MCUXpresso SDK Documentation

Release 25.12.00

NXP
Dec 18, 2025

Table of contents

This documentation contains information specific to the evkbimxrt1050 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

Sim passing Mynewt failing Espressif passing imgtool passing License: Apache 2.0

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 Connectivity

1.2.1 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.

1.3 eIQ

1.3.1 eIQ

eIQ TensorFlow Lite for Micro Library User Guide

Overview TensorFlow Lite is an open source software library for running machine learning models on mobile and embedded devices. For more information, see www.tensorflow.org/lite.

For memory constrained devices, the library contains TensorFlow Lite for Microcontrollers. For more information, see *www.tensorflow.org/lite/microcontrollers*.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated TensorFlow Lite for Microcontrollers based on version 25-04-08 (from the 8th of April 2025 with commit). This document describes the steps required to download and start using the library. Additionally, the document describes the steps required to create an application for running pre-trained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

TensorFlow Lite for Microcontrollers TensorFlow Lite for Microcontrollers is a port of TensorFlow Lite designed to run machine learning models on DSPs, microcontrollers and other devices with limited memory.

Additional Links:

- Tensorflow github repository
- TFLM at tensorflow.org

Build Status

- GitHub Status

Official Builds

Build Type	Status
CI (Linux)	Run-CI passing
Code Sync	Sync from Upstream TF passing

Community Supported TFLM Examples This table captures platforms that TFLM has been ported to. Please see *New Platform Support* for additional documentation.

Platform	Status
Arduino	CI no status Arduino examples tests no status
Coral Dev Board Micro	TFLM + EdgeTPU Examples for Coral Dev Board Micro
Espressif Systems Dev Boards	CI passing
Renesas Boards	TFLM Examples for Renesas Boards
Silicon Labs Dev Kits	TFLM Examples for Silicon Labs Dev Kits
Sparkfun Edge	CI no status
Texas Instruments Dev Boards	

Community Supported Kernels and Unit Tests This is a list of targets that have optimized kernel implementations and/or run the TFLM unit tests using software emulation or instruction set simulators.

Build Type	Status
Cortex-M	Cortex-M passing
Hexagon	Run-Hexagon passing
RISC-V	RISC-V passing
Xtensa	Run-Xtensa passing
Generate Integration Test	Generate Integration Tests passing

Contributing See our *contribution documentation*.

Getting Help A Github issue should be the primary method of getting in touch with the TensorFlow Lite Micro (TFLM) team.

The following resources may also be useful:

1. SIG Micro email group and monthly meetings.
2. SIG Micro gitter chat room.
3. For questions that are not specific to TFLM, please consult the broader TensorFlow project, e.g.:
   - Create a topic on the TensorFlow Discourse forum
   - Send an email to the TensorFlow Lite mailing list
   - Create a TensorFlow issue
   - Create a Model Optimization Toolkit issue

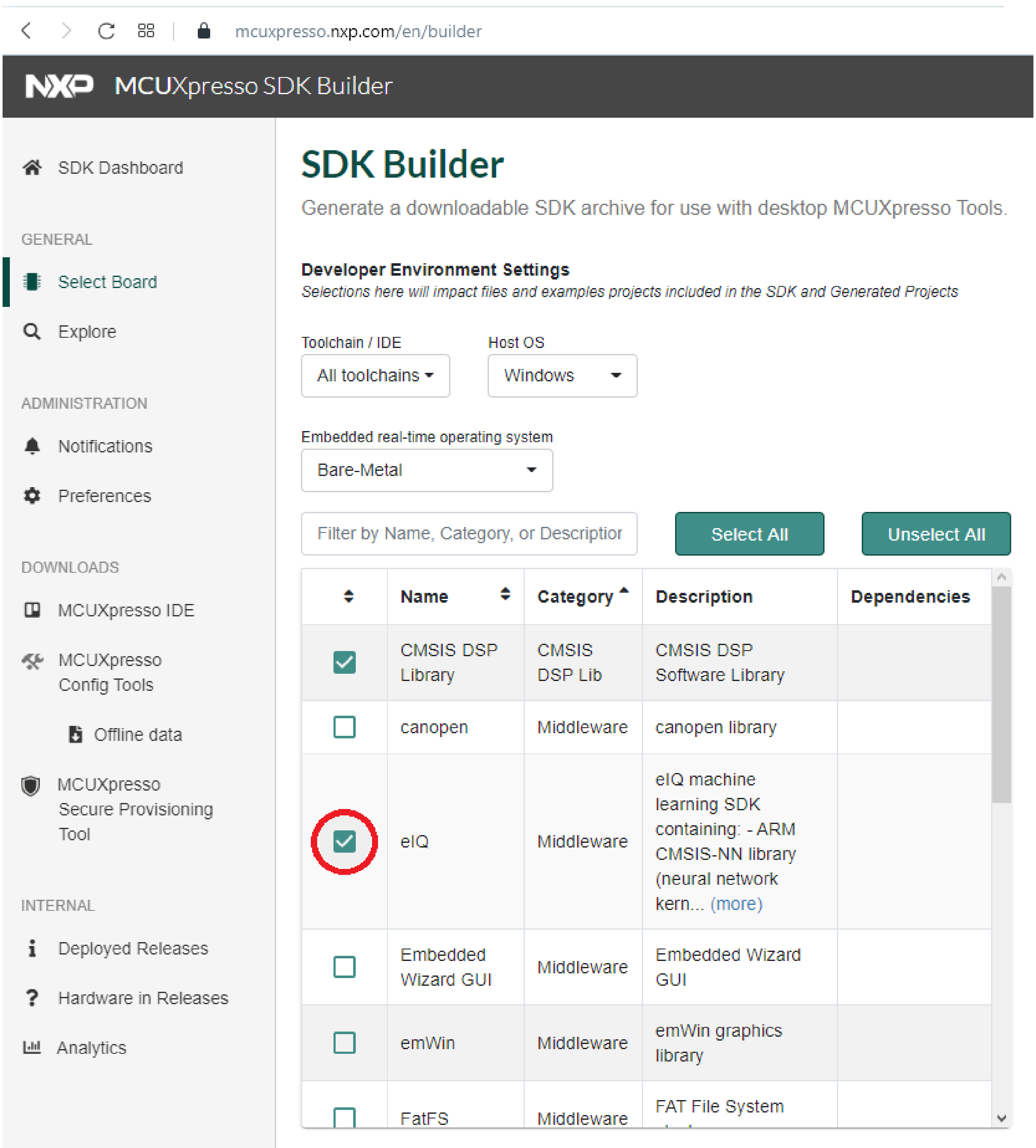
Additional Documentation

- *Continuous Integration*
- *Benchmarks*
- *Profiling*
- *Memory Management*
- *Logging*
- *Porting Reference Kernels from TfLite to TFLM*
- *Optimized Kernel Implementations*
- *New Platform Support*
- Platform/IP support
  - *Arm IP support*
- *Software Emulation with Renode*
- *Software Emulation with QEMU*
- *Python Dev Guide*
- *Automatically Generated Files*
- *Python Interpreter Guide*

RFCs

1. *Pre-allocated tensors*
2. *TensorFlow Lite for Microcontrollers Port of 16x8 Quantized Operators*

Deployment The eIQ TensorFlow Lite for Microcontrollers library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package. See *Figure 1*.

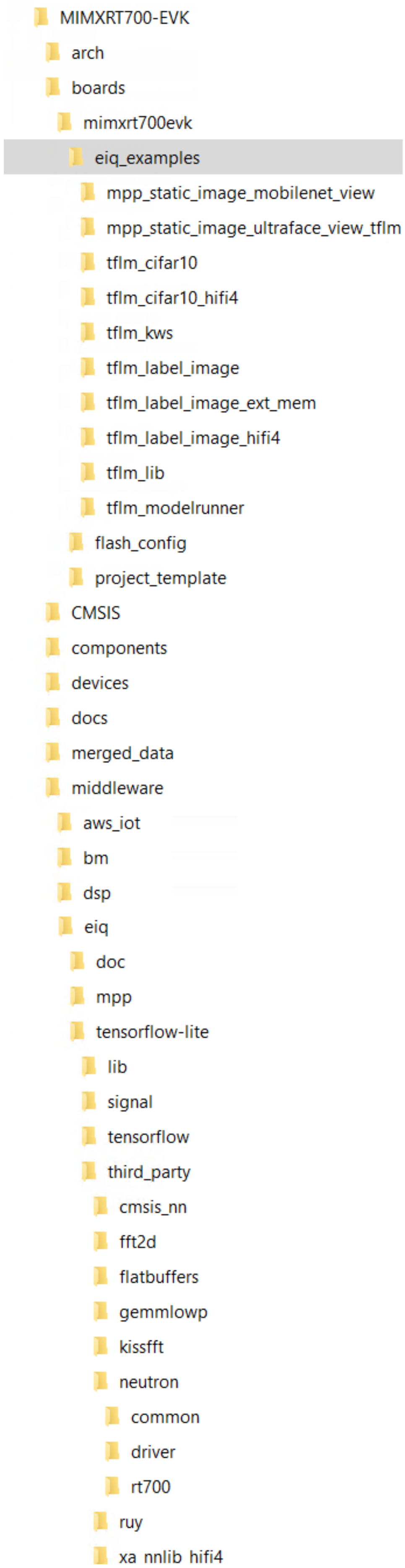


|

|

Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to *Figure 2*. The eIQ TensorFlow Lite library directories are highlighted in red.

- SDK_2_15_000_EVKB-IMXRT1050
  - boards
    - evkbimxrt1050
      - cmsis_driver_examples
      - component_examples
      - demo_apps
      - driver_examples
      - eiq_examples
        - deepviewrt_camera_label_image
        - deepviewrt_image_detection
        - deepviewrt_labelimage
        - glow_cifar10
        - glow_cifar10_camera
        - glow_lenet_mnist
        - glow_lenet_mnist_camera
        - tflm_cifar10
        - tflm_kws
        - tflm_label_image
        - tflm_lib
      - littlefs_examples
      - lwip_examples
      - project_template
      - sdmmc_examples
      - xip
  - CMSIS
  - components
  - devices
  - docs
  - middleware
    - bm
    - cjson
    - eiq
      - deepviewrt
      - doc
      - glow
      - mpp
      - tensorflow-lite
        - lib
        - signal
        - tensorflow
        - third_party


MIMXRT700-EVK
arch
boards
mimxrt700evk
eiq_examples
mpp_static_image_mobilenet_view
mpp_static_image_ultraface_view_tflm
tflm_cifar10
tflm_cifar10_hifi4
tflm_kws
tflm_label_image
tflm_label_image_ext_mem
tflm_label_image_hifi4
tflm_lib
tflm_modelrunner
flash_config
project_template
CMSIS
components
devices
docs
merged_data
middleware
aws_iot
bm
dsp
eiq
doc
mpp
tensorflow-lite
lib
signal
tensorflow
third_party
cmsis_nn
fft2d
flatbuffers
gemmlowp
kissfft
neutron
common
driver
rt700
ruy
xa_nnlib_hifi4

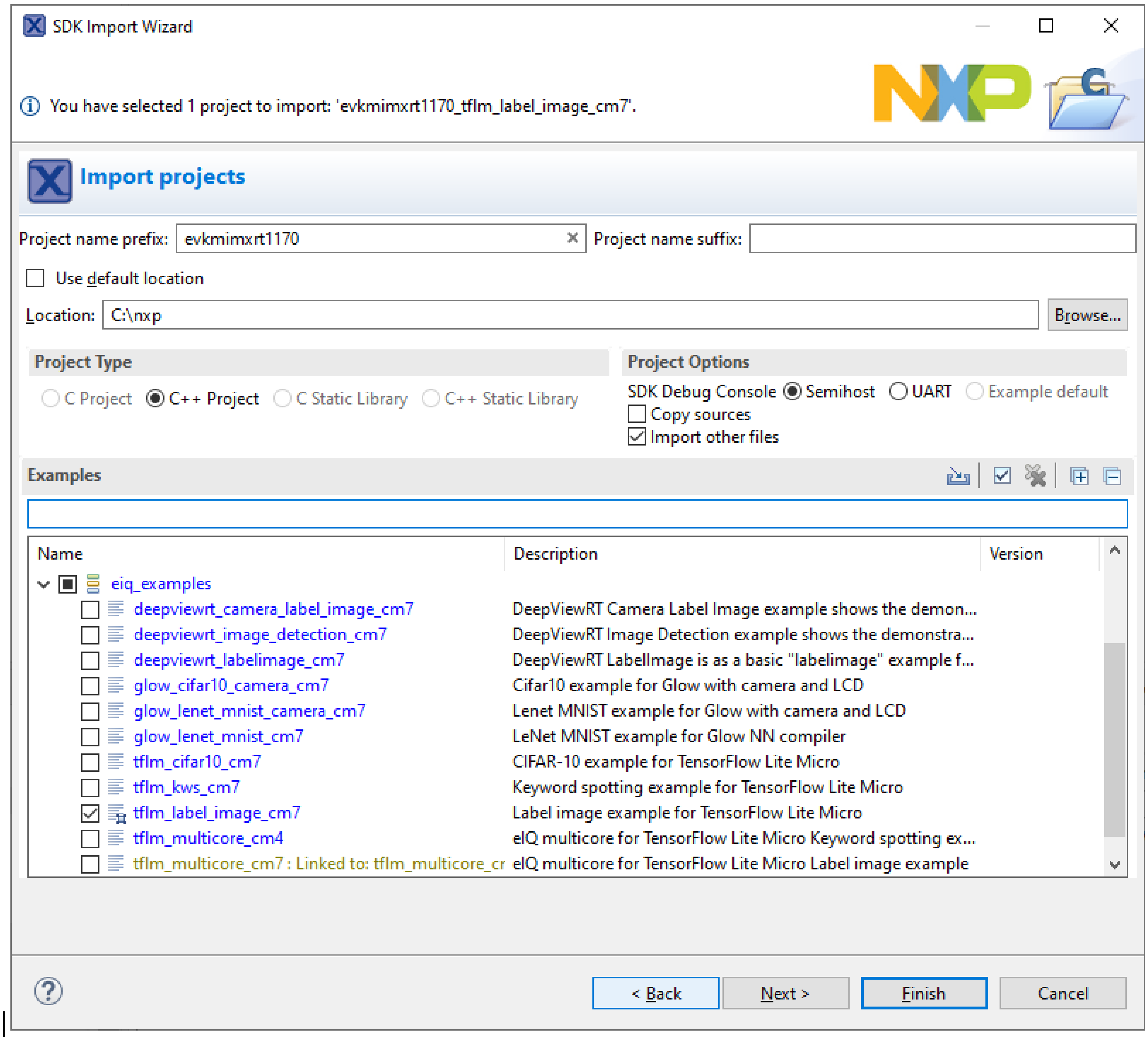
|

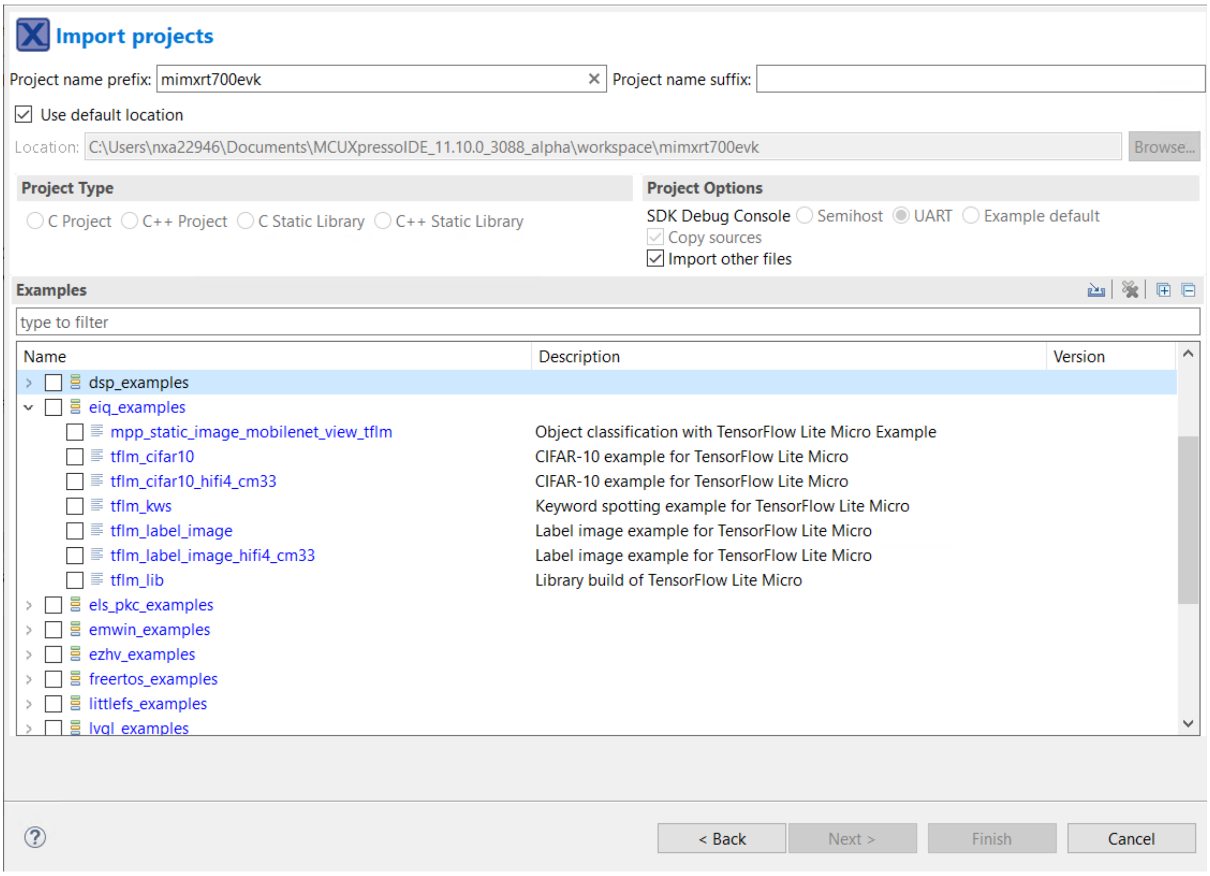
The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

Example applications The eIQ TensorFlow Lite library is provided with a set of example applications. For details, see *Table 1*. The applications demonstrate the usage of the library in several use cases.

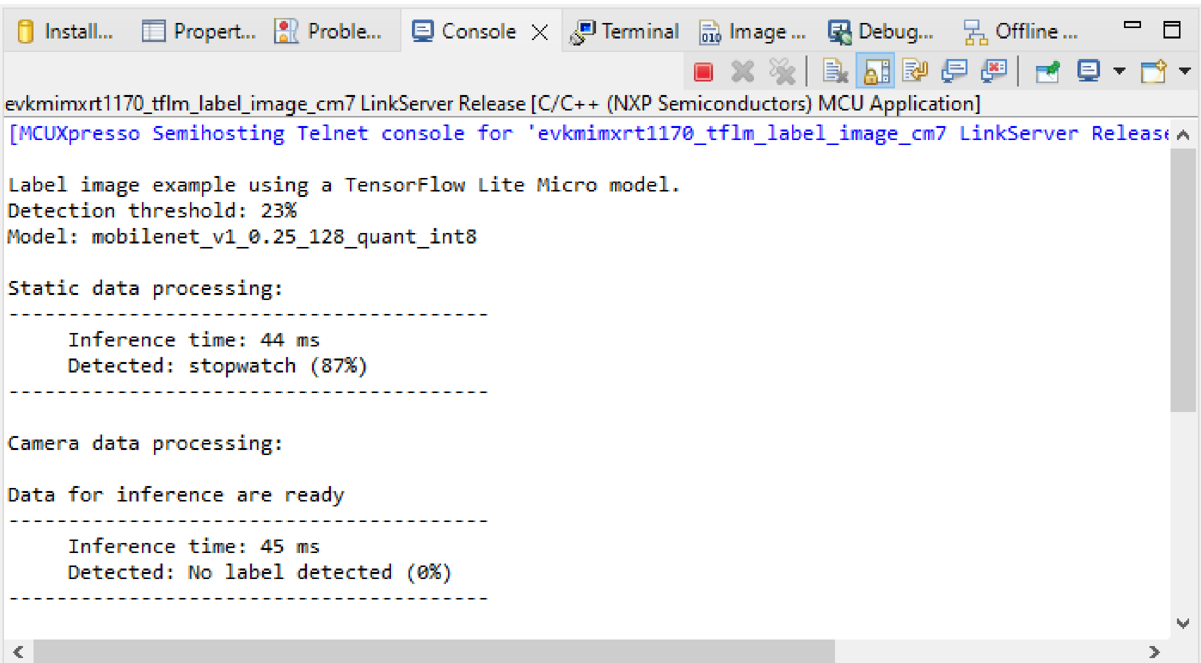
Name	Description	Availability
tflm_c	CIFAR-10 classification of 32 × 32 RGB pixel images into 10 categories using a small Convolutional Neural Network (CNN).	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_k	Keyword spotting application using a neural network for word detection in pre-processed audio input.	MCX-N947-EVK (no audio support) MCX-N947-FRDM (no audio support) MCX-N547-EVK (no audio support) MIMXRT700-EVK (no audio support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categorieswith eIQ Neutron NPU.	MCX-N947-EVK (no camera and display support) MCX-N947-FRDM (no camera and display support) MCX-N547-EVK (no camera and display support) MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 224 × 224 RGB pixel images into 1000 categorieswith eIQ Neutron NPU. In this example, it demonstrates how to fetch model's weight from external memory(xSPI flash) to internal SRAM for Neutron NPU execution.	MIMXRT700-EVK (no camera and display support)
tflm_c	CIFAR-10 classification of 32 × 32 RGBpixel images into 10 categories using a small Convolutional Neural Network. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for CIFAR-10 classification.	MIMXRT700-EVK (no camera and display support)
tflm_l	Image recognition application using a MobileNet model architecture to classify 128 × 128 RGB pixel images into 1000 categories. In this example, M33 core0 starts HiFi4 DSP core with HiFi4 DSP image. HiFi4 DSP does the inference for image recognition application.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG). When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard as shown in *Figure 1*.


SDK Import Wizard
You have selected 1 project to import: 'evkmimxrt1170_tflm_label_image_cm7'.
Import projects
Project name prefix: evkmimxrt1170
Project name suffix:
Use default location
Location: C:\nxp
Browse...
Project Type
C Project
C++ Project
C Static Library
C++ Static Library
Project Options
SDK Debug Console
Semihost
UART
Example default
Copy sources
Import other files
Examples
Name
Description
Version
eiq_examples
deepviewrt_camera_label_image_cm7
DeepViewRT Camera Label Image example shows the demon...
deepviewrt_image_detection_cm7
DeepViewRT Image Detection example shows the demonstra...
deepviewrt_labelimage_cm7
DeepViewRT LabelImage is as a basic "labelimage" example f...
glow_cifar10_camera_cm7
Cifar10 example for Glow with camera and LCD
glow_lenet_mnist_camera_cm7
Lenet MNIST example for Glow with camera and LCD
glow_lenet_mnist_cm7
LeNet MNIST example for Glow NN compiler
tflm_cifar10_cm7
CIFAR-10 example for TensorFlow Lite Micro
tflm_kws_cm7
Keyword spotting example for TensorFlow Lite Micro
tflm_label_image_cm7
Label image example for TensorFlow Lite Micro
tflm_multicore_cm4
eIQ multicore for TensorFlow Lite Micro Keyword spotting ex...
tflm_multicore_cm7 : Linked to: tflm_multicore_cr
eIQ multicore for TensorFlow Lite Micro Label image example
< Back
Next >
Finish
Cancel



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, *Figure 2* shows the output of the tflm_label_image_cm7``tflm_label_image example application printed to the MCUXpresso IDE Console window when semihosting debug console is selected in the SDK Import Wizard.



```
Label image example using a TensorFlow Lite Micro model.
Detection threshold: 23%
Model: mobilenet_v1_0.25_128_quant_int8_npu

Static data processing:
----------------------------------------
     Inference time: 3987 us
     Detected: stopwatch (87%)
----------------------------------------
```

|

Model Conversion to TensorFlow Lite Format The eIQ® Toolkit provides a comprehensive end-to-end environment for machine learning (ML) model development and deployment. Designed for NXP EdgeVerse processors, the toolkit includes both an intuitive GUI-based tool (eIQ Portal) and command-line utilities for advanced workflows.

One key component, the eIQ ModelTool, enables seamless conversion of ML models from popular formats such as TensorFlow, PyTorch, and ONNX into the TensorFlow Lite (TFLite) format. These converted models can be further optimized and deployed on NXP platforms for inference acceleration.

Model Conversion for NXP eIQ Neutron NPU To leverage the NXP eIQ Neutron NPU for hardware acceleration, models must undergo additional processing using the Neutron Converter Tool. This tool transforms standard quantized TensorFlow Lite models into a format optimized for execution on the Neutron NPU.

The key steps involved in this process are as follows:

1. Convert to Quantized TensorFlow Lite Model: Ensure the model is in a quantized TFLite format before running the Neutron Converter.
2. Run the Neutron Converter Tool: The Neutron Converter analyzes the TFLite model, identifies supported operators, and replaces them with specialized NPU-compatible nodes. Unsupported operations are executed using fallback mechanisms, such as:

- CMSIS-NN for optimized CPU execution
- Reference Operators for unsupported cases

3. Execute on Target Platform: The converted model runs efficiently on the Neutron NPU using a custom TFLite Micro-operator implementation.

Example: Converting a Quantized TensorFlow Lite Model for Neutron NPU The following is a sample command-line invocation for the Neutron Converter tool:

```
neutron-converter --input mobilenet_v1_0.25_128_quant.tflite \
                  --output mobilenet_v1_0.25_128_quant_npu.tflite \
                  --target imxrt700 \
                  --dump-header-file-output
```

Note: This will convert the source tflite model to neutron compatable model, meanwhile, it will dump the model as one headfile name as "mobilenet_v1_0.25_128_quant_npu.h".

Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE This section lists the steps to Prepare CM33 Core for the examples and Prepare DSP core for the examples.

Prepare CM33 Core for the examples

1. The tflm_cifar10_hifi4 and tflm_label_image_hifi4 examples consist of two separate applications that run on the CM33 core0 and DSP core. The CM33 core0 application initializes the DSP core and starts it.

   To debug the application:

   1. Set up and execute the CM33 application using an environment of your choice.
   2. Build and execute the examples located in:

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/cm33/

      <SDK_ROOT>/ boards/mimxrt700evk/eiq_examples/tflm_label_image_hifi4/cm33/

2. The tflm_cifar10_hifi1 example consists of three separate applications that run on the CM33 core0, CM33 core1, and DSP core. The CM33 core0 application initializes the CM33 core1 core and starts it. The CM33 core1 application initializes the DSP core and starts it.

   To debug the application:

   1. Set up and build the CM33 core1 application using an environment of your choice.
   2. Set up and execute the CM33 core0 application using an environment of your choice.
   3. Build and execute the example located in:

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core1/

      <SDK_ROOT>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi1/cm33_core0/

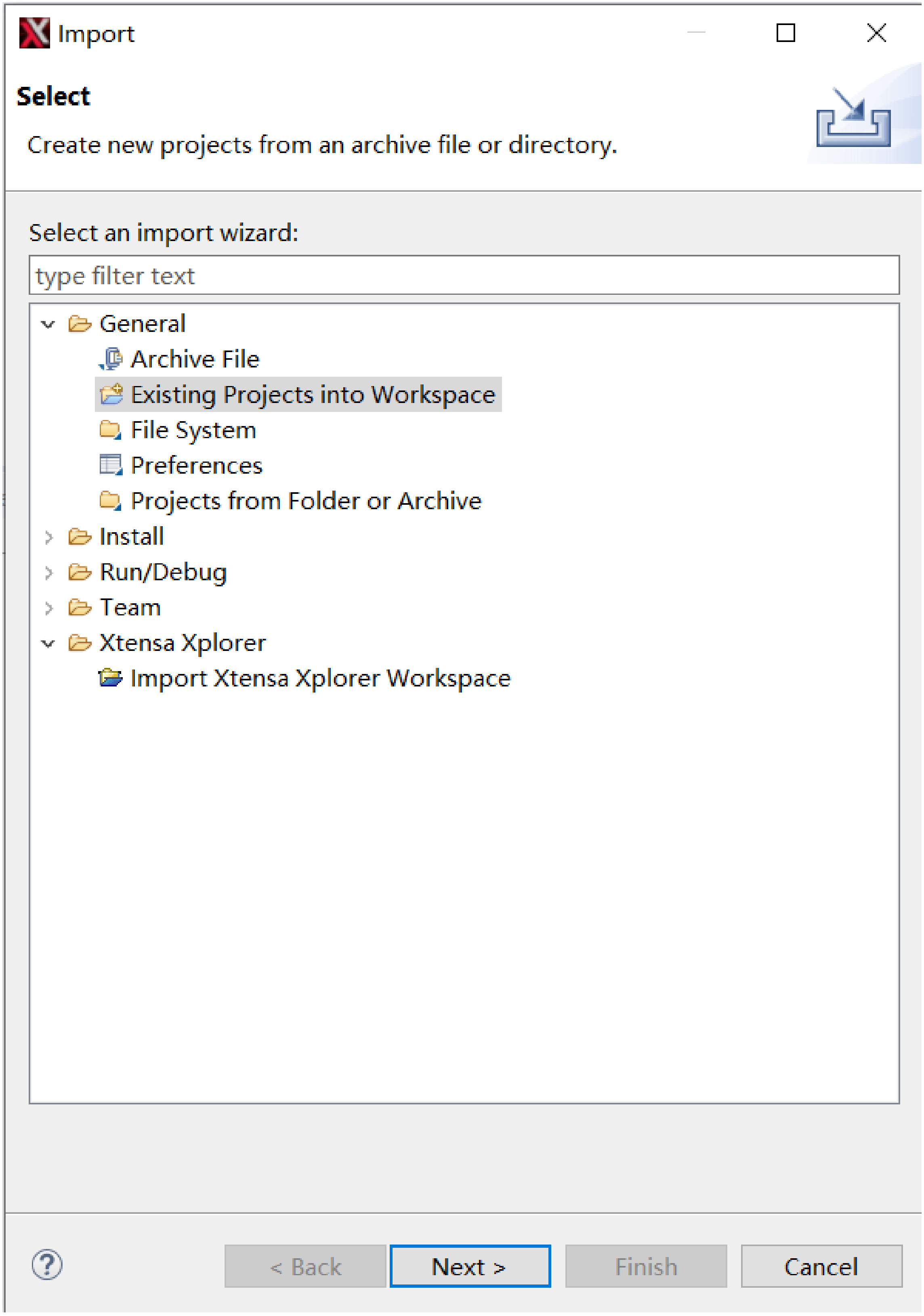
      Note: ARMGCC toolchain and IAR Embedded Workbench are both supported. To enable compatibility with RT700, IAR Embedded Workbench may require a patch. There are default DSP core images in the SDK. For details on how to build the examples, refer to Prepare DSP core for the examples.

Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

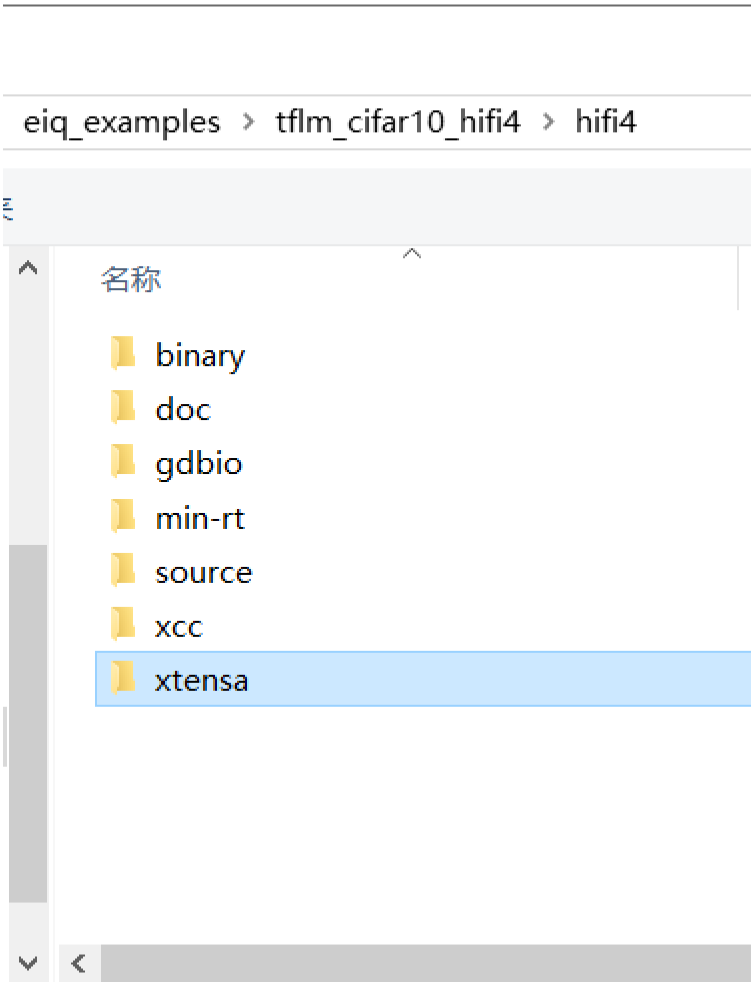
Prepare DSP core for the examples The projects for different supported toolchains are built. The "xcc" project builds on the command line and the "xtensa" directory is an Xplorer IDE project.

To run the tflm_cifar10_hifi4 example, import the SDK sources into the Xplorer IDE.

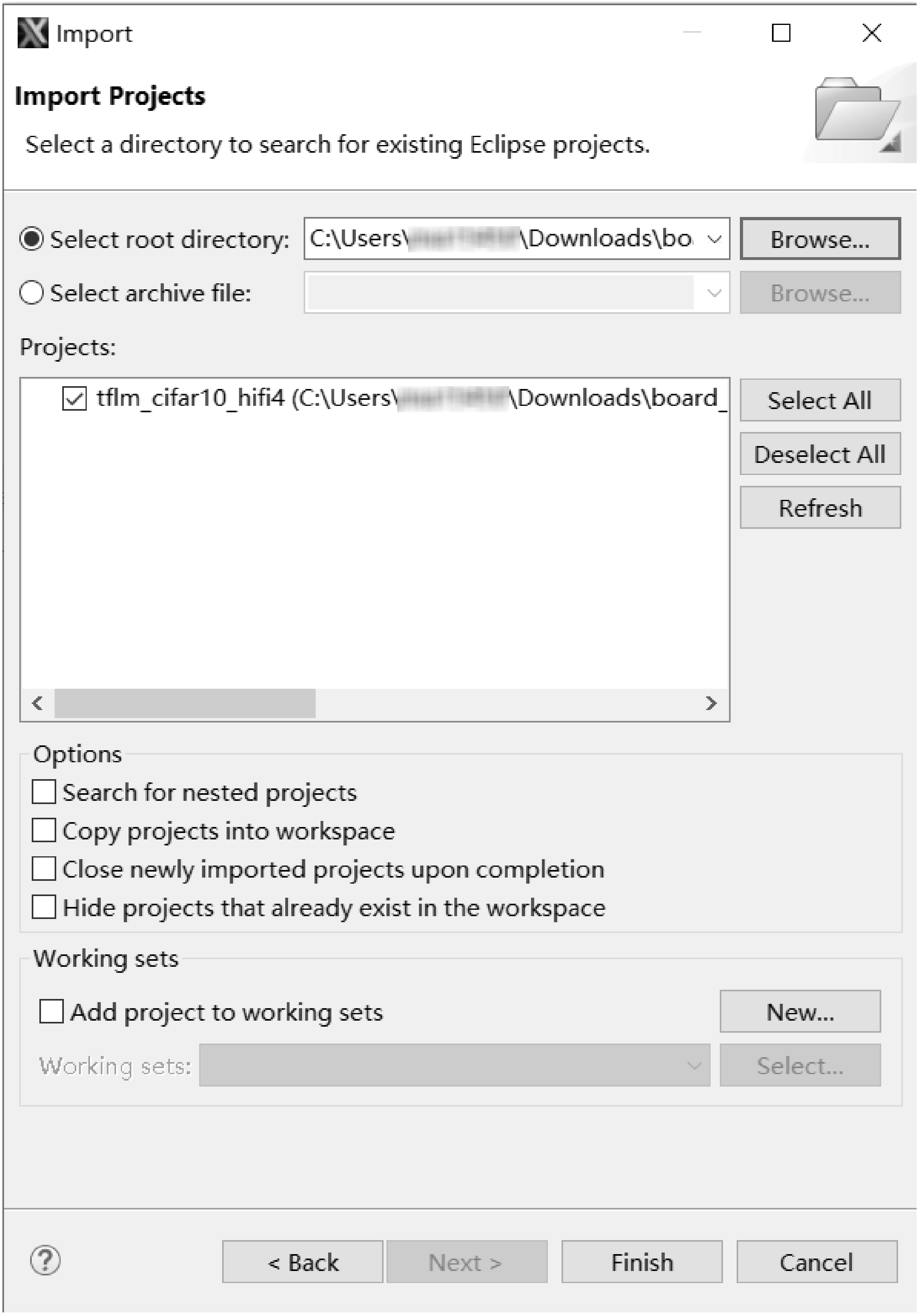
1. Select **File** > **Import** > **General** > **Existing Projects into Workspace**.



2. Click **Next**.
3. Select the SDK directory/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/xtensa as the root directory.

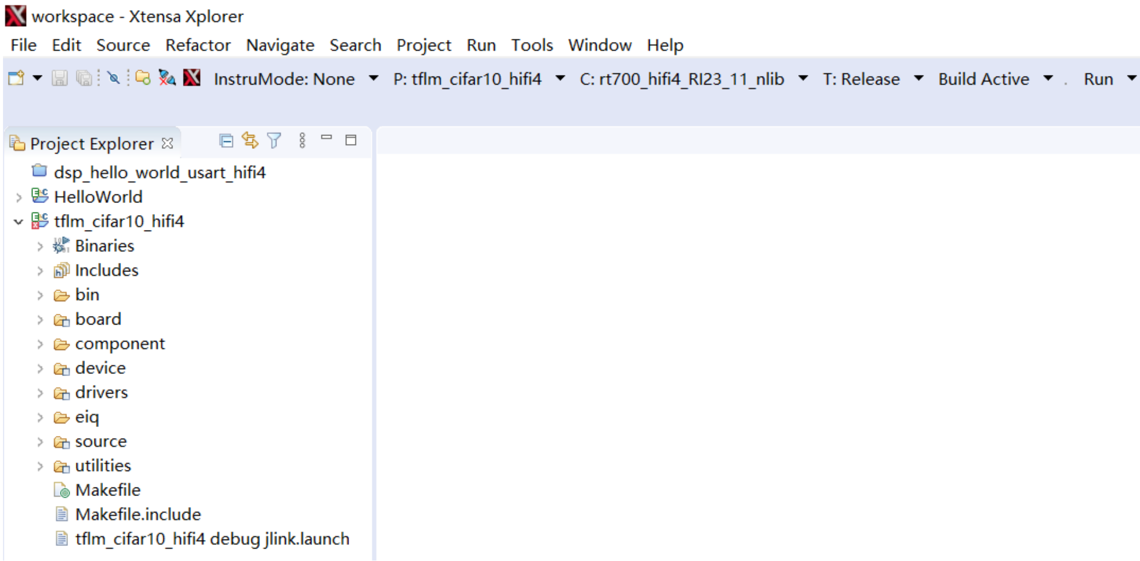


4. Click **Select Folder**.
5. Leave all the other options check boxes blank.

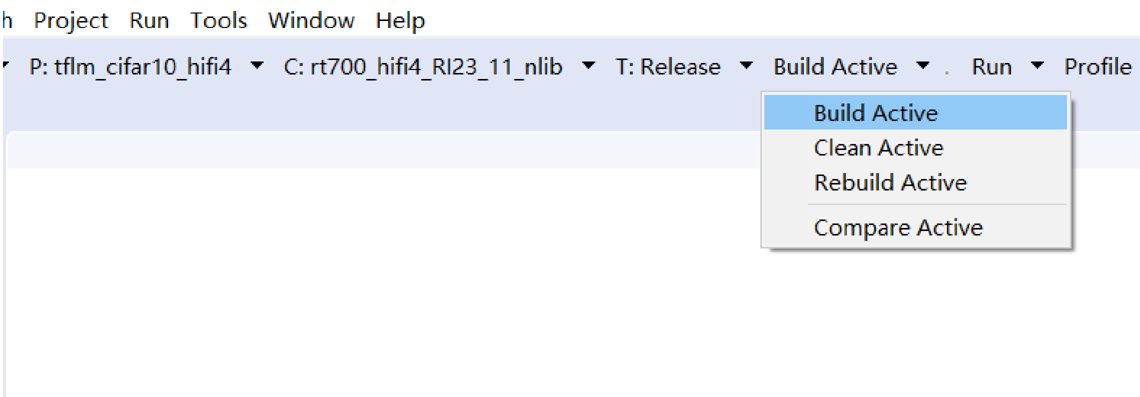


Once imported, the tflm_cifar10_hifi4 example appears in the **Project Explorer**.

6. To make a build selection for the project and hardware target configuration, use the drop-down buttons on the menu bar.



7. To build the DSP application image for the CM33 application, select the **Release target** option in the Xplorer IDE as below.



8. Three DSP binaries are generated and are loaded into different TCM or SRAM address segments:
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_data_release.bin
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_literal_release.bin
   - <SDK_ROOT/>/boards/mimxrt700evk/eiq_examples/tflm_cifar10_hifi4/hifi4/binary/dsp_text_release.bin

Parent topic:*Run and debug eIQ HiFi4 and HiFi1 DSP examples using Xplorer IDE*

Running an inference After converting the model to the TensorFlow Lite format, it is converted into a C language array to include it in the application source code. The *xxd* utility can be used for this purpose (distributed with the *Vim* editor for many platforms on https://www.vim.org/) as shown in *Converting a model to a C language header file*. The utility converts a TensorFlow Lite model into a C header file with an array definition containing the binary image of the model and a variable containing the data size.

Converting a model to a C language header file {#EXAMPLE_4 .section}

```
xxd -i mobilenet_v1_0.25_128_quant.tflite > mobilenet_v1_0.25_128_quant_model.h
```

After the header file is generated, the type of the array is changed from unsigned char to const char to match the library API input parameters and the default array name can be changed to a more convenient one. The user must align the buffer to at least 64-bit boundary (the size of a double-precision floating-point number) to avoid misaligned memory access. The alignment can be achieved by using the __ALIGNED(16) macro from the cmsis_compiler.h header file (available in the MCUXpresso SDK) in the array declaration before the data assignment.

The easiest way to create an application with the proper configuration is to copy and modify an existing example application. To learn where to find the example applications and how to build them, see the *Example applications*.

Running an inference using TensorFlow Lite for Microcontrollers involves several steps (shown for quantized model with signed 8-bit values as input and 32-floating point values as output):

1. Include the necessary eIQ TensorFlow Lite Micro library header files and the converted model.

   Including header files

   ```
   #include "tensorflow/lite/micro/micro_error_reporter.h"
   #include "tensorflow/lite/micro/micro_interpreter.h"
   #include "tensorflow/lite/micro/all_ops_resolver.h"
   #include "mobilenet_v1_0.25_128_quant_model.h"
   ```

2. Allocate a static memory buffer for input and output tensors and intermediate arrays. Load the FlatBuffer model image (assuming the mobilenet_v1_0.25_128_quant_model.h file generated in *Converting a model to a C language header file* defines an array named mobilenet_model and a size variable named mobilenet_model_len), build the interpreter object and allocate memory for tensors.

   Loading the FlatBuffer model

   ```
   constexpr int kTensorArenaSize = 1024 * 1024;
   static uint8_t tensorArena[kTensorArenaSize];
   const tflite::Model* model = tflite::GetModel(mobilenet_model);
   // TODO: Report an error if model->version() != TFLITE_SCHEMA_VERSION
   static tflite::AllOpsResolver microOpResolver;
   static tflite::MicroErrorReporter microErrorReporter;
   static tflite::MicroInterpreter interpreter(model,
     microOpResolver, tensorArena, kTensorArenaSize,
     microErrorReporter);
   interpreter->AllocateTensors();
   // TODO: Check return value for kTfLiteOk
   ```

3. Fillhe input data into the input tensor. For example, if a speech recognition model, image data from a camera or audio data from a microphone. The dimensions of the input data must be the same as the dimensions of the input tensor. These dimensions were specified when the model was created.

   Fill-in input data

   ```
   // Get access to the input tensor data
   TfLiteTensor* inputTensor = interpreter->input(0);
   // Copy the input tensor data from an application buffer
   for (int i = 0; i < inputTensor->bytes; i++)
     inputTensor->data.int8[i] = input_data[i];
   ```

4. Run the inference and read the output data from the output tensor. The dimensions of the output data must be the same as the dimensions of the output tensor. These dimensions were specified when the model was created.

   Running inference and reading output data

```
// Run the inference
interpreter->Invoke();
// TODO: Check the return value for TfLiteOk
// Get access to the output tensor data
TfLiteTensor* outputTensor = interpreter->output(0);
// Copy the output tensor data to an application buffer
for (int i = 0; i < outputTensor->bytes / sizeof(float32); i++)
  output_data[i] = outputTensor->data.f[i];
```

NPU inference {#npu_infer .section} Running an inference using a model converted for the NPU requires registration of a custom operator implementation. First the header file with the custom operator implementation interface must be included.

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/micro/kernels/neutron/neutron.h"
```

Next, the specialized implemetation has to be registered in the operator resolver object.

```
static tflite::AllOpsResolver microOpResolver;
microOpResolver.AddCustom(tflite::GetString_NEUTRON_GRAPH(),
        tflite::Register_NEUTRON_GRAPH());
```

The specialized NPU nodes from the converted model are the executed using this newly registered implementation.

Adjusting the tensor arena size {#adjust_arena .section} The tensor arena is a static memory buffer used for intermediate tensor and scratch buffer allocation. The size of the tensor arena buffer is set by the kTensorArenaSize constant in the example above. The value depends on the tensor sizes used in the model and on the hardware-specific implementations of kernels, which may require various sizes of scratch buffers for intermediate computations. The value can be determined experimentally by running an inference with a small value, so the library fails with an insufficient tensor memory error and prints the missing amount. Continue adjusting the size until the error stops being reported. If the target hardware changes, readjust the value.

Code size optimization Typically, models do not use all the operators that are available in TensorFlow Lite. However, because of the default operator registration mechanism used in the library, the toolchain linker is not able to remove the code of unused operators. In order to reduce code size, it is possible to only register the specific operators used by a model. To determine which operators are used by a particular model, a model visualizer tool like Netron can be used. Then a mutable operator resolver object can be created that only registers the operators that are used by the model being inferenced.

Use the tflite::MicroMutableOpResolver object template, which is later passed to the tflite::MicroInterpreter object. Depending on the list of used operators, the result should be similar to the following code snippet. Make sure to update the MicroMutableOpResolver template parameter to reflect the number of operators that need to be registered.

Register only used operators in TensorFlow Lite Micro {#SECTION_SS1_DJQ_QPB .section}

```
#include "tensorflow/lite/micro/kernels/micro_ops.h"
        #include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
        tflite::MicroMutableOpResolver<6> microOpResolver;
        microOpResolver.AddAveragePool2D();
        microOpResolver.AddConv2D();
        microOpResolver.AddDepthwiseConv2D();
```

(continues on next page)

(continued from previous page)

```
microOpResolver.AddDequantize();
microOpResolver.AddReshape();
microOpResolver.AddSoftmax();
static tflite::MicroInterpreter interpreter(
model, microOpResolver, tensorArena, kTensorArenaSize, microErrorReporter);
```

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

eIQ ExecuTorch Library User Guide

Overview ExecuTorch is an end-to-end solution for enabling on-device inference capabilities across mobile and edge devices including wearables, embedded devices and microcontrollers. It is part of the PyTorch Edge ecosystem and enables efficient deployment of PyTorch models to edge devices. For more information, see https://pytorch.org/executorch-overview.

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides a comprehensive software package with a pre-integrated ExecuTorch based on version v1.0.0 which includes the Neutron Backend. Neutron Backend enables acceleration of ML models on the eIQ® Neutron Neural Processing Unit (NPU).

This document describes the steps required to download and start using the ExecuTorch. Additionally, the document describes the steps required to create an application for running pretrained models.

Note: The document also assumes knowledge of machine learning frameworks for model training.

Supported platforms:

- i.MX RT700

Installation The ExecuTorch, with the Neutron Backend consists of:

- ExecuTorch with Neutron Backend for Ahead of Time ML Model Compilation
- Neutron Converter
- MCUXpresso SDK

Here we briefly describe each components purpose and steps to install them.

The **ExecuTorch AoT** and **Neutron Converter** are needed to convert a PyTorch model to ExecuTorch and Delegate it to eIQ Neutron NPU using the Neutron Backend. The **MCUXpresso SDK** provides project to build the ExecuTorch Runtime Library, the example application with simple CNN, toolchains and other middleware libraries to build and deploy the application on the target platform.

If you want run to prepared example application on the i.MX RT700 platform, and skip the model preparation phase continue with the *MCUXpresso SDK Part*.

ExecuTorch for Ahead of Time model preparation The ExecuTorch enables to deploy PyTorch models on edge devices. For this purpose the PyTorch model must be processed and converter by the ExecuTorch Ahead of Time (AoT) part. You can obtain the full ExecuTorch including the AoT part aligned with this version of MCUX SDK from the mcuxsdk-middleware-executorch release/mcux-full branch.

Installation Prerequisities:

- x86 Linux Machine with GLIBC-2.29 or higher (e.g. Ubuntu 20.04 or higher)
- Python 3.10, 3.11 or 3.12

To build and install the ExecuTorch follow these steps:

1. (Optional) Setup python virtual environment on desired location and activate it.

```
$ python3 -m venv venv
$ source venv/bin/activate
```

2. Clone the ExecuTorch from mcuxsdk-middleware-executorch

```
$ git clone --branch release/mcux-full https://github.com/nxp-mcuxpresso/mcuxsdk-middleware-executorch.git
$ cd mcuxsdk-middleware-executorch
$ git submodule update --init --recursive
```

3. Build and install the ExecuTorch and its dependencies:

```
$ ./install_executorch.sh
```

[!WARNING] The install_requirements.sh installs the CPU version of torch from https://download.pytorch.org/whl/cpu. If you are behind corporate proxy, it might have issues accessing it and you will see warnings like:

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,␣
↪status=None)) after connection broken by 'SSLError(SSLCertVerificationError(1, '[SSL:␣
↪CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer␣
↪certificate (_ssl.c:1006)'))': /whl/test/cpu/torch/
```

In this case the CUDA version of torch is installed and the install_requirements.sh script fails with:

```
PyTorch: CUDA cannot be found.  Depending on whether you are building
```

Make sure the pip can access the https://download.pytorch.org/whl/cpu PyPI.

Next continue with installation of the *Neutron Converter*

Neutron Converter The eIQ Neutron Backend uses the Neutron Converter to convert the ExecuTorch program to the eIQ Neutron NPU microcode.

Installation The Neutron Converter is available as a Python package and can be installed by the pip command from eiq.nxp.com/repository:

```
pip install --index-url https://eiq.nxp.com/repository neutron_converter_SDK_25_12==1.0.0
```

Or you can use the prepared setup script:

```
./examples/nxp/setup.sh
```

The Neutron Converter is used internally by the ExecuTorch, and it is tied to the particular BSP you are using - the suffix of the python package name. In the code snippet above the flavor is the SDK_25_12. In the *aot_neutron_convert.py* example script by the --neutron_converter_flavor parameter.

MCUXpresso SDK The MCUXpresso SDK is used to build, debug and deploy the application using the ExecuTorch on the target platform.

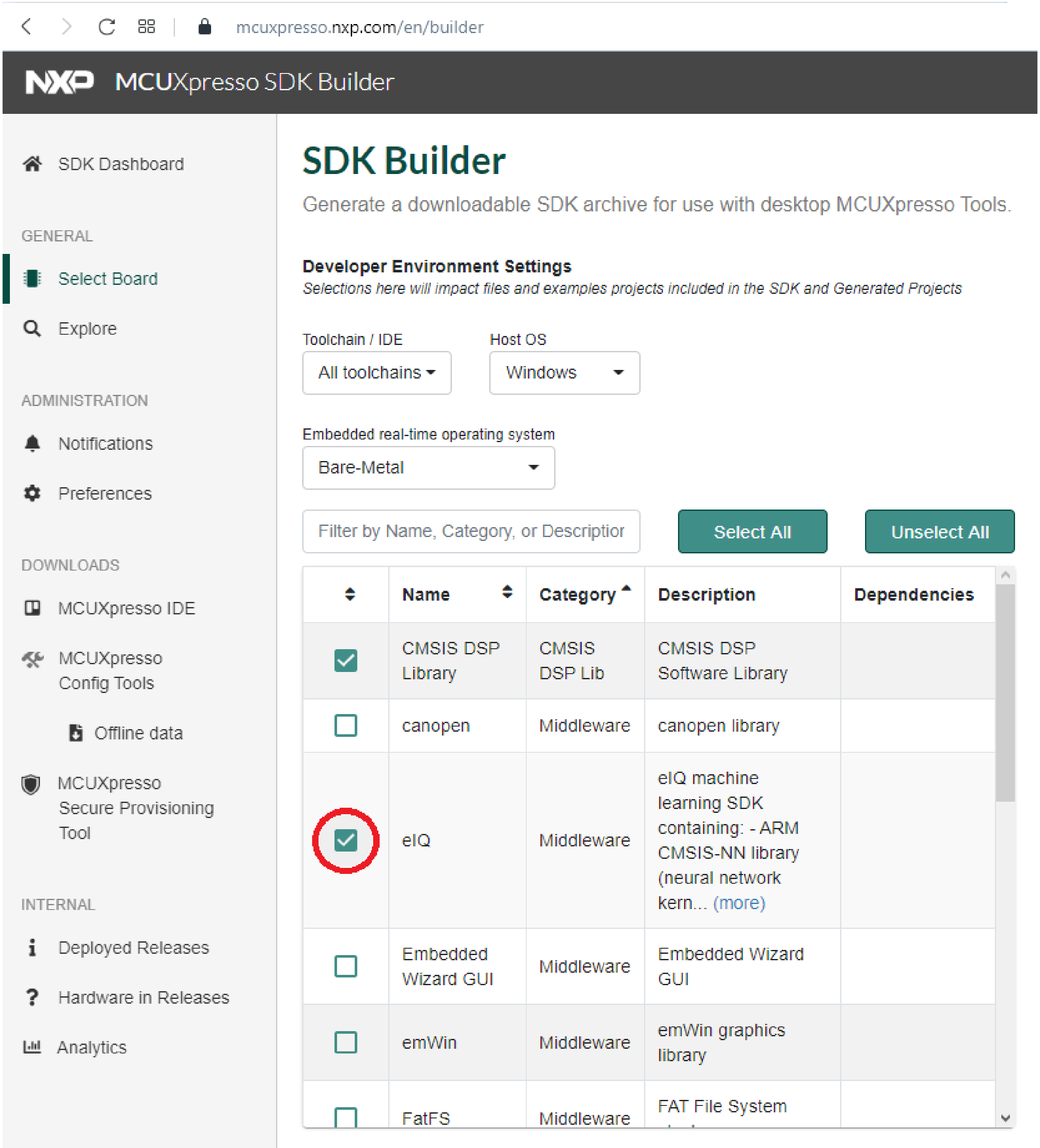
You can obtain the MCUXpresso SDK from MCUXpresso SDK Builder including the IDE. See the *getting_mcuxpress* for details.

In the MCUXpresso SDK, there are 2 projects available related to ExecuTorch:

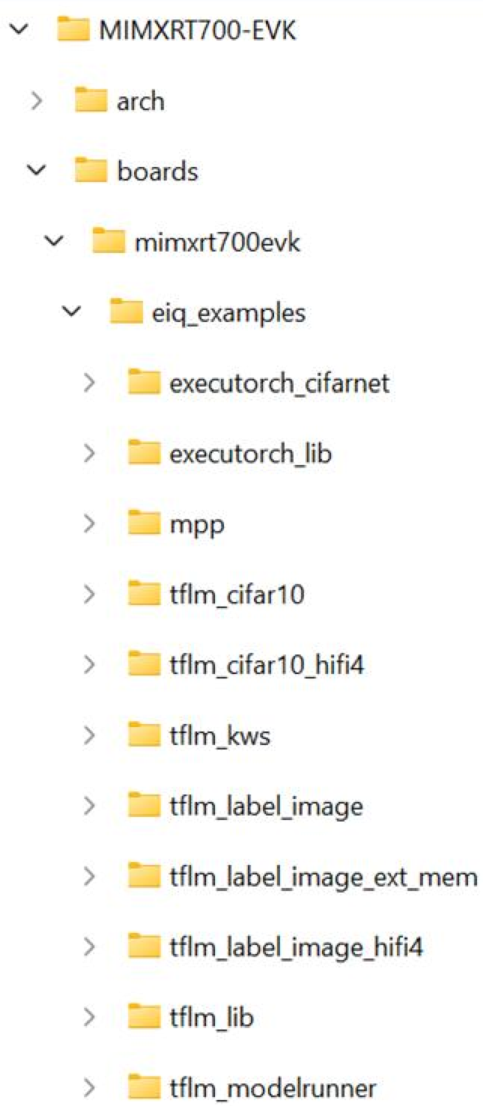
- executorch_lib
- executorch_cifarnet

For more details see *example_applications*. Here you will find the details to run build and run the demo applications.

Getting the MCUXpresso SDK with eIQ ExecuTorch The eIQ ExecuTorch library is part of the eIQ machine learning software package, which is an optional middleware component of MCUXpresso SDK. The eIQ component is integrated into the MCUXpresso SDK Builder delivery system available on mcuxpresso.nxp.com. To include eIQ machine learning into the MCUXpresso SDK package, the eIQ middleware component is selected in the software component selector on the SDK Builder page when building a new package:



Once the MCUXpresso SDK package is downloaded, it can be extracted on a local machine or imported into the MCUXpresso IDE. For more information on the MCUXpresso SDK folder structure, see the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG). The package directory structure is similar to figure bellow:



The *boards* directory contains example application projects for supported toolchains. For the list of supported toolchains, see the *MCUXpresso SDK Release Notes*. The *middleware* directory contains the eIQ library source code and example application source code and data.

PyTorch Model Conversion to ExecuTorch Format In this guideline we will show how to use the ExecuTorch AoT part to convert a PyTorch model to ExecuTorch format and delegate the model computation to eIQ Neutron NPU using the eIQ Neutron Backend.

First we will start with an example script converting the model. This example shows the CifarNet model preparation. It is the same model which is part of the `example_cifarnet`.

1. Run the `aot_neutron_compile.py` example with the `cifar10` model. As the `aot_neutron_compile.py` is already installed as part of the ExecuTorch installation we will run it from there

```
$ python -m examples.nxp.aot_neutron_compile --quantize \
        --delegate --neutron_converter_flavor SDK_25_12 -m cifar10
```

2. It will generate you `cifar10_nxp_delegate.pte` file which can be used with the MXUXpresso SDK `cifarnet_example` project.

The generated PTE file is used in the executorch_cifarnet example application, see *example_application*.

MCUXpresso SDK Example applications The MCUXpresso SDK provides a set of projects and example application with the eIQ ExecuTorch. These demonstrate the functionality of the ExecuTorch with the Neutron Backend, or enable to build the executorch library itself, if code changes or customization is needed. See table bellow:

Name	Description	Availability
executorch_	This project contains the ExecuTorch Runtime Library source code and is used to build the ExecuTorch Runtime Library. The library is further used to build a full application using the leveraging ExecuTorch.	MIMXRT700-EVK (no camera and display support)
executorch_	Example application demonstrating the use of the ExecuTorch running a CifarNet classification model accelerated on the eIQ Neutron NPU. The Cifarnet is a small Convolutional Neural Network (CNN), trained on CIFAR-10 [1] dataset. The model clasifies the input images into 10 caterories.	MIMXRT700-EVK (no camera and display support)

For details on how to build and run the example applications with supported toolchains, see *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG).

How to build and run `executorch_cifarnet` **example** The example needs ExecuTorch Runtime Library and Neutron Libraries.

ExecuTorch Runtime Library:

- `middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a` for Cortex-M33 Core
- `middleware/eiq/executorch/lib/hifi4/xcc/imxrt700/libexecutorch.a` for HiFi4 Core

Neutron Libraries:

- Cortex-M33
  - `/middleware/eiq/neutron/rt700/cm33/libNeutronDriver.a` and
  - `/middleware/eiq/neutron/rt700/cm33/libNeutronFirmware.a`
- HiFi4 DSP
  - `/middleware/eiq/neutron/rt700/hifi4/libNeutronDriver.a` and
  - `/middleware/eiq/neutron/rt700/hifi4/libNeutronFirmware.a`

In the example the model and the input image is already embedded into the program and ready to build and deploy to i.MX RT700, so you can continue right to the *building and deployment* section.

Convert the model and example input to C array In this section we describe where the model and example input is located in the example application sources, and how it was generated.

The **cifar10 model** ExecuTorch model is stored in `boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/model_pte.h`. and was generated from the `cifar10_nxp_delegate.pte` (see *convert_model*).

We use the `xxd` command to get the C array containing the model data and array size:

```
$ xxd -i cifar10_nxp_delegate.pte > model_pte_data.h
```

then use the array data and size in the `model_pte.h`.

