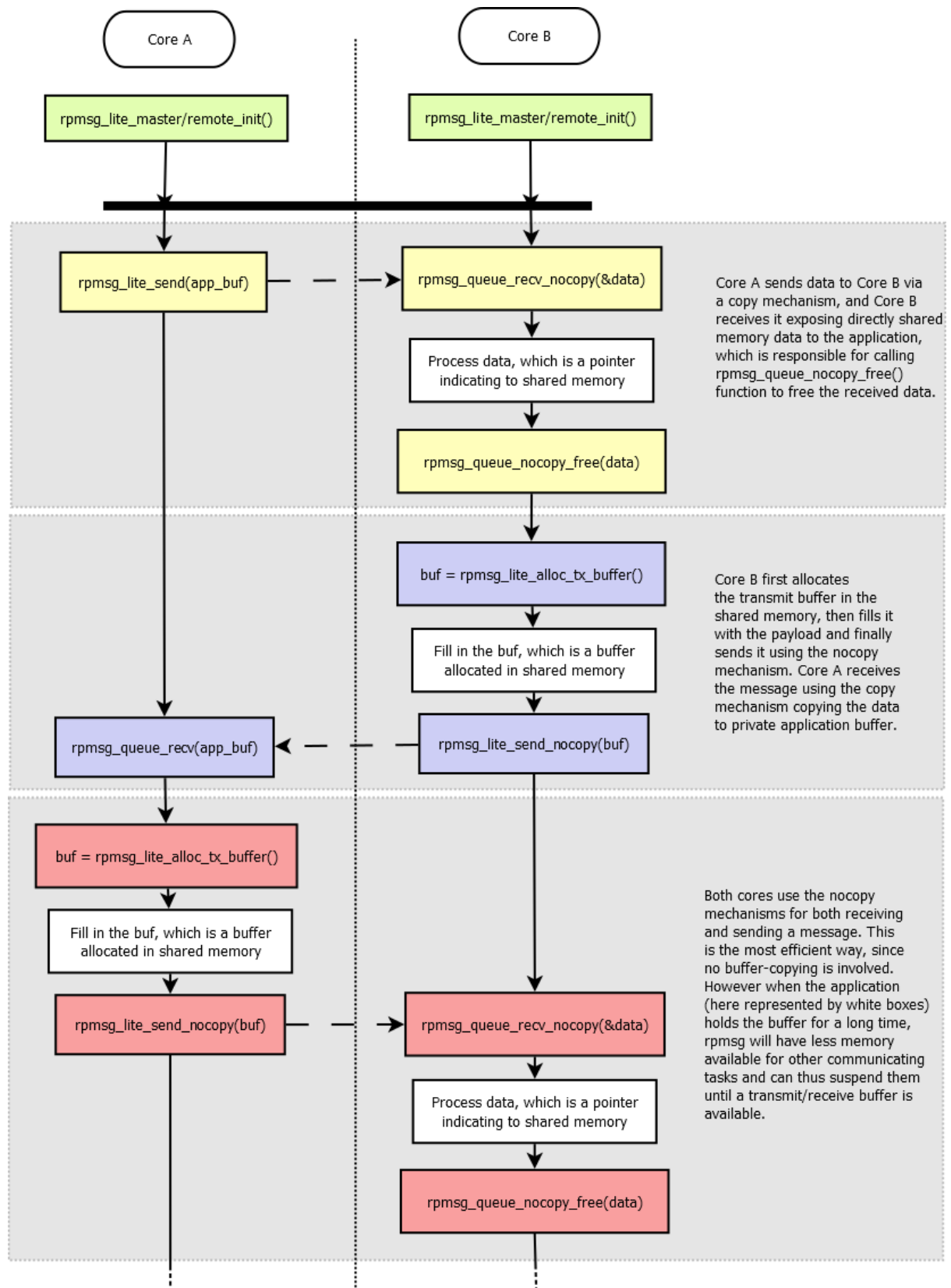
no-copy-send mechanism: This mechanism allows sending messages without the cost for copying data from the application buffer to the RPMsg/virtio buffer in the shared memory. The sequence of no-copy sending steps to be performed is as follows:

- Call the `rpmsg_lite_alloc_tx_buffer()` function to get the virtio buffer and provide the buffer pointer to the application.
- Fill the data to be sent into the pre-allocated virtio buffer. Ensure that the filled data does not exceed the buffer size (provided as the `rpmsg_lite_alloc_tx_buffer()` size output parameter).
- Call the `rpmsg_lite_send_nocopy()` function to send the message to the destination endpoint. Consider the cache functionality and the virtio buffer alignment. See the `rpmsg_lite_send_nocopy()` function description below.

no-copy-receive mechanism: This mechanism allows reading messages without the cost for copying data from the virtio buffer in the shared memory to the application buffer. The sequence of no-copy receiving steps to be performed is as follows:

- Call the `rpmsg_queue_rcv_nocopy()` function to get the virtio buffer pointer to the received data.
- Read received data directly from the shared memory.
- Call the `rpmsg_queue_nocopy_free()` function to release the virtio buffer and to make it available for the next data transfer.

The user is responsible for destroying any RPMsg-Lite objects he has created in case of deinitialization. In order to do this, the function `rpmsg_queue_destroy()` is used to destroy a queue, `rpmsg_lite_destroy_ept()` is used to destroy an endpoint and finally, `rpmsg_lite_deinit()` is used to deinitialize the RPMsg-Lite intercore communication stack. Deinitialize all endpoints using a queue before deinitializing the queue. Otherwise, you are actively invalidating the used queue handle, which is not allowed. RPMsg-Lite does not check this internally, since its main aim is to be lightweight.



Examples RPMsg_Lite multicore examples are part of NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages. To get the board list with multicore support (RPMsg_Lite included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded,

see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for RPMsg_Lite multicore examples with 'rpmsg_lite_' name prefix.

Another way of getting NXP MCUXpressoSDK RPMsg_Lite multicore examples is using the [mcuxsdk-manifests](#) Github repo. Follow the description how to use the West tool to clone and update the mcuxsdk-manifests repo in [readme section](#). Once done the armgcc rpmsg_lite examples can be found in

mcuxsdk/examples/_<board_name>/multicore_examples

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the RPMsg_Lite examples use the 'rpmsg_lite_' name prefix.

Notes

Environment layers implementation Several environment layers are provided in lib/rpmsg_lite/porting/environment folder. Not all of them are fully tested however. Here is the list of environment layers that passed testing:

- rpmsg_env_bm.c
- rpmsg_env_freertos.c
- rpmsg_env_xos.c
- rpmsg_env_threadx.c

The rest of environment layers has been created and used in some experimental projects, it has been running well at the time of creation but due to the lack of unit testing there is no guarantee it is still fully functional.

Shared memory configuration It is important to correctly initialize/configure the shared memory for data exchange in the application. The shared memory must be accessible from both the master and the remote core and it needs to be configured as Non-Cacheable memory. Dedicated shared memory section in linker file is also a good practise, it is recommended to use linker files from MCUXpressoSDK packages for NXP devices based applications. It needs to be ensured no other application part/component is unintentionally accessing this part of memory.

Configuration options The RPMsg-Lite can be configured at the compile time. The default configuration is defined in the rpmsg_default_config.h header file. This configuration can be customized by the user by including rpmsg_config.h file with custom settings. The following table summarizes all possible RPMsg-Lite configuration options.

Config- uration option	De- fault value	Usage
RL_MS_PE (1)		Delay in milliseconds used in non-blocking API functions for polling.
RL_BUFFE (496)		Size of the buffer payload, it must be equal to (240, 496, 1008, ...) $[2^n - 16]$
RL_BUFFE (2)		Number of the buffers, it must be power of two (2, 4, ...)
RL_API_H (1)		Zero-copy API functions enabled/disabled.
RL_USE_S' (0)		Static API functions (no dynamic allocation) enabled/disabled.
RL_USE_D (0)		Memory cache management of shared memory. Use in case of data cache is enabled for shared memory.
RL_CLEAF (0)		Clearing used buffers before returning back to the pool of free buffers enabled/disabled.
RL_USE_M (0)		When enabled IPC interrupts are managed by the Multicore Manager (IPC interrupts router), when disabled RPMsg-Lite manages IPC interrupts by itself.
RL_USE_E (0)		When enabled the environment layer uses its own context. Required for some environments (QNX). The default value is 0 (no context, saves some RAM).
RL_DEBU (0)		When enabled buffer pointers passed to <code>rpmsg_lite_send_nocopy()</code> and <code>rpmsg_lite_release_rx_buffer()</code> functions (enabled by <code>RL_API_HAS_ZEROCOPY</code> config) are checked to avoid passing invalid buffer pointer. The default value is 0 (disabled). Do not use in RPMsg-Lite to Linux configuration.
RL_ALLO (0)		When enabled the opposite side is notified each time received buffers are consumed and put into the queue of available buffers. Enable this option in RPMsg-Lite to Linux configuration to allow unblocking of the Linux blocking send. The default value is 0 (RPMsg-Lite to RPMsg-Lite communication).
RL_ALLO (0)		It allows to define custom shared memory configuration and replacing the shared memory related global settings from <code>rpmsg_config.h</code> . This is useful when multiple instances are running in parallel but different shared memory arrangement (vring size & alignment, buffers size & count) is required. The default value is 0 (all RPMsg-Lite instances use the same shared memory arrangement as defined by common config macros).
RL_ASSERT	see <code>rpmsg</code>	Assert implementation.

How to format rpmsg-lite code To format code, use the application developed by Google, named *clang-format*. This tool is part of the [llvm](#) project. Currently, the clang-format 10.0.0 version is used for rpmsg-lite. The set of style settings used for clang-format is defined in the `.clang-format` file, placed in a root of the rpmsg-lite directory where Python script `run_clang_format.py` can be executed. This script executes the application named *clang-format.exe*. You need to have the path of this application in the OS's environment path, or you need to change the script.

References

[1] M. Novak, M. Cingel, **Lockless Shared Memory Based Multicore Communication Protocol**
Copyright © 2016 Freescale Semiconductor, Inc. Copyright © 2016-2025 NXP

Changelog RPMMSG-Lite 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

Fixed

- Fixed CERT-C INT31-C violation in platform_notify function in rpmsg_platform.c for imxrt700_m33, imxrt700_hifi4, imxrt700_hifi1 platforms

v5.2.0

Added

- Add MCXL20 porting layer and unit testing
- New utility macro RL_CALCULATE_BUFFER_COUNT_DOWN_SAFE to safely determine maximum buffer count within shared memory while preventing integer underflow.
- RT700 platform add support for MCMGR in DSPs

Changed

- Change rpmsg_platform.c to support new MCMGR API
- Improved input validation in initialization functions to properly handle insufficient memory size conditions.
- Refactored repeated buffer count calculation pattern for better code maintainability.
- To make sure that remote has already registered IRQ there is required App level IPC mechanism to notify master about it

Fixed

- Fixed env_wait_for_link_up function to handle timeout in link state checks for baremetal and qnx environment, RL_BLOCK mode can be used to wait indefinitely.
- Fixed CERT-C INT31-C violation by adding compile-time check to ensure RL_PLATFORM_HIGHEST_LINK_ID remains within safe range for 16-bit casting in virtqueue ID creation.
- Fixed CERT-C INT30-C violations by adding protection against unsigned integer underflow in shared memory calculations, specifically in shmem_length - (uint32_t)RL_VRING_OVERHEAD and shmem_length - 2U * shmem_config.vring_size expressions.
- Fixed CERT INT31-C violation in platform_interrupt_disable() and similar functions by replacing unsafe cast from uint32_t to int32_t with a return of 0 constant.
- Fixed unsigned integer underflow in rpmsg_lite_alloc_tx_buffer() where subtracting header size from buffer size could wrap around if buffer was too small, potentially leading to incorrect buffer sizing.
- Fixed CERT-C INT31-C violation in rpmsg_lite.c where size parameter was cast from uint32_t to uint16_t without proper validation.
 - Applied consistent masking approach to both size and flags parameters: (uint16_t)(value & 0xFFFFU).
 - This fix prevents potential data loss when size values exceed 65535.

- Fixed CERT INT31-C violation in `env_memset` functions by explicitly converting `int32_t` values to unsigned char using bit masking. This prevents potential data loss or misinterpretation when passing values outside the unsigned char range (0-255) to the standard `memset()` function.
- Fixed CERT-C INT31-C violations in RPMsg-Lite environment porting: Added validation checks for signed-to-unsigned integer conversions to prevent data loss and misinterpretation.
 - `rpmsg_env_freertos.c`: Added validation before converting `int32_t` to `UBaseType_t`.
 - `rpmsg_env_qnx.c`: Fixed format string and added validation before assigning to `mqstat` fields.
 - `rpmsg_env_threadx.c`: Added validation to prevent integer overflow and negative values.
 - `rpmsg_env_xos.c`: Added range checking before casting to `uint16_t`.
 - `rpmsg_env_zephyr.c`: Added validation before passing values to `k_msgq_init`.
- Fixed a CERT INT31-C compliance issue in `env_get_current_queue_size()` function where an unsigned queue count was cast to a signed `int32_t` without proper validation, which could lead to lost or misinterpreted data if queue size exceeded `INT32_MAX`.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where `memcmp()` return value (signed int) was compared with unsigned constant without proper type handling.
- Fixed CERT INT31-C violation in `rpmsg_platform.c` where casting from `uint32_t` to `uint16_t` could potentially result in data loss. Changed length variable type from `uint16_t` to `uint32_t` to properly handle memory address differences without truncation.
- Fixed potential integer overflow in `env_sleep_msec()` function in ThreadX environment implementation by rearranging calculation order in the sleep duration formula.
- Fixed CERT-C INT31-C violation in RPMsg-Lite where bitwise NOT operations on integer constants were performed in signed integer context before being cast to unsigned. This could potentially lead to misinterpreted data on `imx943` platform.
- Added `RL_MAX_BUFFER_COUNT` (32768U) and `RL_MAX_VRING_ALIGN` (65536U) limit to ensure alignment values cannot contribute to integer overflow
- Fixed CERT INT31-C violation in `vring_need_event()`, added cast to `uint16_t` for each operand.

v5.1.4 - 27-Mar-2025

Added

- Add KW43B43 porting layer

Changed

- Doxygen bump to version 1.9.6

v5.1.3 - 13-Jan-2025

Added

- Memory cache management of shared memory. Enable with `#define RL_USE_DCACHE (1)` in `rpmsg_config.h` in case of data cache is used.
- Cmake/Kconfig support added.
- Porting layers for imx95, imxrt700, mcmxw71x, mcmxw72x, kw47b42 added.

v5.1.2 - 08-Jul-2024

Changed

- Zephyr-related changes.
- Minor Misra corrections.

v5.1.1 - 19-Jan-2024

Added

- Test suite provided.
- Zephyr support added.

Changed

- Minor changes in platform and env. layers, minor test code updates.

v5.1.0 - 02-Aug-2023

Added

- RPMsg-Lite: Added aarch64 support.

Changed

- RPMsg-Lite: Increased the queue size to $(2 * RL_BUFFER_COUNT)$ to cover zero copy cases.
- Code formatting using LLVM16.

Fixed

- Resolved issues in ThreadX env. layer implementation.

v5.0.0 - 19-Jan-2023

Added

- Timeout parameter added to `rpmsg_lite_wait_for_link_up` API function.

Changed

- Improved debug check buffers implementation - instead of checking the pointer fits into shared memory check the presence in the VirtIO ring descriptors list.
- VRING_SIZE is set based on number of used buffers now (as calculated in vring_init) - updated for all platforms that are not communicating to Linux rpmsg counterpart.

Fixed

- Fixed wrong RL_VRING_OVERHEAD macro comment in platform.h files
- Misra corrections.

v4.0.0 - 20-Jun-2022

Added

- Added support for custom shared memory arrangement per the RPMsg_Lite instance.
- Introduced new rpmsg_lite_wait_for_link_up() API function - this allows to avoid using busy loops in rtos environments, GitHub PR [#21](#).

Changed

- Adjusted rpmsg_lite_is_link_up() to return RL_TRUE/RL_FALSE.

v3.2.0 - 17-Jan-2022

Added

- Added support for i.MX8 MP multicore platform.

Changed

- Improved static allocations - allow OS-specific objects being allocated statically, GitHub PR [#14](#).
- Aligned rpmsg_env_xos.c and some platform layers to latest static allocation support.

Fixed

- Minor Misra and typo corrections, GitHub PR [#19](#), [#20](#).

v3.1.2 - 16-Jul-2021

Added

- Addressed MISRA 21.6 rule violation in rpmsg_env.h (use SDK's PRINTF in MCUXpressoSDK examples, otherwise stdio printf is used).
- Added environment layers for XOS.
- Added support for i.MX RT500, i.MX RT1160 and i.MX RT1170 multicore platforms.

Fixed

- Fixed incorrect description of the `rpmsg_lite_get_endpoint_from_addr` function.

Changed

- Updated `RL_BUFFER_COUNT` documentation (issue [#10](#)).
- Updated `imxrt600_hifi4` platform layer.

v3.1.1 - 15-Jan-2021

Added

- Introduced `RL_ALLOW_CONSUMED_BUFFERS_NOTIFICATION` config option to allow opposite side notification sending each time received buffers are consumed and put into the queue of available buffers.
- Added environment layers for Threadx.
- Added support for i.MX8QM multicore platform.

Changed

- Several MISRA C-2012 violations addressed.

v3.1.0 - 22-Jul-2020

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed (7.4).
- Fixed missing lock in `rpmsg_lite_rx_callback()` for QNX env.
- Correction of `rpmsg_lite_instance` structure members description.
- Address -Waddress-of-packed-member warnings in GCC9.

Changed

- Clang update to v10.0.0, code re-formatted.

v3.0.0 - 20-Dec-2019

Added

- Added support for several new multicore platforms.

Fixed

- MISRA C-2012 violations fixed, incl. data types consolidation.
- Code formatted.

v2.2.0 - 20-Mar-2019

Added

- Added configuration macro `RL_DEBUG_CHECK_BUFFERS`.
- Several MISRA violations fixed.
- Added environment layers for QNX and Zephyr.
- Allow environment context required for some environment (controlled by the `RL_USE_ENVIRONMENT_CONTEXT` configuration macro).
- Data types consolidation.

v1.1.0 - 28-Apr-2017

Added

- Supporting i.MX6SX and i.MX7D MPU platforms.
- Supporting LPC5411x MCU platform.
- Baremetal and FreeRTOS support.
- Support of copy and zero-copy transfer.
- Support of static API (without dynamic allocations).

Multicore Manager

MCUXpresso SDK : `mcuxsdk-middleware-mcmgr` (Multicore Manager)

Overview This repository is for MCUXpresso SDK Multicore Manager middleware delivery and it contains Multicore Manager component 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](#) for the complete delivery of MCUXpresso SDK to be able to build and run Multicore Manager examples that are based on `mcux-sdk-middleware-mcmgr` component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

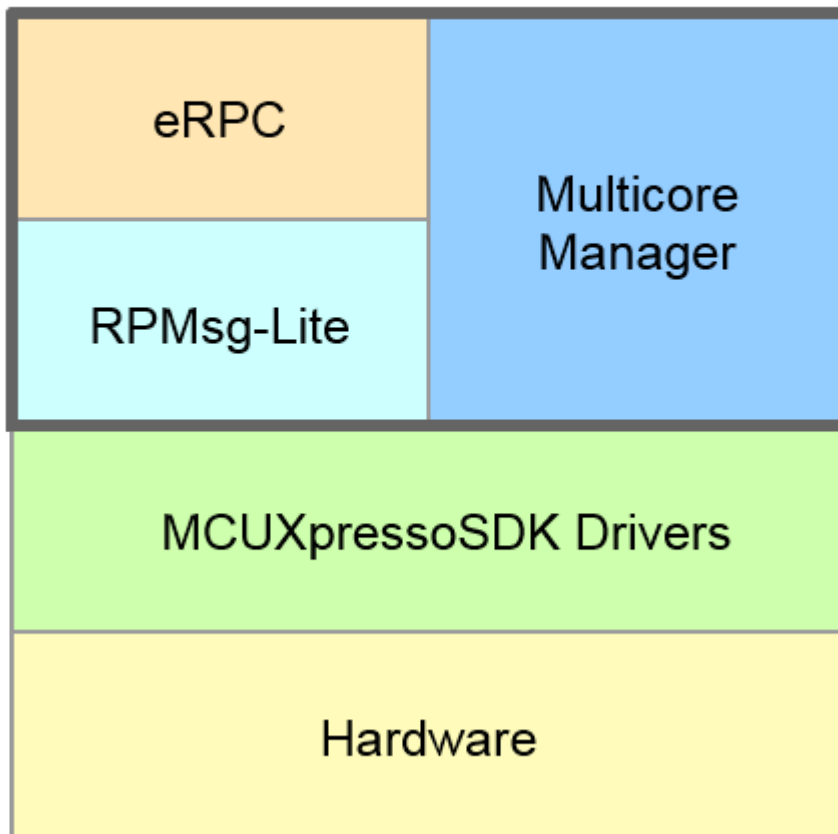
Visit [Multicore Manager - Documentation](#) to review details on the contents in this sub-repo.

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 mcmgr project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

Multicore Manager (MCMGR) The Multicore Manager (MCMGR) software library provides a number of services for multicore systems. This library is distributed as a part of the Multicore SDK (MCSDK). Together, the MCSDK and the MCUXpresso SDK (SDK) form a framework for development of software for NXP multicore devices.

The MCMGR component is located in the <MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr directory.



The Multicore Manager provides the following major functions:

- Maintains information about all cores in system.
- Secondary/auxiliary core(s) startup and shutdown.
- Remote core monitoring and event handling.

Usage of the MCMGR software component The main use case of MCMGR is the secondary/auxiliary core start. This functionality is performed by the public API function.

Example of MCMGR usage to start secondary core:

```
#include "mcmgr.h"

void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();

    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass "1" as startup data,
    ↪ starting synchronously. */
    MCMGR_StartCore(kMCMGR_Core1, CORE1_BOOT_ADDRESS, 1, kMCMGR_Start_Synchronous);
    .
    .
    .
    /* Stop secondary core execution. */
    MCMGR_StopCore(kMCMGR_Core1);
}
```

Some platforms allow stopping and re-starting the secondary core application again, using the MCMGR_StopCore / MCMGR_StartCore API calls. It is necessary to ensure the initially loaded image is not corrupted before re-starting, especially if it deals with the RAM target. Cache coherence has to be considered/ensured as well.

Another important MCMGR feature is the ability for remote core monitoring and handling of events such as reset, exception, and application events. Application-specific callback functions for events are registered by the MCMGR_RegisterEvent() API. Triggering these events is done using the MCMGR_TriggerEvent() API. mcmgr_event_type_t enums all possible event types.

An example of MCMGR usage for remote core monitoring and event handling. Code for the primary side:

```
#include "mcmgr.h"

#define APP_RPMSG_READY_EVENT_DATA (1)
#define APP_NUMBER_OF_CORES (2)
#define APP_SECONDARY_CORE kMCMGR_Core1

/* Callback function registered via the MCMGR_RegisterEvent() and triggered by MCMGR_TriggerEvent()
↪ called on the secondary core side */
void RPMsgRemoteReadyEventHandler(mcmgr_core_t coreNum, uint16_t eventData, void *context)
{
    uint16_t *data = &((uint16_t *)context)[coreNum];

    *data = eventData;
}

void main()
{
    uint16_t RPMsgRemoteReadyEventData[NUMBER_OF_CORES] = {0};

    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
```

(continues on next page)

(continued from previous page)

```

/* Register the application event before starting the secondary core */
MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPSMsgRemoteReadyEventHandler, (void_)
↳)RPSMsgRemoteReadyEventData);

/* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpsmsg_lite_base address_
↳as startup data, starting synchronously. */
MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpsmsg_lite_
↳base, kMCMGR_Start_Synchronous);

/* Wait until the secondary core application signals the rpsmsg remote has been initialized and is ready to_
↳communicate. */
while(APP_RPSMSG_READY_EVENT_DATA != RPSMsgRemoteReadyEventData[APP_SECONDARY_
↳CORE]) {};
.
.
.
}

```

Code for the secondary side:

```

#include "mcmgr.h"

#define APP_RPSMSG_READY_EVENT_DATA (1)

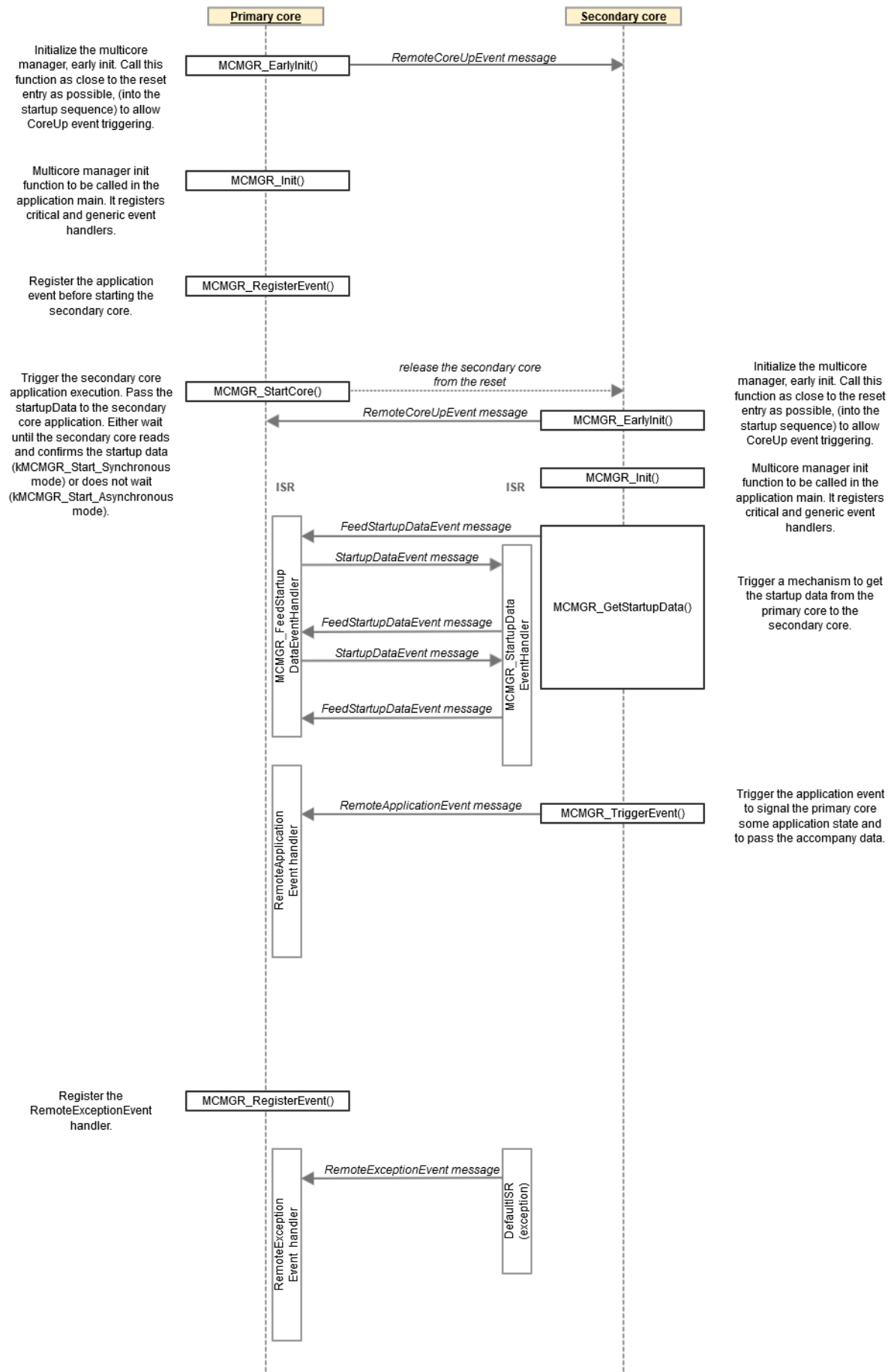
void main()
{
    /* Initialize MCMGR - low level multicore management library.
       Call this function as close to the reset entry as possible,
       (into the startup sequence) to allow CoreUp event triggering. */
    MCMGR_EarlyInit();

    /* Initialize MCMGR, install generic event handlers */
    MCMGR_Init();
    .
    .
    .

    /* Signal the to other core that we are ready by triggering the event and passing the APP_RPSMSG_
    ↳READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPSMSG_
    ↳READY_EVENT_DATA);
    .
    .
    .
}

```

MCMGR Data Exchange Diagram The following picture shows how the handshakes are supposed to work between the two cores in the MCMGR software.



Changelog Multicore Manager 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

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.

v5.0.0

Added

- Added MCMGR_BUSY_POLL_COUNT macro to prevent infinite polling loops in MCMGR operations.
- Implemented timeout mechanism for all polling loops in MCMGR code.
- Added support to handle more than two cores. Breaking API change by adding parameter `coreNum` specifying core number in functions bellow.
 - MCMGR_GetStartupData(uint32_t *startupData, mcmgr_core_t coreNum)
 - MCMGR_TriggerEvent(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
 - MCMGR_TriggerEventForce(mcmgr_event_type_t type, uint16_t eventData, mcmgr_core_t coreNum)
 - typedef void (*mcmgr_event_callback_t)(uint16_t data, void *context, mcmgr_core_t coreNum);

When registering the event with function MCMGR_RegisterEvent() user now needs to provide `callbackData` pointer to array of elements per every core in system (see README.md for example). In case of systems with only two cores the `coreNum` in callback can be ignored as events can arrive only from one core. Please see Porting guide for more details: Porting-GuideTo_v5.md

- Updated all porting files to support new MCMGR API.
- Added new platform specific include file `mcmgr_platform.h`. It will contain common platform specific macros that can be then used in `mcmgr` and application. e.g. platform core count MCMGR_CORECOUNT 4.
- Move all header files to new `inc` directory.
- Added new platform-specific include files `inc/platform/<platform_name>/mcmgr_platform.h`.

Added

- Add MCXL20 porting layer and unit testing

v4.1.7

Fixed

- `mcmgr_stop_core_internal()` function now returns `kStatus_MCMGR_NotImplemented` status code instead of `kStatus_MCMGR_Success` when device does not support stop of secondary core. Ports affected: kw32w1, kw45b41, kw45b42, mcxw716, mcxw727.

[v4.1.6]

Added

- Multicore Manager moved to standalone repository.
- Add porting layers for imxrt700, mcmxw727, kw47b42.
- New `MCMGR_ProcessDeferredRxIsr()` API added.

[v4.1.5]

Added

- Add notification into `MCMGR_EarlyInit` and `mcmgr_early_init_internal` functions to avoid using uninitialized data in their implementations.

[v4.1.4]

Fixed

- Avoid calling tx isr callbacks when respective Messaging Unit Transmit Interrupt Enable flag is not set in the CR/TCR register.
- Messaging Unit RX and status registers are cleared after the initialization.

[v4.1.3]

Added

- Add porting layers for imxrt1180.

Fixed

- `mu_isr()` updated to avoid calling tx isr callbacks when respective Transmit Interrupt Enable flag is not set in the CR/TCR register.
- `mcmgr_mu_internal.c` code adaptation to new supported SoCs.

[v4.1.2]

Fixed

- Update `mcmgr_stop_core_internal()` implementations to set core state to `kMCMGR_ResetCoreState`.

[v4.1.0]

Fixed

- Code adjustments to address MISRA C-2012 Rules

[v4.0.3]

Fixed

- Documentation updated to describe handshaking in a graphic form.
- Minor code adjustments based on static analysis tool findings

[v4.0.2]

Fixed

- Align porting layers to the updated MCUXpressoSDK feature files.

[v4.0.1]

Fixed

- Code formatting, removed unused code

[v4.0.0]

Added

- Add new MCMGR_TriggerEventForce() API.

[v3.0.0]

Removed

- Removed MCMGR_LoadApp(), MCMGR_MapAddress() and MCMGR_SignalReady()

Modified

- Modified MCMGR_GetStartupData()

Added

- Added MCMGR_EarlyInit(), MCMGR_RegisterEvent() and MCMGR_TriggerEvent()
- Added the ability for remote core monitoring and event handling

[v2.0.1]

Fixed

- Updated to be Misra compliant.

[v2.0.0]

Added

- Support for lpcxpresso54114 board.

[v1.1.0]

Fixed

- Ported to KSDK 2.0.0.

[v1.0.0]

Added

- Initial release.

eRPC

MCUXpresso SDK : mcuxsdk-middleware-erpc

Overview This repository is for MCUXpresso SDK eRPC middleware delivery and it contains eRPC component 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](#) for the complete delivery of MCUXpresso SDK to be able to build and run eRPC examples that are based on mcux-sdk-middleware-erpc component.

Documentation Overall details can be reviewed here: [MCUXpresso SDK Online Documentation](#)

Visit [eRPC - 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 eRPC project placed on github. Contributing can be managed via pull-requests. Before a pull-request is created the code should be tested and properly formatted.

eRPC

- [MCUXpresso SDK : mcuxsdk-middleware-erpc](#)

- [Overview](#)
- [Documentation](#)
- [Setup](#)
- [Contribution](#)

- [eRPC](#)

- [About](#)
- [Releases](#)
 - * [Edge releases](#)
- [Documentation](#)
- [Examples](#)
- [References](#)
- [Directories](#)
- [Building and installing](#)
 - * [Requirements](#)
 - [Windows](#)
 - [Mac OS X](#)
 - * [Building](#)
 - [CMake and KConfig](#)
 - [Make](#)
 - * [Installing for Python](#)
- [Known issues and limitations](#)
- [Code providing](#)

About

eRPC (Embedded RPC) is an open source Remote Procedure Call (RPC) system for multichip embedded systems and heterogeneous multicore SoCs.

Unlike other modern RPC systems, such as the excellent [Apache Thrift](#), eRPC distinguishes itself by being designed for tightly coupled systems, using plain C for remote functions, and having a small code size (<5kB). It is not intended for high performance distributed systems over a network.

eRPC does not force upon you any particular API style. It allows you to export existing C functions, without having to change their prototypes. (There are limits, of course.) And although the

internal infrastructure is written in C++, most users will be able to use only the simple C setup APIs shown in the examples below.

A code generator tool called `erpcgen` is included. It accepts input IDL files, having an `.erpc` extension, that have definitions of your data types and remote interfaces, and generates the shim code that handles serialization and invocation. `erpcgen` can generate either C/C++ or Python code.

Example `.erpc` file:

```
// Define a data type.
enum LEDName { kRed, kGreen, kBlue }

// An interface is a logical grouping of functions.
interface IO {
    // Simple function declaration with an empty reply.
    set_led(LEDName whichLed, bool onOrOff) -> void
}
```

Client side usage:

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* init eRPC client IO service */
    initIO_client(client_manager);

    // Now we can call the remote function to turn on the green LED.
    set_led(kGreen, true);

    /* deinit objects */
    deinitIO_client();
    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}
```

```
void example_client(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_client_t client_manager;

    /* Init eRPC client infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    client_manager = erpc_client_init(transport, message_buffer_factory);

    /* scope for client service */
    {
        /* init eRPC client IO service */
        IO_client client(client_manager);

        // Now we can call the remote function to turn on the green LED.
        client.set_led(kGreen, true);
    }

    /* deinit objects */
```

(continues on next page)

(continued from previous page)

```

    erpc_client_deinit(client_manager);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

Server side usage:

```

// Implement the remote function.
void set_led(LEDName whichLed, bool onOrOff) {
    // implementation goes here
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    erpc_service_t service = create_IO_service();

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, service);

    // Run the server.
    erpc_server_run();

    /* deinit objects */
    destroy_IO_service(service);
    erpc_server_deinit(server);
    erpc_mbf_dynamic_deinit(message_buffer_factory);
    erpc_transport_tcp_deinit(transport);
}

```

```

// Implement the remote function.
class IO : public IO_interface
{
    /* eRPC call definition */
    void set_led(LEDName whichLed, bool onOrOff) override {
        // implementation goes here
    }
}

void example_server(void) {
    erpc_transport_t transport;
    erpc_mbf_t message_buffer_factory;
    erpc_server_t server;
    IO IOImpl;
    IO_service io(&IOImpl);

    /* Init eRPC server infrastructure */
    transport = erpc_transport_cmsis_uart_init(Driver_USART0);
    message_buffer_factory = erpc_mbf_dynamic_init();
    server = erpc_server_init(transport, message_buffer_factory);

    /* add custom service implementation to the server */
    erpc_add_service_to_server(server, &io);

    /* poll for requests */
}

```

(continues on next page)

(continued from previous page)

```
erpc_status_t err = server.run();

/* deinit objects */
erpc_server_deinit(server);
erpc_mbf_dynamic_deinit(message_buffer_factory);
erpc_transport_tcp_deinit(transport);
}
```

A number of transports are supported, and new transport classes are easy to write.

Supported transports can be found in *erpc/erpc_c/transport* folder. E.g:

- CMSIS UART
- NXP Kinetis SPI and DSPI
- POSIX and Windows serial port
- TCP/IP (mostly for testing)
- [NXP RPMsg-Lite / RPMsg TTY](#)
- SPIdev Linux
- USB CDC
- NXP Messaging Unit

eRPC is available with an unrestrictive BSD 3-clause license. See the [LICENSE file](#) for the full license text.

Releases [eRPC releases](#)

Edge releases Edge releases can be found on [eRPC CircleCI](#) webpage. Choose build of interest, then platform target and choose ARTIFACTS tab. Here you can find binary application from chosen build.

Documentation [Documentation](#) is in the wiki section.

[eRPC Infrastructure documentation](#)

Examples *Example IDL* is available in the *examples/* folder.

Plenty of eRPC multicore and multiprocessor examples can be also found in NXP MCUXpressoSDK packages. Visit <https://mcuxpresso.nxp.com> to configure, build and download these packages.

To get the board list with multicore support (eRPC included) use filtering based on Middleware and search for 'multicore' string. Once the selected package with the multicore middleware is downloaded, see

<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples for eRPC multicore examples (RPMsg_Lite or Messaging Unit transports used) or

<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples for eRPC multiprocessor examples (UART or SPI transports used).

eRPC examples use the 'erpc_' name prefix.

Another way of getting NXP MCUXpressoSDK eRPC multicore and multiprocessor examples is using the [mcux-sdk](#) Github repo. Follow the description how to use the West tool to clone and

update the mcuxsdk repo in [readme Overview section](#). Once done the armgcc eRPC examples can be found in

mcuxsdk/examples/<board_name>/multicore_examples or in

mcuxsdk/examples/<board_name>/multiprocessor_examples folders.

You can use the evkmimxrt1170 as the board_name for instance. Similar to MCUXpressoSDK packages the eRPC examples use the 'erpc_' name prefix.

References This section provides links to interesting erpc-based projects, articles, blogs or guides:

- [erpc \(EmbeddedRPC\) getting started notes](#)
- [ERPC Linux Local Environment Construction and Use](#)
- [The New Wio Terminal eRPC Firmware](#)

Directories *doc* - Documentation.

doxygen - Configuration and support files for running Doxygen over the eRPC C++ infrastructure and erpcgen code.

erpc_c - Holds C/C++ infrastructure for eRPC. This is the code you will include in your application.

erpc_python - Holds Python version of the eRPC infrastructure.

erpcgen - Holds source code for erpcgen and makefiles or project files to build erpcgen on Windows, Linux, and OS X.

erpcsniffer - Holds source code for erpcsniffer application.

examples - Several example IDL files.

mk - Contains common makefiles for building eRPC components.

test - Client/server tests. These tests verify the entire communications path from client to server and back.

utilities - Holds utilities which bring additional benefit to eRPC apps developers.

Building and installing These build instructions apply to host PCs and embedded Linux. For bare metal or RTOS embedded environments, you should copy the *erpc_c* directory into your application sources.

CMake and KConfig build:

It builds a static library of the eRPC C/C++ infrastructure, the *erpcgen* executable, and optionally the unit tests and examples.

CMake is compatible with gcc and clang. On Windows local MingGW downloaded by *script* can be used.

Make build:

It builds a static library of the eRPC C/C++ infrastructure, the *erpcgen* executable, and optionally the unit tests.

The makefiles are compatible with gcc or clang on Linux, OS X, and Cygwin. A Windows build of *erpcgen* using Visual Studio is also available in the *erpcgen/VisualStudio_v14* directory. There is also an Xcode project file in the *erpcgen* directory, which can be used to build *erpcgen* for OS X.

Requirements erPC now support building **erpcgen**, **erpc_lib**, **tests** and **C examples** using CMake.

Requirements when using CMake:

- **CMake** (minimal version 3.20.0)
- Generator - **Make, Ninja, ...**
- **C/C++ compiler** - **GCC, CLANG, ...**
- **Bison** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

Requirements when using Make:

- **Make**
- **C/C++ compiler - GCC, CLANG, ...**
- **Binson** - <https://www.gnu.org/software/bison/>
- **Flex** - <https://github.com/westes/flex/>

Windows Related steps to build **erpcgen** using **Visual Studio** are described in `erpcgen/VisualStudio_v14/readme_erpcgen.txt`.