As **input image** we use the image from CIFAR-10 dataset [1]. After preprocessing and normalization it is converted to bytes and located here `boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/image_data.h`. The preprocessing is performed as follows:

```
import torch
import torchvision
import numpy as np

batch_size = 1

transform = torchvision.transforms.Compose([
  torchvision.transforms.ToTensor(),
  torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)

index = 0
num_images = 10
for data in test_loader:
  images, labels = data
  for image, label in zip(images,labels):
    arr = image.numpy().astype(np.float32)
    arr.tofile("img" + str(index) + "_" + str(int(label)) + ".bin")
    index = index + 1
    if index >= num_images:
      break
  if index >= num_images:
    break
```

This generates the `num_images` count of images from Cifar10 dataset, as input tensors for the cifar10 model and store them in corresponding `.bin` files. Then we use the xxd command to get the C array data and size:

```
$ xxd -i img0_3.bin > image_data_base.h
```

and again copy the array data and size in the `image_data.h`

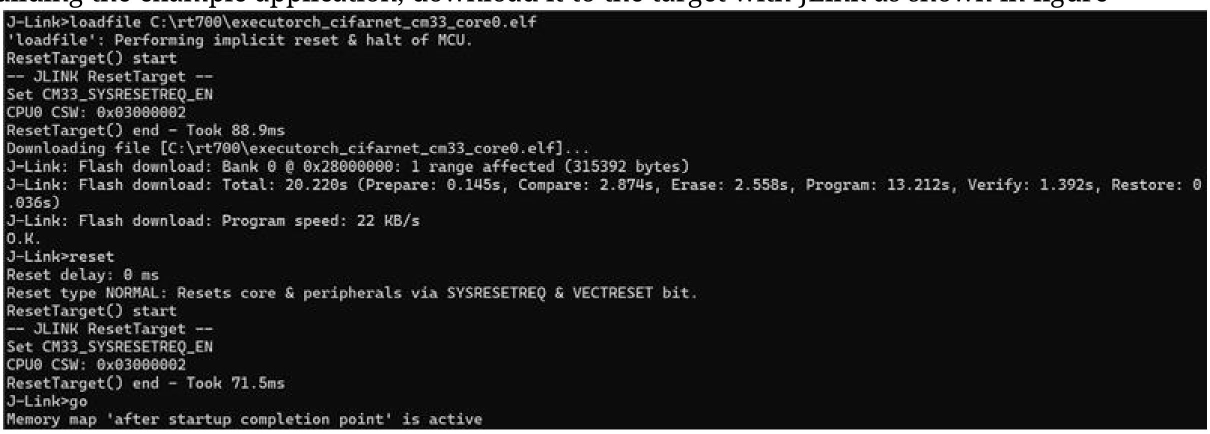
Note, the `img0` is the image picturing a cat, what is a class number 3.

Build, Deploy and Run

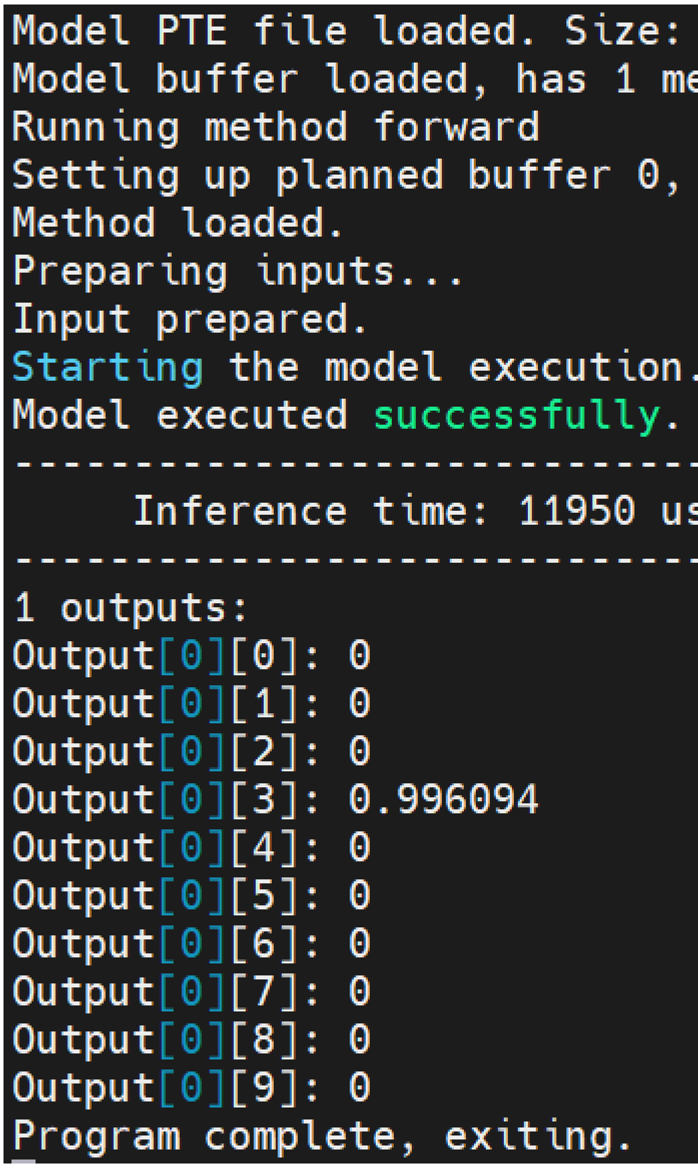
1. When using ARMGCC toolchain, the example application can be built as below:

```
$ boards/mimxrt700evk/eiq_examples/executorch_cifarnet/cm33_core0/armgcc$ ./build_flash_release.sh
```

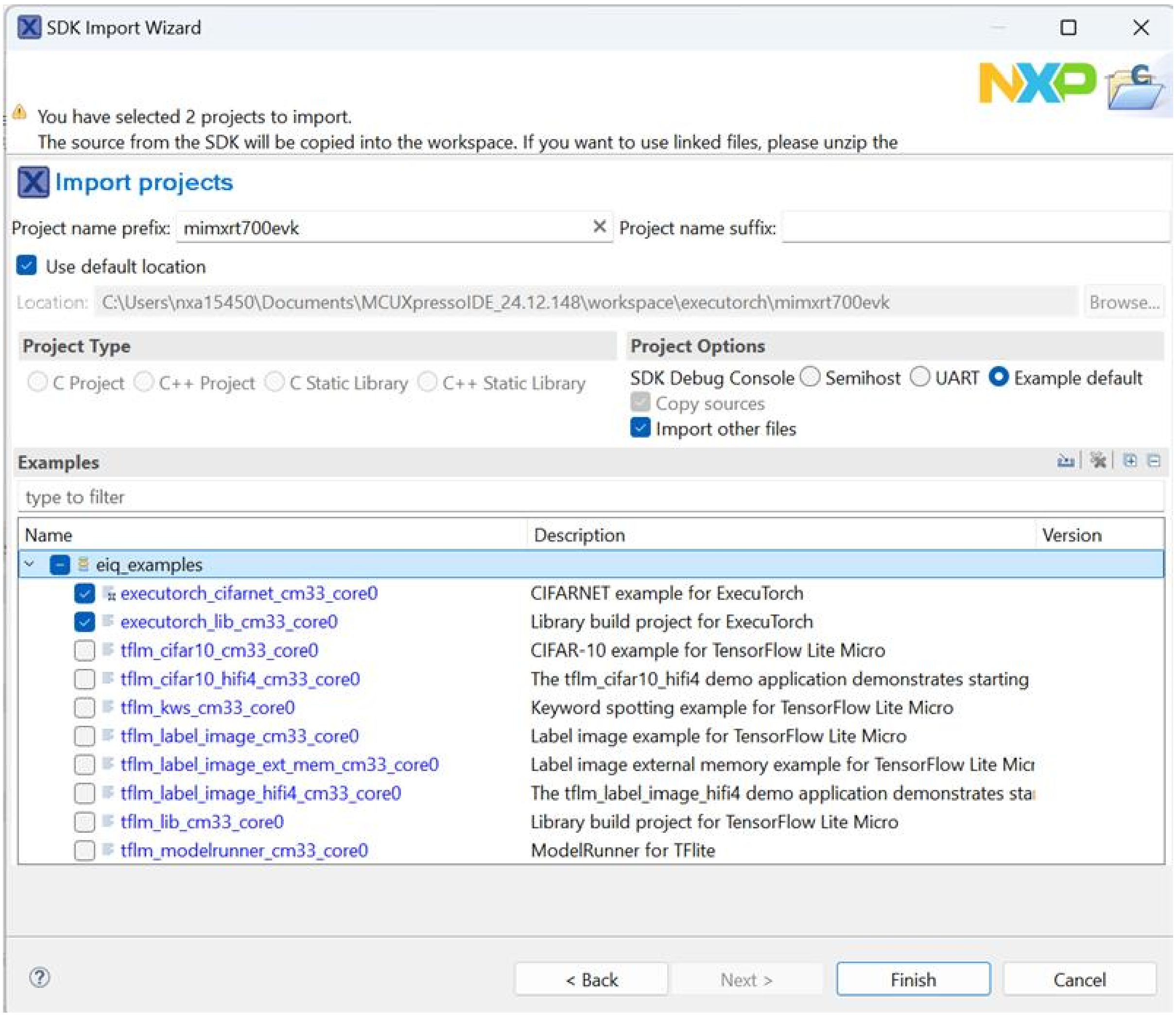
After building the example application, download it to the target with JLink as shown in figure bellow.



The output message displays on the connected terminal:



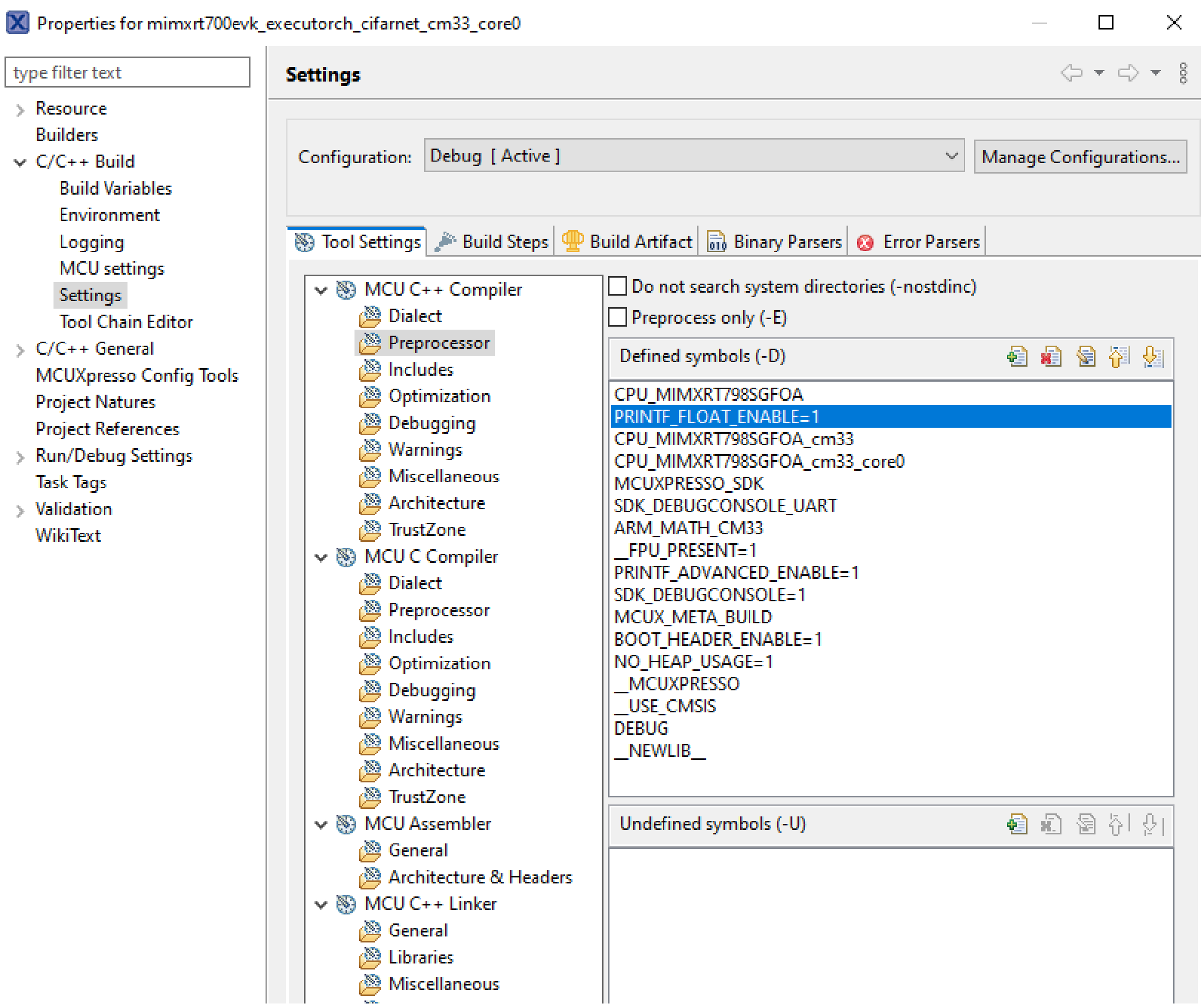
2. When using MCUXpresso IDE, the example applications can be imported through the SDK Import Wizard:



After building the example application and downloading it to the target, the execution stops in the *main* function. When the execution resumes, an output message displays on the connected terminal. For example, bellow figure shows the output of the executorch_cifarnet example application:

```
Model PTE file loaded. Size: 99376 bytes.
Model buffer loaded, has 1 methods
Running method forward
Setting up planned buffer 0, size 53760.
Method loaded.
Preparing inputs...
Input prepared.
Starting the model execution...
Model executed successfully.
----------------------------------------
     Inference time: 14855 us
----------------------------------------
1 outputs:
Output[0][0]: 0
Output[0][1]: 0
Output[0][2]: 0
Output[0][3]: 0.996094
Output[0][4]: 0
Output[0][5]: 0
Output[0][6]: 0
Output[0][7]: 0
Output[0][8]: 0
Output[0][9]: 0
Program complete, exiting.
```

In case of missing probabilities in the printed output, add PRINTF_FLOAT_ENABLE=1 to the Preprocessor settings for C++ and C compiler:

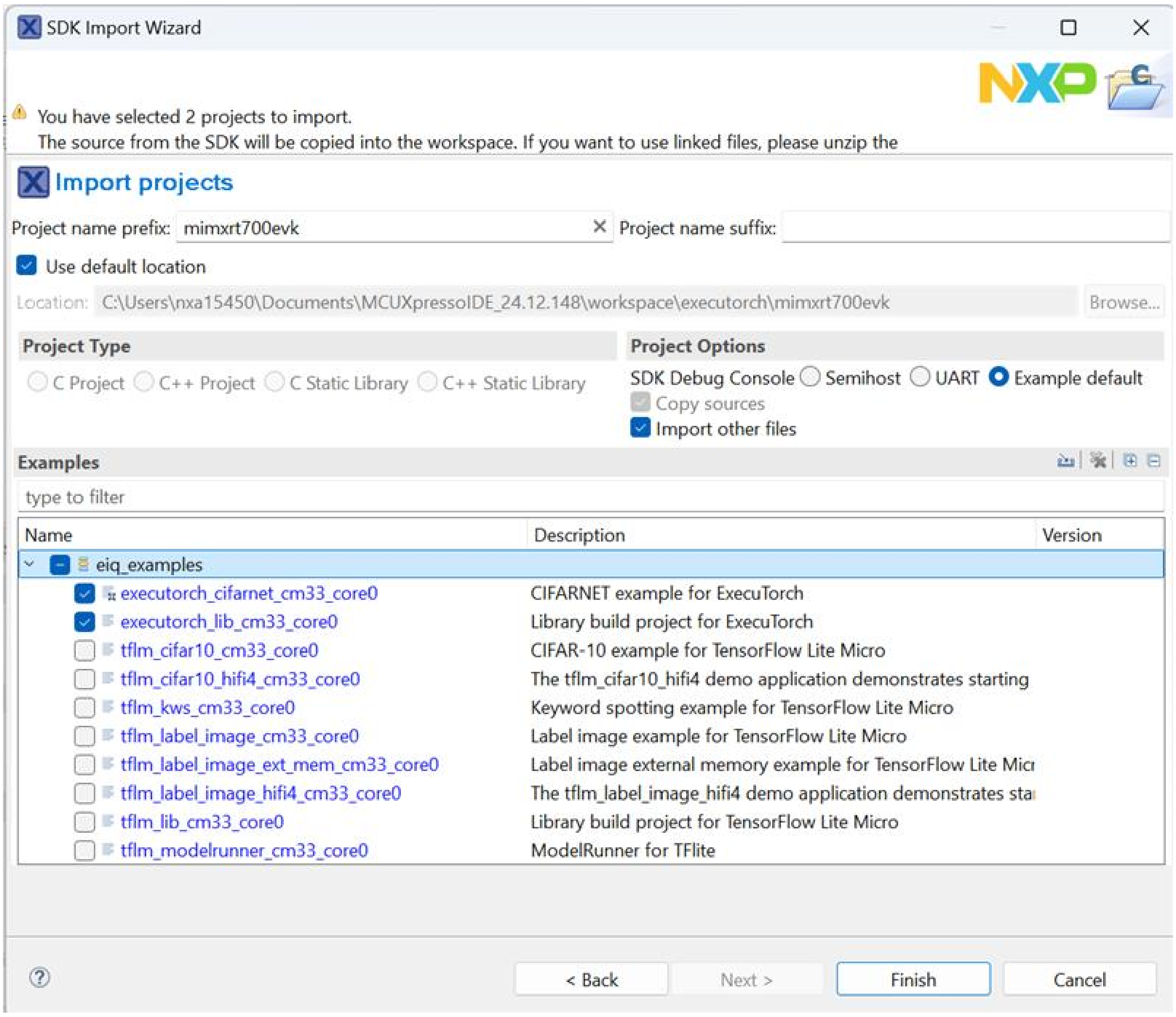


How to build executorch_lib **example** If you want to build a new ExecuTorch Runtime Library, follow the commands as below and use the new library to replace the default Runtime library middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a.

1. When using ARMGCC toolchain, the example application can be built as below.

```
$ boards/mimxrt700evk/eiq_examples/executorch_lib/cm33_core0/armgcc$ ./build_release.sh
$ boards/mimxrt700evk/eiq_examples/executorch_lib/cm33_core0/armgcc$ cp release/libexecutorch_lib_
↪cm33_core0.a ../../../../../../middleware/eiq/executorch/lib/cm33/armgcc/libexecutorch.a
```

2. When using MCUXpresso IDE, you can import the project directly to the IDE through the SDK Import Wizard. The project can be found under eiq_examples:



After building the example application, copy the new library mimxrt700evk_executorch_lib_cm33_core0\Debug\libmimxrt700evk_executorch_lib_cm33_core0.a to replace the default Runtime library mimxrt700evk_executorch_cifarnet_cm33_core0\eiq\executorch\lib\cm33\armgcc\libexecutorch.a.

[1] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009

1.4 File System

1.4.1 FatFs

MCUXpresso SDK : mcuxsdk-middleware-fatfs

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

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

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

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

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

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

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

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

Changelog FatFs

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

The format is based on Keep a Changelog

[R0.15_rev0]

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

[R0.14b_rev1]

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

[R0.14b_rev0]

- Upgraded to version 0.14b

[R0.14a_rev0]

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

[R0.14_rev0]

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

[R0.13c_rev0]

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

[R0.13b_rev0]

- Upgraded to version 0.13b

[R0.13a_rev0]

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

[R0.12c_rev1]

- Add NAND disk support.

[R0.12c_rev0]

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

[R0.12b_rev0]

- Upgraded to version 0.12b.

[R0.11a]

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

1.5 Motor Control

1.5.1 FreeMASTER

Communication Driver User Guide

Introduction

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

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

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

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

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

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

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

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

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

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

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

- **General CPU Platform** (see source code in the src/platforms directory). The code in this layer is only specific to native data type sizes and CPU architectures (for example; alignment-aware memory copy routines). This driver version brings two generic implementations of 32-bit platforms supporting both little-endian and big-endian architectures. There are also implementations customized for the 56F800E family of digital signal controllers and S12Z MCUs. **Zephyr** is treated as a specific CPU platform as it brings unified user configuration (Kconfig) and generic hardware device drivers. With Zephyr, the transport layer and low-level communication layers described below are configured automatically using Kconfig and Device Tree technologies.

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

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

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

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

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

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

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

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

Related links:

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

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

Example applications

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

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

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

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

Description

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

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

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

The FreeMASTER driver features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Zephyr-specific

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

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

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

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

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

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

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

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

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

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

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

Interrupt modes

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

Value Type boolean (0 or 1)

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

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

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

Protocol transport

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

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

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

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

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

```
#define FMSTR_TRANSPORT FMSTR_SERIAL
```

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

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

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

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

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

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

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

FMSTR_SERIAL_BASE

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

Value Type Optional address value (numeric or symbolic)

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

FMSTR_COMM_BUFFER_SIZE

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

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

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

FMSTR_COMM_RQUEUE_SIZE

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

Value Type Value in range 0...255

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

FMSTR_SERIAL_SINGLEWIRE

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

Value Type Boolean 0 or 1.

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

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

```
#define FMSTR_TRANSPORT FMSTR_CAN
```

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

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

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

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

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

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

FMSTR_CAN_BASE

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

Value Type Optional address value (numeric or symbolic)

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

FMSTR_CAN_CMDID

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

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

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

FMSTR_CAN_RSPID

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

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

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

FMSTR_FLEXCAN_TXMB

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

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

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

FMSTR_FLEXCAN_RXMB

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

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

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

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

```
#define FMSTR_TRANSPORT FMSTR_NET
```

FMSTR_NET_DRV Select network interface implementation.

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

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

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

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

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

Add another row below:

FMSTR_NET_PORT

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

Value Type TCP or UDP port number (short integer)

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

FMSTR_NET_BLOCKING_TIMEOUT

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

Value Type Timeout as number of milliseconds

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

FMSTR_NET_AUTODISCOVERY

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

Value Type Boolean 0 or 1.

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

Debugging options

FMSTR_DISABLE

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

Value Type boolean (0 or 1)

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

FMSTR_DEBUG_TX

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

Value Type Boolean 0 or 1.

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

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

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

FMSTR_APPLICATION_STR

```
#define FMSTR_APPLICATION_STR
```

Value Type String.

Description Name of the application visible in FreeMASTER host application.

Memory access

FMSTR_USE_READMEM

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

Value Type Boolean 0 or 1.

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

FMSTR_USE_WRITEMEM

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

Value Type Boolean 0 or 1.

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

Oscilloscope options

FMSTR_USE_SCOPE

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

Value Type Integer number.

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

FMSTR_MAX_SCOPE_VARS

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

Value Type Integer number larger than 2.

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

Recorder options

FMSTR_USE_RECORDER

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

Value Type Integer number.

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

FMSTR_REC_BUFF_SIZE

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

Value Type Integer number larger than 2.

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

FMSTR_REC_TIMEBASE

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

Value Type Number (nanoseconds time).

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

Use one of the following macros:

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

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

FMSTR_REC_FLOAT_TRIG

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

Value Type Boolean 0 or 1.

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

Default value is 0 (false).

Application Commands options

FMSTR_USE_APPCMD

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

Value Type Boolean 0 or 1.

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

FMSTR_APPCMD_BUFF_SIZE

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

Value Type Numeric buffer size in range 1..255

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

FMSTR_MAX_APPCMD_CALLS

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

Value Type Number in range 0..255

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

TSA options

FMSTR_USE_TSA

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

Value Type Boolean 0 or 1.

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

FMSTR_USE_TSA_SAFETY

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

Value Type Boolean 0 or 1.

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

FMSTR_USE_TSA_INROM

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

Value Type Boolean 0 or 1.

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

FMSTR_USE_TSA_DYNAMIC

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

Value Type Boolean 0 or 1.

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

Pipes options

FMSTR_USE_PIPES

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

Value Type Boolean 0 or 1.

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

FMSTR_MAX_PIPES_COUNT

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

Value Type Number in range 1..63.

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

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

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

Completely Interrupt-Driven operation Activated using:

```
#define FMSTR_LONG_INTR 1
```

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

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

Mixed Interrupt and Polling Modes Activated using:

```
#define FMSTR_SHORT_INTR 1
```

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

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

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

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

Completely Poll-driven

```
#define FMSTR_POLL_DRIVEN 1
```

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

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

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

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

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

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

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

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

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

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

FreeMASTER Recorder calls When using the FreeMASTER Recorder in the application (FMSTR_USE_RECORDER > 0), call the FMSTR_RecorderCreate function early after FMSTR_Init to set up each recorder instance to be used in the application. Then call the FMSTR_Recorder function periodically in the code where the data recording should occur. A typical place to call the Recorder routine is at the timer or PWM interrupts, but it can be anywhere else. The example applications provided together with the driver code call the FMSTR_Recorder in the main application loop.

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

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

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

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

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

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

Driver API

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

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

FMSTR_Init

Prototype

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

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

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

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

FMSTR_Poll

Prototype

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

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

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

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

*N * Tchar*

where:

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

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

FMSTR_SerialIsr / FMSTR_CanIsr

Prototype

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

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

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

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

Recorder API

FMSTR_RecorderCreate

Prototype

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

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

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

For more information, see *Configurable items*.

FMSTR_Recorder

Prototype

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

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

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

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

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

FMSTR_RecorderTrigger

Prototype

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

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

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

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

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

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

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

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

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

```
FMSTR_TSA_TABLE_BEGIN(table_id)
```

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

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

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

(continues on next page)

(continued from previous page)

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

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

The table is closed using the FMSTR_TSA_TABLE_END macro:

```
FMSTR_TSA_TABLE_END()
```

TSA descriptor parameters The TSA descriptor macros accept these parameters:

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

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

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

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

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

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

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

```
FMSTR_TSA_TABLE_LIST_BEGIN()

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

The list is closed with the FMSTR_TSA_TABLE_LIST_END macro:

```
FMSTR_TSA_TABLE_LIST_END()
```

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

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

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

```
FMSTR_TSA_TABLE_BEGIN(files_and_links)

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

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

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

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

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

FMSTR_TSA_TABLE_END()
```

TSA API

FMSTR_SetUpTsaBuff

Prototype

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

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

Arguments

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

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

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

FMSTR_TsaAddVar

Prototype

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

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

Arguments

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

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

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

Application Commands API

FMSTR_GetAppCmd

Prototype

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

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

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

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

FMSTR_GetAppCmdData

Prototype

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

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

Arguments

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

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

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

FMSTR_AppCmdAck

Prototype

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

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

Arguments

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

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

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

FMSTR_AppCmdSetResponseData

Prototype

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

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

Arguments

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

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

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

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

FMSTR_RegisterAppCmdCall

Prototype

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

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

Arguments

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

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

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

The prototype of the callback function is

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

Where:

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

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

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

Pipes API

FMSTR_PipeOpen

Prototype

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

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

Arguments

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

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

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

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

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

FMSTR_PipeClose

Prototype

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

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

Arguments

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

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

FMSTR_PipeWrite

Prototype

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

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

Arguments

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

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

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

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

FMSTR_PipeRead

Prototype

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

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

Arguments

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

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

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

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

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

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

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

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

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

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

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

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

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

Document references

Links

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

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

Documents

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

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

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

1.6 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 25.12.00.
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
  - FRDM-IMXRT1186
- 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
  - FRDM-IMXRT1186

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 <MCSDK_dir>	<MCUXpressoSDK_install_dir>/middleware/multicore/
Documentation	<MCSDK_dir>/mcuxsdk-doc/
Embedded Remote Procedure Call component	<MCSDK_dir>/erpc/
Multicore Manager component	<MCSDK_dir>/mcmgr/
RPMsg-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 (RPMsg-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_dsp
- rpmsg_lite_pingpong_tzm

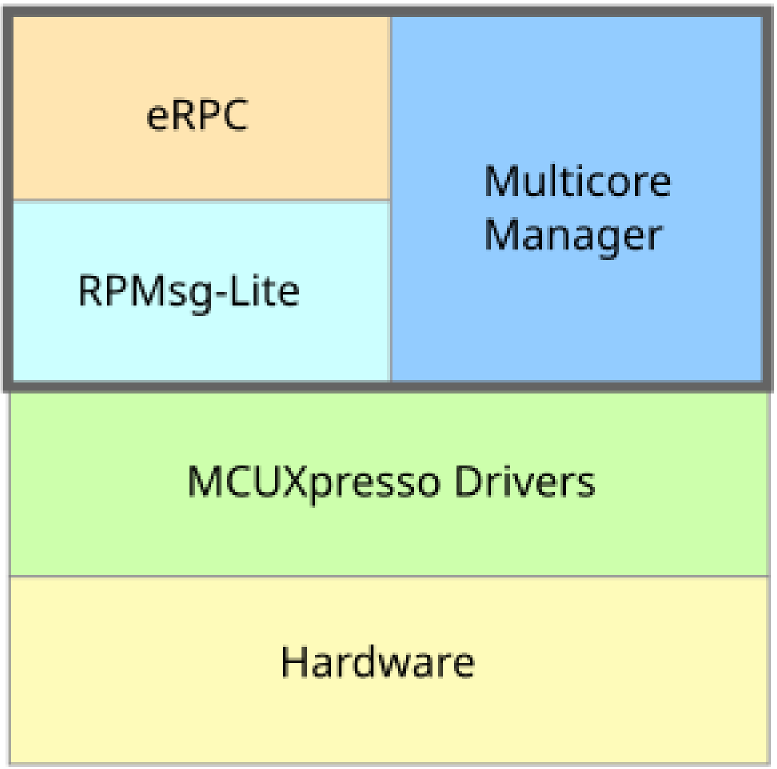
The eRPC multicore component can be leveraged for inter-processor communication and remote procedure calls between SoCs / development boards.
The following multiprocessor demo applications are located together with other MCUXpresso SDK examples in
the <MCUXpressoSDK_install_dir>/examples/multiprocessor_examples 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.

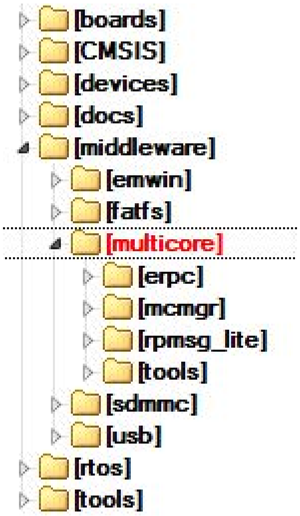


All the MCSDK-related files are located in <MCUXpressoSDK_install_dir>/middleware/multicore folder.

For supported toolchain versions, see the *Multicore SDK v25.12.00 Release Notes* (document MCSDKRN). For the latest version of this and other MCSDK documents, visit www.nxp.com.

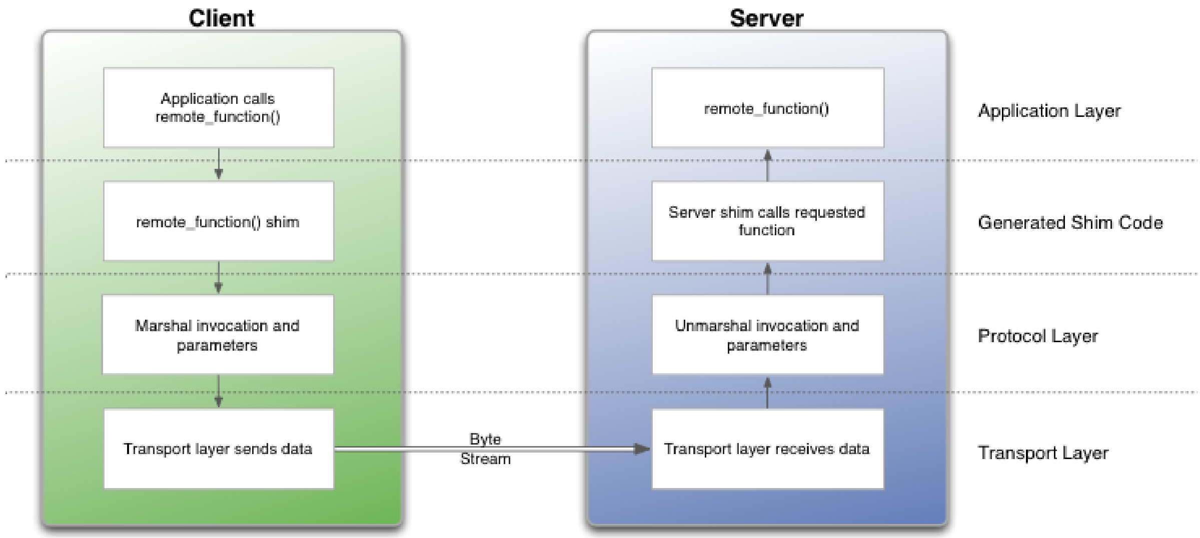
Multicore SDK (MCSDK) components The MCSDK consists of the following software components:

- **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 unmarshaled 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 example projects	<MCUXpressoSDK_install_dir>/examples/multicore_examples/<application_name>/
Multiprocessor example projects	<MCUXpressoSDK_install_dir>/examples/multiprocessor_examples/<application_name>/

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.

[25.12.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0
  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.2
  - RPMsg-Lite v5.3.0

[25.09.00]

- Multicore SDK component versions:
  - embedded Remote Procedure Call (eRPC) v1.14.0

  - eRPC generator (erpcgen) v1.14.0
  - Multicore Manager (MCMgr) v5.0.1
  - RPMsg-Lite v5.2.1

[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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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
  - RPMsg-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 / RPMsg 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 / RPMsg 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.

For Further API documentation, please look at doxygen documentation

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 overview please read RPMSG-Lite VirtIO Overview.

For RPMSG-Lite Design Considerations please read RPMSG-Lite Design Considerations.

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.

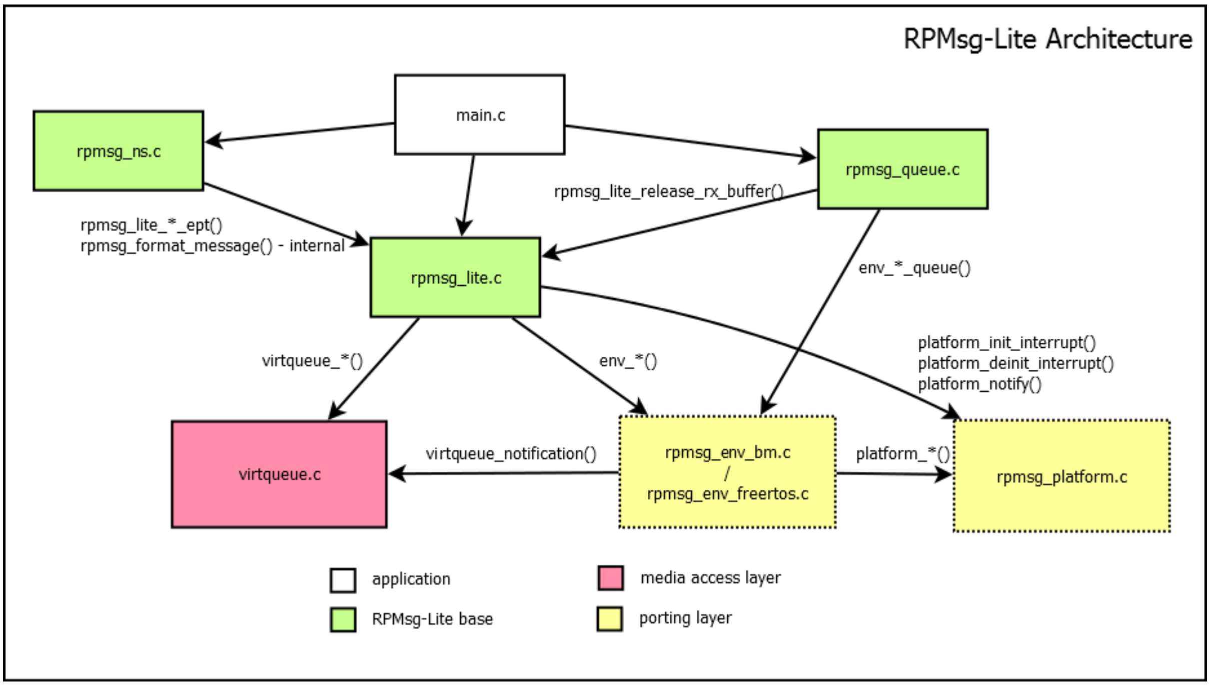
Small MCU-based systems often do not implement dynamic memory allocation. The creation of static API in RPMsg-Lite enables another reduction of resource usage. Not only does the dynamic allocation adds another 5 KB of code size, but also communication is slower and less deterministic, which is a property 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).

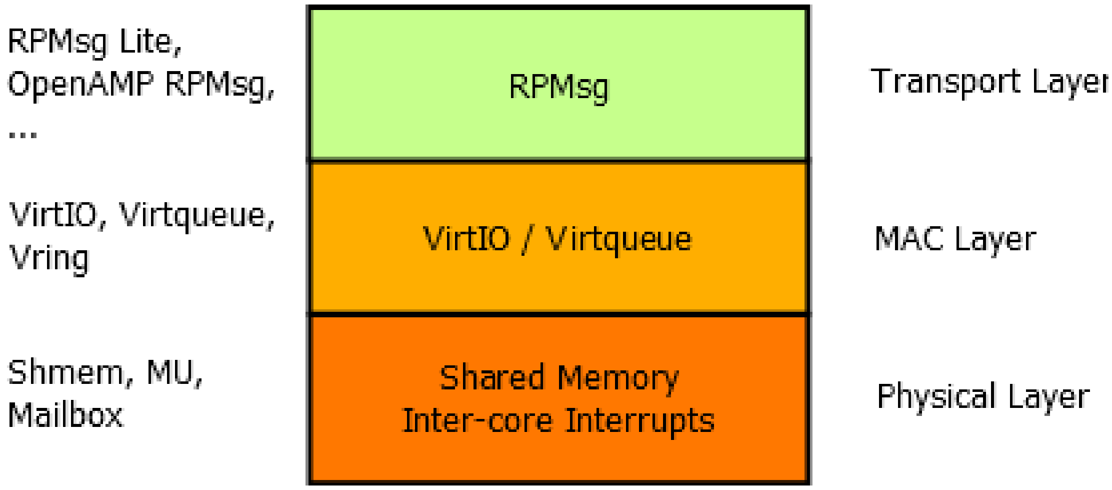
The actual "media access" layer is implemented in virtqueue.c, which is one of the few files shared with the OpenAMP implementation. This layer mainly defines the shared memory model, and internally defines used components such as vring or virtqueue.

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_recv_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 liker file is also a good practise, it is recommended to use linker files from MCUXpressSDK 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.

Configuration option	Default 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 more than 1 byte, and has to be word align (including rpmsg header size 16 bytes), if not it will be aligned up
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 rpmsg_lite_send_nocopy() and rpmsg_lite_release_rx_buffer() functions (enabled by RL_API_HAS_ZEROCOPY 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 rpmsg_config.h 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_ASSER	see rpmsg	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

Changelog RPMSG-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.

[v5.3.0]

Added

- RT700 porting layer added support to send rpmsg messages between CM33_0 <-> Hifi1 and CM33_1 <-> Hifi4 cores.
- Add new platform macro RL_PLATFORM_MAX_ISR_COUNT this will set number of IRQ count per platform. This macro is then used in environment layers to set isr_table size where irq handles are registered. It size should match the bit length of VQ_ID so all combinations can fit into table.
- Unit tests updated to improve code coverage, new unit tests added covering static allocations in rtos environment layers.

Fixed

- virtio.h removed typedef uint8_t boolean and in its place use standard C99 bool type to avoid potential type conflicts.
- env_acquire_sync_lock() and env_release_sync_lock() synchronization primitives removed
- Kconfig consolidation, when RL_ALLOW_CUSTOM_SHMEM_CONFIG enabled the platform_get_custom_shmem_config() function needs to be implemented in platform layer to provide custom shared memory configuration for RPMsg-Lite instance.

v5.2.1

Added

- Doc added RPMSG-Lite VirtIO Overview
- Doc added RPSMG-Lite Design Considerations
- Added frdmimxrt1186 unit testing

Changed

- Remove limitation that RL_BUFFER_SIZE needs to be power of 2. It just has to be more than 16 bytes, e.g. 16 bytes of rpmsg header and payload size at least 1 byte and word aligned, if not it will be aligned up.

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.
- Baremental 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.

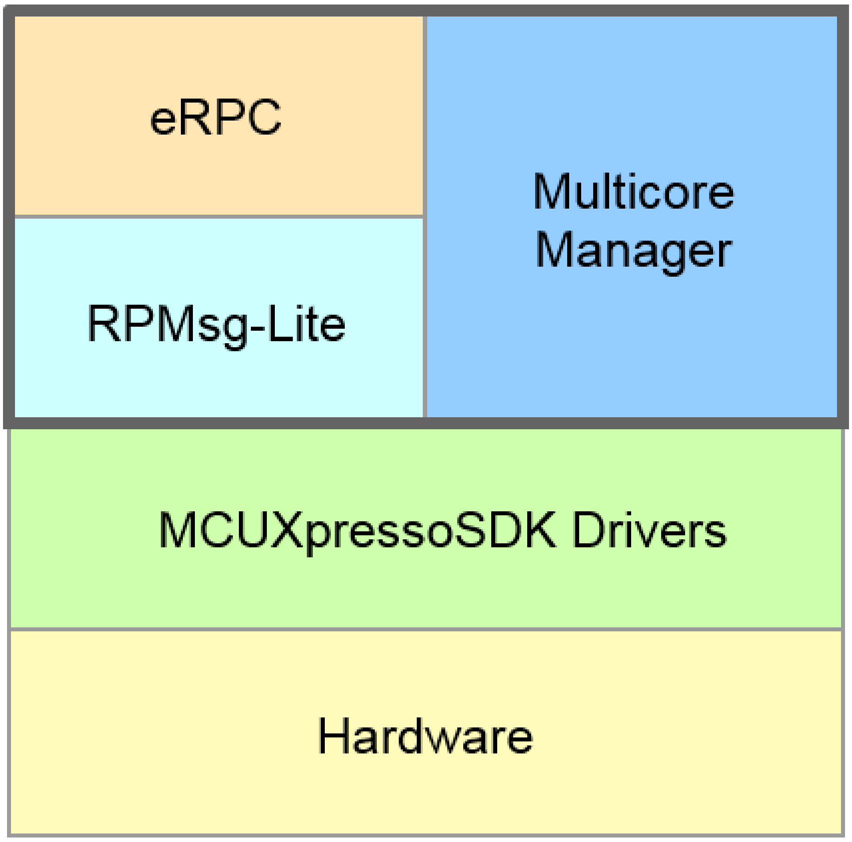
For Further API documentation, please look at doxygen documentation

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();
```

(continues on next page)

(continued from previous page)

```
    /* 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.

It could also happen that the secondary core application stops running correctly and the primary core application does not know about that situation. Therefore, it is beneficial to implement a mechanism for core health monitoring. The *test_heartbeat* unit test can serve as an example how to ensure that: secondary core could periodically send heartbeat signals to the primary core using MCMGR_TriggerEvent() API to indicate that it is alive and functioning properly.

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();

    /* Register the application event before starting the secondary core */
    MCMGR_RegisterEvent(kMCMGR_RemoteApplicationEvent, RPMsgRemoteReadyEventHandler, (void␣
↪*)RPMsgRemoteReadyEventData);
```

(continues on next page)

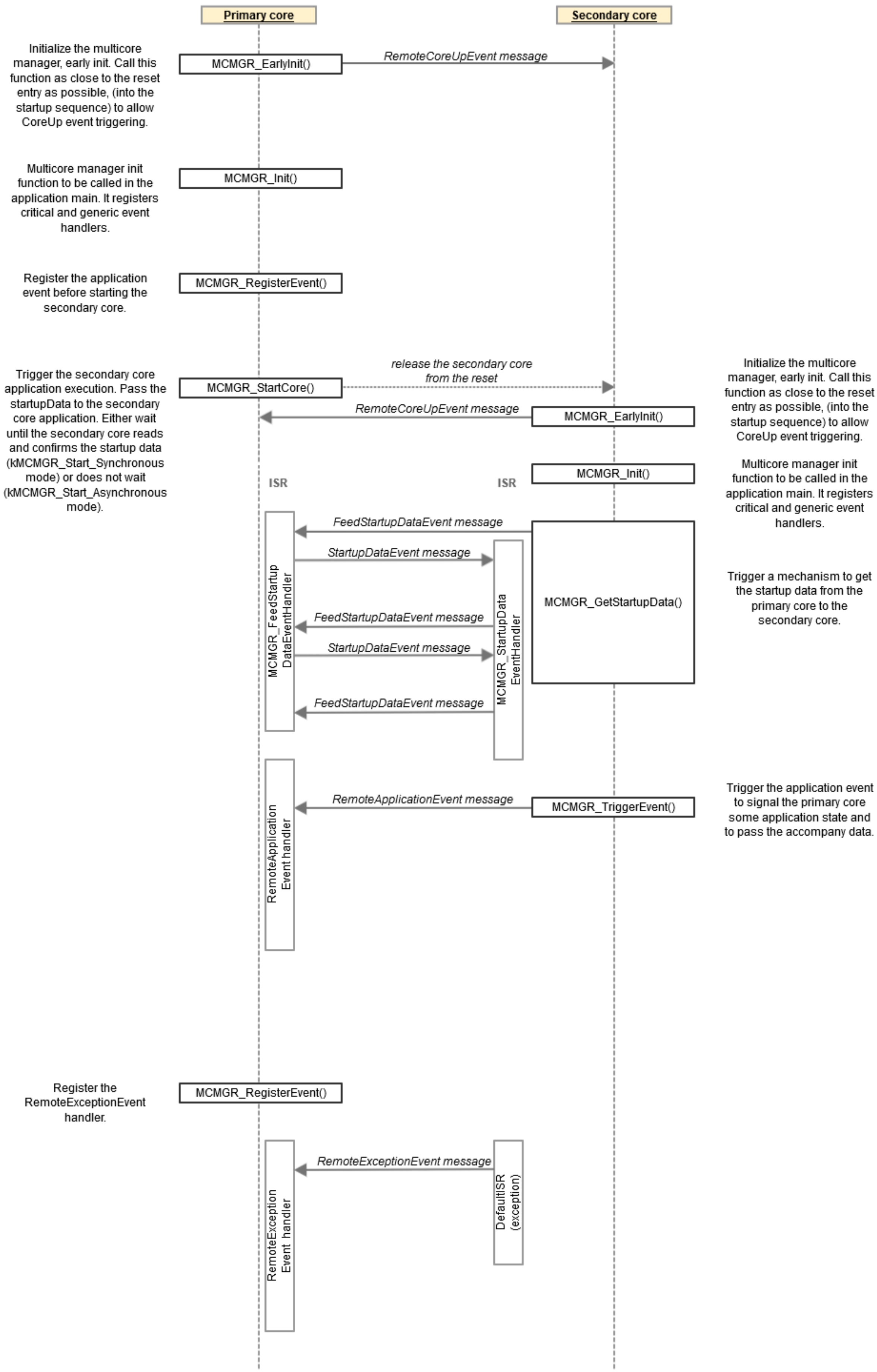
(continued from previous page)

```
    /* Boot secondary core application from the CORE1_BOOT_ADDRESS, pass rpmsg_lite_base address␣
↪as startup data, starting synchronously. */
    MCMGR_StartCore(APP_SECONDARY_CORE, CORE1_BOOT_ADDRESS, (uint32_t)rpmsg_lite_
↪base, kMCMGR_Start_Synchronous);

    /* Wait until the secondary core application signals the rpmsg remote has been initialized and is ready to␣
↪communicate. */
    while(APP_RPMSG_READY_EVENT_DATA != RPMsgRemoteReadyEventData[APP_SECONDARY_
↪CORE]) {};
.
.
.
}
```

Code for the secondary side:

```
#include "mcmgr.h"

#define APP_RPMSG_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_RPMSG_
↪READY_EVENT_DATA */
    MCMGR_TriggerEvent(kMCMGR_Core0, kMCMGR_RemoteApplicationEvent, APP_RPMSG_
↪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.


Primary core
Secondary core
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_EarlyInit()
RemoteCoreUpEvent message
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
MCMGR_Init()
Register the application event before starting the secondary core.
MCMGR_RegisterEvent()
Trigger the secondary core application execution. Pass the startupData to the secondary core application. Either wait until the secondary core reads and confirms the startup data (kMCMGR_Start_Synchronous mode) or does not wait (kMCMGR_Start_Asynchronous mode).
MCMGR_StartCore()
release the secondary core from the reset
RemoteCoreUpEvent message
MCMGR_EarlyInit()
Initialize the multicore manager, early init. Call this function as close to the reset entry as possible, (into the startup sequence) to allow CoreUp event triggering.
MCMGR_Init()
Multicore manager init function to be called in the application main. It registers critical and generic event handlers.
ISR
ISR
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
StartupDataEvent message
FeedStartupDataEvent message
MCMGR_FeedStartup DataEventHandler
MCMGR_StartupData EventHandler
MCMGR_GetStartupData()
Trigger a mechanism to get the startup data from the primary core to the secondary core.
RemoteApplicationEvent message
MCMGR_TriggerEvent()
RemoteApplication Event handler
Trigger the application event to signal the primary core some application state and to pass the accompany data.
Register the RemoteExceptionEvent handler.
MCMGR_RegisterEvent()
RemoteExceptionEvent message
RemoteException Event handler
DefaultISR (exception)

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.

[v5.0.2]

Added

- Added gcov options and configs to support mcmgr code coverage
- Added new test_weak_mu_isr testcase for devices with MU peripheral
- Added new test_heartbeat testcase showing heartbeat mechanism between primary and secondary cores using the MCMGR

v5.0.1

Added

- Added frdmimxrt1186 unit testing

Changed

- [KW43] Rename core#1 reset control register

Fixed

- Added CX flag into CMakeLists.txt to allow c++ build compatibility.
- Fix path to mcmgr headers directory in doxyfile

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 then 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: PortingGuideTo_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

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);
```

(continues on next page)

(continued from previous page)

```
    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 */
    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
```

(continues on next page)