To install MinGW, Bison, Flex locally on Windows:

```
./install_dependencies.ps1
* \ \ \

#### Linux

\ \ \ bash
./install_dependencies.sh
```

Mandatory for case, when build for different architecture is needed

- gcc-multilib, g++-multilib

Mac OS X

```
./install_dependencies.sh
```

Building

CMake and KConfig eRPC use CMake and KConfig to configurate and build eRPC related targets. KConfig can be edited by *prj.conf* or *menuconfig* when building.

Generate project, config and build. In *erpc/* execute:

```
cmake -B ./build # in erpc/build generate cmake project
cmake --build ./build --target menuconfig # Build menuconfig and configure erpcgen, erpc_lib, tests and examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

****CMake will use the system's default compilers and generator**

If you want to use Windows and locally installed MinGW, use *CMake preset* :


```
cmake --preset mingw64 # Generate project in ./build using mingw64's make and compilers
cmake --build ./build --target menuconfig # Build menuconfig and configure erpcgen, erpc_lib, tests and ↵
↵examples
cmake --build ./build # Build all selected target from prj.conf/menuconfig
```

Make To build the library and erpcgen, run from the repo root directory:

```
make
```

To install the library, erpcgen, and include files, run:

```
make install
```

You may need to sudo the make install.

By default this will install into /usr/local. If you want to install elsewhere, set the PREFIX environment variable. Example for installing into /opt:

```
make install PREFIX=/opt
```

List of top level Makefile targets:

- erpc: build the liberpc.a static library
- erpcgen: build the erpcgen tool
- erpcsniffer: build the sniffer tool
- test: build the unit tests under the *test* directory
- all: build all of the above
- install: install liberpc.a, erpcgen, and include files

eRPC code is validated with respect to the C++ 11 standard.

Installing for Python To install the Python infrastructure for eRPC see instructions in the *erpc python readme*.

Known issues and limitations

- Static allocations controlled by the ERPC_ALLOCATION_POLICY config macro are not fully supported yet, i.e. not all erpc objects can be allocated statically now. It deals with the ongoing process and the full static allocations support will be added in the future.

Code providing Repository on Github contains two main branches: **main** and **develop**. Code is developed on **develop** branch. Release version is created via merging **develop** branch into **main** branch.

Copyright 2014-2016 Freescale Semiconductor, Inc.

Copyright 2016-2025 NXP

eRPC Getting Started

Overview This *Getting Started User Guide* shows software developers how to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

The eRPC documentation is located in the `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/doc` folder.

Create an eRPC application This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
 1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
 2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
 3. Add infrastructure files.
 4. Add user code for client and server applications.
 5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

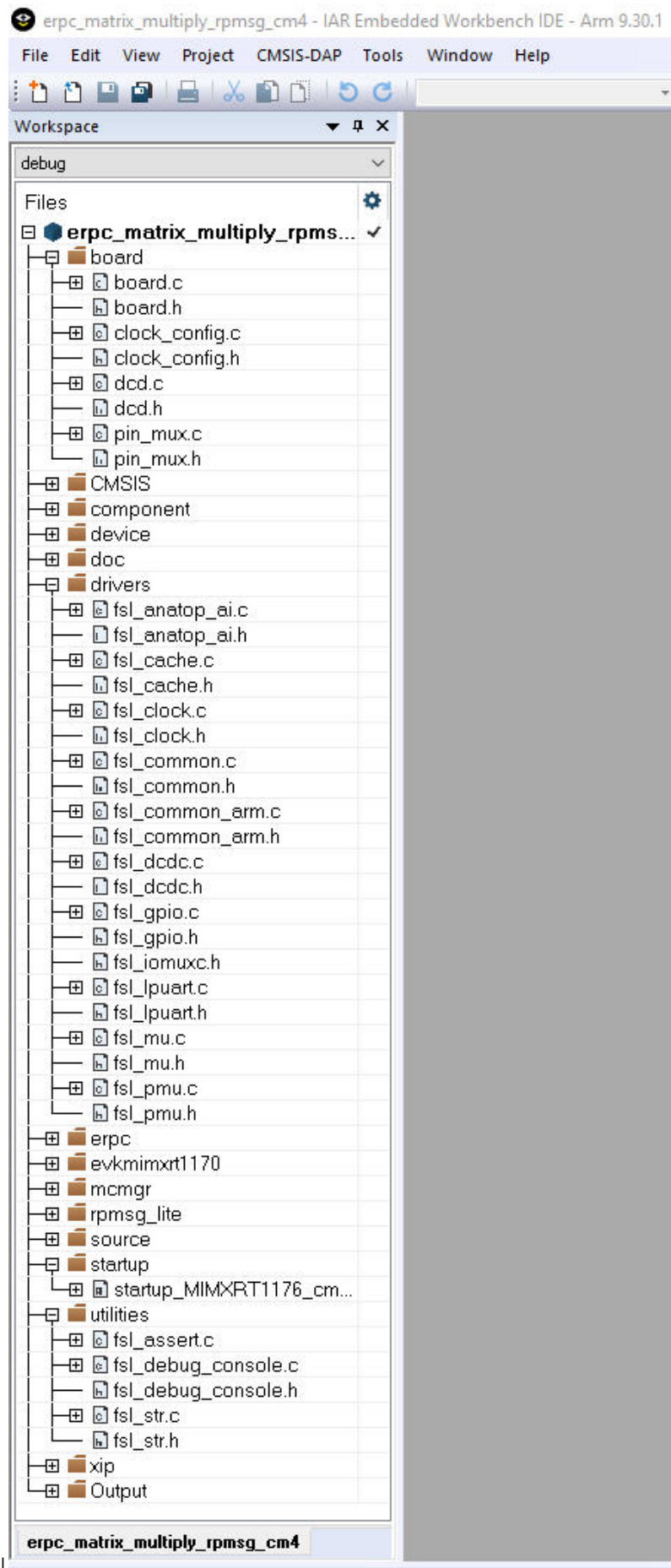
Multicore server application The “Matrix multiply” eRPC server project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmmsg/cm4/iar`

The project files for the eRPC server have the `_cm4` suffix.

Server project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



|

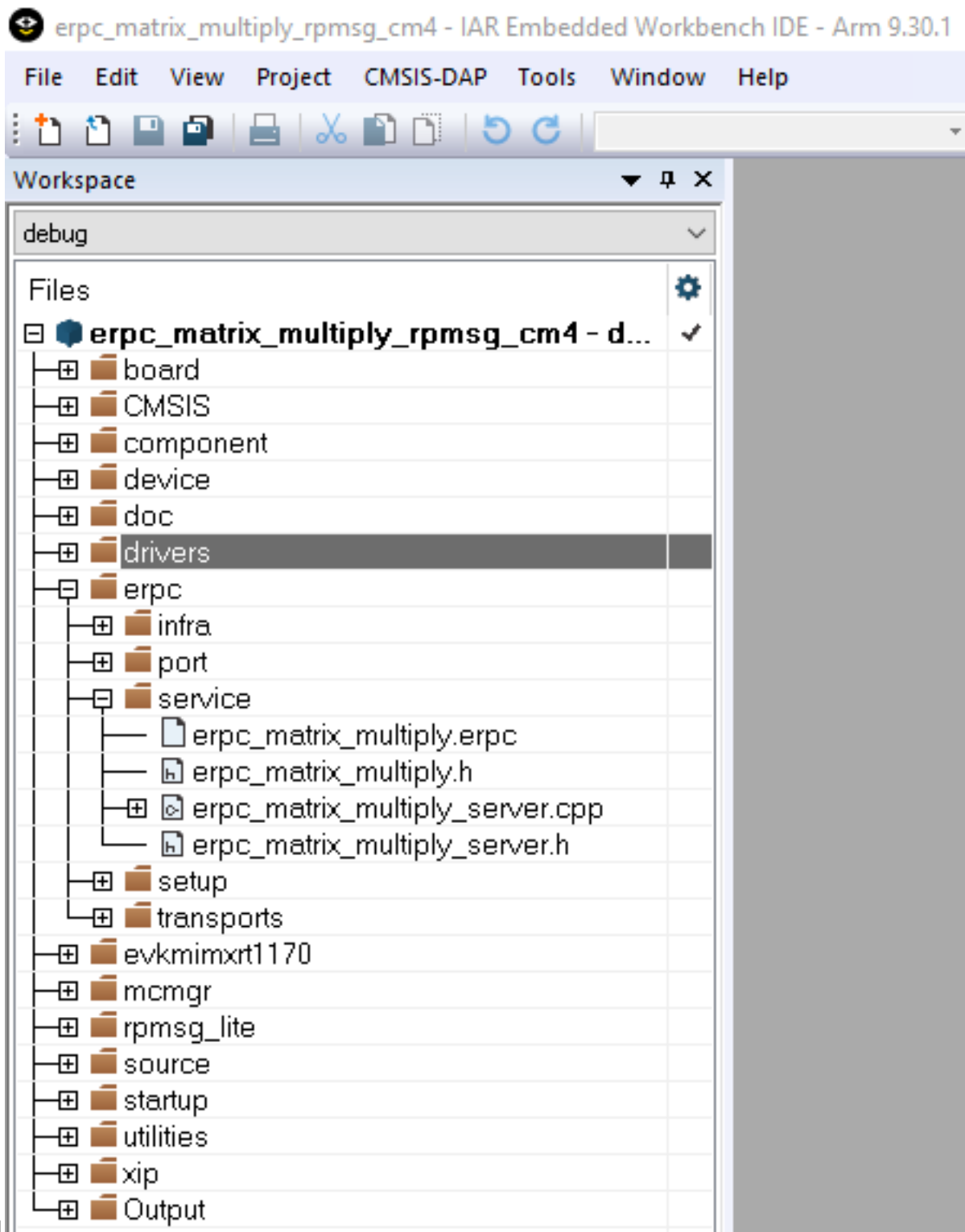
Parent topic: Multicore server application

Server related generated files The server-related generated files are:

- erpc__matric__multiply.h
- erpc__matrix__multiply__server.h
- erpc__matrix__multiply__server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/



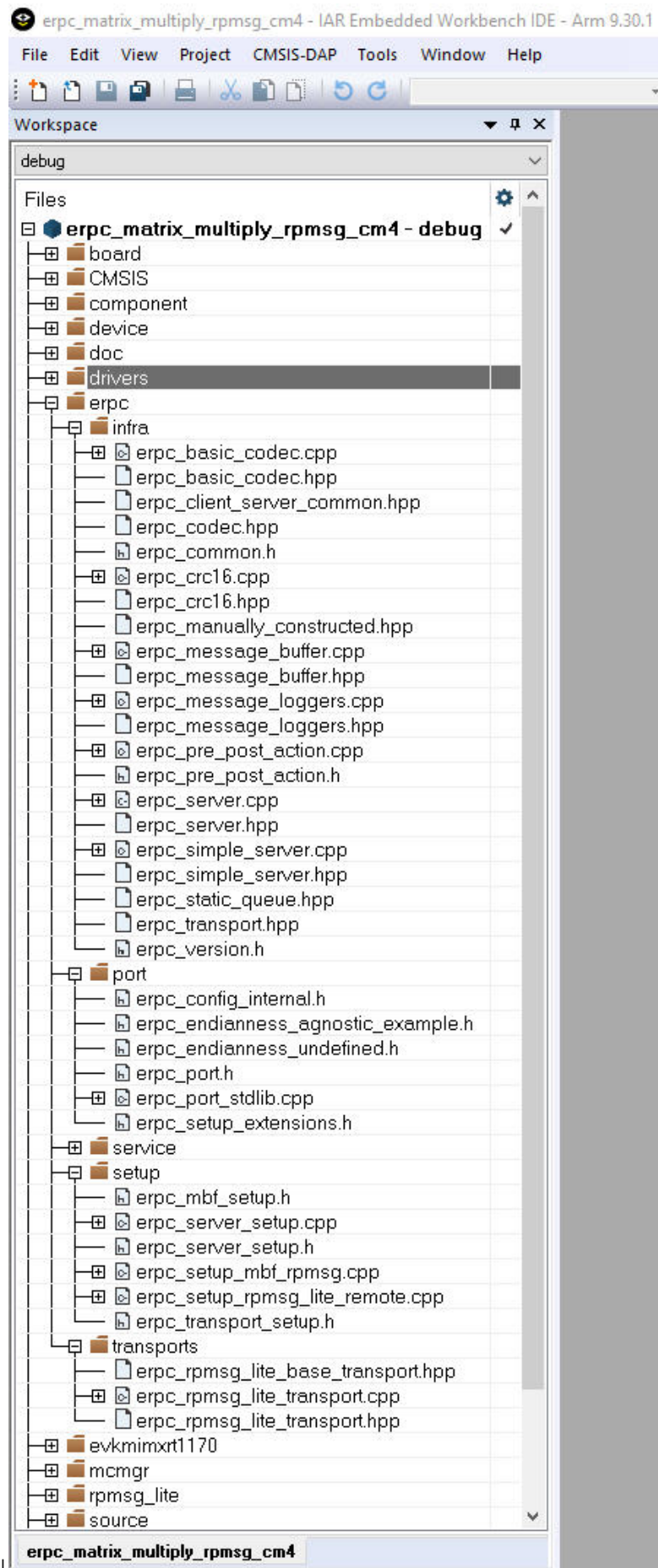
Parent topic: Multicore server application

Server infrastructure files The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
 - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
 - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
 - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
 - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
 - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
 - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
 - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
 - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
 - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
 - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
 - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
 - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
 - RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.



|

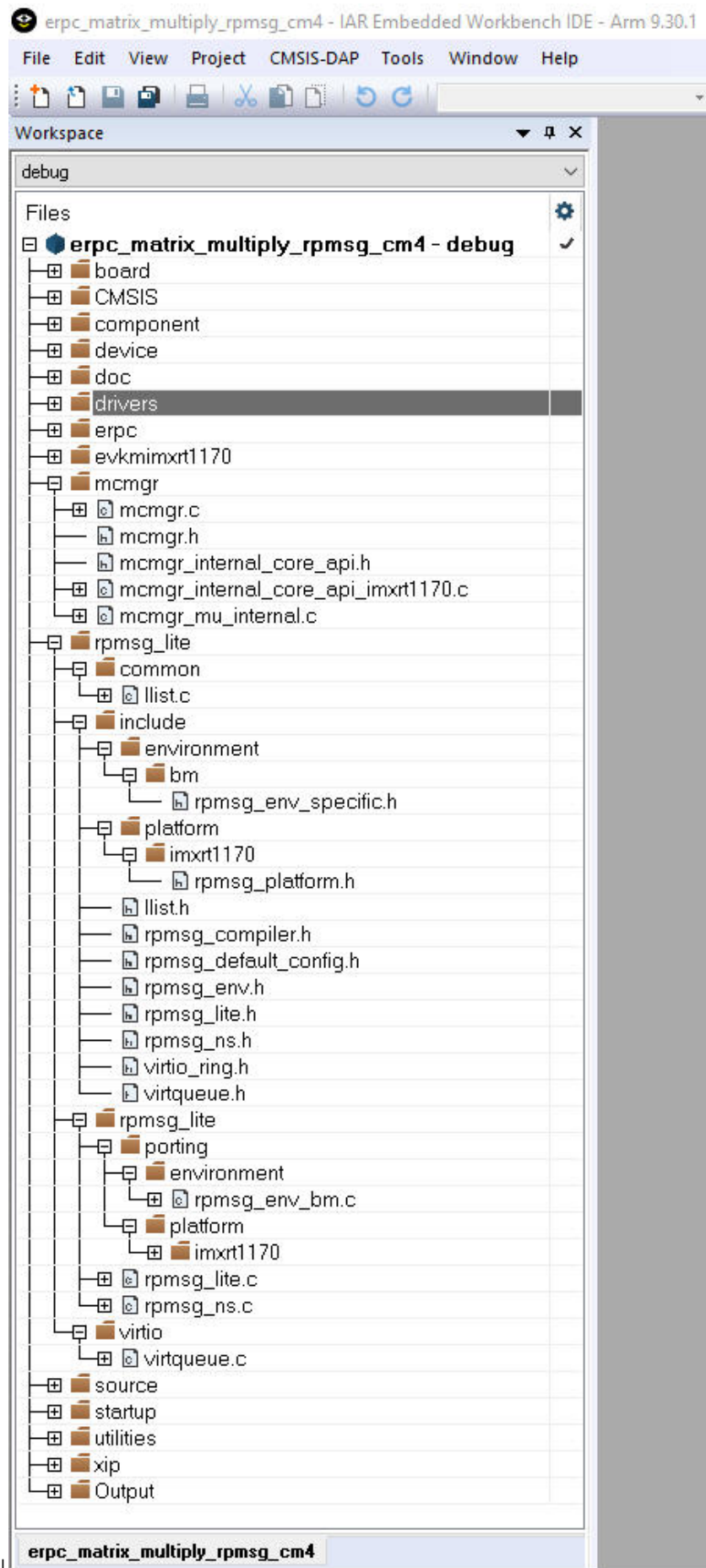
Parent topic: Multicore server application

Server multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



|

Parent topic: Multicore server application

Server user code The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

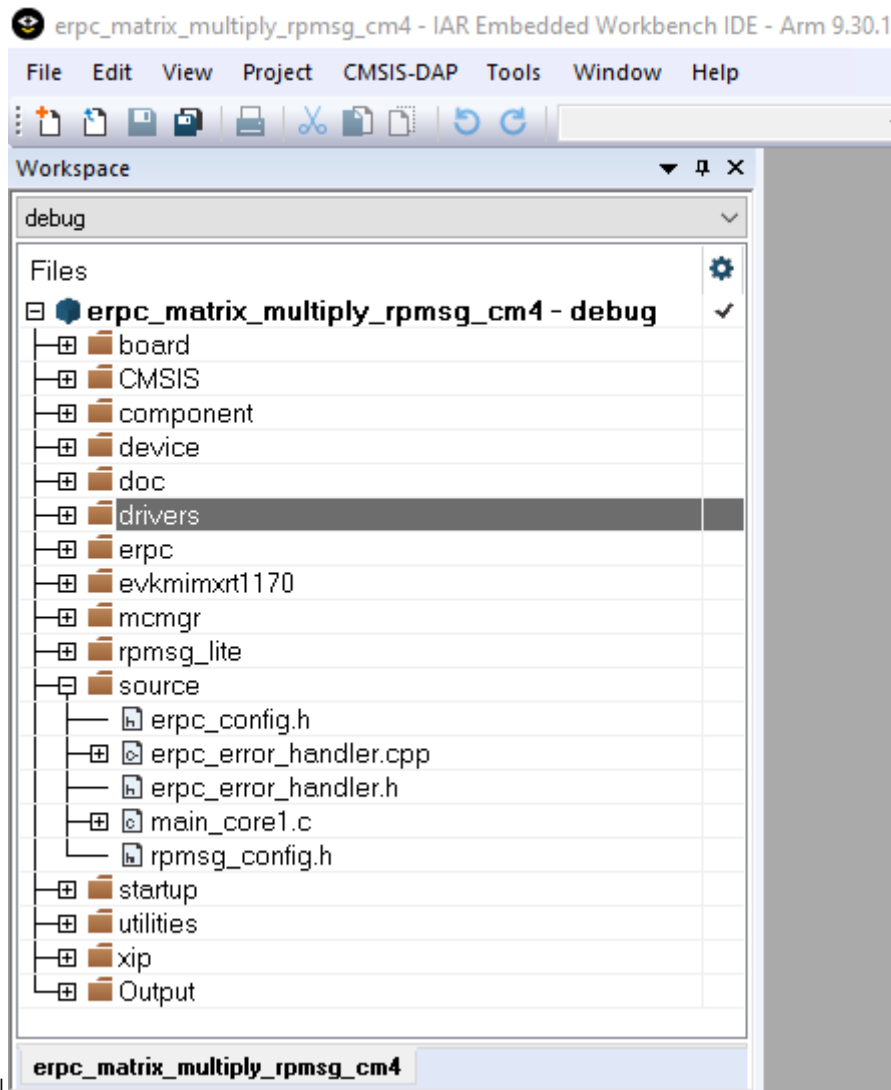
The `main_core1.c` file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
    ...
}
int main()
{
    ...
    /* RPSMsg-Lite transport layer initialization */
    erpc_transport_t transport;
    transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
    ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_rpmsg_init(transport);
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server);
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg` folder.



Parent topic:Multicore server application

Parent topic:[Create an eRPC application](#)

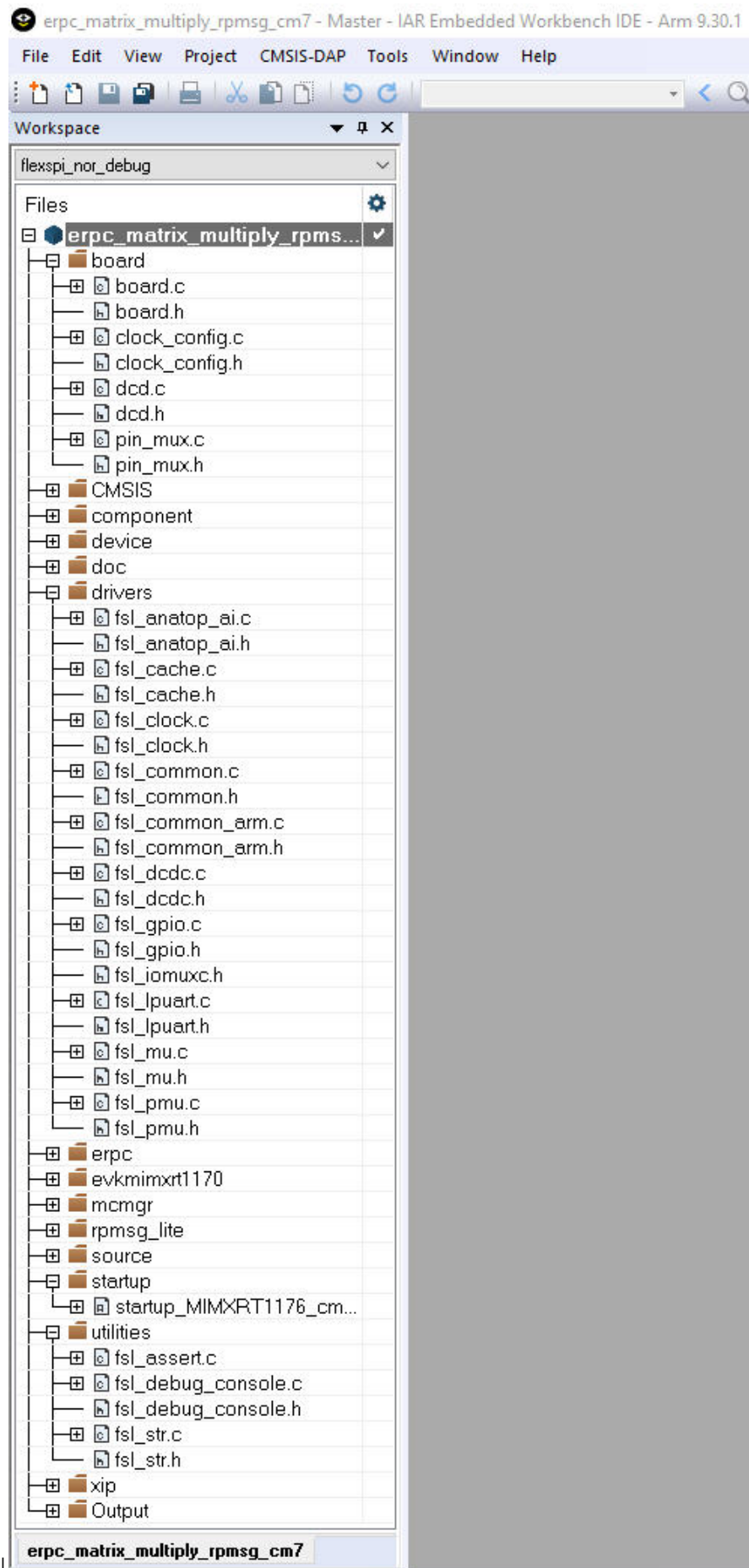
Multicore client application The “Matrix multiply” eRPC client project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar`

Project files for the eRPC client have the `_cm7` suffix.

Client project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



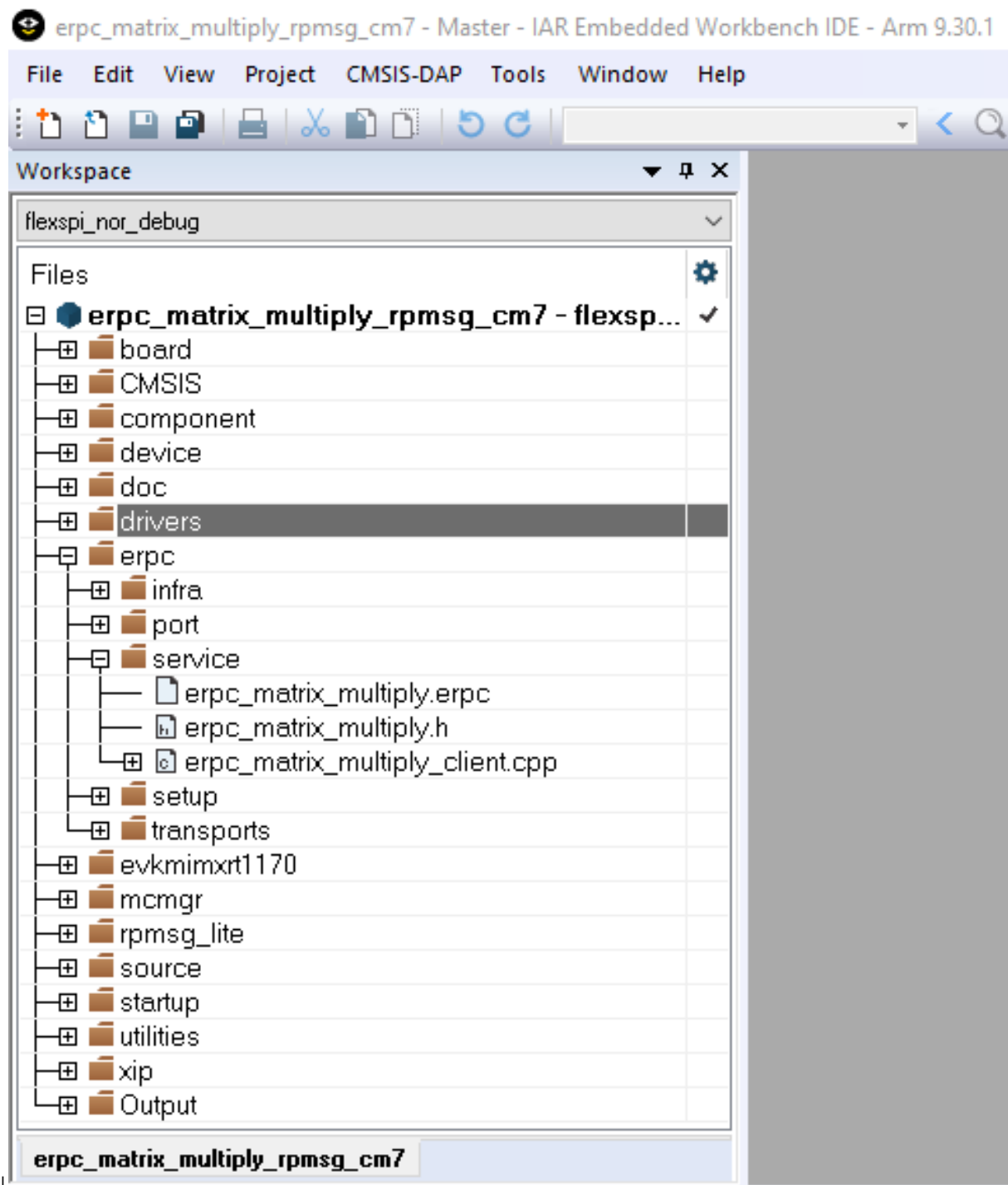
|

Parent topic:Multicore client application

Client-related generated files The client-related generated files are:

- erpc__matric__multiply.h
- erpc__matrix__multiply__client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



Parent topic:Multicore client application

Client infrastructure files The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

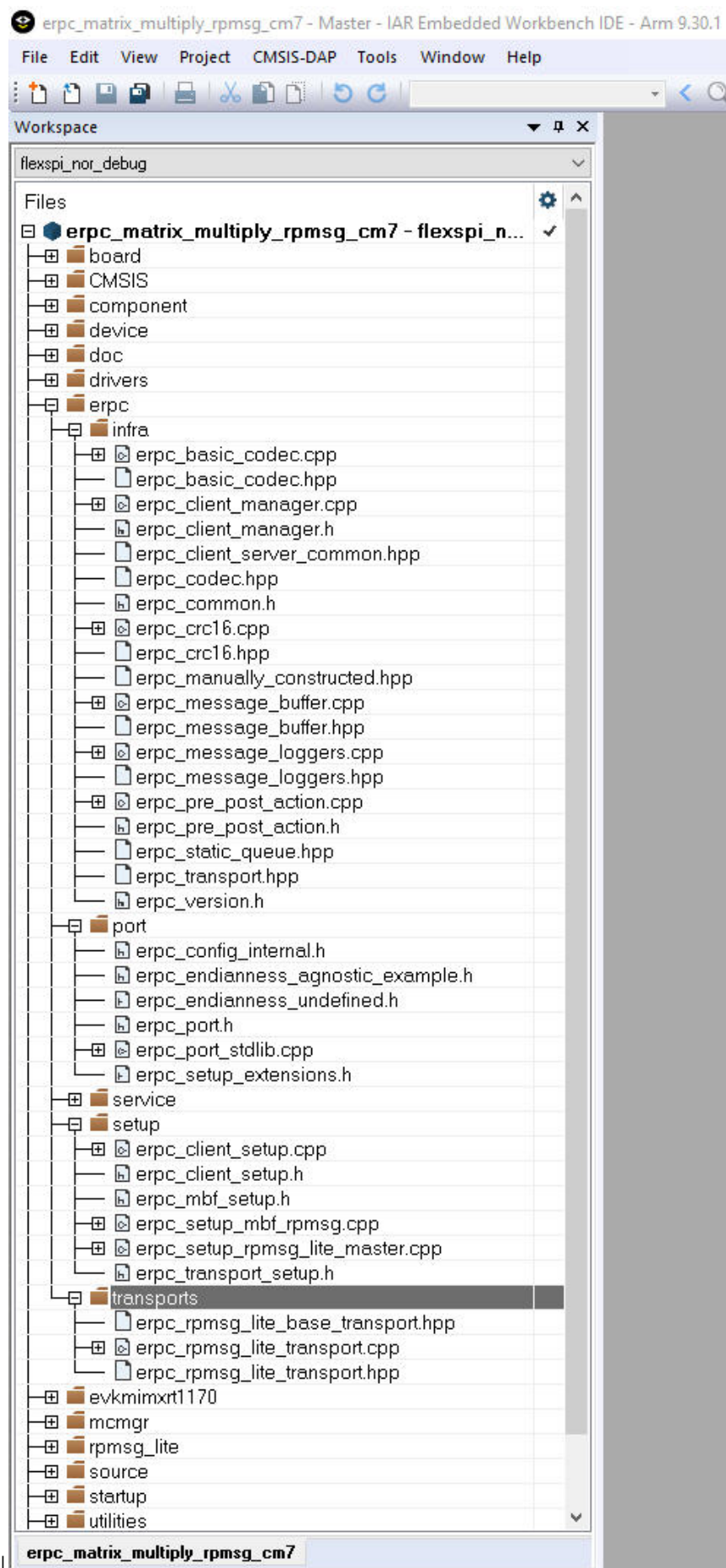
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

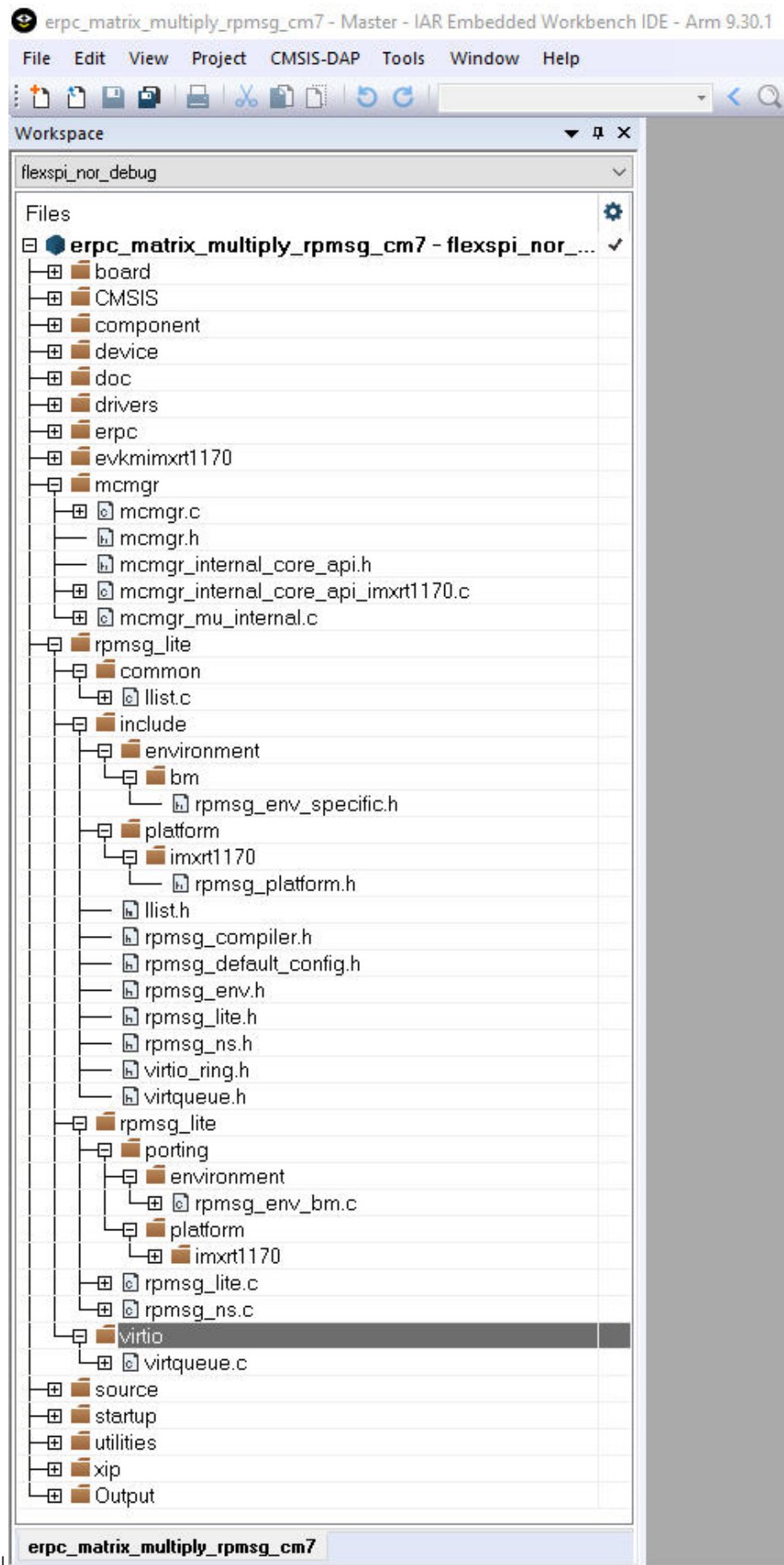
Parent topic: Multicore client application

Client multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/



Parent topic: Multicore client application

Client user code The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

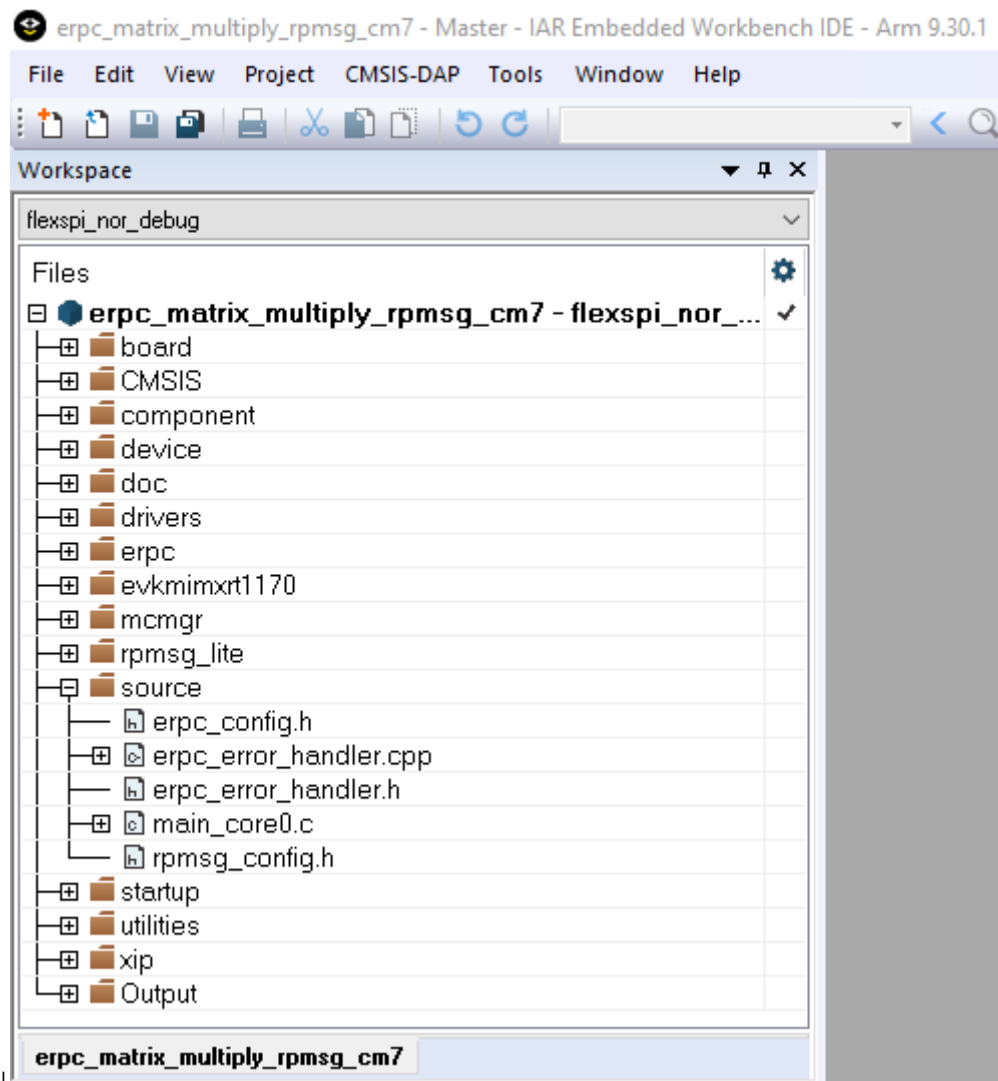
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



Parent topic: Multicore client application

Parent topic: [Create an eRPC application](#)

Multiprocessor server application The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMs-Lite). The RPMs-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transport/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transport/erpc_uart_cmsis_transport.cpp

|

Server user code The server's user code is stored in the main_server.c file, located in the <MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
    ...
}
int main()
{
    ...
    /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver.
    ↪operations */
    erpc_transport_t transport;
    transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_dynamic_init();
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server)
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

Parent topic:Multiprocessor server application

Multiprocessor client application The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPSMsg-Lite). The RPSMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp
<eRPC base directory>/erpc_c/	transports/ erpc_(d)spi_master_transport.hpp
<eRPC base directory>/erpc_c/	transports/ erpc_(d)spi_master_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp
<eRPC base directory>/erpc_c/	transports/erpc_uart_cmsis_transport.hpp
<eRPC base directory>/erpc_c/	transports/erpc_uart_cmsis_transport.cpp

Client user code The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

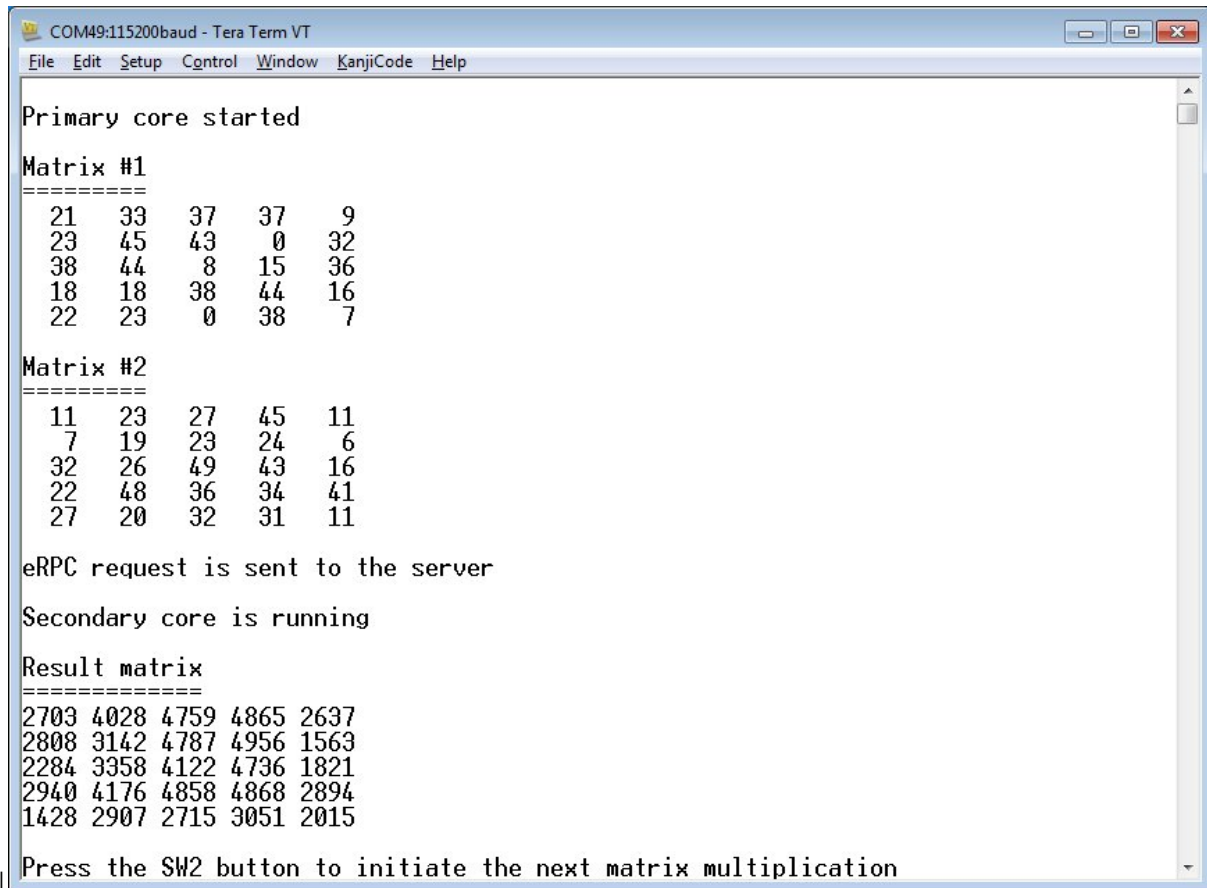
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

Parent topic:Multiprocessor client application

Parent topic:Multiprocessor server application

Parent topic:[Create an eRPC application](#)

Running the eRPC application Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK_install_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Primary core started

Matrix #1
=====
 21  33  37  37   9
 23  45  43   0  32
 38  44   8  15  36
 18  18  38  44  16
 22  23   0  38   7

Matrix #2
=====
 11  23  27  45  11
  7  19  23  24   6
 32  26  49  43  16
 22  48  36  34  41
 27  20  32  31  11

eRPC request is sent to the server

Secondary core is running

Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015

Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

eRPC example This section shows how to create an example eRPC application called “Matrix multiply”, which implements one eRPC function (matrix multiply) with two function parameters (two matrices). The client-side application calls this eRPC function, and the server side performs the multiplication of received matrices. The server side then returns the result.

For example, use the NXP MIMXRT1170-EVK board as the target dual-core platform, and the IAR Embedded Workbench for ARM (EWARM) as the target IDE for developing the eRPC example.

- The primary core (CM7) runs the eRPC client.
- The secondary core (CM4) runs the eRPC server.
- RPMsg-Lite (Remote Processor Messaging Lite) is used as the eRPC transport layer.

The “Matrix multiply” application can be also run in the multi-processor setup. In other words, the eRPC client running on one SoC communicates with the eRPC server that runs on another SoC, utilizing different transport channels. It is possible to run the board-to-PC example (PC as the eRPC server and a board as the eRPC client, and vice versa) and also the board-to-board example. These multiprocessor examples are prepared for selected boards only.

| Multicore application source and project files | `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore/`
| Multiprocessor application source and project files | `<MCUXpressoSDK_install_dir>/boards/<board_name>/multi`
`<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<tr`
| | eRPC source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/` | | RPLite
source files | `<MCUXpressoSDK_install_dir>/middleware/multicore/rplite/`

Designing the eRPC application The matrix multiply application is based on calling single eRPC function that takes 2 two-dimensional arrays as input and returns matrix multiplication results as another 2 two-dimensional array. The IDL file syntax supports arrays with the dimension length set by the number only (in the current eRPC implementation). Because of this, a variable is declared in the IDL dedicated to store information about matrix dimension length, and to allow easy maintenance of the user and server code.

For a simple use of the two-dimensional array, the alias name (new type definition) for this data type has been declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and server applications.

Parent topic: [eRPC example](#)

Creating the IDL file The created IDL file is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/`

The created IDL file contains the following code:

```
program erpc_matrix_multiply
/*! This const defines the matrix size. The value has to be the same as the
Matrix array dimension. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
const int32 matrix_size = 5;
/*! This is the matrix array type. The dimension has to be the same as the
matrix size const. Do not forget to re-generate the erpc code once the
matrix size is changed in the erpc file */
type Matrix = int32[matrix_size][matrix_size];
interface MatrixMultiplyService {
erpcMatrixMultiply(in Matrix matrix1, in Matrix matrix2, out Matrix result_matrix) ->
void
}
```

Details:

- The IDL file starts with the program name (*erpc_matrix_multiply*), and this program name is used in the naming of all generated outputs.
- The declaration and definition of the constant variable named *matrix_size* follows next. The *matrix_size* variable is used for passing information about the length of matrix dimensions to the client/server user code.
- The alias name for the two-dimensional array type (*Matrix*) is declared.
- The interface group *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration *erpcMatrixMultiply*.
- As shown above, the function’s declaration contains three parameters of *Matrix* type: *matrix1* and *matrix2* are input parameters, while *result_matrix* is the output parameter. Additionally, the returned data type is declared as *void*.

When writing the IDL file, the following order of items is recommended:

1. Program name at the top of the IDL file.
2. New data types and constants declarations.
3. Declarations of interfaces and functions at the end of the IDL file.

Parent topic: [eRPC example](#)

Using the eRPC generator tool | Windows OS | <MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64
 | Linux OS | <MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86
 <MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86
 | | Mac OS | <MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Mac |

The files for the “Matrix multiply” example are pre-generated and already a part of the application projects. The following section describes how they have been created.

- The easiest way to create the shim code is to copy the erpcgen application to the same folder where the IDL file (*.erpc) is located; then run the following command:

```
erpcgen <IDL_file>.erpc
```

- In the “Matrix multiply” example, the command should look like:

```
erpcgen erpc_matrix_multiply.erpc
```

Additionally, another method to create the shim code is to execute the eRPC application using input commands:

- “-?”/”—help” – Shows supported commands.
- “-o <filePath>”/”—output<filePath>” – Sets the output directory.

For example,

```
<path_to_erpcgen>/erpcgen -o <path_to_output>  
<path_to_IDL>/<IDL_file_name>.erpc
```

For the “Matrix multiply” example, when the command is executed from the default erpcgen location, it looks like:

```
erpcgen -o
```

```
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service  
../../../../../boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/erpc_matrix_mu
```

In both cases, the following four files are generated into the <MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service folder:

- erpc_matrix_multiply.h
- erpc_matrix_multiply_client.cpp
- erpc_matrix_multiply_server.h
- erpc_matrix_multiply_server.cpp

For multiprocessor examples, the eRPC file and pre-generated files can be found in the <MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_common/erpc_matrix_multiply/service folder.

For Linux OS users:

- Do not forget to set the permissions for the eRPC generator application.
- Run the application as ./erpcgen... instead of as erpcgen

Parent topic: [eRPC example](#)

Create an eRPC application This section describes a generic way to create a client/server eRPC application:

1. **Design the eRPC application:** Decide which data types are sent between applications, and define functions that send/receive this data.
2. **Create the IDL file:** The IDL file contains information about data types and functions used in an eRPC application, and is written in the IDL language.
3. **Use the eRPC generator tool:** This tool takes an IDL file and generates the shim code for the client and the server-side applications.
4. **Create an eRPC application:**
 1. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
 2. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
 3. Add infrastructure files.
 4. Add user code for client and server applications.
 5. Set the client and server project options.
5. **Run the eRPC application:** Run both the server and the client applications. Make sure that the server has been run before the client request was sent.

A specific example follows in the next section.

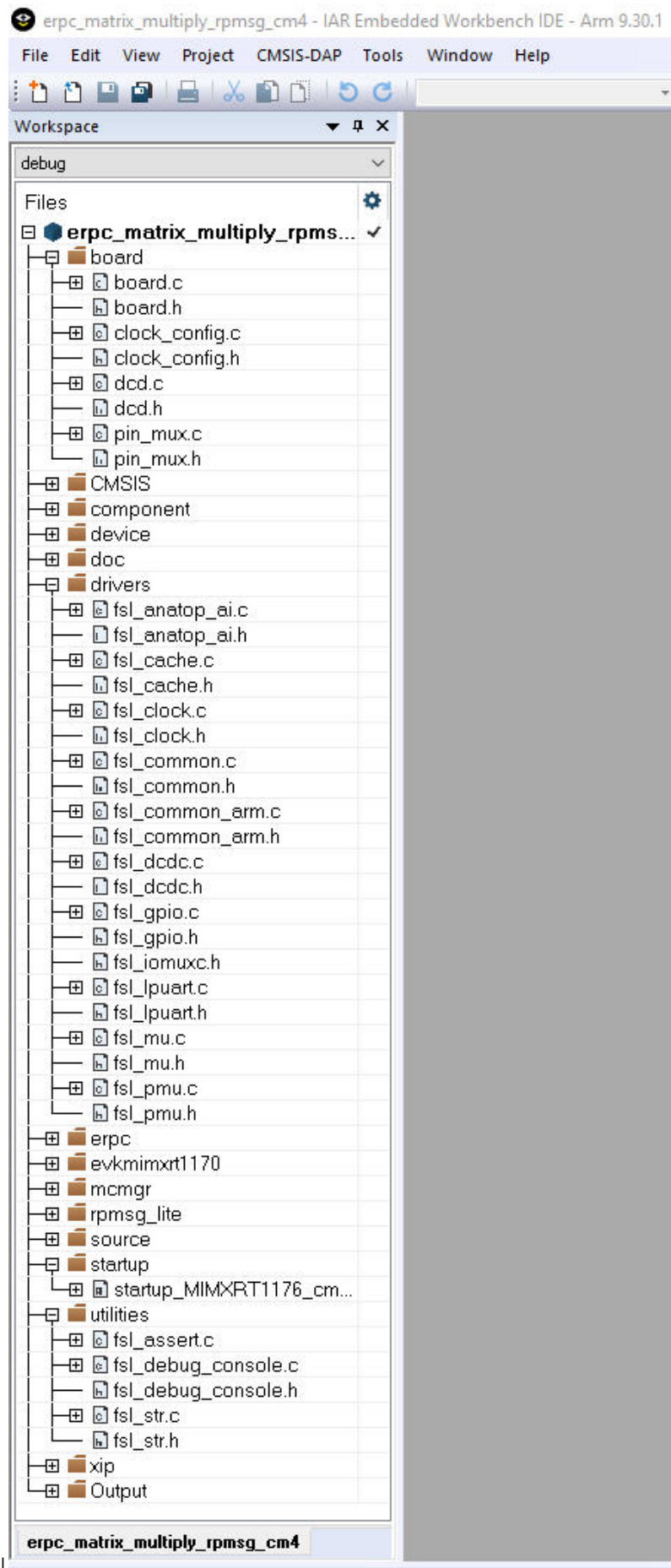
Multicore server application The “Matrix multiply” eRPC server project is located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmmsg/cm4/iar/`

The project files for the eRPC server have the `_cm4` suffix.

Server project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in:

- `<MCUXpressoSDK_install_dir>/devices/<device>`
- `<MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/`



|

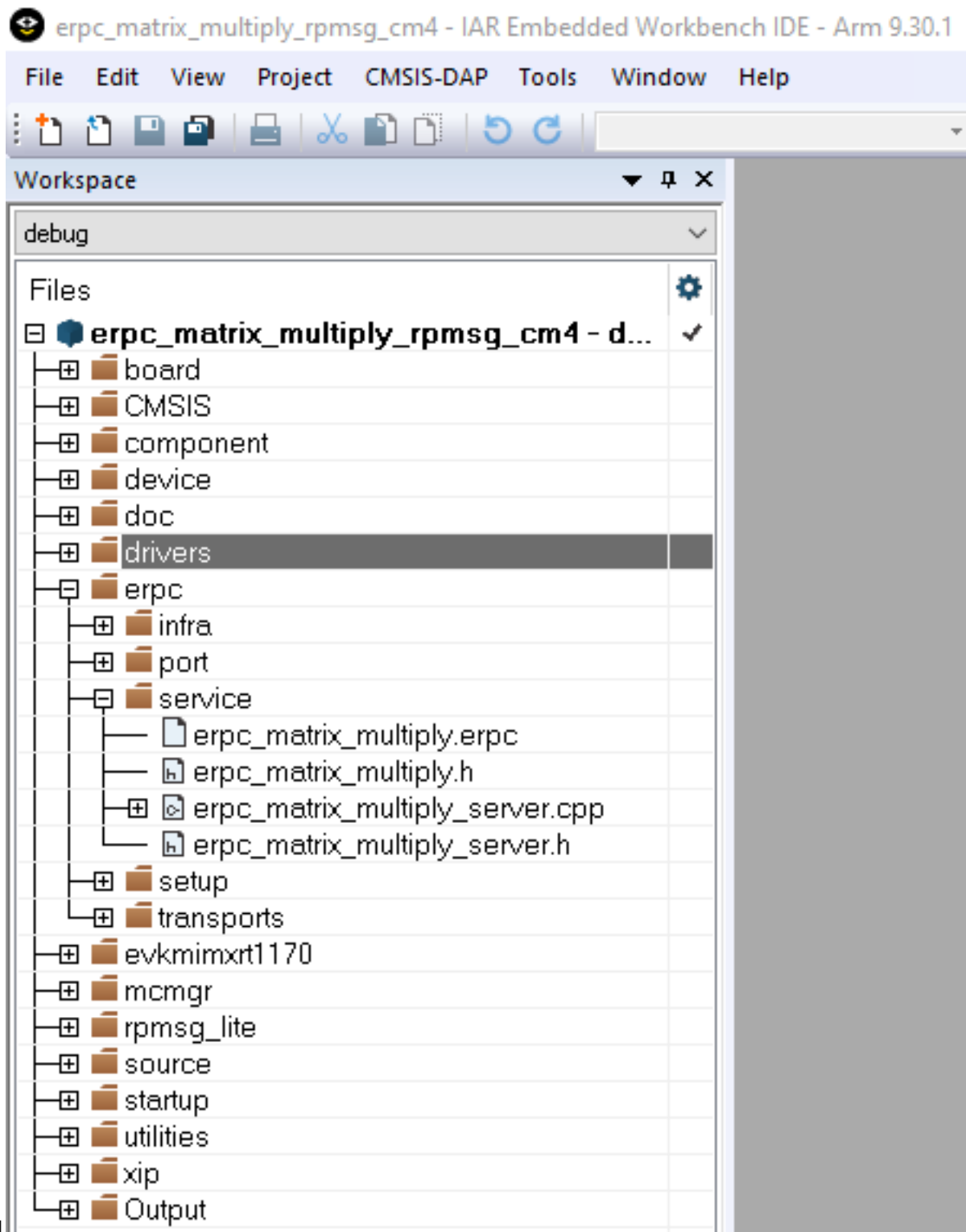
Parent topic: Multicore server application

Server related generated files The server-related generated files are:

- erpc__matric__multiply.h
- erpc__matrix__multiply__server.h
- erpc__matrix__multiply__server.cpp

The server-related generated files contain the shim code for functions and data types declared in the IDL file. These files also contain functions for the identification of client requested functions, data deserialization, calling requested function's implementations, and data serialization and return, if requested by the client. These shim code files can be found in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/



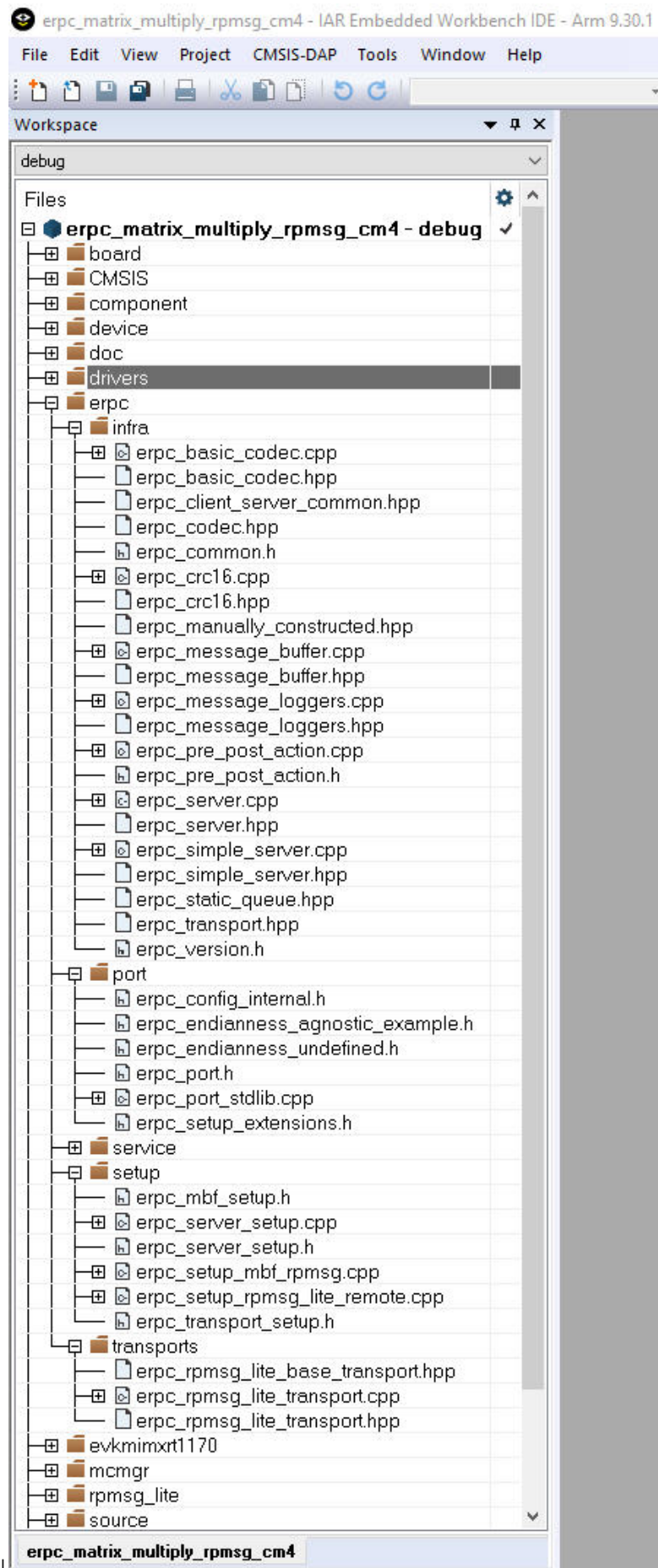
Parent topic: Multicore server application

Server infrastructure files The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
 - Four files, `erpc_server.hpp`, `erpc_server.cpp`, `erpc_simple_server.hpp`, and `erpc_simple_server.cpp`, are used for running the eRPC server on the server-side applications. The simple server is currently the only implementation of the server, and its role is to catch client requests, identify and call requested functions, and send data back when requested.
 - Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
 - The `erpc_common.hpp` file is used for common eRPC definitions, typedefs, and enums.
 - The `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
 - Message buffer files are used for storing serialized data: `erpc_message_buffer.h` and `erpc_message_buffer.cpp`.
 - The `erpc_transport.h` file defines the abstract interface for transport layer.
- The **port** subfolder contains the eRPC porting layer to adapt to different environments.
 - `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
 - `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
 - `erpc_config_internal.h` internal erpc configuration file.
- The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.
 - The `erpc_server_setup.h` and `erpc_server_setup.cpp` files need to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
 - The `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_remote.cpp` files need to be added into the project in order to allow the C-wrapped function for transport layer setup.
 - The `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files need to be added into the project in order to allow message buffer factory usage.
- The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions in the setup folder.
 - RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files need to be added into the server project.



|

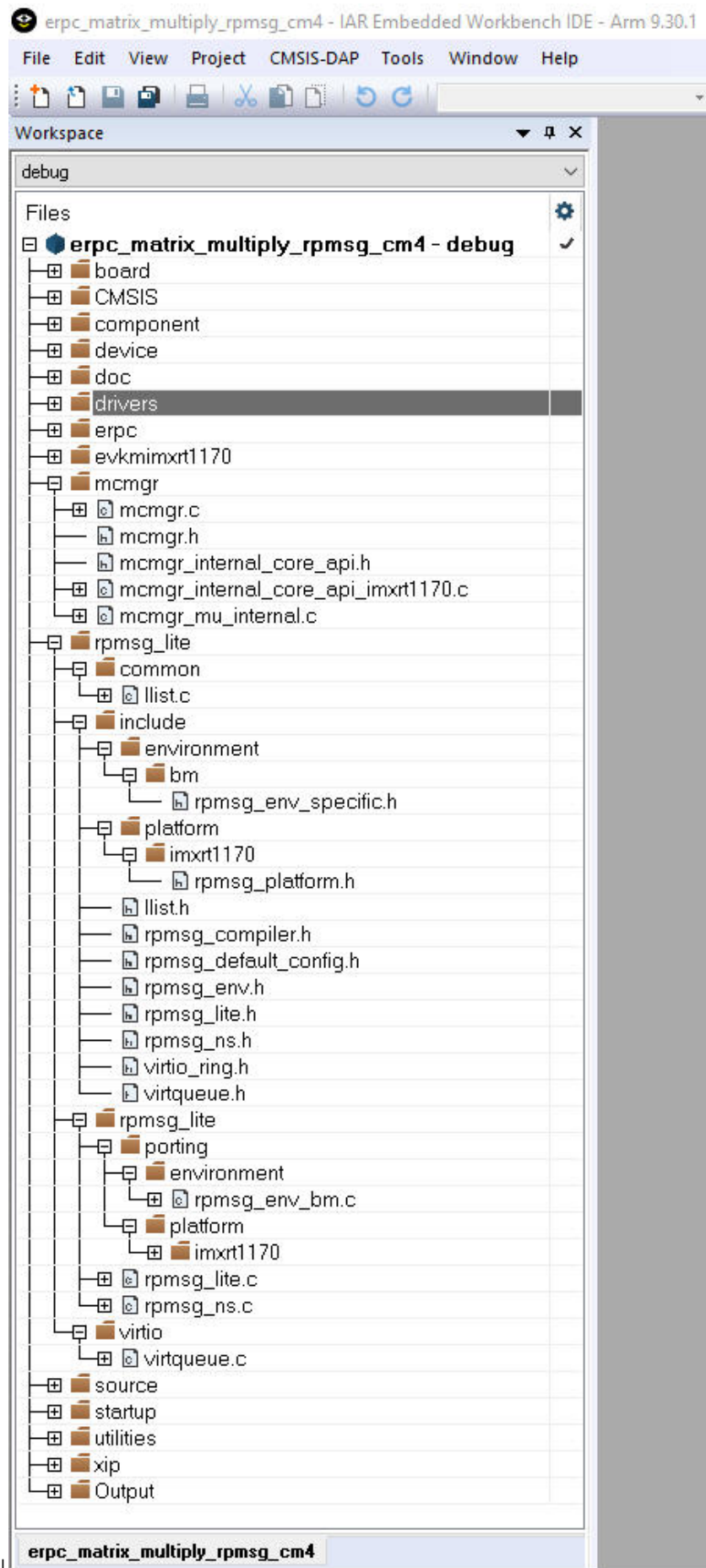
Parent topic: Multicore server application

Server multicore infrastructure files Because of the RPSMsg-Lite (transport layer), it is also necessary to include RPSMsg-Lite related files, which are in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/rpsmsg_lite/`

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/`



|

Parent topic: Multicore server application

Server user code The server's user code is stored in the `main_core1.c` file, located in the following folder:

`<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4`

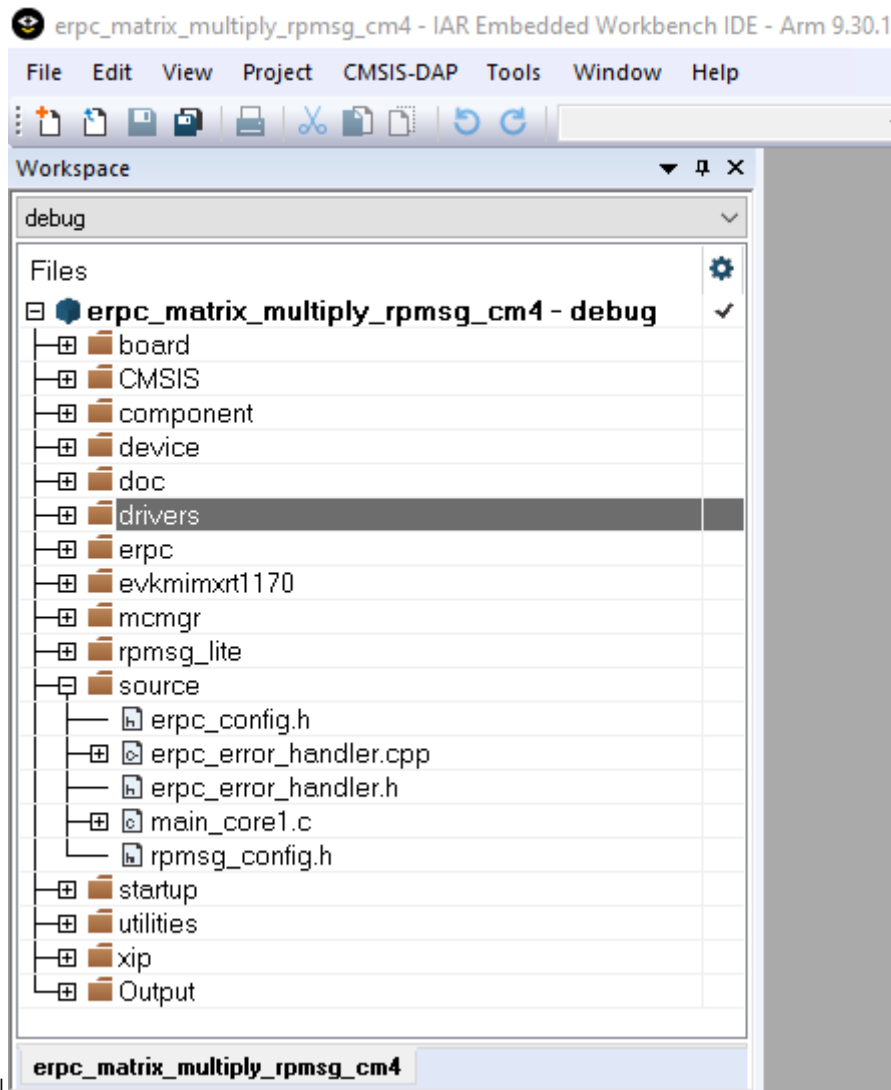
The `main_core1.c` file contains two functions:

- The **main()** function contains the code for the target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server waits for client's requests in the while loop.
- The **erpcMatrixMultiply()** function is the user implementation of the eRPC function defined in the IDL file.
- There is the possibility to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in the `erpc_error_handler.h` and `erpc_error_handler.cpp` files.

The eRPC-relevant code is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(const Matrix *matrix1, const Matrix *matrix2, Matrix *result_matrix)
{
    ...
}
int main()
{
    ...
    /* RPSMsg-Lite transport layer initialization */
    erpc_transport_t transport;
    transport = erpc_transport_rpmsg_lite_remote_init(src, dst, (void*)startupData,
    ERPC_TRANSPORT_RPMSG_LITE_LINK_ID, SignalReady, NULL);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_rpmsg_init(transport);
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server);
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the <MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/ erpc_matrix_multiply_rpmsg folder.



Parent topic:Multicore server application

Parent topic:[Create an eRPC application](#)

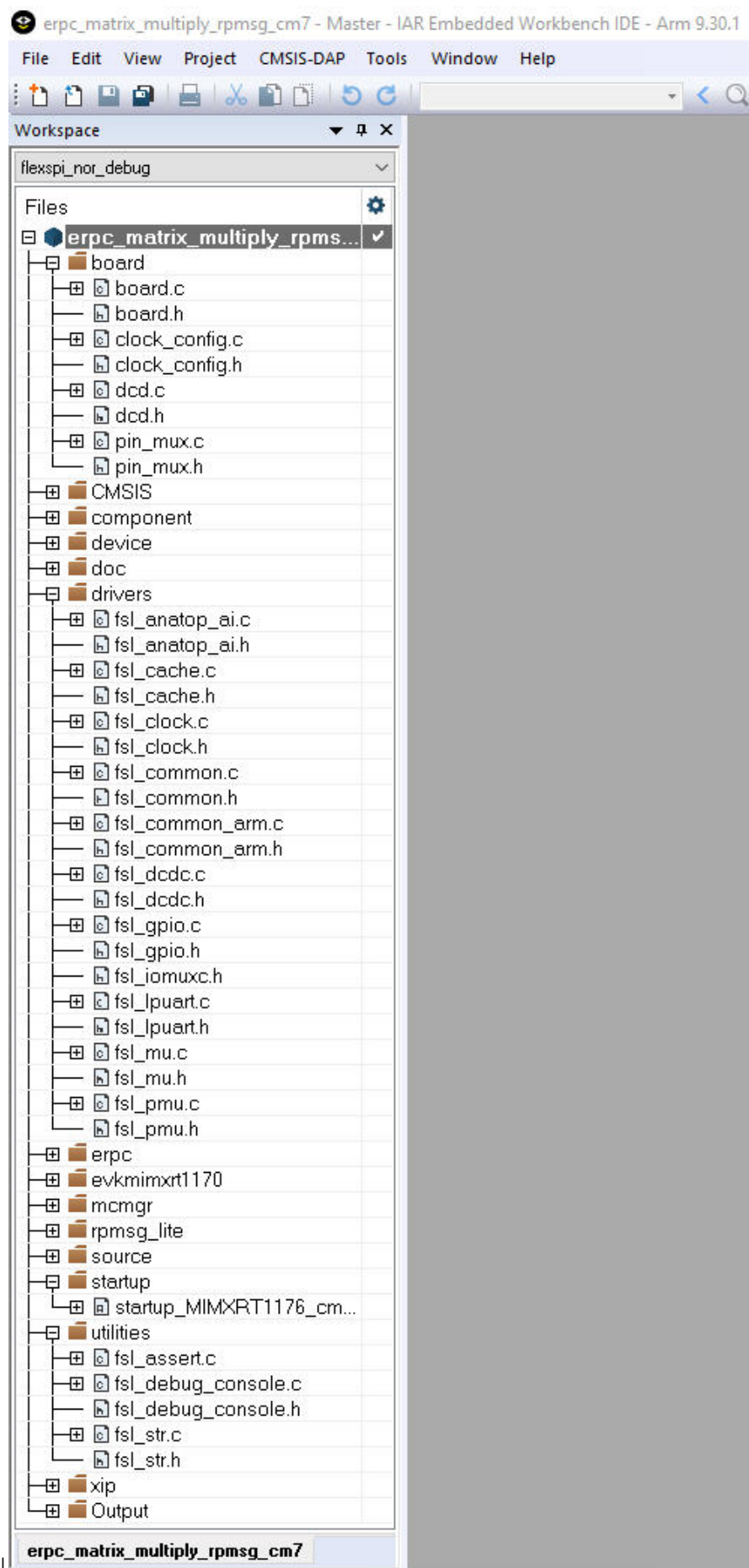
Multicore client application The “Matrix multiply” eRPC client project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm7/iar

Project files for the eRPC client have the _cm7 suffix.

Client project basic source files The startup files, board-related settings, peripheral drivers, and utilities belong to the basic project source files and form the skeleton of all MCUXpresso SDK applications. These source files are located in the following folders:

- <MCUXpressoSDK_install_dir>/devices/<device>
- <MCUXpressoSDK_install_dir>/boards/<board_name>/multicore_examples/<example_name>/



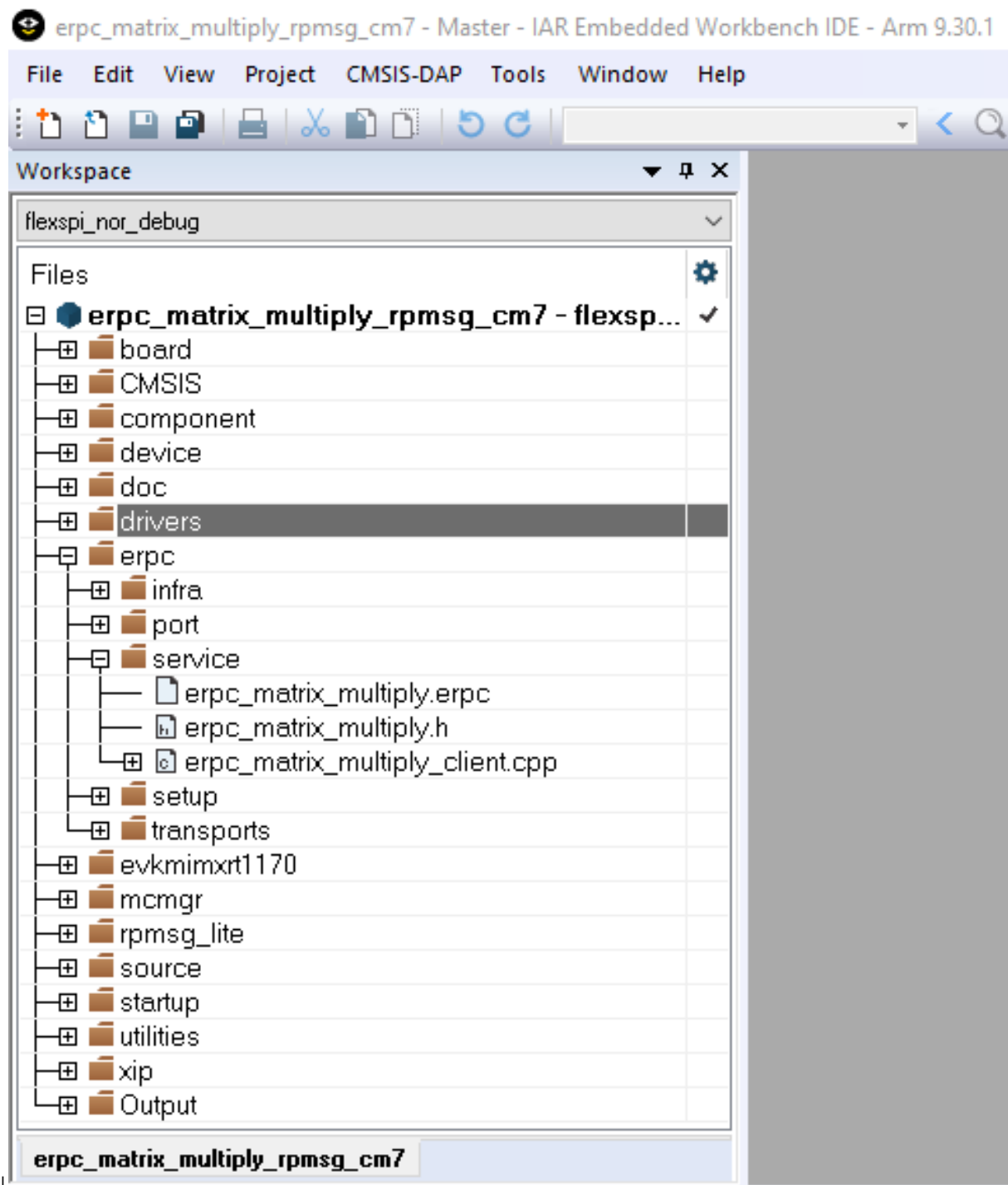
|

Parent topic: Multicore client application

Client-related generated files The client-related generated files are:

- erpc__matric__multiply.h
- erpc__matrix__multiply__client.cpp

These files contain the shim code for the functions and data types declared in the IDL file. These functions also call methods for codec initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures (if return values are expected). These shim code files can be found in the `<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_common/erpc_matrix_multiply/service/` folder.



Parent topic: Multicore client application

Client infrastructure files The eRPC infrastructure files are located in the following folder:

`<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/erpc_c`

The **erpc_c** folder contains files for creating eRPC client and server applications in the C/C++ language. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.

- Two files, `erpc_client_manager.h` and `erpc_client_manager.cpp`, are used for managing the client-side application. The main purpose of the client files is to create, perform, and release eRPC requests.
- Three files (`erpc_codec.hpp`, `erpc_basic_codec.hpp`, and `erpc_basic_codec.cpp`) are used for codecs. Currently, the basic codec is the initial and only implementation of the codecs.
- `erpc_common.h` file is used for common eRPC definitions, typedefs, and enums.
- `erpc_manually_constructed.hpp` file is used for allocating static storage for the used objects.
- Message buffer files are used for storing serialized data: `erpc_message_buffer.hpp` and `erpc_message_buffer.cpp`.
- `erpc_transport.hpp` file defines the abstract interface for transport layer.

The **port** subfolder contains the eRPC porting layer to adapt to different environments.

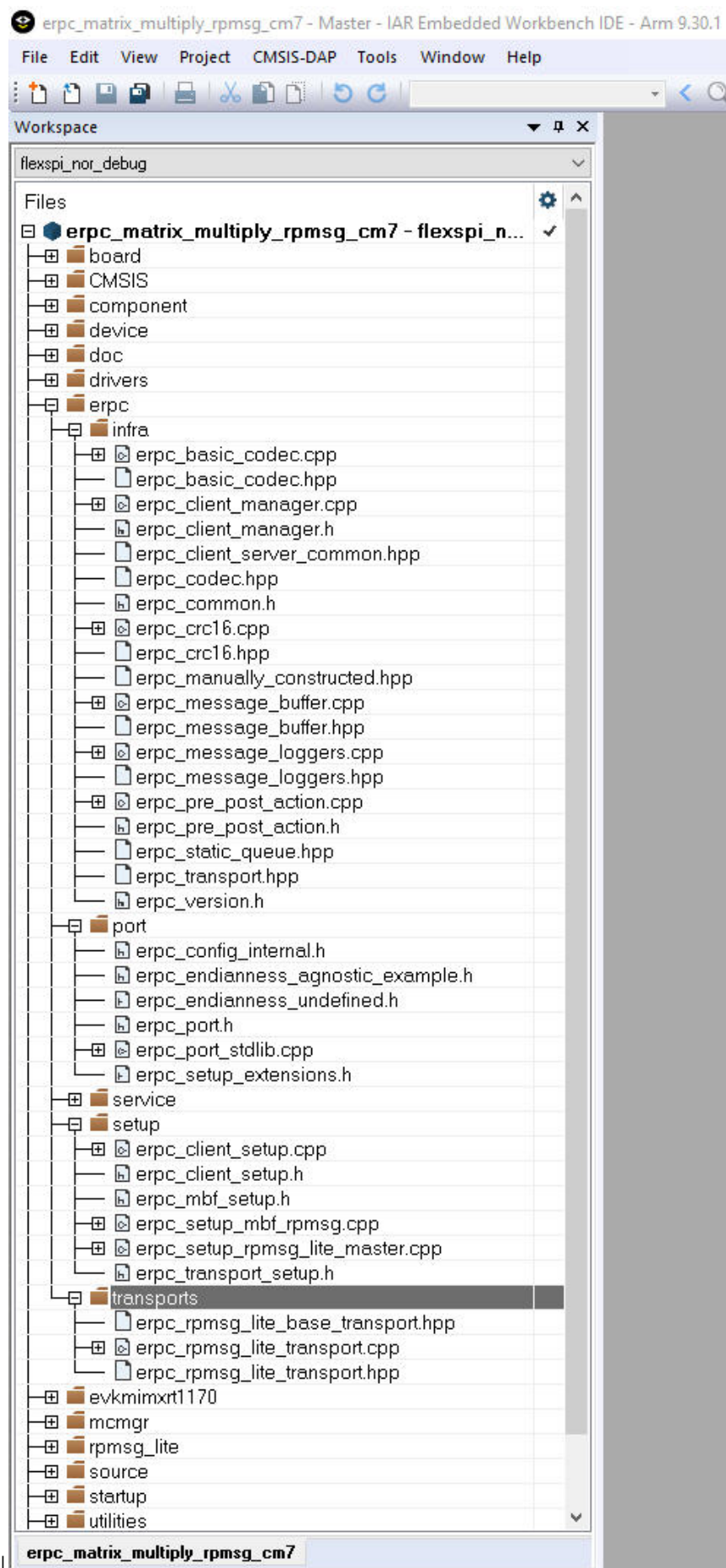
- `erpc_port.h` file contains definition of `erpc_malloc()` and `erpc_free()` functions.
- `erpc_port_stdlib.cpp` file ensures adaptation to `stdlib`.
- `erpc_config_internal.h` internal eRPC configuration file.

The **setup** subfolder contains a set of plain C APIs that wrap the C++ infrastructure, providing client and server init and deinit routines that greatly simplify eRPC usage in C-based projects. No knowledge of C++ is required to use these APIs.

- `erpc_client_setup.h` and `erpc_client_setup.cpp` files needs to be added into the “Matrix multiply” example project to demonstrate the use of C-wrapped functions in this example.
- `erpc_transport_setup.h` and `erpc_setup_rpmsg_lite_master.cpp` files needs to be added into the project in order to allow C-wrapped function for transport layer setup.
- `erpc_mbf_setup.h` and `erpc_setup_mbf_rpmsg.cpp` files needs to be added into the project in order to allow message buffer factory usage.

The **transports** subfolder contains transport classes for the different methods of communication supported by eRPC. Some transports are applicable only to host PCs, while others are applicable only to embedded or multicore systems. Most transports have corresponding client and server setup functions, in the setup folder.

- RPMsg-Lite is used as the transport layer for the communication between cores, `erpc_rpmsg_lite_base_transport.hpp`, `erpc_rpmsg_lite_transport.hpp`, and `erpc_rpmsg_lite_transport.cpp` files needs to be added into the client project.



|

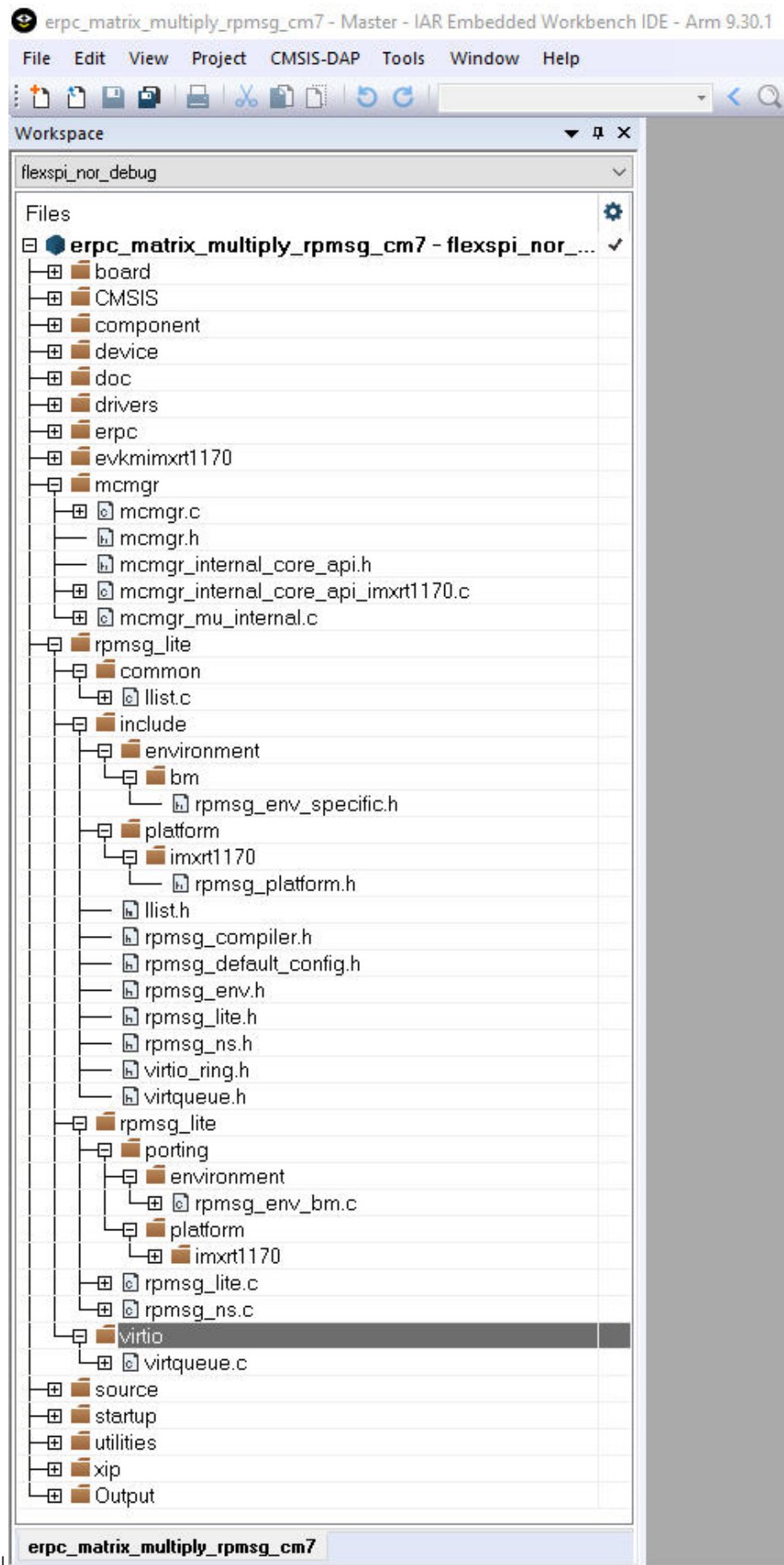
Parent topic: Multicore client application

Client multicore infrastructure files Because of the RPMsg-Lite (transport layer), it is also necessary to include RPMsg-Lite related files, which are in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/

The multicore example applications also use the Multicore Manager software library to control the secondary core startup and shutdown. These source files are located in the following folder:

<MCUXpressoSDK_install_dir>/middleware/multicore/mcmgr/



Parent topic:Multicore client application

Client user code The client's user code is stored in the main_core0.c file, located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_example/erpc_matrix_multiply_rpmsg/cm7

The main_core0.c file contains the code for target board and eRPC initialization.

- After initialization, the secondary core is released from reset.
- When the secondary core is ready, the primary core initializes two matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

It is possible to write the application-specific eRPC error handler. The eRPC error handler of the matrix multiply application is implemented in erpc_error_handler.h and erpc_error_handler.cpp files.

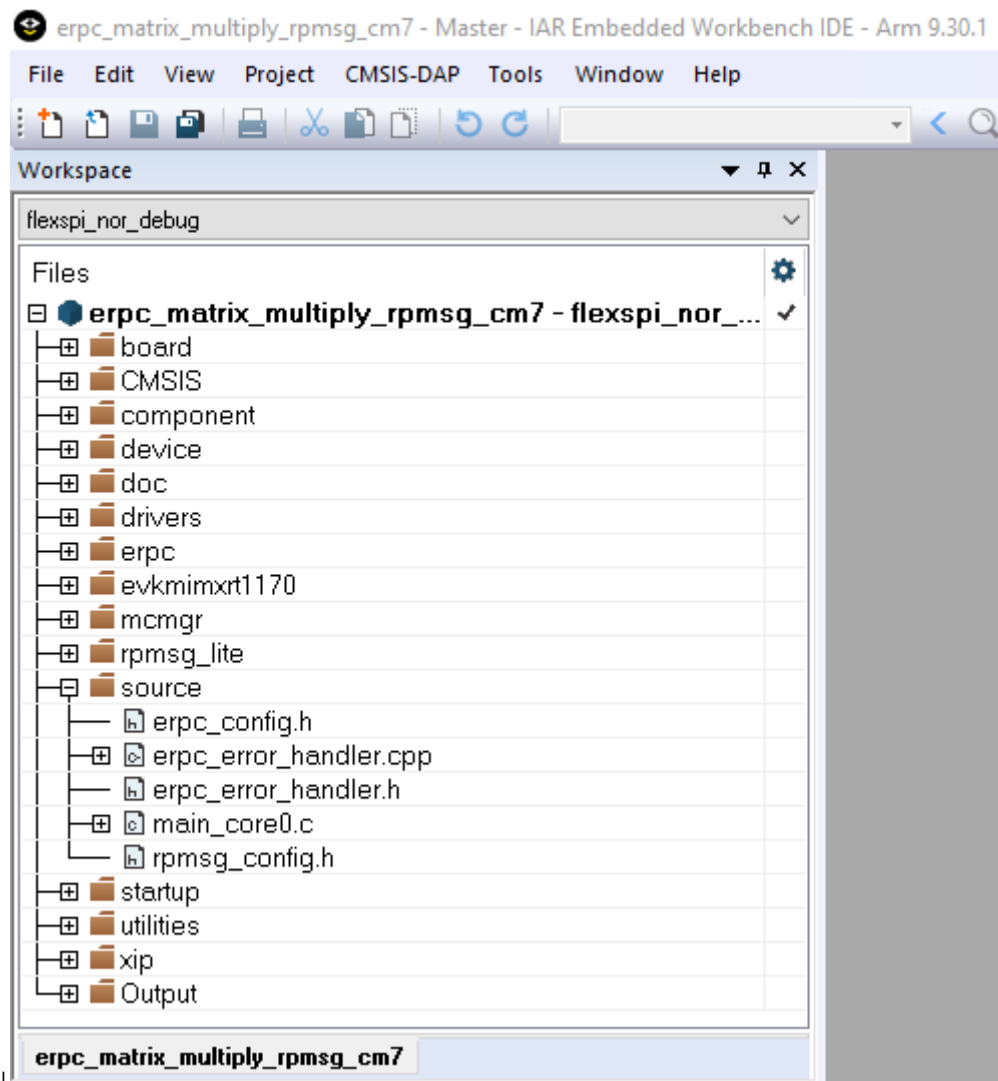
The matrix multiplication can be issued repeatedly, when pressing a software board button.

The eRPC-relevant code is captured in the following code snippet:

```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst,
ERPC_TRANSPORT_RPMSG_LITE_LINK_ID);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_rpmsg_init(transport);
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport, message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

Except for the application main file, there are configuration files for the RPMsg-Lite (rpmsg_config.h) and eRPC (erpc_config.h), located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg



Parent topic:Multicore client application

Parent topic:[Create an eRPC application](#)

Multiprocessor server application The “Matrix multiply” eRPC server project for multiprocessor applications is located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>` folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires server-related generated files (server shim code), server infrastructure files, and the server user code. There is no need for server multicore infrastructure files (MCMGR and RPMs-Lite). The RPMs-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_slave.cpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.hpp
	<eRPC base directory>/erpc_c/transports/erpc_(d)spi_slave_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp

<eRPC base directory>/erpc_c/transport/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transport/erpc_uart_cmsis_transport.cpp

|

Server user code The server's user code is stored in the main_server.c file, located in the <MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_server_matrix_multiply_<transport_layer>/ folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

```
/* erpcMatrixMultiply function user implementation */
void erpcMatrixMultiply(Matrix matrix1, Matrix matrix2, Matrix result_matrix)
{
    ...
}
int main()
{
    ...
    /* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
    ↪operations */
    erpc_transport_t transport;
    transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
    ...
    /* MessageBufferFactory initialization */
    erpc_mbf_t message_buffer_factory;
    message_buffer_factory = erpc_mbf_dynamic_init();
    ...
    /* eRPC server side initialization */
    erpc_server_t server;
    server = erpc_server_init(transport, message_buffer_factory);
    ...
    /* Adding the service to the server */
    erpc_service_t service = create_MatrixMultiplyService_service();
    erpc_add_service_to_server(server, service);
    ...
    while (1)
    {
        /* Process eRPC requests */
        erpc_status_t status = erpc_server_poll(server)
        /* handle error status */
        if (status != kErpcStatus_Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            ...
        }
        ...
    }
}
```

Parent topic:Multiprocessor server application

Multiprocessor client application The “Matrix multiply” eRPC client project for multiprocessor applications is located in the <MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/iar/ folder.

Most of the multiprocessor application setup is the same as for the multicore application. The multiprocessor server application requires client-related generated files (server shim code),

client infrastructure files, and the client user code. There is no need for client multicore infrastructure files (MCMGR and RPSMsg-Lite). The RPSMsg-Lite transport layer is replaced either by SPI or UART transports. The following table shows the required transport-related files per each transport type.

SPI	<eRPC base directory>/erpc_c/setup/erpc_setup_(d)spi_master.cpp
<eRPC base directory>/erpc_c/	transports/ erpc_(d)spi_master_transport.hpp
<eRPC base directory>/erpc_c/	transports/ erpc_(d)spi_master_transport.cpp
UART	<eRPC base directory>/erpc_c/setup/erpc_setup_uart_cmsis.cpp
<eRPC base directory>/erpc_c/	transports/erpc_uart_cmsis_transport.hpp
<eRPC base directory>/erpc_c/	transports/erpc_uart_cmsis_transport.cpp

Client user code The client's user code is stored in the `main_client.c` file, located in the `<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_examples/erpc_client_matrix_multiply_<transport_layer>/` folder.

The eRPC-relevant code with UART as a transport is captured in the following code snippet:

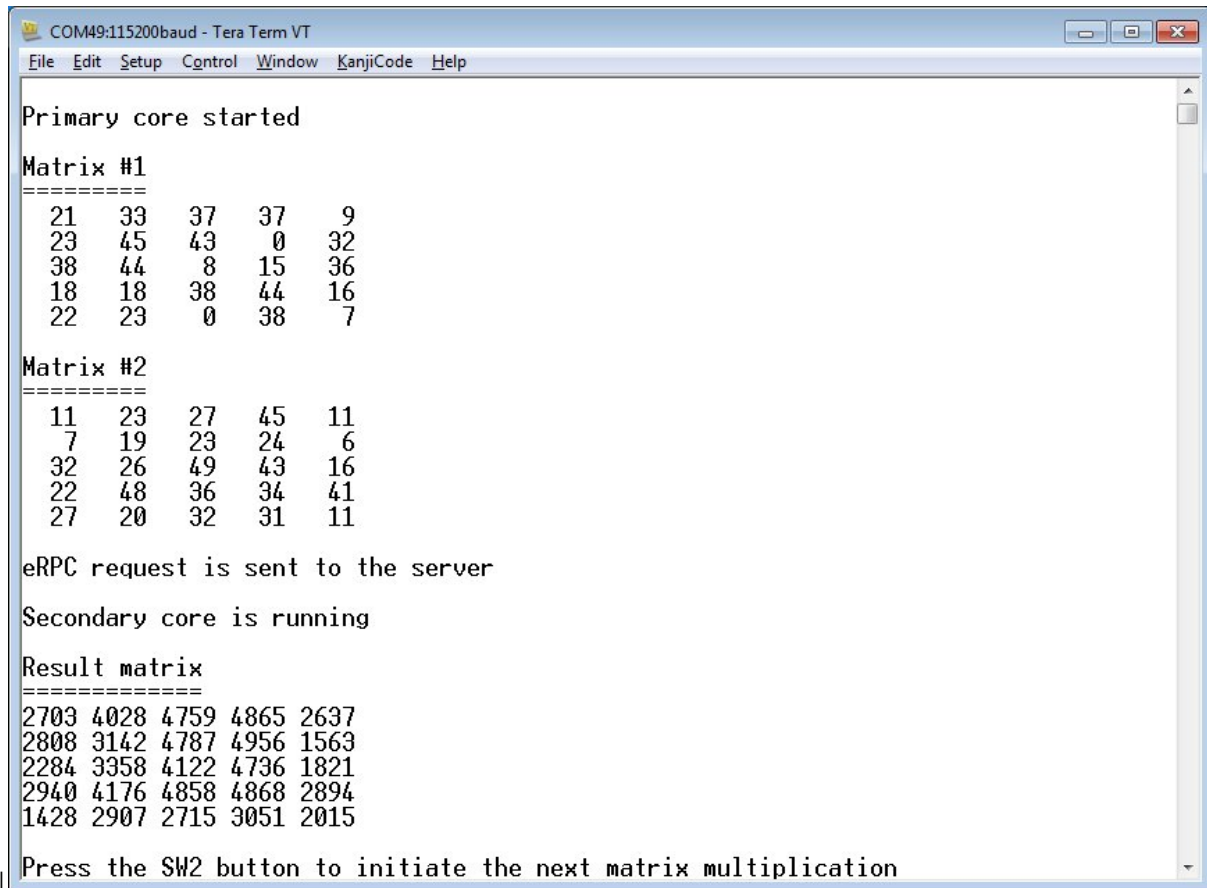
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver
↳operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&ERPC_DEMO_UART);
...
/* MessageBufferFactory initialization */
erpc_mbf_t message_buffer_factory;
message_buffer_factory = erpc_mbf_dynamic_init();
...
/* eRPC client side initialization */
erpc_client_t client;
client = erpc_client_init(transport,message_buffer_factory);
...
/* Set default error handler */
erpc_client_set_error_handler(client, erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply(matrix1, matrix2, result_matrix);
    ...
    /* Check if some error occurred in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

Parent topic:Multiprocessor client application

Parent topic:Multiprocessor server application

Parent topic:[Create an eRPC application](#)

Running the eRPC application Follow the instructions in *Getting Started with MCUXpresso SDK* (document MCUXSDKGSUG) (located in the <MCUXpressoSDK_install_dir>/docs folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:



```

COM49:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

Primary core started

Matrix #1
=====
 21  33  37  37   9
 23  45  43   0  32
 38  44   8  15  36
 18  18  38  44  16
 22  23   0  38   7

Matrix #2
=====
 11  23  27  45  11
  7  19  23  24   6
 32  26  49  43  16
 22  48  36  34  41
 27  20  32  31  11

eRPC request is sent to the server

Secondary core is running

Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015

Press the SW2 button to initiate the next matrix multiplication

```

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow the instructions in the accompanied example readme files that provide details about the proper board setup and the PC side setup (Python).

Parent topic:[Create an eRPC application](#)

Parent topic:[eRPC example](#)

Other uses for an eRPC implementation The eRPC implementation is generic, and its use is not limited to just embedded applications. When creating an eRPC application outside the embedded world, the same principles apply. For example, this manual can be used to create an eRPC application for a PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or new transport layers can be implemented.

For more information and erpc updates see the github.com/EmbeddedRPC.

Note about the source code in the document Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Changelog eRPC 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

1.14.0

Added

- Added Cmake/Kconfig support.
- Made java code jdk11 compliant, GitHub PR #432.
- Added imxrt1186 support into mu transport layer.
- erpcgen: Added assert for listType before usage, GitHub PR #406.

Fixed

- eRPC: Sources reformatted.
- erpc: Fixed typo in semaphore get (mutex -> semaphore), and write it can fail in case of timeout, GitHub PR #446.
- erpc: Free the arbitrated client token from client manager, GitHub PR #444.
- erpc: Fixed Makefile, install the erpc_simple_server header, GitHub PR #447.
- erpc_python: Fixed possible AttributeError and OSError on calling TCPTransport.close(), GitHub PR #438.
- Examples and tests consolidated.

1.13.0

Added

- erpc: Add BSD-3 license to endianness agnostic files, GitHub PR #417.
- eRPC: Add new Zephyr-related transports (zephyr_uart, zephyr_mbox).
- eRPC: Add new Zephyr-related examples.

Fixed

- eRPC,erpcgen: Fixing/improving markdown files, GitHub PR #395.
- eRPC: Fix Python client TCPTransports not being able to close, GitHub PR #390.
- eRPC,erpcgen: Align switch brackets, GitHub PR #396.
- erpc: Fix zephyr uart transport, GitHub PR #410.
- erpc: UART ZEPHYR Transport stop to work after a few transactions when using USB-CDC resolved, GitHub PR #420.

Removed

- eRPC,erpcgen: Remove cstbool library, GitHub PR #403.

1.12.0**Added**

- eRPC: Add dynamic/static option for transport init, GitHub PR #361.
- eRPC,erpcgen: Winsock2 support, GitHub PR #365.
- eRPC,erpcgen: Feature/support multiple clients, GitHub PR #271.
- eRPC,erpcgen: Feature/buffer head - Framed transport header data stored in Message-Buffer, GitHub PR #378.
- eRPC,erpcgen: Add experimental Java support.

Fixed

- eRPC: Fix receive error value for spidev, GitHub PR #363.
- eRPC: UartTransport::init adaptation to changed driver.
- eRPC: Fix typo in assert, GitHub PR #371.
- eRPC,erpcgen: Move enums to enum classes, GitHub PR #379.
- eRPC: Fixed rpmsg tty transport to work with serial transport, GitHub PR #373.