(continued from previous page)

```
{
    /* 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 */
    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 by 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**, …
- **Binson** - 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 configurate 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 configurate 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.

---

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.

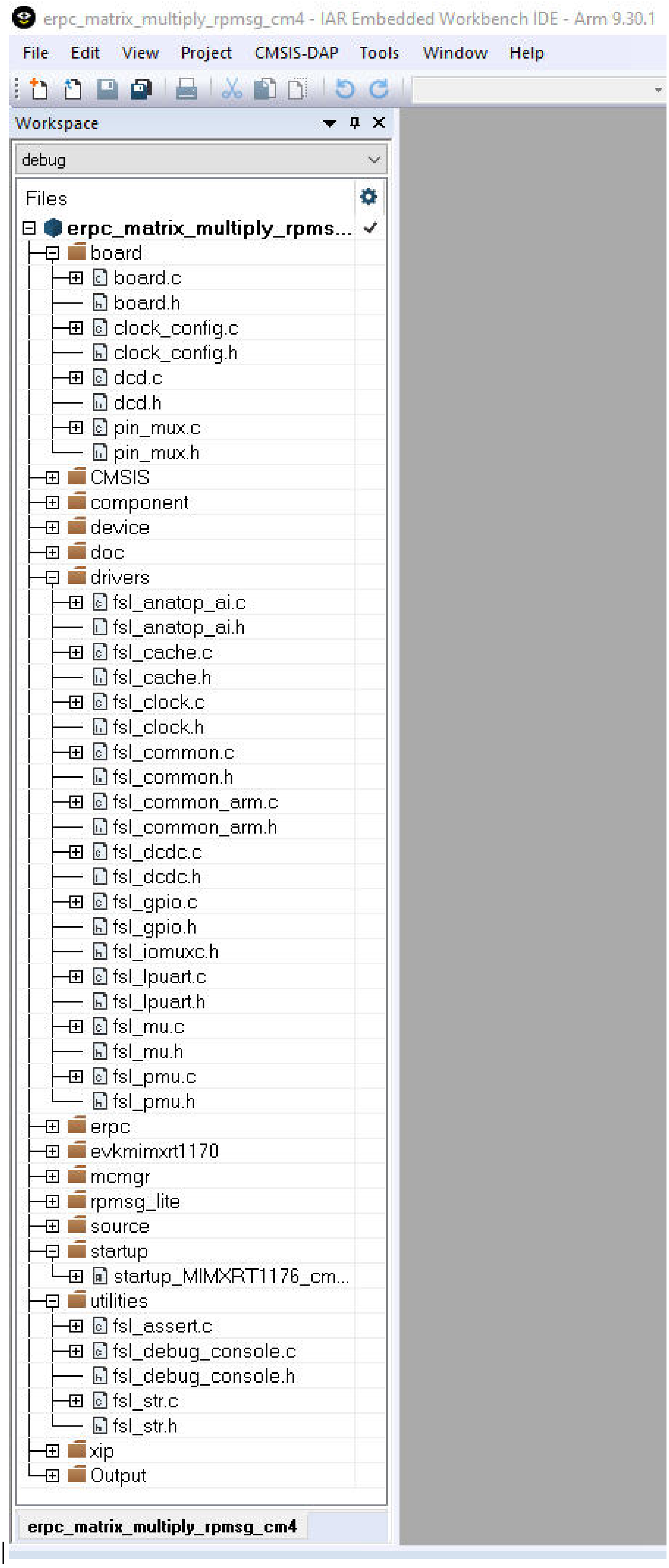
Multicore server application The “Matrix multiply” eRPC server project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/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>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

Parent topic: Multicore server application

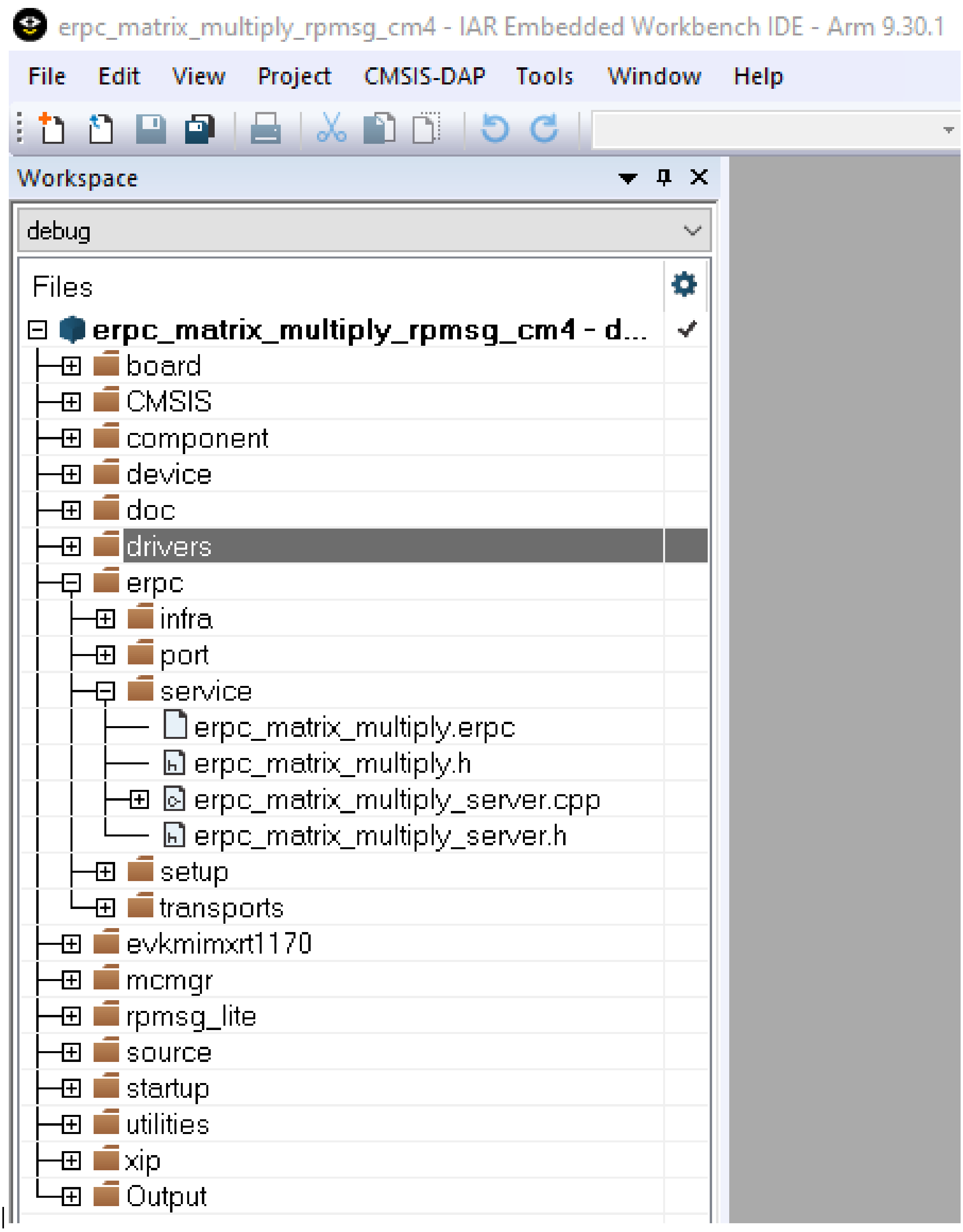
Server related generated files

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/s



|

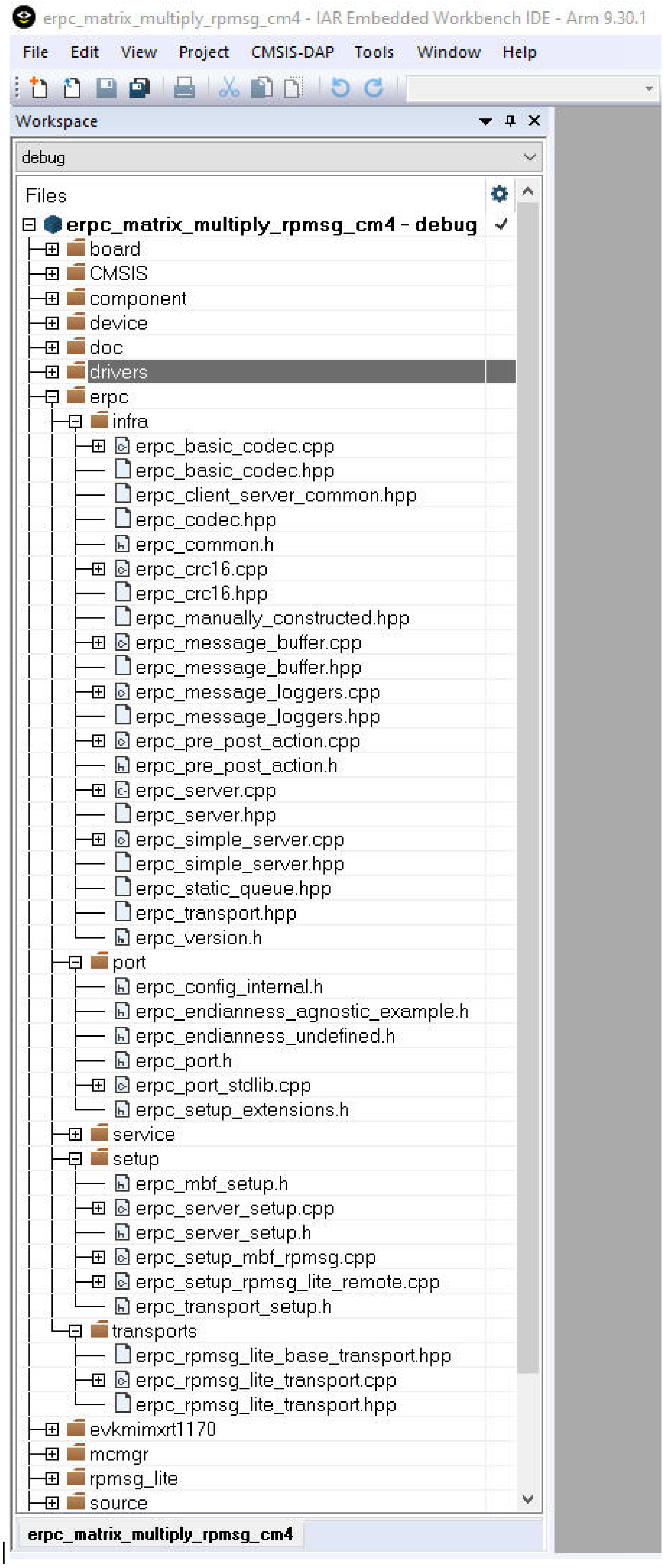
Parent topic: Multicore server application

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 needs 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 needs 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 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 need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

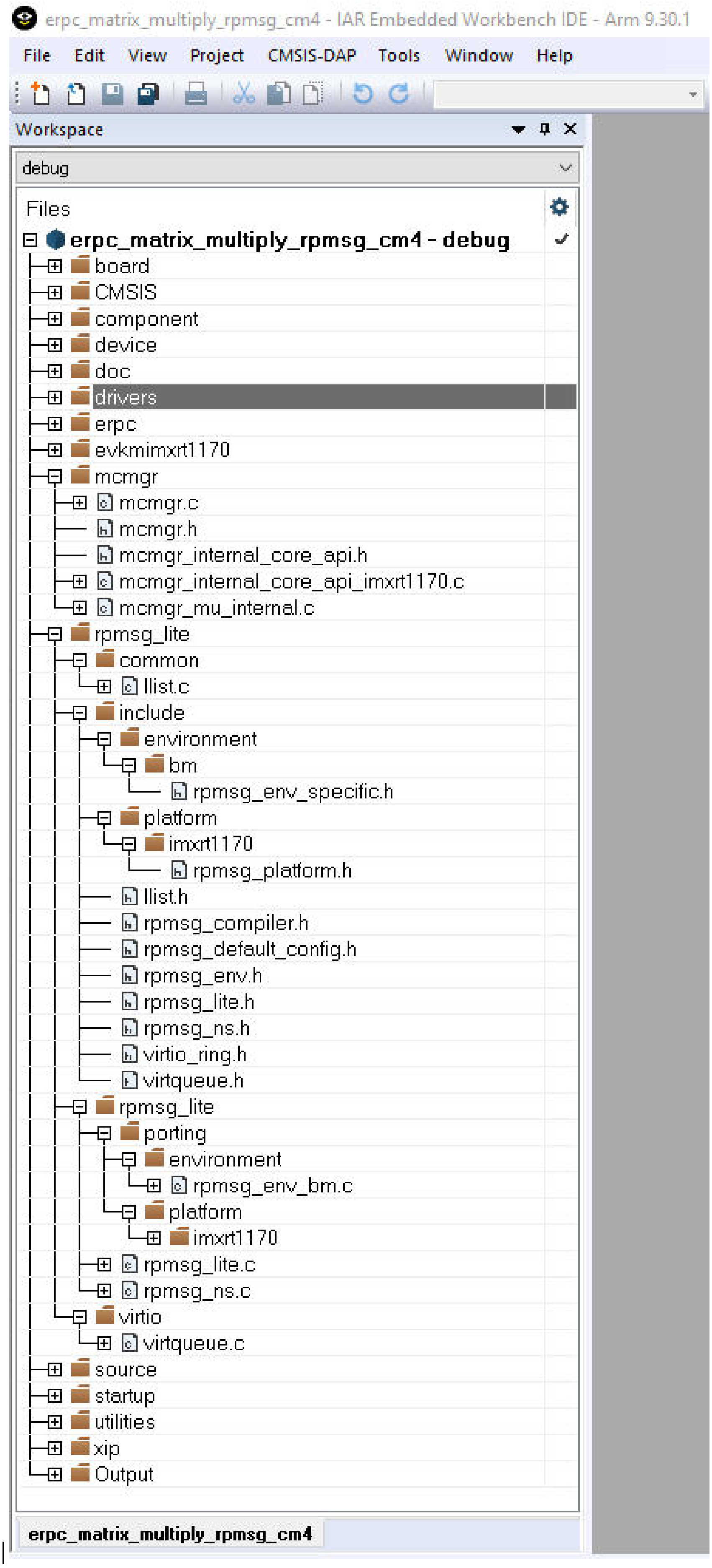
Parent topic: Multicore server application

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/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

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

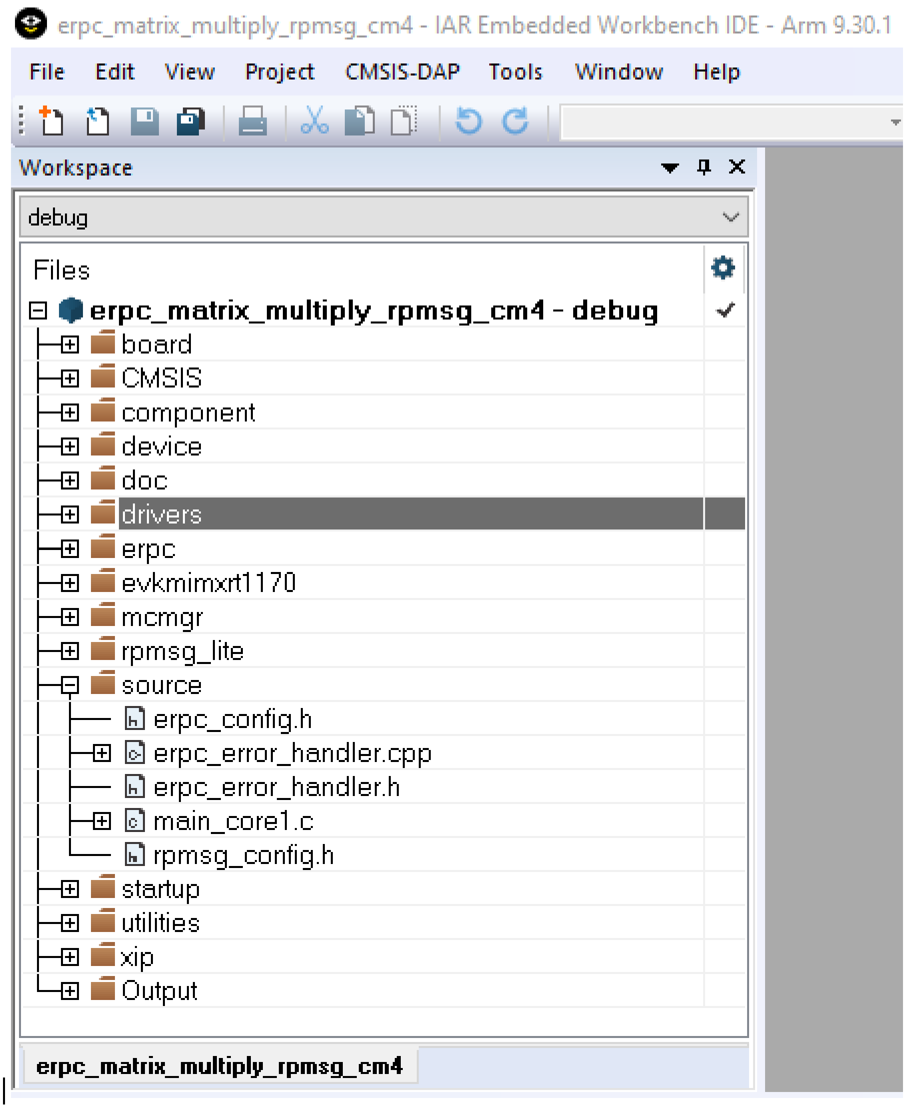
The main_core1.c file contains two functions:

- 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()
{
  ...
  /* RPMsg-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*

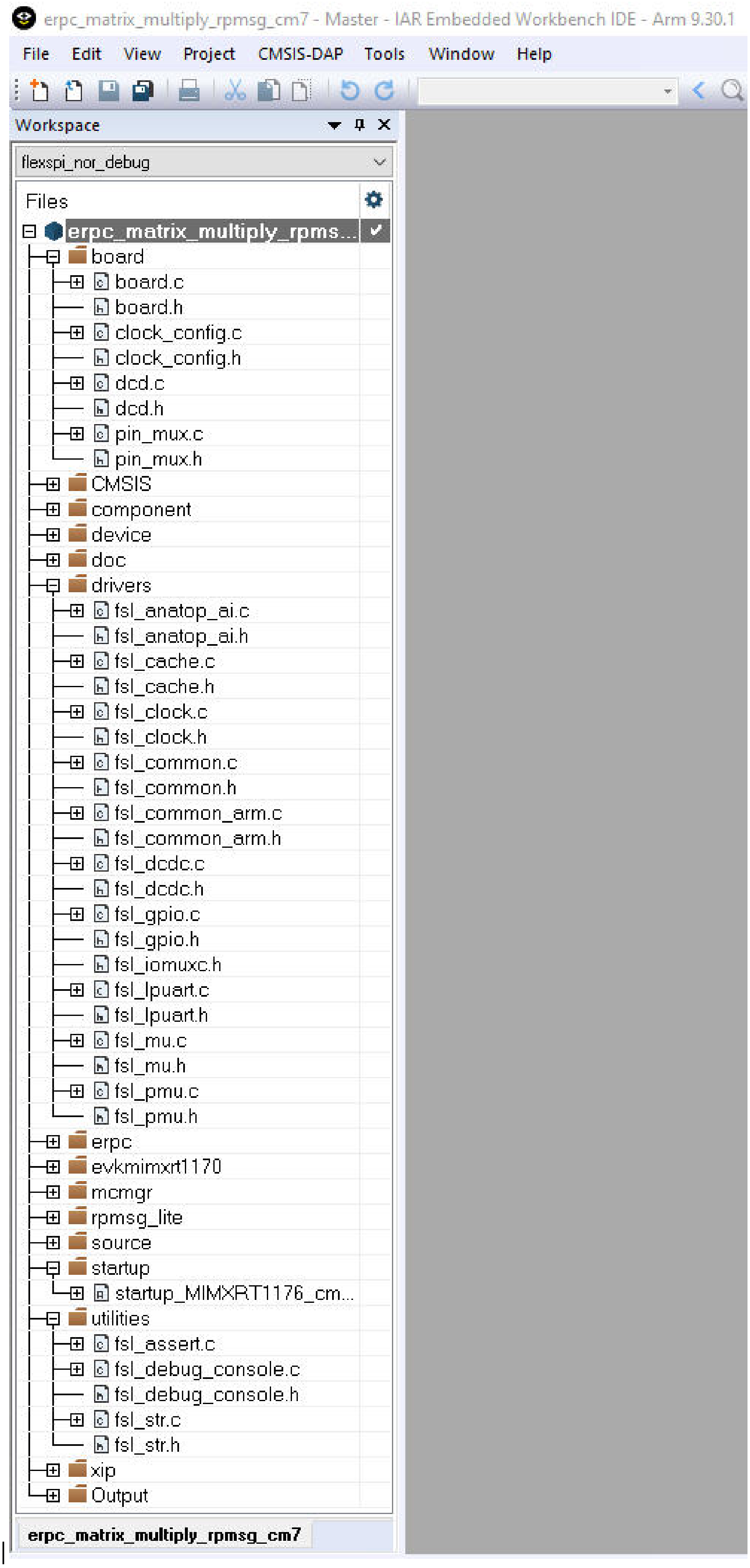
Multicore client application The "Matrix multiply" eRPC client project is located in the following folder:

<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>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

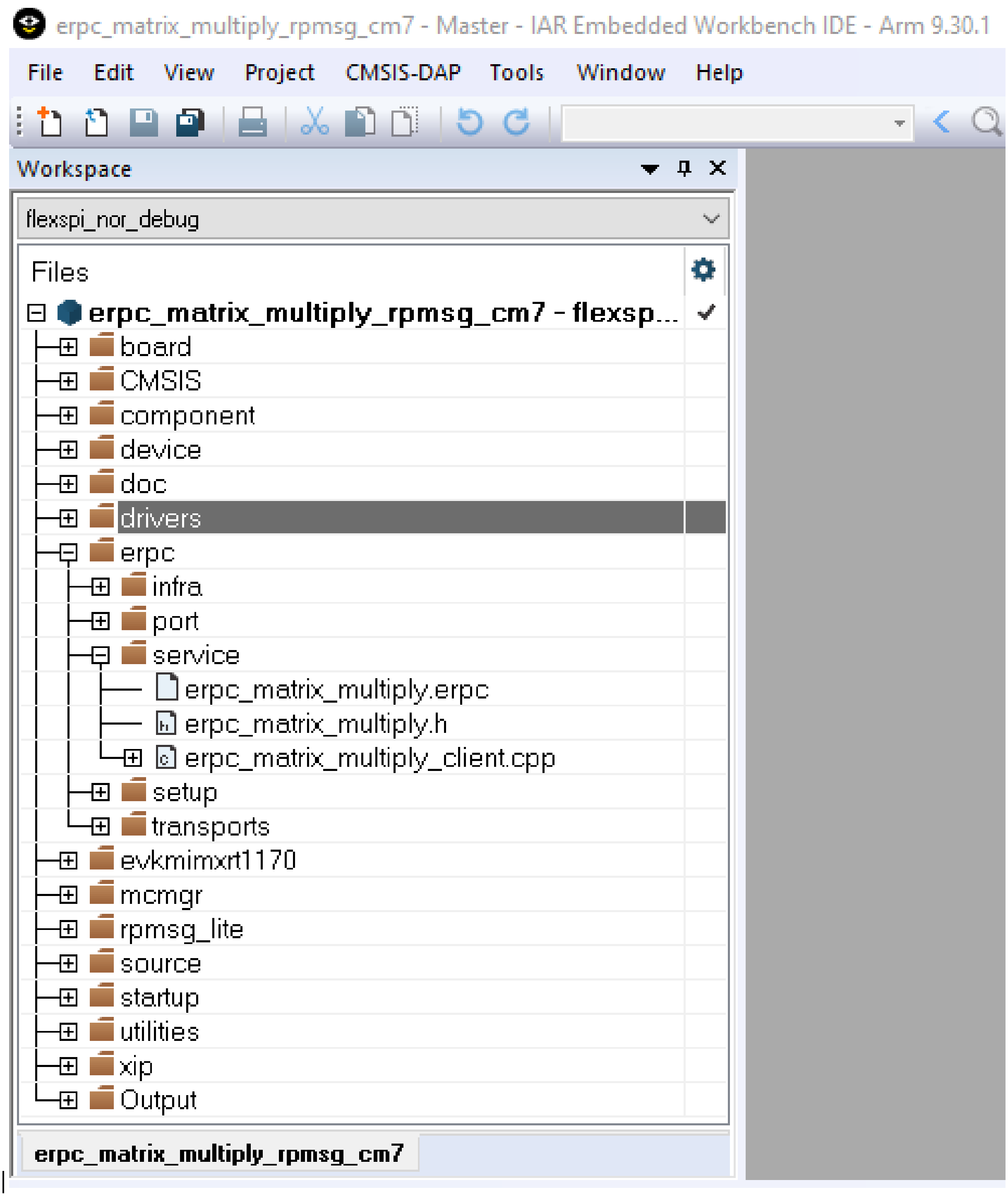
|

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.

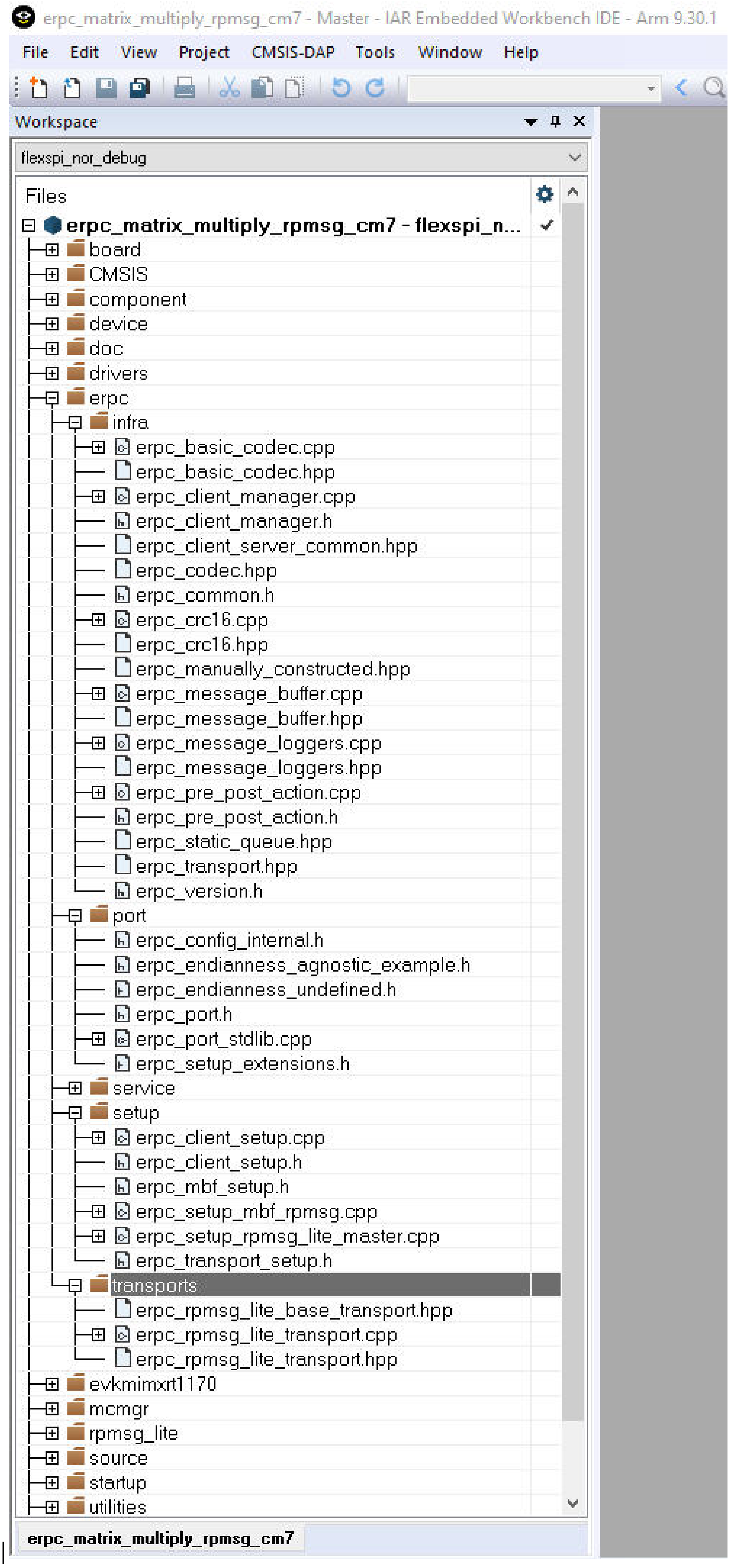
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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.



|

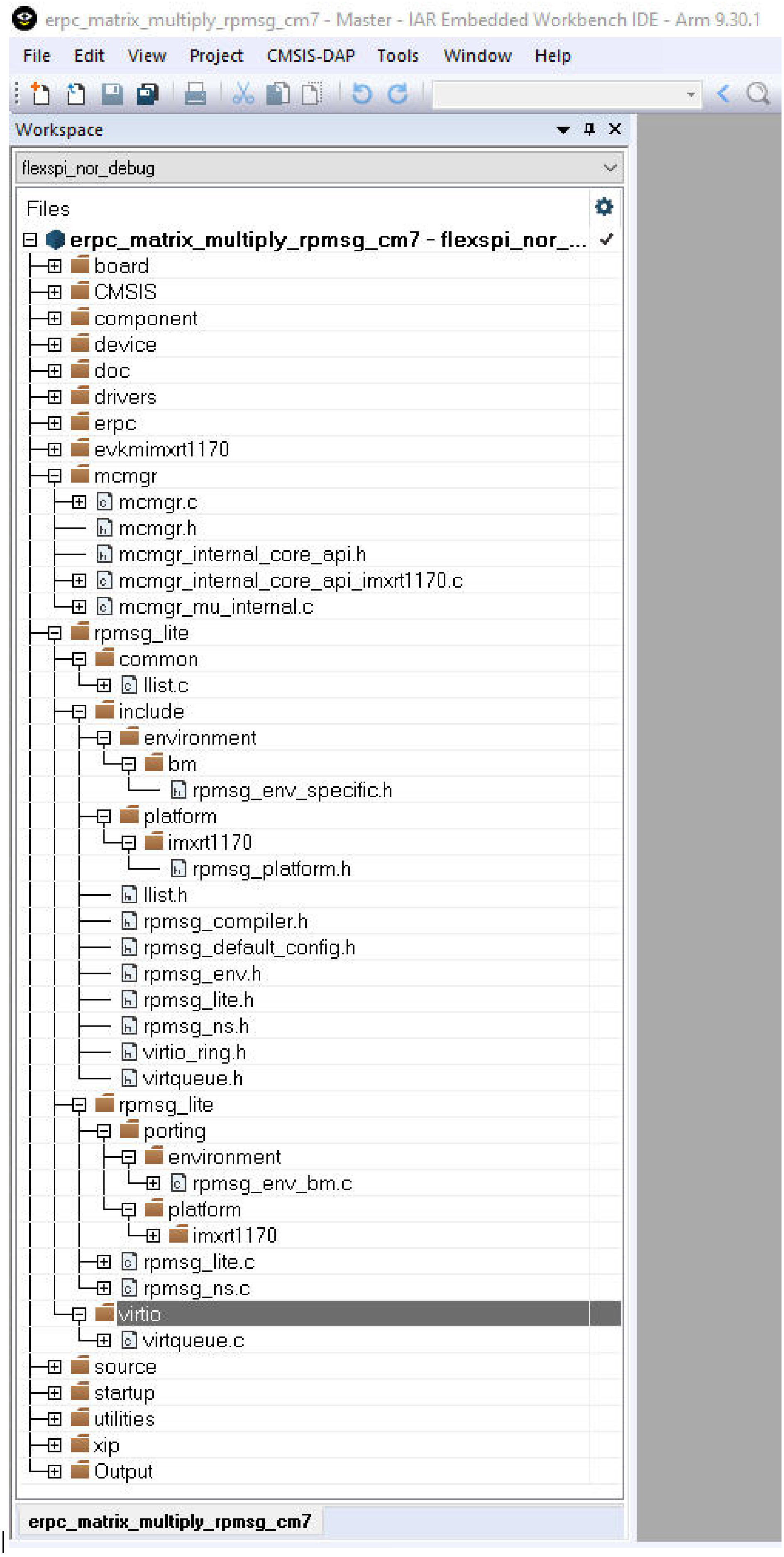
Parent topic: Multicore client application

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/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

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.

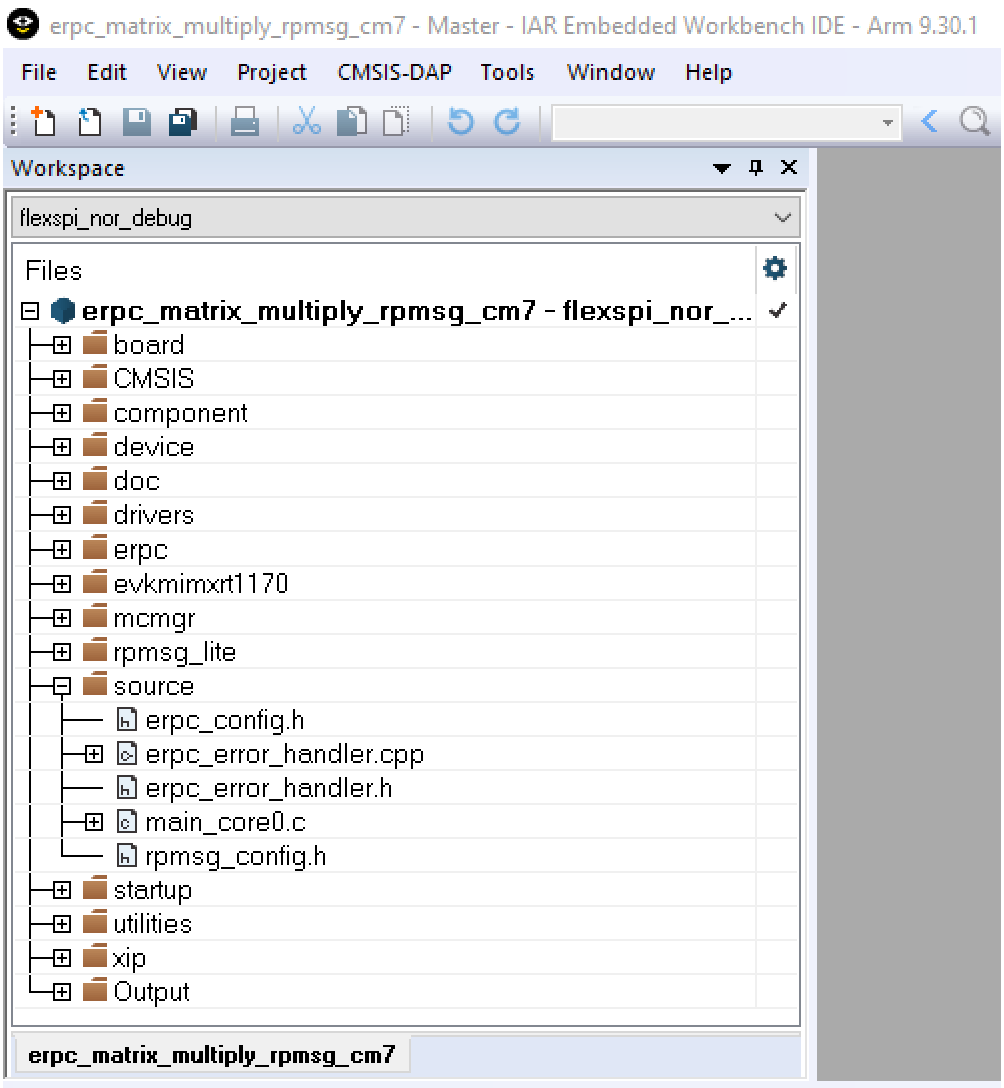
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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 occured 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 RPMsg-Lite). The RPMsg-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/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/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 RPMsg-Lite). The RPMsg-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:

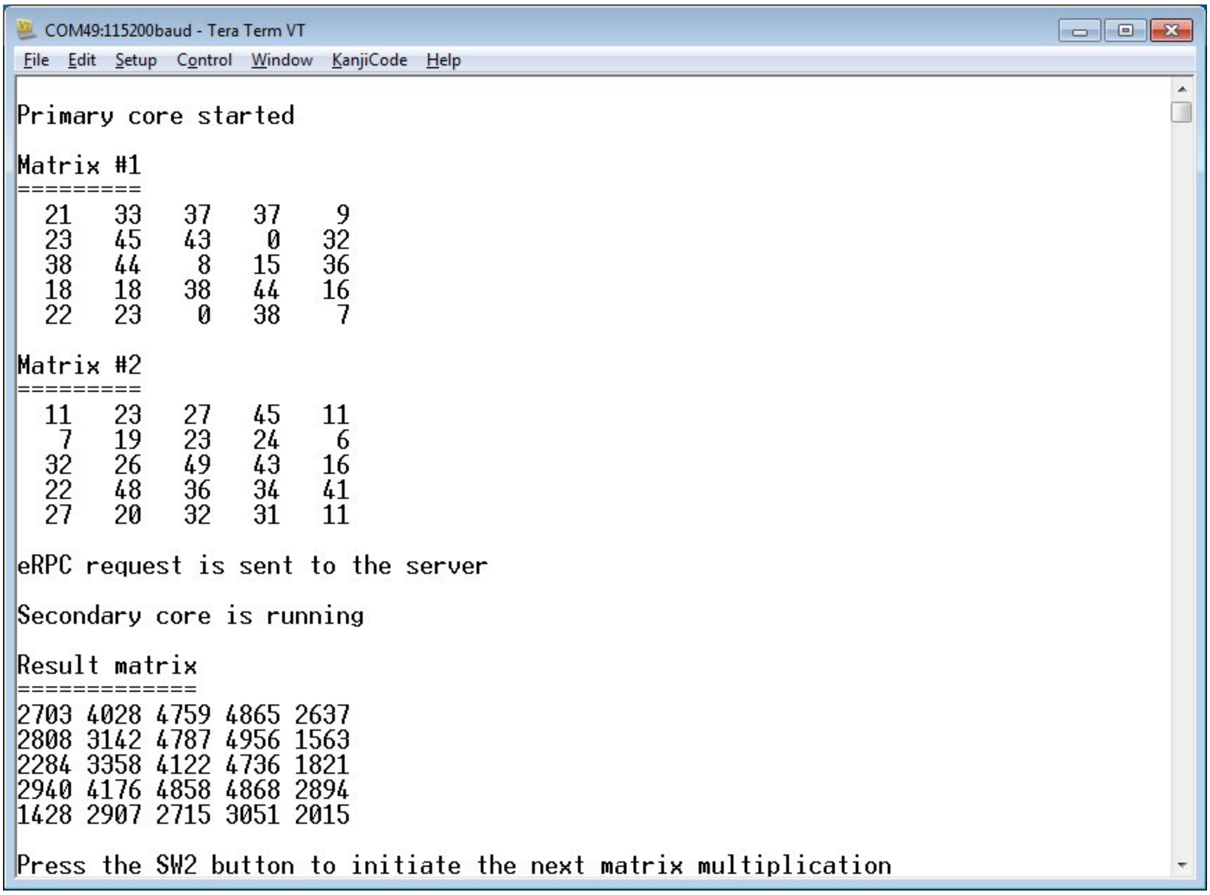
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver␣
↪operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&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 occured 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:



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 comunicates with the eRPC server that runs on anothe 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/* | | RPMsg-Lite source files | *<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/* |

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 is 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/s

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/erp
|Linux OS|<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64

<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/serv* 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:

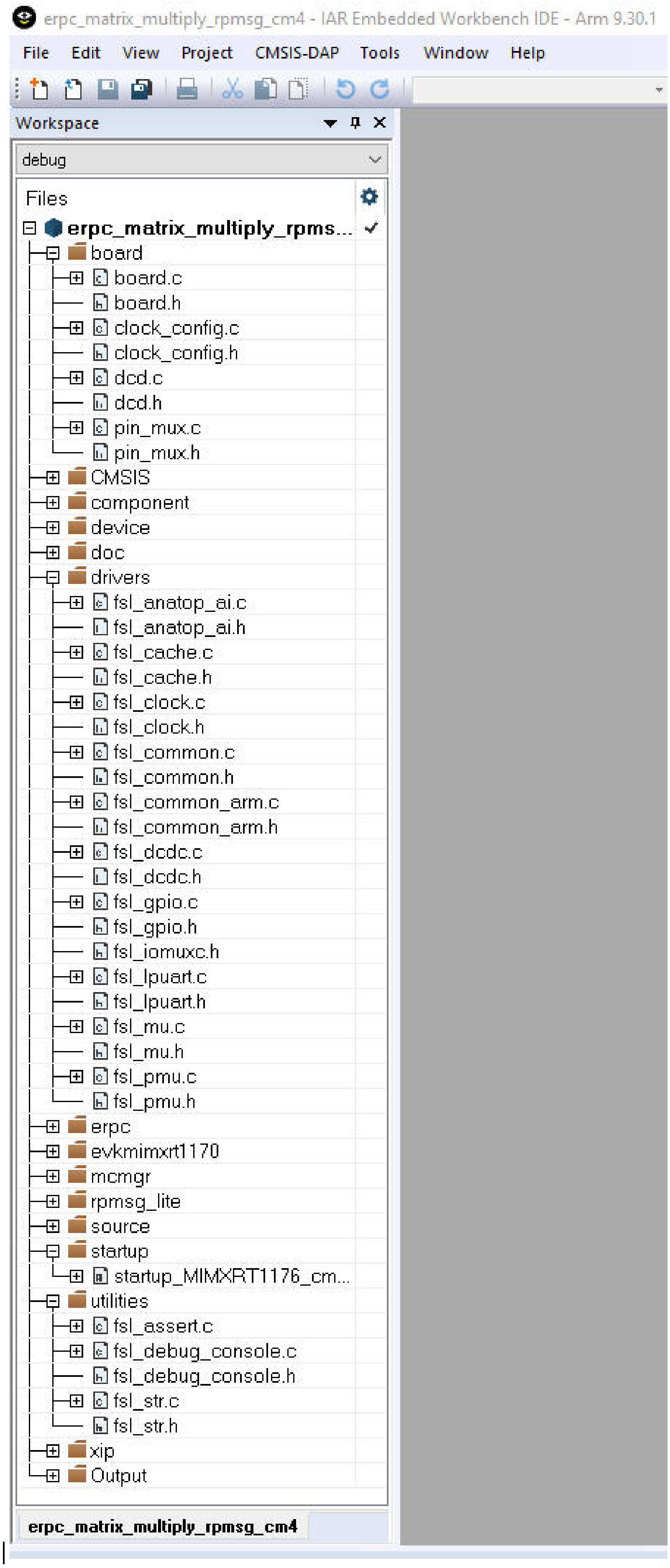
<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar/

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>/*


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

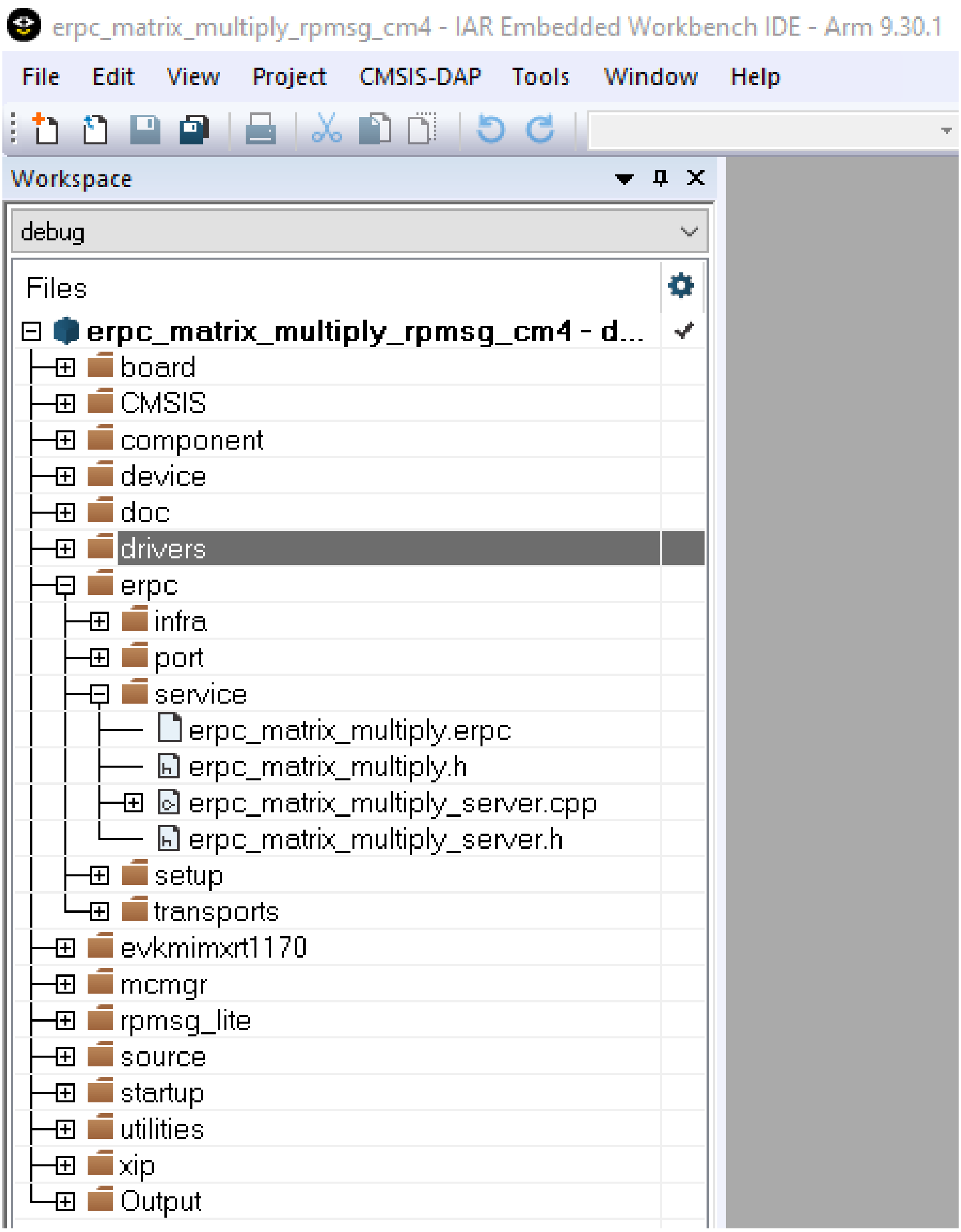
Parent topic: Multicore server application

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/s



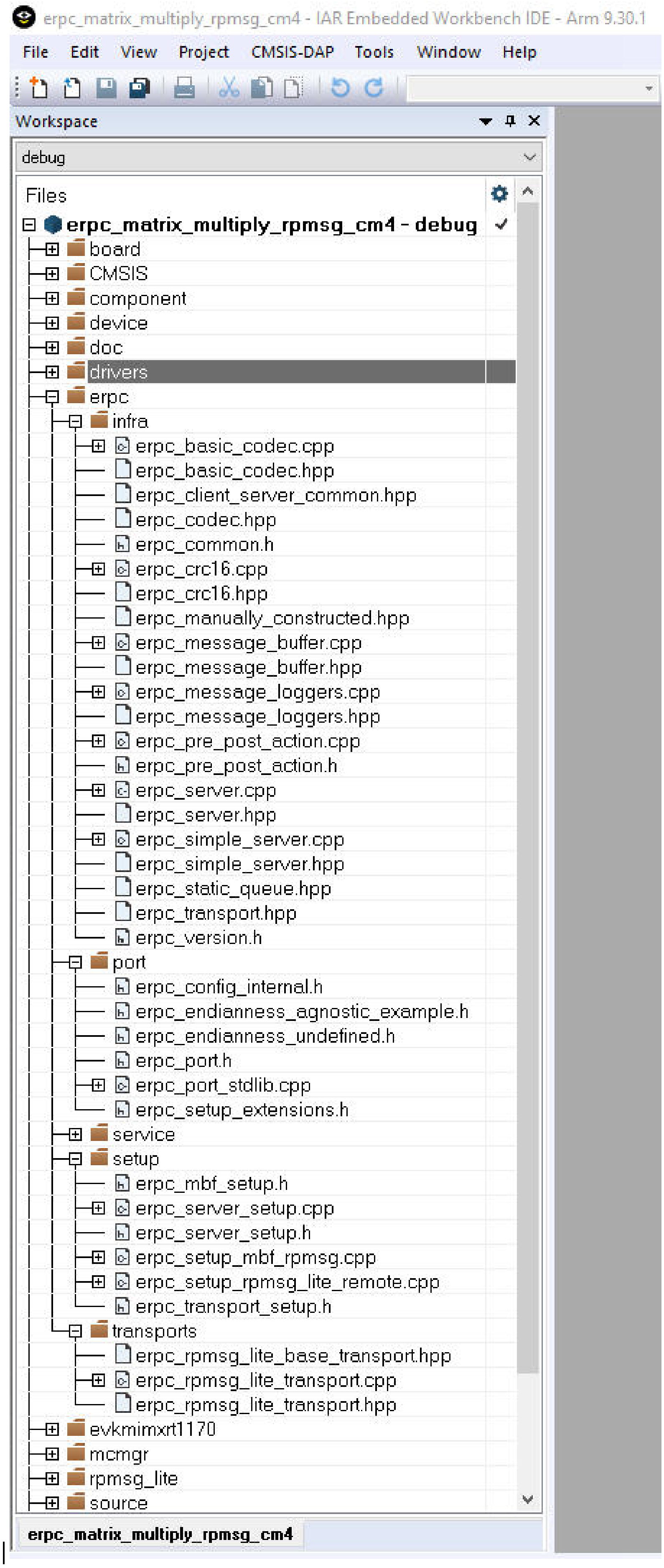
Parent topic: Multicore server application

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 needs 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 needs 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 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 need to be added into the server project.


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_server.cpp
erpc_server.hpp
erpc_simple_server.cpp
erpc_simple_server.hpp
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_mbf_setup.h
erpc_server_setup.cpp
erpc_server_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_remote.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
erpc_matrix_multiply_rpmsg_cm4

|

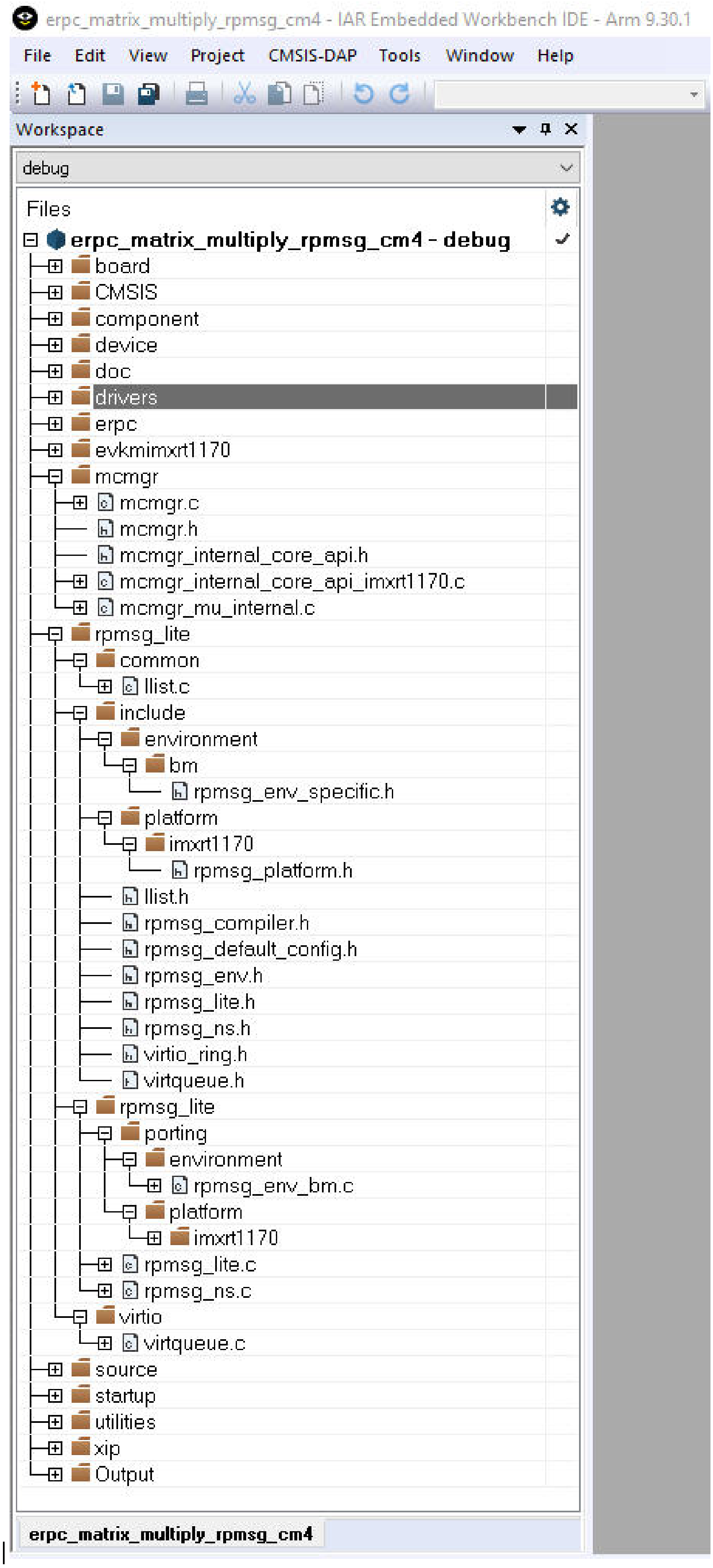
Parent topic: Multicore server application

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/


erpc_matrix_multiply_rpmsg_cm4 - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
debug
Files
erpc_matrix_multiply_rpmsg_cm4 - debug
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm4

|

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

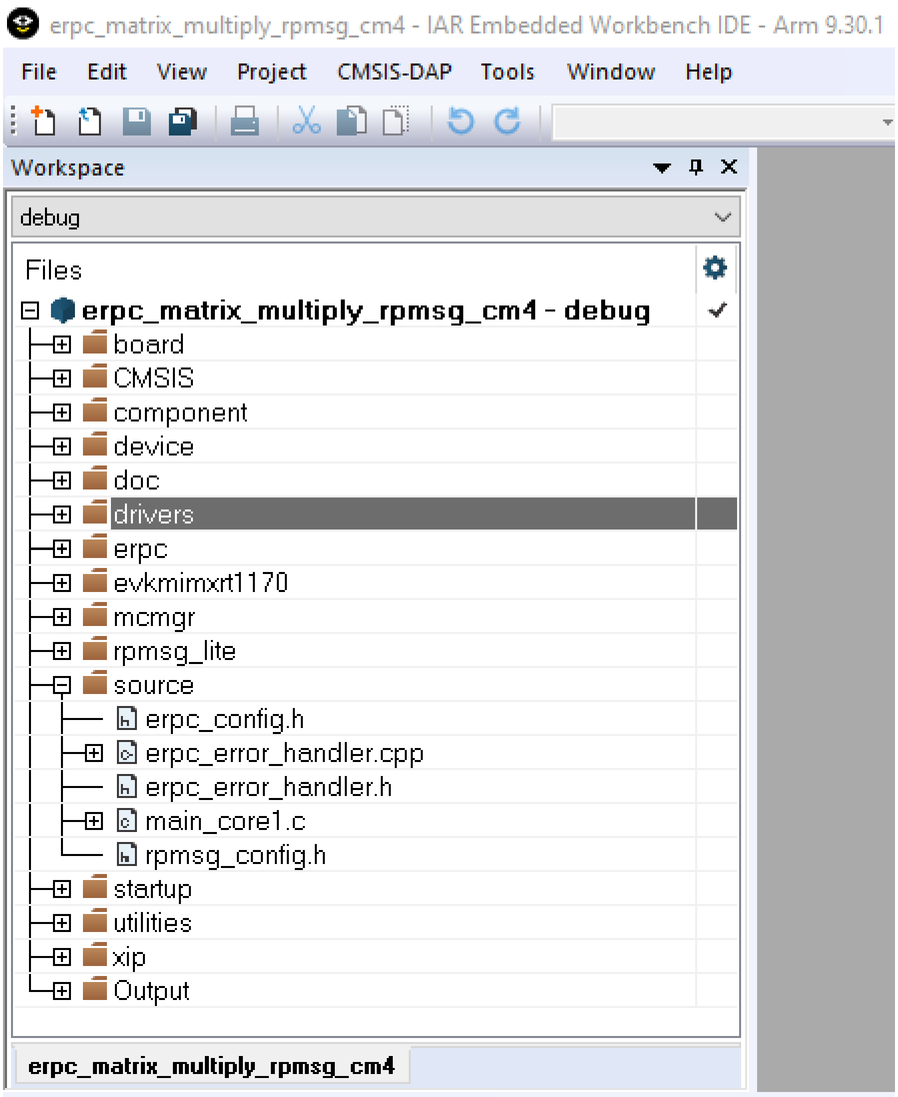
The main_core1.c file contains two functions:

- 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()
{
  ...
  /* RPMsg-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*

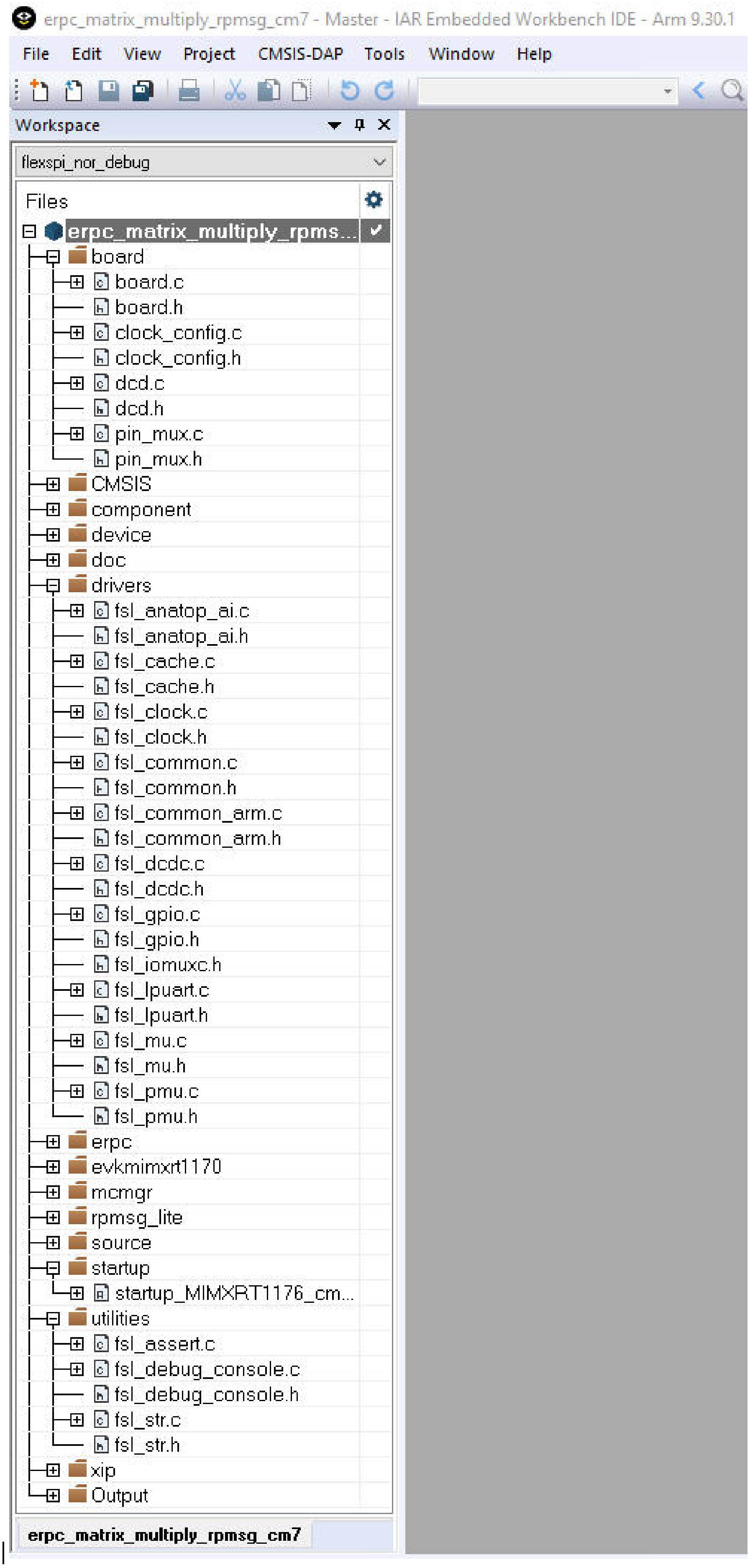
Multicore client application The "Matrix multiply" eRPC client project is located in the following folder:

<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>/*


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpms...
board
board.c
board.h
clock_config.c
clock_config.h
dcd.c
dcd.h
pin_mux.c
pin_mux.h
CMSIS
component
device
doc
drivers
fsl_anatop_ai.c
fsl_anatop_ai.h
fsl_cache.c
fsl_cache.h
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_common_arm.c
fsl_common_arm.h
fsl_dcdc.c
fsl_dcdc.h
fsl_gpio.c
fsl_gpio.h
fsl_iomuxc.h
fsl_lpuart.c
fsl_lpuart.h
fsl_mu.c
fsl_mu.h
fsl_pmu.c
fsl_pmu.h
erpc
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
startup_MIMXRT1176_cm...
utilities
fsl_assert.c
fsl_debug_console.c
fsl_debug_console.h
fsl_str.c
fsl_str.h
xip
Output
erpc_matrix_multiply_rpmsg_cm7

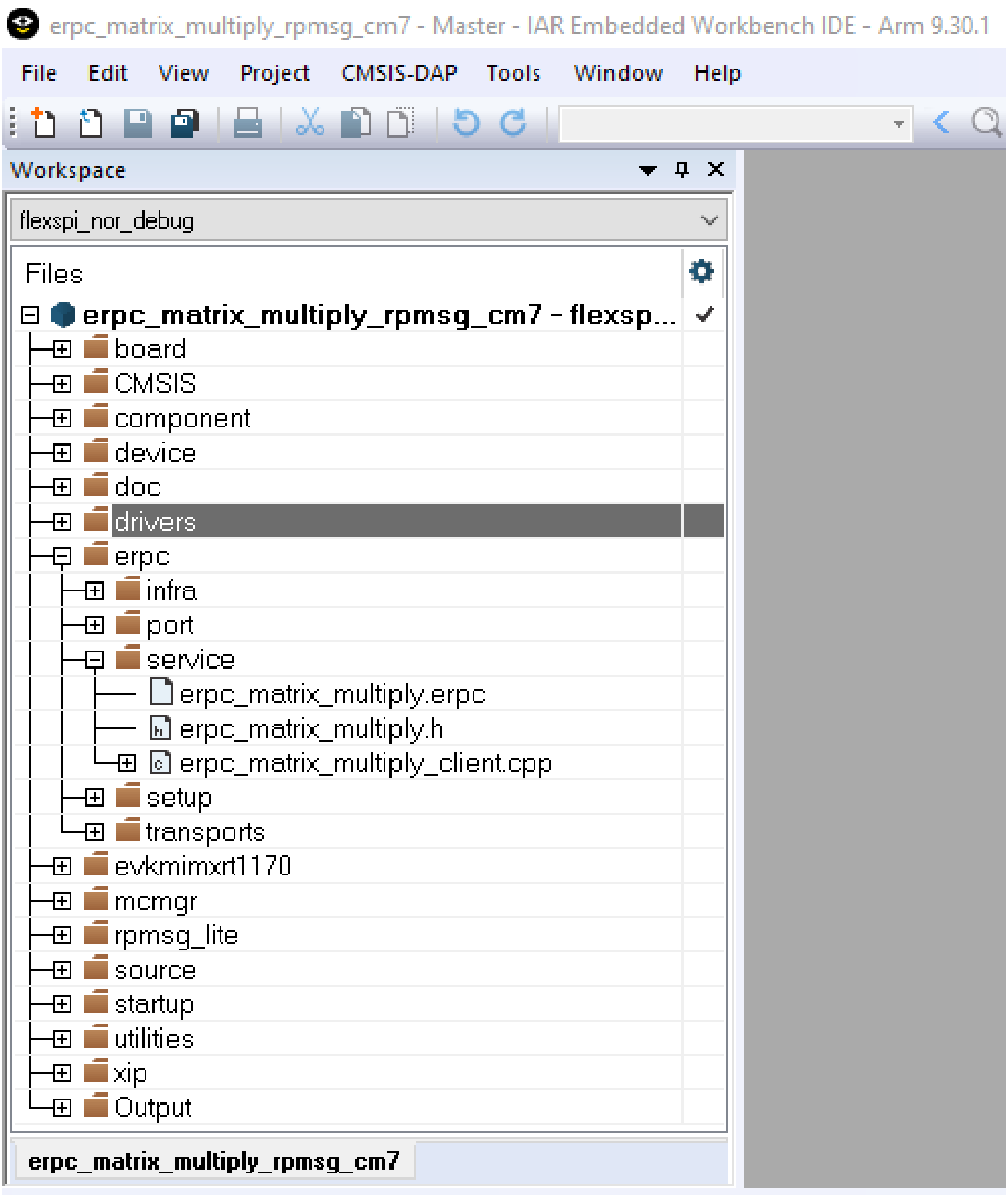
|

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.

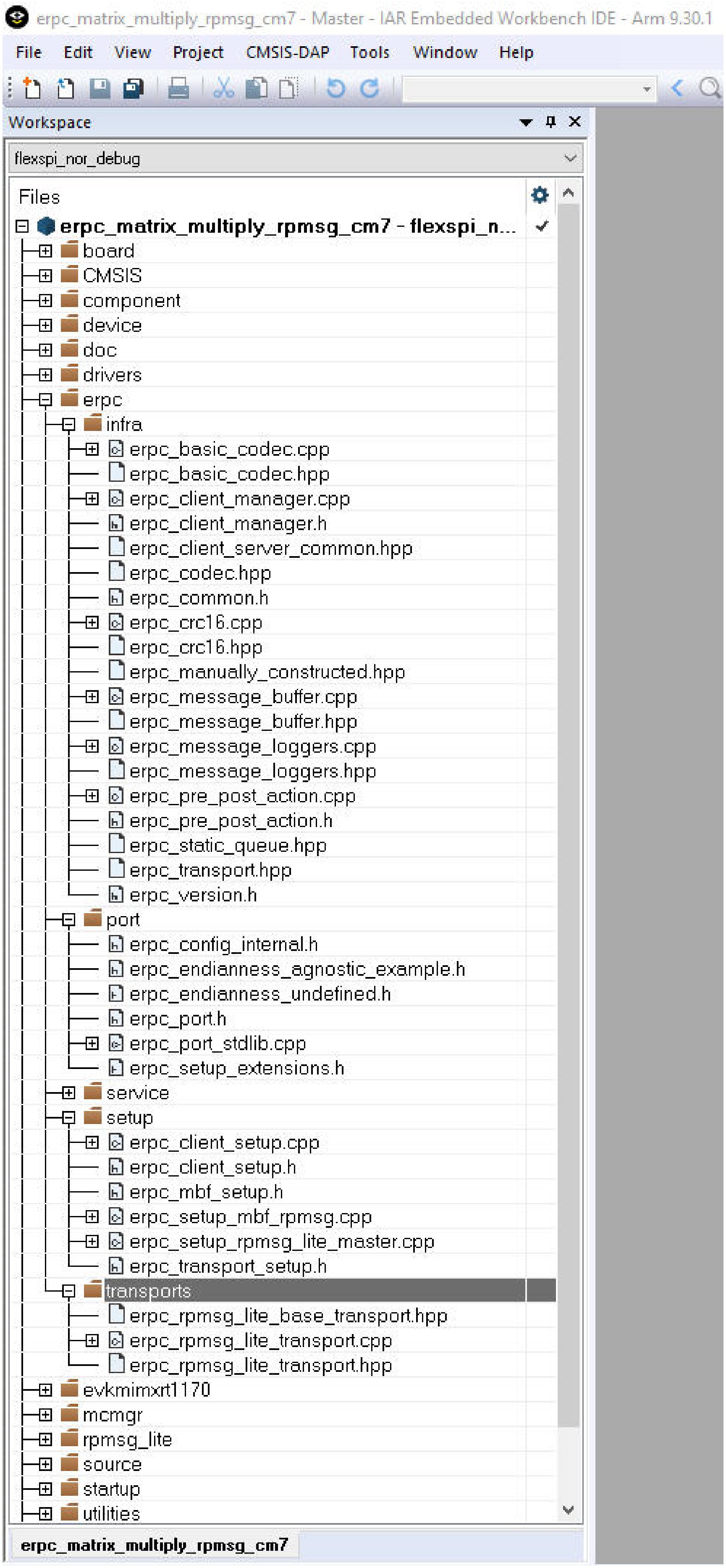
- erpc_port.h file contains definition of erpc_malloc() and erpc_free() functions.
- erpc_port_stdlib.cpp file ensures adaptation to stdlib.
- erpc_config_internal.h internal eRPC configuration file.

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.


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_n...
board
CMSIS
component
device
doc
drivers
erpc
infra
erpc_basic_codec.cpp
erpc_basic_codec.hpp
erpc_client_manager.cpp
erpc_client_manager.h
erpc_client_server_common.hpp
erpc_codec.hpp
erpc_common.h
erpc_crc16.cpp
erpc_crc16.hpp
erpc_manually_constructed.hpp
erpc_message_buffer.cpp
erpc_message_buffer.hpp
erpc_message_loggers.cpp
erpc_message_loggers.hpp
erpc_pre_post_action.cpp
erpc_pre_post_action.h
erpc_static_queue.hpp
erpc_transport.hpp
erpc_version.h
port
erpc_config_internal.h
erpc_endianness_agnostic_example.h
erpc_endianness_undefined.h
erpc_port.h
erpc_port_stdlib.cpp
erpc_setup_extensions.h
service
setup
erpc_client_setup.cpp
erpc_client_setup.h
erpc_mbf_setup.h
erpc_setup_mbf_rpmsg.cpp
erpc_setup_rpmsg_lite_master.cpp
erpc_transport_setup.h
transports
erpc_rpmsg_lite_base_transport.hpp
erpc_rpmsg_lite_transport.cpp
erpc_rpmsg_lite_transport.hpp
evkmimxrt1170
mcmgr
rpmsg_lite
source
startup
utilities
erpc_matrix_multiply_rpmsg_cm7

|

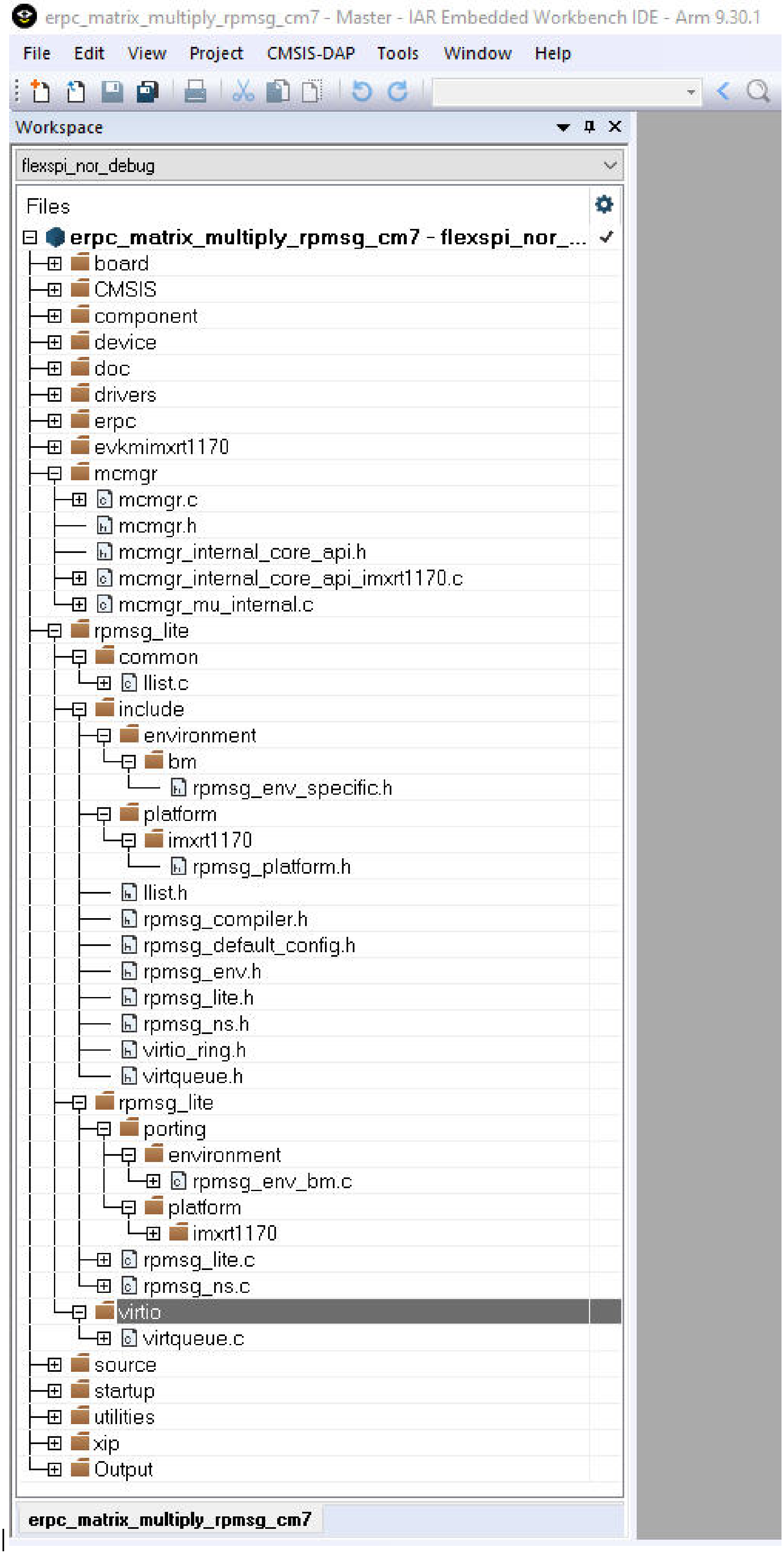
Parent topic: Multicore client application

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/


erpc_matrix_multiply_rpmsg_cm7 - Master - IAR Embedded Workbench IDE - Arm 9.30.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
flexspi_nor_debug
Files
erpc_matrix_multiply_rpmsg_cm7 - flexspi_nor_...
board
CMSIS
component
device
doc
drivers
erpc
evkmimxrt1170
mcmgr
mcmgr.c
mcmgr.h
mcmgr_internal_core_api.h
mcmgr_internal_core_api_imxrt1170.c
mcmgr_mu_internal.c
rpmsg_lite
common
llist.c
include
environment
bm
rpmsg_env_specific.h
platform
imxrt1170
rpmsg_platform.h
llist.h
rpmsg_compiler.h
rpmsg_default_config.h
rpmsg_env.h
rpmsg_lite.h
rpmsg_ns.h
virtio_ring.h
virtqueue.h
rpmsg_lite
porting
environment
rpmsg_env_bm.c
platform
imxrt1170
rpmsg_lite.c
rpmsg_ns.c
virtio
virtqueue.c
source
startup
utilities
xip
Output
erpc_matrix_multiply_rpmsg_cm7

|

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.

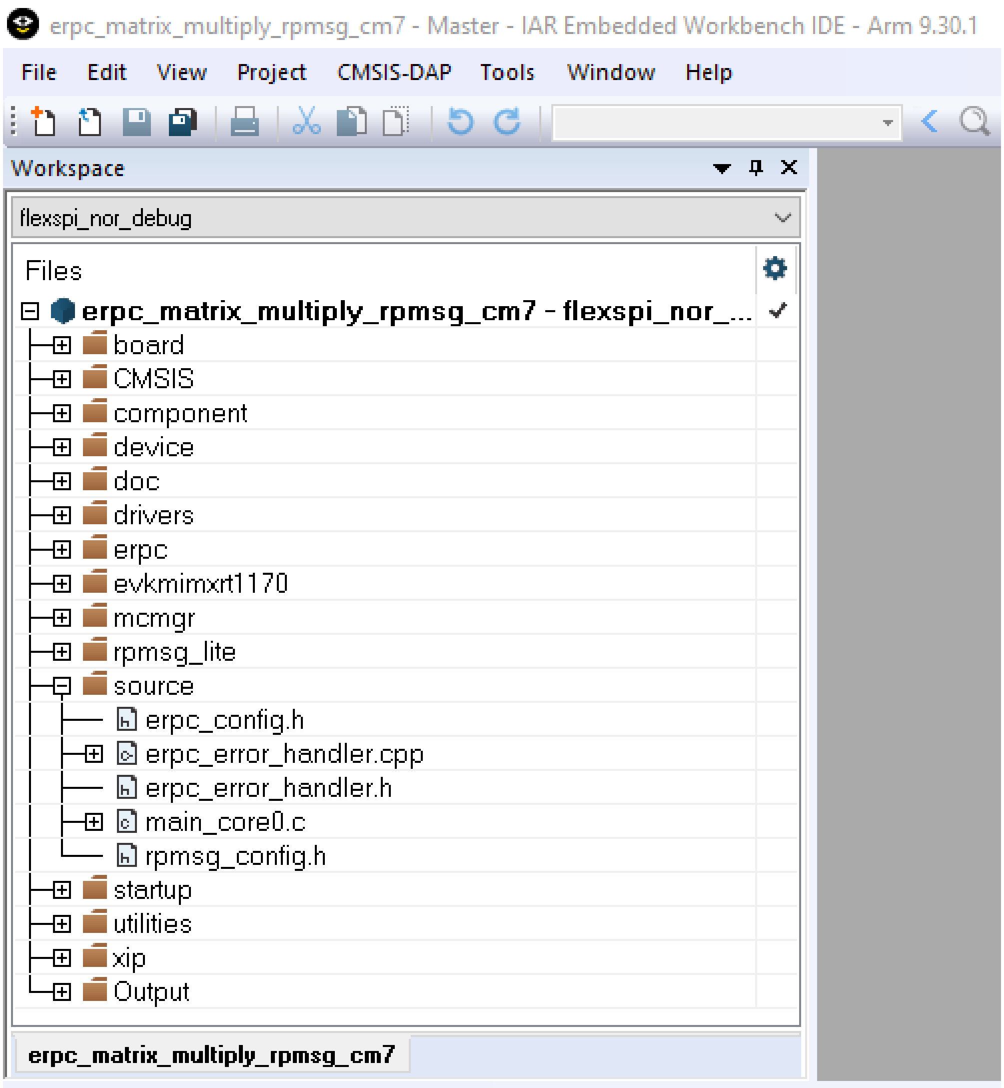
The matrix multiplication can be issued repeatedly, when pressing a software board button.

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 occured 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 RPMsg-Lite). The RPMsg-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/transports/erpc_uart_cmsis_transport.hpp

<eRPC base directory>/erpc_c/transports/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 RPMsg-Lite). The RPMsg-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:

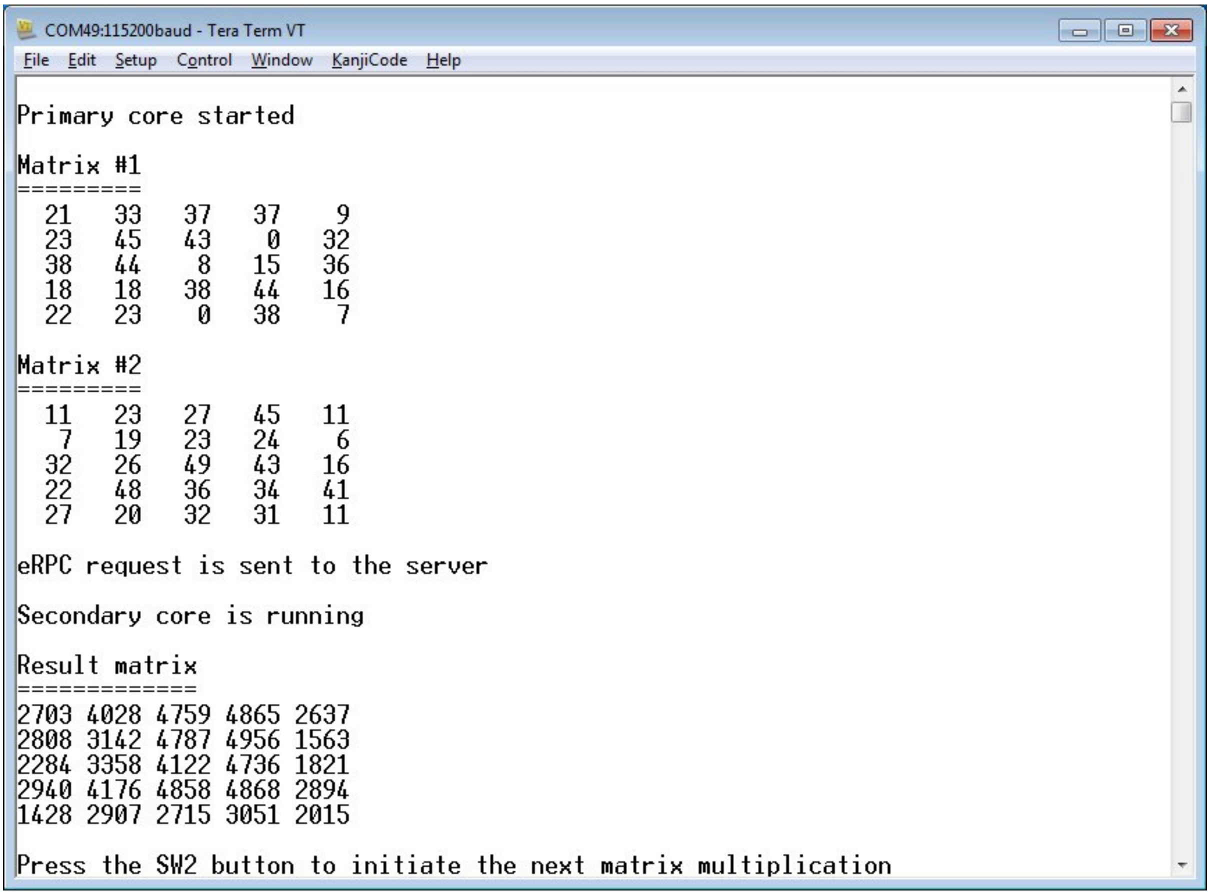
```
...
extern bool g_erpc_error_occurred;
...
/* Declare matrix arrays */
Matrix matrix1 = {0}, matrix2 = {0}, result_matrix = {0};
...
/* UART transport layer initialization, ERPC_DEMO_UART is the structure of CMSIS UART driver operations */
erpc_transport_t transport;
transport = erpc_transport_cmsis_uart_init((void *)&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 occured 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:



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:

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

Added

Fixed

- Python code of the eRPC infrastructure was updated to match the proper python code style, add type annotations and improve readability.

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 MessageBuffer, 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 LIBUSBSIO.GetNumPorts API use with libusbsio 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: MUTransport 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 distuingish which file can and cannot by 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 for 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 LIBUSBSIO 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 trasnport 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 Multimedia

1.7.1 Audio Voice

Audio Voice Components

MCUXpresso SDK : audio-voice-components

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

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

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

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

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

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

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

Content

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

Following table contains information about libraries and source files availability:

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

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

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

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

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

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

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

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

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

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

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

List of supported libraries in Zephyr:

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

AAC decoder

AAC decoder features

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

Specification and reference

Performance

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

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

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

CPU usage

- CPU core clock in MHz: 20.97.

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

API Usage of AAC Decoder

Overview

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

Configuration

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

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

Buffer Allocation

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

Initialization

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

Decoding

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

Seeking

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

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

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

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

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

FLAC decoder

FLAC decoder features

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

Specification and reference

Official website

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

Inbound licensing

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

Performance

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

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

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

CPU usage

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

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

Following test cases are performed:

- Audio format listening test
- Audio quality test

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

API Usage of FLAC Decoder

Overview

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

Configuration

Build Options

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

Buffer Allocation

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

Initialization

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

Decoding

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

Seeking

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

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

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

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

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

MP3 decoder

MP3 decoder features

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

Performance

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

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

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

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

- CPU core clock in MHz: 600.

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

API Usage of MP3 Decoder

Overview

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

Configuration

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

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

Buffer Allocation

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

Initialization

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

Decoding

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

Seeking

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

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

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

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

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

WAV decoder

WAV decoder features

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

Performance

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

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

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

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

- CPU core clock in MHz: 20.97 MHz.

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

Following test cases were performed:

- Audio format listening test
- Audio quality test

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

API Usage of WAV Decoder

Overview

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

Configuration

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

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

Buffer Allocation

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

Initialization

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

Decoding

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

Seeking

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

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

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

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

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

Synchronous Sample Rate Converter

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

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

The SSRC module supports the following features.

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

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

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

Parent topic:*Introduction*

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

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

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

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

Parent topic:*Introduction*

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

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

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

Parent topic: Resource usage

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

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

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

On Arm7 and Arm9

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

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

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

Parent topic: Processing requirements

On Arm9e and XScale

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

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

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

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

Parent topic: Processing requirements

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

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

Parent topic: Processing requirements

Parent topic: Resource usage

Parent topic:*Introduction*

Application programmers interface (API)

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

Type definitions

This section describes the type definitions of the SSRC module.

Types for allocation of instance and scratch memory

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

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

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

Parent topic: Type definitions

LVM_Fs_en

Definition:

```
typedef enum
{
    LVM_FS_8000     = 0,
    LVM_FS_11025    = 1,
```

(continues on next page)

(continued from previous page)

```
    LVM_FS_12000    = 2,
    LVM_FS_16000    = 3,
    LVM_FS_22050    = 4,
    LVM_FS_24000    = 5,
    LVM_FS_32000    = 6,
    LVM_FS_44100    = 7,
    LVM_FS_48000    = 8
} LVM_Fs_en;
```

Description:

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

Parent topic: Type definitions

LVM_Format_en **Definition:**

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

Description:

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

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

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

Parent topic: Type definitions

SSRC_Quality_en **Definition:**

```
typedef enum
{
    SSRC_QUALITY_HIGH           = 0,
```

(continues on next page)

(continued from previous page)

```
    SSRC_QUALITY_VERY_HIGH      = 1,
    SSRC_QUALITY_DUMMY          = LVM_MAXENUM
} SSRC_Quality_en;
```

Description:

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

Parent topic: Type definitions

Instance parameters **Definition:**

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

Description:

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

Parent topic: Type definitions

Nr of samples mode **Definition:**

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

Description:

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

Parent topic: Type definitions

Function return status **Definition:**

```
typedef enum
{
    SSRC_OK                     = 0,
    SSRC_INVALID_FS             = 1,
    SSRC_INVALID_NR_CHANNELS    = 2,
    SSRC_NULL_POINTER           = 3,
    SSRC_WRONG_NR_SAMPLES       = 4,
    SSRC_ALLINGMENT_ERROR       = 5,
```

(continues on next page)

(continued from previous page)

```
    SSRC_INVALID_MODE           = 6,
    SSRC_INVALID_VALUE          = 7,
    SSRC_ALLINGMENT_ERROR       = 8,
    LVXXX_RETURNSTATUS_DUMMY = LVM_MAXENUM
} SSRC_ReturnStatus_en;
```

Description:

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

Parent topic: Type definitions

Parent topic:*Application programmers interface (API)*

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

SSRC_GetNrSamples **Prototype:**

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

Description:

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

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

Returns:

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

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

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

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

Parent topic: Functions

SSRC_GetScratchSize **Prototype:**

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

Description:

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

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

|

Returns:

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

Parent topic: Functions

SSRC_Init **Prototype:**

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

Description:

The SSRC_Init function initializes an instance of the SSRC module.

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

Returns:

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

Parent topic: Functions

SSRC_SetGains

Prototype:

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

Description:

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

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

Returns:

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

Parent topic: Functions

SSRC_Process **Prototype:**

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

Description:

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

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

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

Returns:

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

Parent topic: Functions

SSRC_Process_D32 **Prototype:**

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

Description:

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

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

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

Returns:

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

Parent topic: Functions

Parent topic:*Application programmers interface (API)*

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

Define the number of samples to be used on input and output Call the function SSRC_GetNrSamples. Each integer multiple of the returned number of samples can be used.

Parent topic: Dynamic function usage

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

Parent topic: Dynamic function usage

Initialize the SSRC instance Call the SSRC_Init function.

Parent topic: Dynamic function usage

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

Parent topic: Dynamic function usage

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

Parent topic: Dynamic function usage

Parent topic:*Application programmers interface (API)*

Reentrancy None of the SSRC functions are re-entrant.

Parent topic:*Application programmers interface (API)*

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

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

Parent topic:*Additional user information*

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

Parent topic:*Additional user information*

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

The example application takes as command-line input parameters:

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

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

- Bit accurate test
- THD+N measurement

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

Parent topic:*Integration test*

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

Parent topic:*Integration test*

Maestro Audio Framework

MCUXpresso SDK : Maestro

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

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

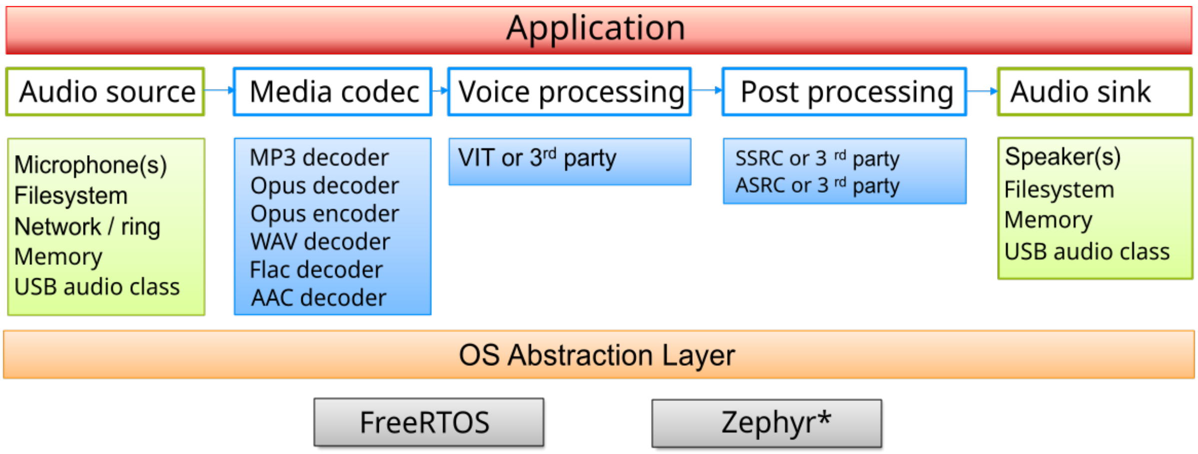
Visit Maestro - Documentation to review details on the contents in this sub-repo.

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

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

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

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



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

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

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

To see what is new, see *changelog*.

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

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

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

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

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

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

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

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

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

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

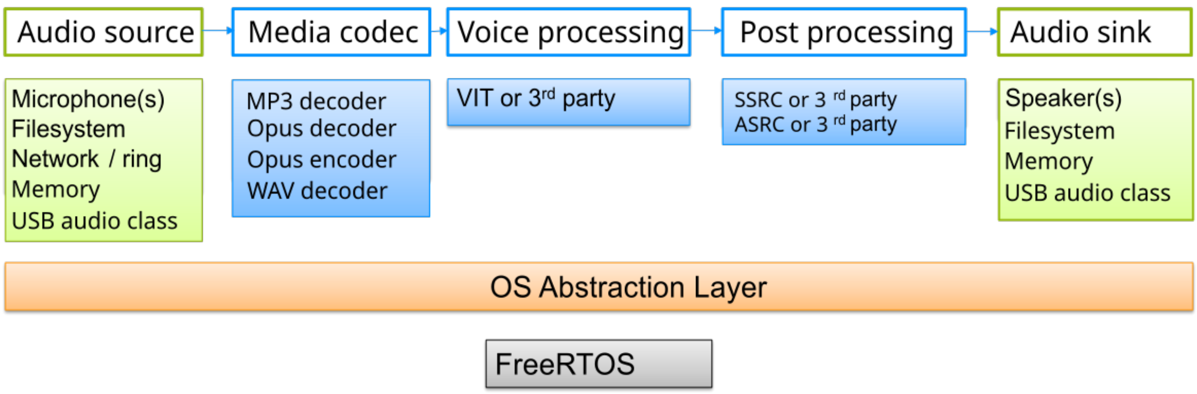
File structure

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

Maestro Audio Framework Programmer's Guide

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

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



Pipeline

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

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

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

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

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

Each element index (ElementIndex) has its own properties and a list of these properties can be found in the streamer_element_properties.h file. These properties are divided into groups and each group is identified by a property mask (e.g. for speaker it is PROP_SPEAKER_MASK). Then the property_lookup_table in the streamer_msg.c file determines which property group relates to which element index (ElementIndex). When an element is created and added to the pipeline, its properties are set to their defalut values. Default values can be seen in the initialization function of a particular element. The initialization functions are specified in the element_list array in the streamer_element.c file (e.g. for the audio_proc element it is the audio_proc_init_element function). The user can get the value of the property using the streamer_get_property function or change its value using the streamer_set_property function.

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

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

- Add a new element type to the StreamElementType enum type in the streamer_api.h.
- Create a new *.c and *.h files for the new element type in the middleware\audio_voice\maestro\src\elements\ folder. All necessary structures and functions (functions for src pads, sink pads and element itself) needs to be defined in these files. Inspiration can be found in other elements.
- Link the initialization function to the element type in the element_list array in the streamer_element.c file. To do this, a new definition that enables the element needs to be created (e.g. there is a STREAMER_ENABLE_AUDIO_PROC definition for the audio_proc element).

- Associate the newly created element type with an element index (ElementIndex) by adding a new pair to the type_lookup_table in the streamer.c file.
- If the user wants to use the newly created element in an application, the definiton that enables the element must be defined at the project level.

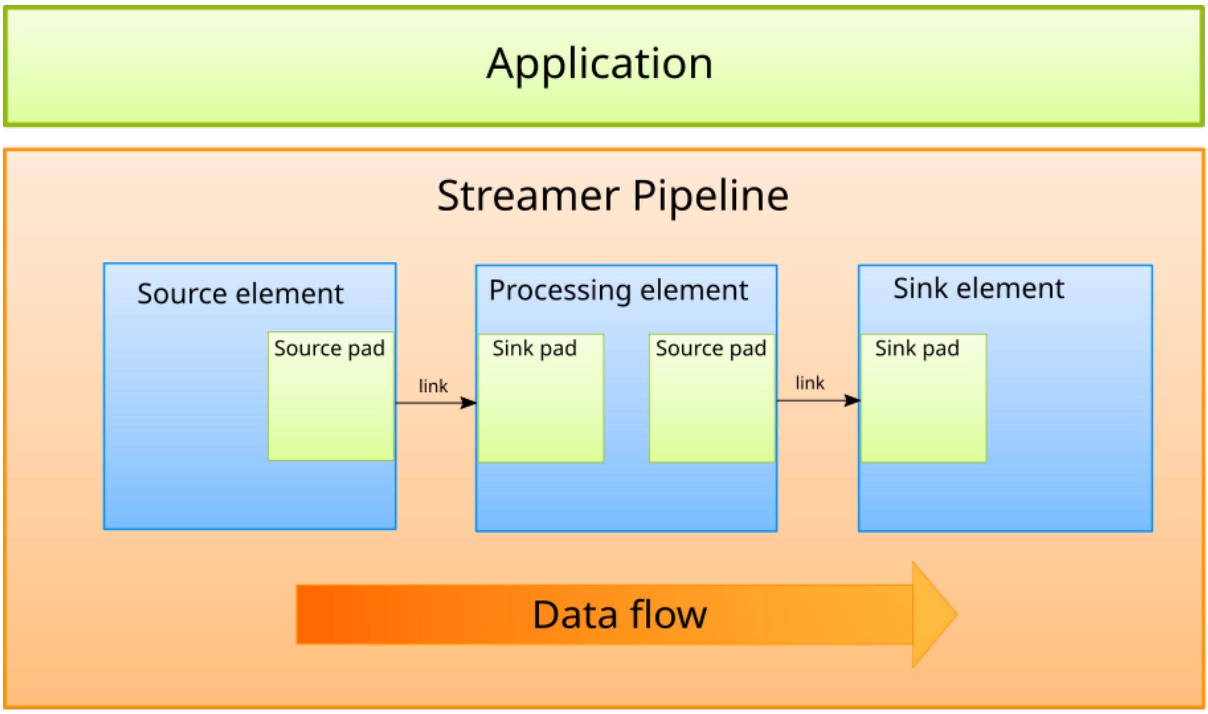
Mostly the user doesn't need to create a new element type, but just create an element index.

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

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

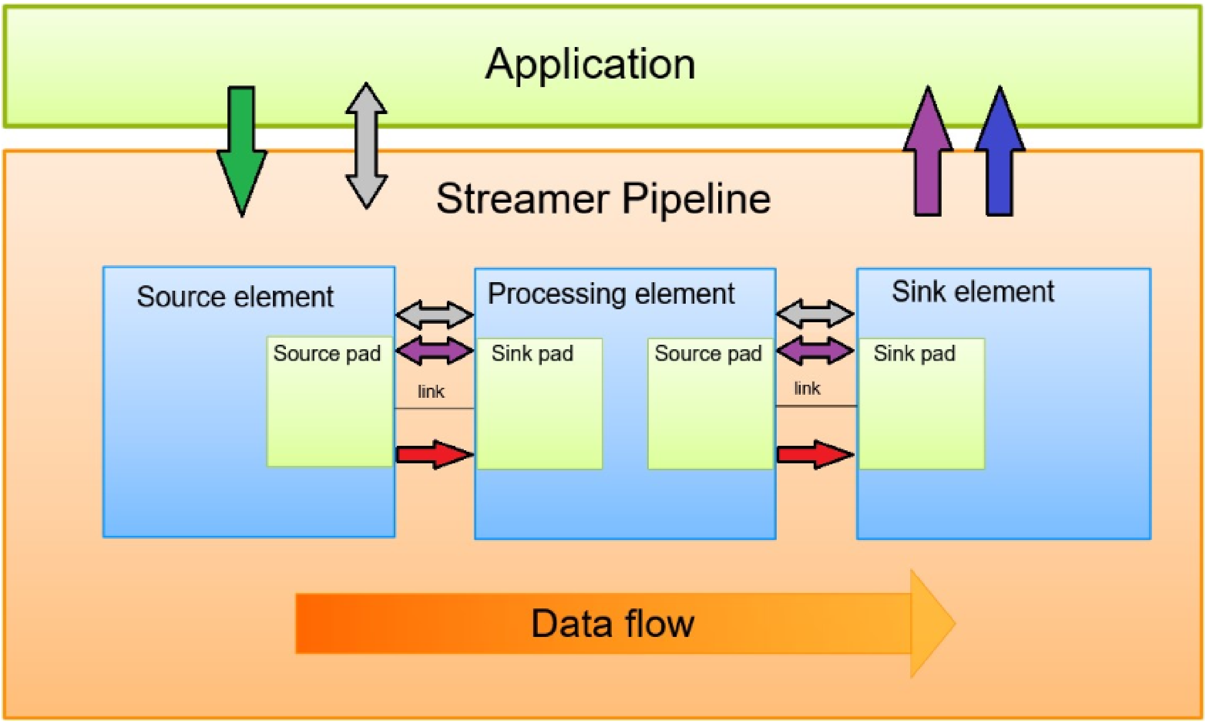
It is important to know that each element type (StreamElementType) can be associated with more than one element index (ElementIndex), but each element index (ElementIndex) can be associated with only one element type (StreamElementType).

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



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

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



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

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

Using the CCI to interface with Metadata

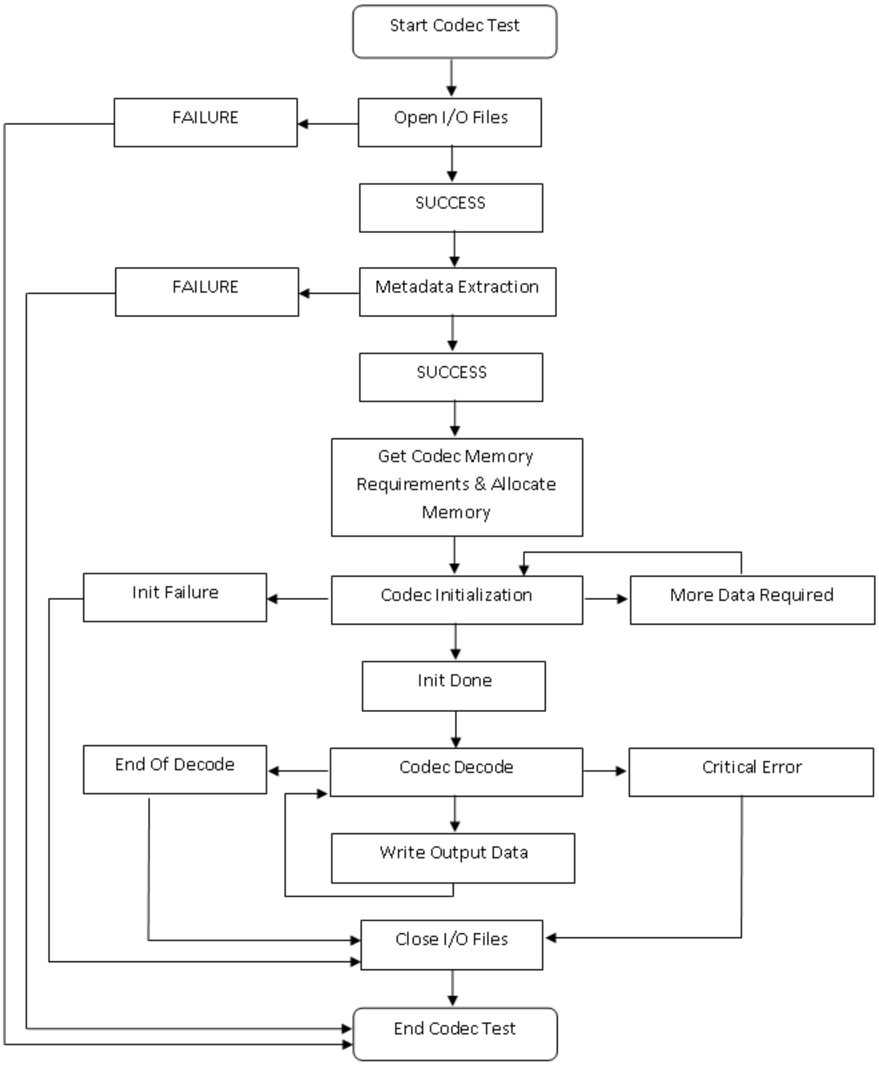
- cci_extract_meta_data must be called before any other Codec Interface APIs. This API extracts the metadata information of the codec and fills this information in the file_meta_data_t structure. The file_meta_data_t structure must be allocated by the application.

- This function first extracts the input file extension and based on that it calls the specific codec's metadata extraction function. If it finds an invalid extension or unsupported extension then it returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.
- If this API finds the valid metadata then it returns with META_DATA_FOUND code. If this API does not find any metadata information then it returns with META_DATA_NOT_FOUND code. It also returns with META_DATA_FILE_NOT_SUPPORTED code for any unsupported file format.

Using the CCI to interface with Decoders

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

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



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

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

```
const codec_interface_function_table_t g_codec_function_table[STREAM_TYPE_COUNT] = {
#ifdef VORBIS_CODEC
  {
      &VORBISDecoderGetMemorySize,
      &VORBISDecoderInit,
      &VORBISDecoderDecode,
      NULL,
```

(continues on next page)

(continued from previous page)

```
        NULL,
        &VORBISDecoderSeek,
        &VORBISDecoderGetIOFrameSize,
    },
#else
    {
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
    }
#endif
};
```

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

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

Maestro performance

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

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

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

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

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

- CPU core clock in MHz: 600.

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

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

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

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

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

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

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

- PROP_ENCODER_CHUNK_SIZE

  - **Synopsis:** Determines the length of a chunk pulled from the sibling of the source pad and essentially influences the size of allocated buffers. If the actual amount of data pulled is smaller, the rest is zero-filled.
  - **Type:** unsigned 32-bit integer
  - **Default value:** 1920
  - **Constraints:**
    * Must be bigger than zero, otherwise STREAM_ERR_INVALID_ARGS is returned.
    * Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS is returned.

- PROP_ENCODER_TYPE
  - **Synopsis:** Determines the exact encoder (CEI implementation) to be used.
  - **Type:** CeiEncoderType (cei_enctypes.h)
  - **Default value:** CEIENC_LAST
  - **Constraints:**
    * Must not be equal to CEIENC_LAST, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    * Selected encoder must be implemented, otherwise STREAM_ERR_INVALID_ARGS will be returned.
    * Cannot be changed if the actual encoder has been created. If done so, STREAM_ERR_ELEMENT_BAD_STATUS will be returned.
  - **Behaviour influenced:** The encoder element process function will return FLOW_ERROR if this property isn't set.

- PROP_ENCODER_CONFIG
  - **Synopsis:** Determines encoder-specific configuration (application, bitrate, …).
  - **Type:** Pointer to the encoder-specific configuration structure.
  - **Default value:** Determined by the encoder.
  - **Constraints**:
    * The encoder has to be configurable. If it is not, STREAM_ERR_ERR_GENERAL will be returned on any access.
    * The structure has to conform to the encoder requirements. If the encoder returns an error code, STREAM_ERR_GENERAL will be returned.

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

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

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

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

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

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

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

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

Maestro playback example

Table of content

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

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

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

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

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

Limitations:

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

        * *LPCXpresso55s69* - When a memory allocation ERROR occurs, it is necessary to disable the SSRC element due to insufficient memory.

Known issues:

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

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

Hardware requirements

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

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

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

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board.
2. Open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits
    - No parity
    - One stop bit
    - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.

5. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

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

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

[APP_Main_Task] started

Copyright  2022  NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

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

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

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"file": Perform audio file decode and playback

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

Details of commands can be found *here*.