1.11.0**Fixed**

- eRPC: Makefiles update, GitHub PR #301.
- eRPC: Resolving warnings in Python, GitHub PR #325.
- eRPC: Python3.8 is not ready for usage of typing.Any type, GitHub PR #325.
- eRPC: Improved codec function to use reference instead of address, GitHub PR #324.

- eRPC: Fix NULL check for pending client creation, GitHub PR #341.
- eRPC: Replace sprintf with snprintf, GitHub PR #343.
- eRPC: Use MU_SendMsg blocking call in MU transport.
- eRPC: New LPSPI and LPI2C transport layers.
- eRPC: Freeing static objects, GitHub PR #353.
- eRPC: Fixed casting in deinit functions, GitHub PR #354.
- eRPC: Align LIBUSBIO.GetNumPorts API use with libusb python module v. 2.1.11.
- erpcgen: Renamed temp variable to more generic one, GitHub PR #321.
- erpcgen: Add check that string read is not more than max length, GitHub PR #328.
- erpcgen: Move to g++ in pytest, GitHub PR #335.
- erpcgen: Use build=release for make, GitHub PR #334.
- erpcgen: Removed boost dependency, GitHub PR #346.
- erpcgen: Mingw support, GitHub PR #344.
- erpcgen: VS build update, GitHub PR #347.
- erpcgen: Modified name for common types macro scope, GitHub PR #337.
- erpcgen: Fixed memcpy for template, GitHub PR #352.
- eRPC,erpcgen: Change default build target to release + adding artefacts, GitHub PR #334.
- eRPC,erpcgen: Remove redundant includes, GitHub PR #338.
- eRPC,erpcgen: Many minor code improvements, GitHub PR #323.

1.10.0

Fixed

- eRPC: MU transport layer switched to blocking MU_SendMsg() API use.

1.10.0

Added

- eRPC: Add TCP_NODELAY option to python, GitHub PR #298.

Fixed

- eRPC: MUPort adaptation to new supported SoCs.
- eRPC: Simplifying CI with installing dependencies using shell script, GitHub PR #267.
- eRPC: Using event for waiting for sock connection in TCP python server, formatting python code, C specific includes, GitHub PR #269.
- eRPC: Endianness agnostic update, GitHub PR #276.
- eRPC: Assertion added for functions which are returning status on freeing memory, GitHub PR #277.
- eRPC: Fixed closing arbitrator server in unit tests, GitHub PR #293.
- eRPC: Makefile updated to reflect the correct header names, GitHub PR #295.

- eRPC: Compare value length to used length() in reading data from message buffer, GitHub PR #297.
- eRPC: Replace EXPECT_TRUE with EXPECT_EQ in unit tests, GitHub PR #318.
- eRPC: Adapt rpmsg_lite based transports to changed rpmsg_lite_wait_for_link_up() API parameters.
- eRPC, erpcgen: Better distinguish which file can and cannot be linked by C linker, GitHub PR #266.
- eRPC, erpcgen: Stop checking if pointer is NULL before sending it to the erpc_free function, GitHub PR #275.
- eRPC, erpcgen: Changed api to count with more interfaces, GitHub PR #304.
- erpcgen: Check before reading from heap the buffer boundaries, GitHub PR #287.
- erpcgen: Several fixes for tests and CI, GitHub PR #289.
- erpcgen: Refactoring erpcgen code, GitHub PR #302.
- erpcgen: Fixed assigning const value to enum, GitHub PR #309.
- erpcgen: Enable runTesttest_enumErrorCode_allDirection, serialize enums as int32 instead of uint32.

1.9.1

Fixed

- eRPC: Construct the USB CDC transport, rather than a client, GitHub PR #220.
- eRPC: Fix premature import of package, causing failure when attempting installation of Python library in a clean environment, GitHub PR #38, #226.
- eRPC: Improve python detection in make, GitHub PR #225.
- eRPC: Fix several warnings with deprecated call in pytest, GitHub PR #227.
- eRPC: Fix freeing union members when only default need be freed, GitHub PR #228.
- eRPC: Fix making test under Linux, GitHub PR #229.
- eRPC: Assert costumizing, GitHub PR #148.
- eRPC: Fix corrupt clientList bug in TransportArbitrator, GitHub PR #199.
- eRPC: Fix build issue when invoking g++ with -Wno-error=free-nonheap-object, GitHub PR #233.
- eRPC: Fix inout cases, GitHub PR #237.
- eRPC: Remove ERPC_PRE_POST_ACTION dependency on return type, GitHub PR #238.
- eRPC: Adding NULL to ptr when codec function failed, fixing memcpy when fail is present during deserialization, GitHub PR #253.
- eRPC: MessageBuffer usage improvement, GitHub PR #258.
- eRPC: Get rid of serial and enum34 dependency (enum34 is in python3 since 3.4 (from 2014)), GitHub PR #247.
- eRPC: Several MISRA violations addressed.
- eRPC: Fix timeout for Freertos semaphore, GitHub PR #251.
- eRPC: Use of rpmsg_lite_wait_for_link_up() in rpmsg_lite based transports, GitHub PR #223.
- eRPC: Fix codec nullptr dereferencing, GitHub PR #264.

- erpcgen: Fix two syntax errors in erpcgen Python output related to non-encapsulated unions, improved test for union, GitHub PR #206, #224.
- erpcgen: Fix serialization of list/binary types, GitHub PR #240.
- erpcgen: Fix empty list parsing, GitHub PR #72.
- erpcgen: Fix templates for malloc errors, GitHub PR #110.
- erpcgen: Get rid of encapsulated union declarations in global scale, improve enum usage in unions, GitHub PR #249, #250.
- erpcgen: Fix compile error:UniqueIdChecker.cpp:156:104:'sort' was not declared, GitHub PR #265.

1.9.0

Added

- eRPC: Allow used LIBUSB_SIO device index being specified from the Python command line argument.

Fixed

- eRPC: Improving template usage, GitHub PR #153.
- eRPC: run_clang_format.py cleanup, GitHub PR #177.
- eRPC: Build TCP transport setup code into liberpc, GitHub PR #179.
- eRPC: Fix multiple definitions of g_client error, GitHub PR #180.
- eRPC: Fix memset past end of buffer in erpc_setup_mbf_static.cpp, GitHub PR #184.
- eRPC: Fix deprecated error with newer pytest version, GitHub PR #203.
- eRPC, erpcgen: Static allocation support and usage of rpmsg static FreeRTOSs related API, GitHub PR #168, #169.
- erpcgen: Remove redundant module imports in erpcgen, GitHub PR #196.

1.8.1

Added

- eRPC: New i2c_slave_transport transport introduced.

Fixed

- eRPC: Fix misra erpc c, GitHub PR #158.
- eRPC: Allow conditional compilation of message_loggers and pre_post_action.
- eRPC: (D)SPI slave transports updated to avoid busy loops in rtos environments.
- erpcgen: Re-implement EnumMember::hasValue(), GitHub PR #159.
- erpcgen: Fixing several misra issues in shim code, erpcgen and unit tests updated, GitHub PR #156.
- erpcgen: Fix bison file, GitHub PR #156.

1.8.0

Added

- eRPC: Support win32 thread, GitHub PR #108.
- eRPC: Add mbed support for malloc() and free(), GitHub PR #92.
- eRPC: Introduced pre and post callbacks for eRPC call, GitHub PR #131.
- eRPC: Introduced new USB CDC transport.
- eRPC: Introduced new Linux spidev-based transport.
- eRPC: Added formatting extension for VSC, GitHub PR #134.
- erpcgen: Introduce ustring type for unsigned char and force cast to char*, GitHub PR #125.

Fixed

- eRPC: Update makefile.
- eRPC: Fixed warnings and error with using MessageLoggers, GitHub PR #127.
- eRPC: Extend error msg for python server service handle function, GitHub PR #132.
- eRPC: Update CMSIS UART transport layer to avoid busy loops in rtos environments, introduce semaphores.
- eRPC: SPI transport update to allow usage without handshaking GPIO.
- eRPC: Native _WIN32 erpc serial transport and threading.
- eRPC: Arbitrator deadlock fix, TCP transport updated, TCP setup functions introduced, GitHub PR #121.
- eRPC: Update of matrix_multiply.py example: Add -serial and -baud argument, GitHub PR #137.
- eRPC: Update of .clang-format, GitHub PR #140.
- eRPC: Update of erpc_framed_transport.cpp: return error if received message has zero length, GitHub PR #141.
- eRPC, erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136, #139.
- eRPC, erpcgen: Core re-formatted using Clang version 10.
- erpcgen: Enable deallocation in server shim code when callback/function pointer used as out parameter in IDL.
- erpcgen: Removed '\$' character from generated symbol name in '_\$union' suffix, GitHub PR #103.
- erpcgen: Resolved mismatch between C++ and Python for callback index type, GitHub PR #111.
- erpcgen: Python generator improvements, GitHub PR #100, #118.
- erpcgen: Fixed error messages produced by -Wall -Wextra -Wshadow -pedantic-errors compiler flags, GitHub PR #136.

1.7.4

Added

- eRPC: Support MU transport unit testing.
- eRPC: Adding mbed os support.

Fixed

- eRPC: Unit test code updated to handle service add and remove operations.
- eRPC: Several MISRA issues in rpmsg-based transports addressed.
- eRPC: Fixed Linux/TCP acceptance tests in release target.
- eRPC: Minor documentation updates, code formatting.
- erpcgen: Whitespace removed from C common header template.

1.7.3

Fixed

- eRPC: Improved the test_callbacks logic to be more understandable and to allow requested callback execution on the server side.
- eRPC: TransportArbitrator::prepareClientReceive modified to avoid incorrect return value type.
- eRPC: The ClientManager and the ArbitratedClientManager updated to avoid performing client requests when the previous serialization phase fails.
- erpcgen: Generate the shim code for destroy of statically allocated services.

1.7.2

Added

- eRPC: Add missing doxygen comments for transports.

Fixed

- eRPC: Improved support of const types.
- eRPC: Fixed Mac build.
- eRPC: Fixed serializing python list.
- eRPC: Documentation update.

1.7.1

Fixed

- eRPC: Fixed semaphore in static message buffer factory.
- erpcgen: Fixed MU received error flag.
- erpcgen: Fixed tcp transport.

1.7.0

Added

- eRPC: List names are based on their types. Names are more deterministic.
- eRPC: Service objects are as a default created as global static objects.
- eRPC: Added missing doxygen comments.
- eRPC: Added support for 64bit numbers.
- eRPC: Added support of program language specific annotations.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Generating crc value is optional.
- eRPC: Fixed CMSIS Uart driver. Removed dependency on KSDK.
- eRPC: Forbid users use reserved words.
- eRPC: Removed outByref for function parameters.
- eRPC: Optimized code style of callback functions.

1.6.0

Added

- eRPC: Added @nullable support for scalar types.

Fixed

- eRPC: Improved code size of generated code.
- eRPC: Improved eRPC nested calls.
- eRPC: Improved eRPC list length variable serialization.

1.5.0

Added

- eRPC: Added support for unions type non-wrapped by structure.
- eRPC: Added callbacks support.
- eRPC: Added support @external annotation for functions.
- eRPC: Added support @name annotation.
- eRPC: Added Messaging Unit transport layer.
- eRPC: Added RPMSG Lite RTOS TTY transport layer.
- eRPC: Added version verification and IDL version verification between eRPC code and eRPC generated shim code.
- eRPC: Added support of shared memory pointer.

- eRPC: Added annotation to forbid generating const keyword for function parameters.
- eRPC: Added python matrix multiply example.
- eRPC: Added nested call support.
- eRPC: Added struct member “byref” option support.
- eRPC: Added support of forward declarations of structures
- eRPC: Added Python RPMsg Multiendpoint kernel module support
- eRPC: Added eRPC sniffer tool

1.4.0

Added

- eRPC: New RPMsg-Lite Zero Copy (RPMsgZC) transport layer.

Fixed

- eRPC: win_flex_bison.zip for windows updated.
- eRPC: Use one codec (instead of inCodec outCodec).

[1.3.0]

Added

- eRPC: New annotation types introduced (@length, @max_length, ...).
- eRPC: Support for running both erpc client and erpc server on one side.
- eRPC: New transport layers for (LP)UART, (D)SPI.
- eRPC: Error handling support.

[1.2.0]

Added

- eRPC source directory organization changed.
- Many eRPC improvements.

[1.1.0]

Added

- Multicore SDK 1.1.0 ported to KSDK 2.0.0.

[1.0.0]

Added

- Initial Release

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
- IW611¹
- IW612²
- AW611³
- 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 MHz
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 support
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) 4k TX and RX support
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 (wireless
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 MHz
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 (wireless multimedia)
802.11i security	Hostapd Support
802.11i security	WPA2-PSK AES hostapd
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) Hostapd
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) Hostapd

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	88W89	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	88W8987	IW416	IW611/IW612	RW610/RW612	IW612	AW611
Generic	Firmware download (parallel) ^[^1]	Y	Y	Y	N	N	Y
Generic	Secure boot	N	N	Y	Y	Y	Y
Generic	Kconfig memory optimizer	Y	Y	Y	Y	Y	Y
Generic	Firmware Compression ^[^2]	N	Y	N	N	N	N
Generic	u-AP intra-BSS	Y	N	Y	Y	Y	Y
Generic	Net Monitor Mode	N	N	N	Y	Y	N
Generic	Net Monitor Mode with packet transmission	N	N	N	Y	Y	N
Generic	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 ^[^1]	IW611/IW612	RW610/RW612	IW612 ^[^1]	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	fea-	Bluetooth Class 1.5 and Class 2 support	Y	Y	Y	N	N	Y
General features	fea-	Scatternet support	Y	Y	Y	N	N	Y
General features	fea-	Maximum of seven simultaneous ACL connections – Central links	Y	Y	Y	N	N	Y
General features	fea-	Automatic packet type selection	Y	Y	Y	N	N	Y
General features	fea-	Bluetooth - 2.1 to 5.0 specification support	Y	Y	Y	N	N	Y
General features	fea-	Low power sniff	Y	Y	Y	N	N	Y
General features	fea-	Deep sleep using out-of-band	Y	Y	N	N	N	N
General features	fea-	Wake on Bluetooth (SoC to host)	Y	Y	Y	N	N	Y
General features	fea-	Independent reset (in-band) ^[^1]	Y	Y	Y	Y	N	Y
General features	fea-	Independent reset (out-band) ^[^1]	Y	Y	N	N	N	N
General features	fea-	Firmware download (parallel) ^[^1]	Y	Y	N	N	N	N
General features	fea-	RF test mode	Y	Y	Y	N	N	Y
Bluetooth packet type supported	type	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	type	SCO (HV1, HV3)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	type	eSCO (EV3, EV4, EV5, 2EV3, 3EV3, 2EV5, 3EV5)	Y	Y	Y	N	N	Y
Bluetooth profiles supported	sup-	A2DP source/sink	Y	Y	Y	N	N	Y
Bluetooth profiles supported	sup-	AVRCP target/controller	Y	Y	Y	N	N	Y
Bluetooth profiles supported	sup-	HFP Dev/AG	Y	Y	Y	N	N	Y
Bluetooth profiles supported	sup-	OPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	sup-	SPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	sup-	HID target/device	Y	Y	Y	N	N	Y
Bluetooth audio features	au-	PCM NBS central/peripheral	Y	Y	Y	N	N	Y
Bluetooth audio features	au-	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, 32kHz
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 tures	fea- Spinel over SPI	Y	N	N
General tures	fea- OpenThread RCP Mode implementing Thread1.3	Y	N	N
General tures	fea- 802.15.4-2015 MAC/PHY as required by Thread 1.3	Y	Y	Y
General tures	fea- OpenThread Border Router (OTBR) v1.1	Y	Y	Y
General tures	fea- Direct/indirect transmission with/without ACK	Y	Y	Y
General tures	fea- 802.15.4 CSL parent feature implementation	Y	Y	Y
General tures	fea- Enhanced Frame Pending	Y	Y	Y
General tures	fea- Enhanced keep alive	Y	Y	Y
General tures	fea- Router	Y	Y	Y
General tures	fea- Leader	Y	Y	Y
General tures	fea- Router Eligible End Device (REED)	Y	Y	Y
General tures	fea- 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
CON- FIG_WIFI_SL]	CONFIG_ROAMING CONFIG_11R
CON- FIG_WIFI_SL]	CONFIG_CLOUD_KEEP_ALIVE CONFIG_WIFI_EU_CRYPTO CON- FIG_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 CON- FIG_COMBO_SCAN CONFIG_SCAN_CHANNEL_GAP
CON- FIG_WIFI_SL]	CONFIG_UAP_STA_MAC_ADDR_FILTER CONFIG_WIFI_MAX_CLIENTS_CNT
CON- FIG_FREERTC	If the macro is enabled, the heap memory usage is reduced by 10 KB (from 70 KB to 60 KB).
CON- FIG_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**

Compo- po- nent	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**

Compo- nent	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**

Compo- nent	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**

Compo- nent	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}

Com- ponent	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}

Com- ponent	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

Compo- nent	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_throughput}](#)

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 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
Bluetooth	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)		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)		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)		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)		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}

Component	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)		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}

Com po- nent	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.
Blue tooth	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.
Blue-tooth	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
Blue tooth	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.

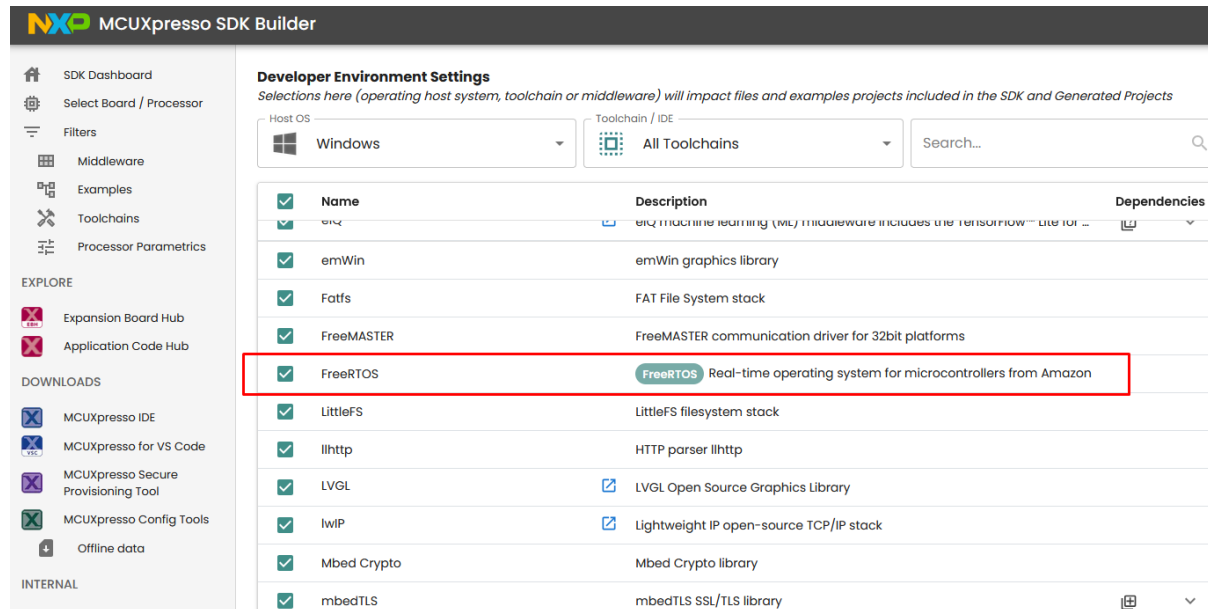
The MCUXpresso SDK framework also contains a set of FreeRTOS examples which show basic FreeRTOS OS features. This makes it easy to start a new FreeRTOS project or begin experimenting with FreeRTOS OS. Selected drivers and middleware are RTOS ready with related FreeRTOS adaptation layer.

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.

Task List (FreeRTOS)							
TCB#	Task Name	Task Handle	Task State	Priority	Stack Usage	Event Object	Runtime
1	task_one	0x1fffecc8	Blocked	1 (1)	0 B / 880 B	MyCountingSemaphore (Rx)	0x0 (0.0%)
2	task_two	0x1ffff130	Blocked	2 (2)	0 B / 888 B	MyCountingSemaphore (Rx)	0x1 (0.1%)
3	IDLE	0x1ffff330	Running	0 (0)	0 B / 296 B		0x3e5 (99.6%)
4	Tmr Svc	0x1ffff6b8	Blocked	17 (17)	28 B / 672 B	TmrQ (Rx)	0x3 (0.3%)

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 appropriate 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)(<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.userName = 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](#).