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

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

5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.

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

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

Playback itself can be started with the STREAMER_Start function.

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

```
ELEMENT_PROPERTY_T prop;

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

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

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

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

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

States The application can be in 3 different states:

- Idle

- Running
- Paused

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

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

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

Legend for diagrams:

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

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

help, version

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

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

file stop

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

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

file pause

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

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

file volume <volume>

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

B((Idle)):::state --> M[Error: Play a track first]:::error
C((Running)):::state --> G{Volume
parameter
empty?}:::condition
D((Paused)):::state --> G
G -- Yes -->H[Error: Enter volume parameter]:::error
G -- No -->I{Volume
in range?}:::condition
I -- No -->J[Error: invalid value]:::error
I -- Yes -->K[Set volume]:::function
J --> L((No state
change)):::state
K --> L
H--> L

file seek <seek_time>

The seek argument is only supported in the Standard mode.

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

B((Idle)):::state --> E[Error: First select
an audio track to play]:::error
E-->B
C((Running)):::state --> F[Error: First
pause the track]:::error
F --> C
D((Paused)):::state --> G{Seek
parameter
empty?}:::condition
G --No --> H{AAC file?}:::condition

G --Yes --> I[Error: Enter
a seek time value]:::error
I-->N((Paused)):::state;
H --Yes -->J[Error: The AAC decoder
does not support
the seek command]:::error
J-->N
H --No -->K{Seek
parameter
positive?}:::condition
K --No -->L[Error: The seek
time must be
a positive value]:::error
L-->N
K --Yes -->M[Seek the file]:::function
M-->N

file play <filename>

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

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

file list

flowchart TD
classDef function fill:#69CA00

```
    classDef condition fill:#0EAFE0
    classDef state fill:#F9B500
    classDef error fill:#F54D4D

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

file info

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

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

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

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

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

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

Maestro record example

Table of content

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

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

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

- **Loopback** - The application demonstrates a loopback from the microphone to the speaker without any audio processing. Mono, stereo or multichannel mode can be used, depending on the hardware, see *table* below.
- **File recording** - The application takes audio samples from the microphone inputs and stores them to an SD card as an PCM file. The PCM file has following parameters:
  - Mono and stereo : 2 channels, 16kHz, 16bit width
  - Multi-channel (AUD-EXP-42448): 6 channels, 16kHz, 32bit width
- **Voice control** - The application takes audio samples from the microphone input and uses the VIT library to recognize wake words and voice commands. If a wake word or a voice command is recognized, the application write it to the serial terminal.
- **Encoding** - The application takes PCM samples from memory and sends them to the Opus encoder. The encoded data is stored in memory and compared to a reference. The result of the comparison is finally written into the serial terminal.

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

Limitations:

- Note:
  - *LPCXPresso55s69* - MCUXpresso IDE project default debug console is semihost
- Addition labraries
  - **VIT:**
    * The VIT is supported only in the MCUXpresso IDE and ARMGCC.
    * *LPCXpresso55s69* - The VIT is disabled by default due to insufficient memory. To enable it, see the *Example configuration* section.

        * *EVK-MCXN5XX* - Some VIT models can't fit into memory. In order to free some space it is necessary to disable SD card handling and opus encoder. To disable it, see the *Example configuration* section.

- Encoder
  - **OPUS:**
    * *LPCXpresso55s69* - The encoder is not supported due to insufficient memory.
- The File recording mode is not supported on *RW612BGA* development board due to missing SD card slot.

Known issues:

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

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

Hardware requirements

- Desired development board
- Micro USB cable
- Headphones with 3.5 mm stereo jack
- Personal computer
- Optional:
  - SD card for file output
  - Audio expansion board AUD-EXP-42448 (REV B)
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

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

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

Preparation

1. Connect a micro USB cable between the PC host and the debug USB port on the development board
2. Open a serial terminal with the following settings:
   - 115200 baud rate

   - 8 data bits
   - No parity
   - One stop bit
   - No flow control
3. Download the program to the target board.
4. Insert the headphones into the Line-Out connector (headphone jack) on the development board.
5. *LPCXpresso55s69:*
   - Insert source of sound to audio Line-In connector (headphone jack) on the development board.
6. Either press the reset button on your development board or launch the debugger in your IDE to begin running the demo.

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

```
******************************
Maestro audio record demo start
******************************

Copyright  2022  NXP
[APP_SDCARD_Task] start
[APP_Shell_Task] start

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

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"record_mic": Record MIC audio and perform one (or more) of following actions:
- playback on codec
- perform voice recognition (VIT)
- store samples to a file.

USAGE: record_mic [audio|file|<file_name>|vit] 20 [<language>]
The number defines length of recording in seconds.

Please see the project defined symbols for the languages supported.
Then specify one of: en/cn/de/es/fr/it/ja/ko/pt/tr as the language parameter.
For voice recognition say supported WakeWord and in 3s frame supported command.
Please note that this VIT demo is near-field and uses 1 on-board microphone.

NOTES: This command returns to shell after the recording is finished.
       To store samples to a file, the "file" option can be used to create a file
       with a predefined name, or any file name (without whitespaces) can be specified
       instead of the "file" option.

"opus_encode": Initializes the streamer with the Opus memory-to-memory pipeline and
encodes a hardcoded buffer.
```

Details of commands can be found *here*.

Example configuration The example can be configured by user. There are several options how to configure the example settings, depending on the environment. For configuration using west and Kconfig, please follow the instructions here. Before configuration, please check the *table* to see if the feature is supported on the development board.

- **Connect AUD-EXP-42448:**
  - *EVKC-MIMXRT1060:*
    1. Disconnect the power supply for safety reasons.
    2. Insert AUD-EXP-42448 into J19 to be able to use the CS42448 codec for multichannel output.
    3. Uninstall J99.
    4. Set the DEMO_CODEC_WM8962 macro to 0 in the app_definitions.h file
    5. Set the DEMO_CODEC_CS42448 macro to 1 in the app_definitions.h file.
  - *Note:*
    * The audio stream is as follows:
      - Stereo INPUT 1 (J12) -> LINE 1&2 OUTPUT (J6)
      - Stereo INPUT 2 (J15) -> LINE 3&4 OUTPUT (J7)
      - MIC1 & MIC2 (P1, P2) -> LINE 5&6 OUTPUT (J8)
      - Insert the headphones into the different line outputs to hear the inputs.
      - To use the Stereo INPUT 1, 2, connect an audio source LINE IN jack.
- **Enable VIT:**
  - *LPCXpresso55s69 and MCX-N5XX:*
    * In MCUXPresso IDE (SDK package):
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level.
      2. Add VIT_PROC symbol to preprocessor defines on project level:
         - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
    * In armgcc in SDK package:
      1. Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
      2. Remove OPUS_ENCODE=1 and STREAMER_ENABLE_ENCODER preprocessor defines in flags.cmake file.
      3. Add VIT_PROC symbol to preprocessor defines in flags.cmake file.
      4. Remove sdmmc_config.c,.h files from CMakeLists.txt file.
    * In Kconfig:
      1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
      2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd and MCUX_COMPONENT_middleware.sdmmc.host.usdhc
      3. Make sure sdmmc_config files (.c, .h) is excluded from project build

- remove mcux_add_source function that adds the sources in reconfig.cmake in maestro_record/cm33_core0 folder

4. Disable fatfs MCUX_COMPONENT_middleware.fatfs and MCUX_COMPONENT_middleware.fatfs.sd
5. Disable file utils MCUX_COMPONENT_middleware.audio_voice.maestro.file_utils.enable
6. Make sure Opus encoder is disabled MCUX_COMPONENT_middleware.audio_voice.maestro.element.encoder.opus.enable
7. Make sure VIT_PROC symbol is defined
   - remove mcux_remove_macro function that removes the VIT_PROC preprocessor definition in reconfig.cmake in maestro_record folder
8. Make sure VIT processing is enabled MCUX_PRJSEG_middleware.audio_voice.components.vit

- **VIT model generation:**
  - For custom VIT model generation (defining own wake words and voice commands) please use https://vit.nxp.com/
- **Disable SD card handling:**
  - In MCUXPresso IDE:
    * Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines on project level:
      - (Project -> Properties -> C/C++ Build -> Settings -> MCU C Compiler -> Preprocessor)
  - In armgcc in SDK package:
    * Remove SD_ENABLED and STREAMER_ENABLE_FILE_SINK symbols from preprocessor defines in flags.cmake file.
  - In Kconfig:
    1. Disable File sink MCUX_COMPONENT_middleware.audio_voice.maestro.element.file_sink.enable
    2. Make sure SD card support is disabled MCUX_COMPONENT_middleware.sdmmc.sd

Functionality The record_mic or opus_encode command calls the STREAMER_mic_Create or STREAMER_opusmem2mem_Create function from the app_streamer.c file depending on the selected mode.

- When the *Loopback* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements:
  - ELEMENT_MICROPHONE_INDEX
  - ELEMENT_SPEAKER_INDEX
- When the *File recording* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_FILE_SINK_INDEX
- When the *Voice control* mode is selected, the command calls the STREAMER_mic_Create function that creates a pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_VIT_INDEX

- When the Encoding mode is selected, the command calls the STREAMER_opusmem2mem_Create function that creates a pipeline with the following elements: - ELEMENT_MEM_SRC_INDEX - ELEMENT_ENCODER_INDEX - ELEMENT_MEM_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

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

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val  = DEMO_MIC_CHANNEL_NUM;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_BITS_PER_SAMPLE;
prop.val  = DEMO_AUDIO_BIT_WIDTH;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
prop.val  = DEMO_MIC_FRAME_SIZE;
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val  = DEMO_AUDIO_SAMPLE_RATE;
streamer_set_property(handle->streamer, 0, prop, true);
```

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

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *record_mic audio <time>*
- *record_mic file <time>*
- *record_mic <file_name> <time>*
- *record_mic vit <time> <language>*
- *opus_encode*

Legend for diagrams:

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

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

help, version

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

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

record_mic audio <time>

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

    B((Idle)):::state --> D{time
    > 0 ?}:::condition
    D -- Yes --> F[recording]:::function
    D -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    F --> C((Running)):::state
    C -->G{time
    expired?}:::condition
    G -- No --> C
    G -- Yes --> B
```

record_mic file <time>/record_mic <file_name> <time>

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

    B((Idle)):::state --> C{time
    > 0 ?}:::condition
    C -- Yes --> D{SD card
    inserted?}:::condition
    C -- No --> E[Error: Record length
    must be greater than 0]:::error
    E --> B
    D -- Yes --> G{Custom
    file name?}:::condition
    G -- Yes --> H[Create custom
    file name]:::function
    G -- No --> I[Create default
    file name]:::function
    H --> J[Recording]:::function
    I --> J
    J --> K((Running)):::state
```

K -->L{time
expired?}:::condition
L -- No --> K
L -- Yes --> B
D -- No --> F[Error: Insert SD
card first]:::error
F --> B

record_mic vit <time> <language>

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

B((Idle)):::state --> C{time
> 0 ?}:::condition
C -- Yes --> E{Selected
language?}:::condition
C -- No --> D[Error: Record length
must be greater than 0]:::error
D --> B
E -- Yes --> G{Supported
language?}:::condition
E -- No --> F[Error: Language
not selected]:::error
F -->B
G -- Yes -->I[Recording with
voice recognition]:::function
G -- No -->H[Error: Language not supported]:::error
H --> B
I --> J((Running)):::state
J -->K{time
expired?}:::condition
K -- No --> J
K -- Yes --> B

opus_encode

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

B((Idle)):::state -->C[Encode file]:::function
C -->D[Check result]:::function
D -->B

Processing Time Typical execution times of the streamer pipeline for the EVKC-MIMXRT1060 development board are detailed in the following table. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines were considered. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks.

For further details of execution times on individual elements, please refer to the *Processing Time* document.

	streamer
microphone -> speaker 1 channel	40 µs
microphone -> speaker 2 channels	115 µs
microphone -> VIT	7.4 ms

Maestro USB microphone example

Table of content

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

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

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the microphone inputs and sends them to the USB host via the USB bus. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

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

Limitations:

- *Note:*

  1. When connected to MacBook, change the PCM format from (0x02,0x00,) to (0x01,0x00,) in the g_config_descriptor[CONFIG_DESC_SIZE] in the `usb_descriptor.c` file. Otherwise, it can't be enumerated and noise is present when recording with the QuickTime player because the sampling frequency and bit resolution do not match.

2. When device functionality is changed, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.
3. If you're having audio problems on Windows 10 for recorder, please disable signal enhancement as the following if it is enabled and have a try again.

Known issues:

- No known issues.

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

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- *LPCXpresso55s69:*
  - Source of sound with 3.5 mm stereo jack connector

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

Preparation

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

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

```
******************************************************
Maestro audio USB microphone solutions demo start
******************************************************

Copyright 2022 NXP
```

(continues on next page)

(continued from previous page)

```
[APP_Shell_Task] start

>> usb_mic -1

Starting maestro usb microphone application
The application will run until the board restarts
[STREAMER] Message Task started
Starting recording
[STREAMER] start usb microphone
Set Cur Volume : 1f00
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_mic": Record MIC audio and playback to the USB port as an audio 2.0
          microphone device.

  USAGE: usb_mic <seconds>
    <seconds>  Time in seconds how long the application should run.
              When you enter a negative number the application will
              run until the board restarts.
  EXAMPLE:   The application will run for 20 seconds: usb_mic 20
```

Details of commands can be found *here*.

Example configuration The example only supports one mode and do not support any additional libraries, so the example can't be configured by user.

Functionality The usb_mic command calls the STREAMER_mic_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_MICROPHONE_INDEX - ELEMENT_USB_SINK_INDEX

Recording itself can be started with the STREAMER_Start function.

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

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_MICROPHONE_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_NUM_CHANNELS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_MICROPHONE_SET_FRAME_MS;
```

(continues on next page)

(continued from previous page)

```
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

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

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_mic <seconds>*

Legend for diagrams:

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

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

help, version

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

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

usb_mic <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{seconds
    == 0?}:::condition
    C -- No --> E{seconds
    < 0?}:::condition
    C -- Yes --> D[Error: Incorrect
```

command parameter]:::error
D -->B
E -- Yes --> G[recording]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[recording]:::function
F --> I((Running)):::state
I --> J{seconds
expired?}:::condition
J -- No -->I
J -- Yes --> B

Maestro USB speaker example

Table of content

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

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

The development board will be enumerated as a USB audio class 2.0 device on the USB host. The application takes audio samples from the USB host and sends them to the audio Line-Out port. User will see the volume levels obtained from the USB host but this is only an example application. To leverage the volume values, the demo has to be modified.

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

- **Standard** - The mode demonstrates playback with up to 2 channels, up to 48 kHz sample rate and up to 16 bit width. This mode is enabled by default.
- **Multi-Channel** - In this mode the device is enumerated as a UAC 5.1. This mode is disabled by default. See the *Example configuration* section to see how to enable the mode.
  - When playing an 5.1 audio file, the example sends only the front-left and front-right channels to the audio Line-Out port (the other channels are ignored), since this example only supports on-board codecs with stereo audio output.

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

Limitations:

- *Note:*
  - If the USB device audio speaker example uses an ISO IN feedback endpoint, please attach the device to a host like PC which supports feedback function. Otherwise, there might be attachment issue or other problems.

Known issues:

- No known issues.

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

Hardware requirements

- Desired development board
- 2x Micro USB cable
- Personal Computer
- Headphones with 3.5 mm stereo jack

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

Preparation

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

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

```
*******************************************************
Maestro audio USB speaker solutions demo start
*******************************************************


Copyright  2022  NXP
[APP_Shell_Task] start

>> usb_speaker -1
```

(continues on next page)

(continued from previous page)

```
Starting maestro usb speaker application
The application will run until the board restarts
[STREAMER] Message Task started
Starting playing
[STREAMER] start usb speaker
Set Cur Volume : fbd5
```

Type help to see the command list. Similar description will be displayed on serial console:

```
>> help

"help": List all the registered commands

"exit": Exit program

"version": Display component versions

"usb_speaker": Play data from the USB port as an audio 2.0
               speaker device.

  USAGE:     usb_speaker <seconds>
   <seconds>  Time in seconds how long the application should run.
              When you enter a negative number the application will
              run until the board restarts.
   EXAMPLE:   The application will run for 20 seconds: usb_speaker 20
```

Details of commands can be found *here*.

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

- **Enable Multi-channel mode:**
  - The feature can be enabled by set the USB_AUDIO_CHANNEL5_1 macro to 1U in the usb_device_descriptor.h file.
  - *Note:* When device functionality is changed, such as UAC 5.1, please uninstall the previous PC driver to make sure the device with changed functionality can run normally.

Functionality The Usb_speaker command calls the STREAMER_speaker_Create function from the app_streamer.cfile that creates pipeline with the following elements: - ELEMENT_USB_SRC_INDEX - ELEMENT_SPEAKER_INDEX

Playback itself can be started with the STREAMER_Start function.

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

```
ELEMENT_PROPERTY_T prop;

prop.prop = PROP_USB_SRC_SET_SAMPLE_RATE;
prop.val  = AUDIO_SAMPLING_RATE;

streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_NUM_CHANNELS;
prop.val  = 2;
```

(continues on next page)

(continued from previous page)

```
streamer_set_property(handle->streamer, 0, prop, true);

prop.prop = PROP_USB_SRC_SET_FRAME_MS;
prop.val  = 1;

streamer_set_property(handle->streamer, 0, prop, true);
```

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

States The application can be in 2 different states:

- Idle
- Running

Commands in detail

- *help, version*
- *usb_speaker <seconds>*

Legend for diagrams:

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

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

help, version

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

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

usb_speaker <seconds>

```
flowchart TD
    classDef function fill:#c6d22c
    classDef condition fill:#7cb2de
    classDef state fill:#fcb415
    classDef error fill:#FF999C

    B((Idle)):::state -->C{Duration
    == 0?}:::condition
```

C -- No --> E{Duration
< 0?}:::condition
C -- Yes --> D[Error: Incorrect
command parameter]:::error
D -->B
E -- Yes --> G[playing]:::function
G --> H((Running)):::state
H --> H
E -- No --> F[playing]:::function
F --> I((Running)):::state
I --> J{Duration
expired?}:::condition
J -- No -->I
J -- Yes --> B

Supported features The current version of the audio framework supports several optional features. These can be limited to some MCU cores or development boards variants. More information about support can be found on the specific example page:

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

Some features are delivered as prebuilt library and the binaries can be found in the \middleware\audio_voice\components*component*\libs folder. The source code of some features can be found in the \middleware\audio_voice\maestro\src folder.

Decoders Supported decoders and its options are:

Decoder	Sample rates [kHz]	Number of channels	Bit depth
AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
FLAC	8, 11.025, 12, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	16
MP3	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48	1, 2 (mono/stereo)	16
OPUS	8, 16, 24, 48	1, 2 (mono/stereo)	16
WAV	8, 11.025, 16, 22.05, 32, 44.1, 48	1, 2 (mono/stereo)	8, 16, 24

For more details about the reference decoders please see audio-voice-components repository documentation \middleware\audio_voice\components\.

Encoders

- **OPUS encoder** - The current verion of the audio framework only supports a OPUS encoder. For more details about the encoder please see the following link.

Sample rate converters

- **SSRC** - Synchronous sample rate converter. More details about SSRC are available in the User Guide, which is located in middleware\audio_voice\components\ssrc\doc\.
- **ASRC** - Asynchronous sample rate converter is not used in our examples, but it is part of the maestro middleware and can be enabled. To enable ASRC, the maestro_framework_asrc and CMSIS_DSP_Library_Source components must be added to the project. Furthermore, it is necessary to switch from Redlib to Newlib (semihost) library and add a platform definition

to the project (e.g. for RT1170: PLATFORM_RT1170_CORTEXM7). Supported platforms can be found in the PL_platformTypes.h file. More details about ASRC are available in the User Guide, which is located in middleware\audio_voice\components\asrc\doc\.

Additional libraries

- **VIT** - Voice Intelligent Technology (VIT) Wake Word and Voice Command Engines provide free, ready to use voice UI enablement for developers. It enables customer-defined wake words and commands using free online tools. More details about VIT are available in the VIT package, which is located in middleware\audio_voice\components\vit\{platform}\Doc\(depending on the platform) or via following link.

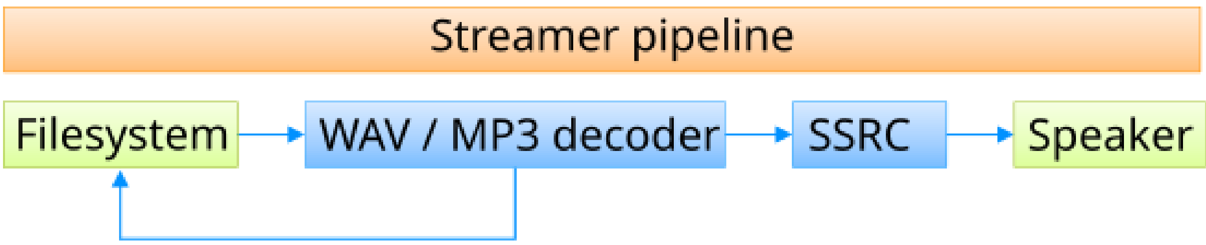
Processing Time

Table of content

- *Maestro playback example*
- *Maestro record example*

The individual time measurements were conducted using a logic analyzer by monitoring changes in the GPIO port levels on the EVKC-MIMXRT1060 development board. These measurements were executed for each individual pipeline run, capturing the timing at each corresponding element, and, when relevant, the interconnections between these elements.

Maestro playback example For the Maestro playback example the following reference audio file was used: test_48khz_16bit_2ch.wav. In this example, the pipeline depicted in the diagram was considered. Media codecs WAV and MP3 were taken into account. To compare the times spent on the SSRC block, sampling rates for both codecs were selected: 44.1 kHz and 48 kHz.



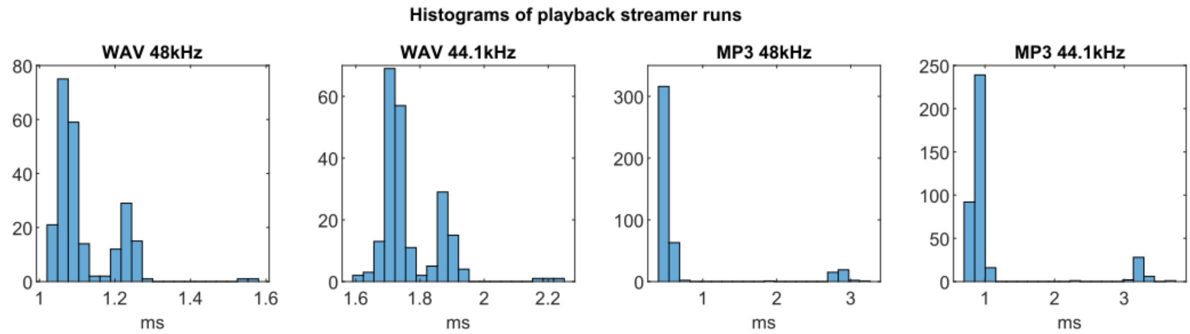
The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines(): streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.

In the scenario involving the WAV codec, the audio file was accessed in every iteration of the streamer pipeline. Meaning, during each run, the file was read directly from the SD card. However, in the case of the MP3 codec, where data processing necessitates complete MP3 frames, the file wasn't read during every run. Rather, it was accessed periodically, triggered when the codec buffer lacked a complete MP3 frame of data. The total time spent on codec processing varies significantly depending on the type and implementation of the codec. For certain types of codecs, like FLAC, there may be multiple file accesses during a single pipeline run. The provided values are specific to the reference implementation. For details about the codecs please see see audio-voice-components documentation middleware\audio_voice\components\.

The duration of the streamer pipeline illustrates that with a sampling frequency of 48 kHz, there is no resampling occurring at the SSRC element. Consequently, the overall pipeline time is lower than in the case of 44.1 kHz audio, where resampling takes place.

To enhance comprehension of the system's behavior, histograms of the pipeline run times and its elements are included. The greater time variance with the MP3 codec is precisely due to the absence of file reads in every run. In clusters with shorter times, there are no file accesses, while in clusters with longer times, file reads occur. This indicates that the majority of runs do not involve file access.

	WAV 48 kHz	WAV 44 kHz	MP3 48 kHz file read	MP3 48 kHz w/o file read	MP3 44 kHz file read	MP3 44 kHz w/o file read
mear	1.11 ms	1.76 ms	2.87 ms	0.51 ms	3.22 ms	0.89 ms
min	1.03 ms	1.60 ms	2.74 ms	0.41 ms	2.33 ms	0.74 ms
max	1.29 ms	2.23 ms	3.24 ms	1.83 ms	3.73 ms	1.12 ms



Time on each element In the tables and histograms below, the timings for individual elements and their connections are provided. Given that the file reading function was invoked during the codec's operation, the tables for individual elements display the total time on the codec element, the time on the codec element before the file read, and the time on the codec element after the file read. The individual blocks in the tables are as follows:

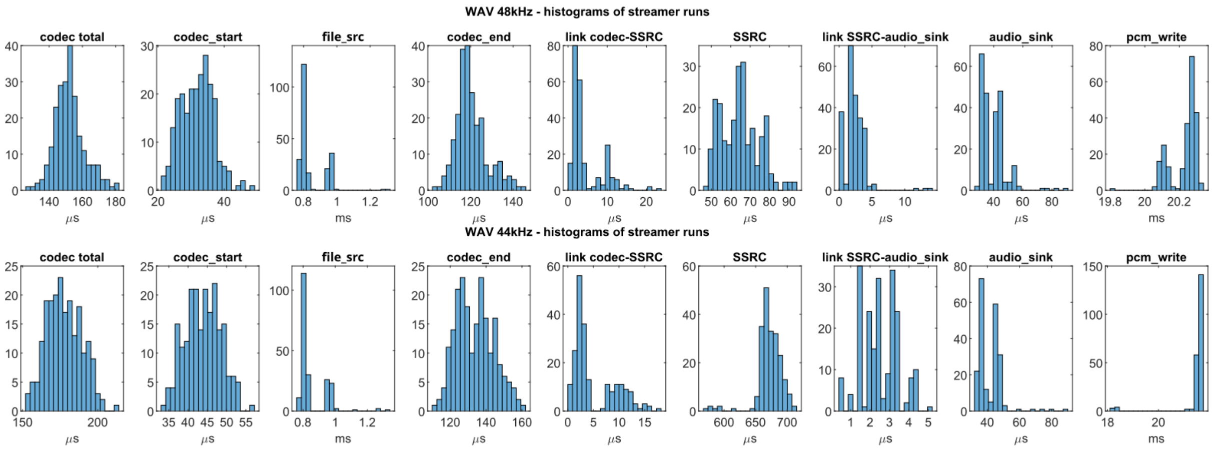
- **streamer** - total time of one pipeline run without time on output buffers
- **codec start** - time on decoder before file read
- **codec end** - time on decoder after file read
- **codec total** - codec_start+codec_end
- **file_src** - file reading time
- **SSRC_proc** - time on SSRC element
- **audio_sink** - time on audio sink without ouput buffers
- **pcm_write** - time on output buffers
- **link** - time on element links

The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **codec** - decoder_sink_pad_process_handler():decoder_pads.c
- **file_src** - filesrc_read():file_src_rtos.c
- **SSRC_proc** - SSRC_Proc_Execute():ssrc_proc.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

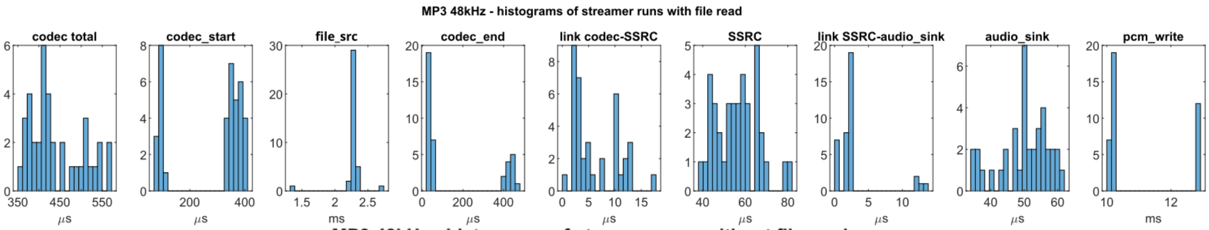
WAV 48kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.119 ms	152 µs	31 µs	0.843 ms	120 µs	5 µs	64 µs	2 µs	40 µs	20.228 ms
min	1.026 ms	125 µs	21 µs	0.773 ms	104 µs	<1 µs	47 µs	<1 µs	30 µs	19.805 ms
max	1.290 ms	193 µs	49 µs	1.311 ms	144 µs	23 µs	93 µs	14 µs	91 µs	20.324 ms

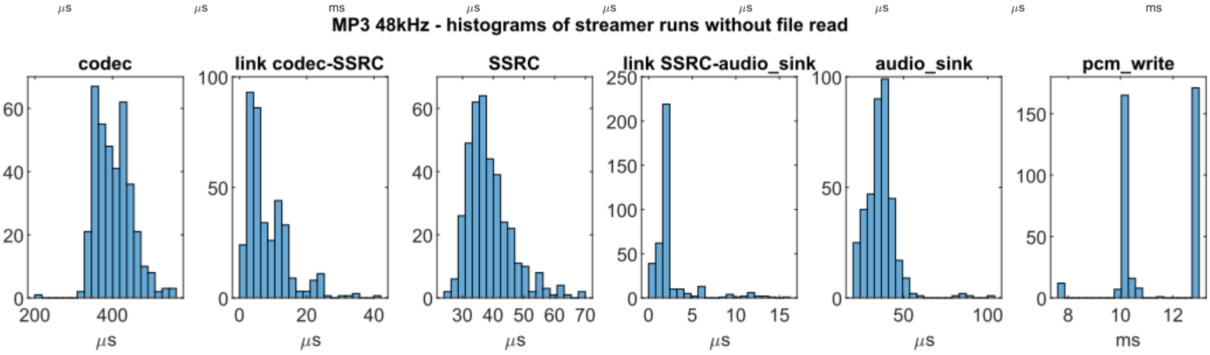
WAV 44kHz	strean	codec total	codec start	file_sr	codec end	link codec-SSRC	SSRC_p	link SSRC-audio_sink	au-dio_sin	pcm_write
mean	1.765 ms	178 µs	44 µs	0.853 ms	134 µs	5 µs	671 µs	3 µs	42 µs	21.472 ms
min	1.604 ms	145 µs	33 µs	0.770 ms	112 µs	<1 µs	574 µs	<1 µs	33 µs	18.163 ms
max	2.233 ms	218 µs	57 µs	1.335 ms	161 µs	18 µs	715 µs	5 µs	89 µs	21.746 ms



MP3 48 kHz w/ file read	strear	codec total	codec start	file_sı	codec end	link codec-SSRC	SSRC_	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	2.871 ms	441 µs	279 µs	2.271 ms	162 µs	6 µs	56 µs	3 µs	50 µs	11.019 ms
min	2.739 ms	353 µs	74 µs	1.353 ms	26 µs	<1 µs	40 µs	<1 µs	34 µs	10.091 ms
max	3.244 ms	570 µs	409 µs	2.728 ms	467 µs	18 µs	80 µs	14 µs	62 µs	12.910 ms

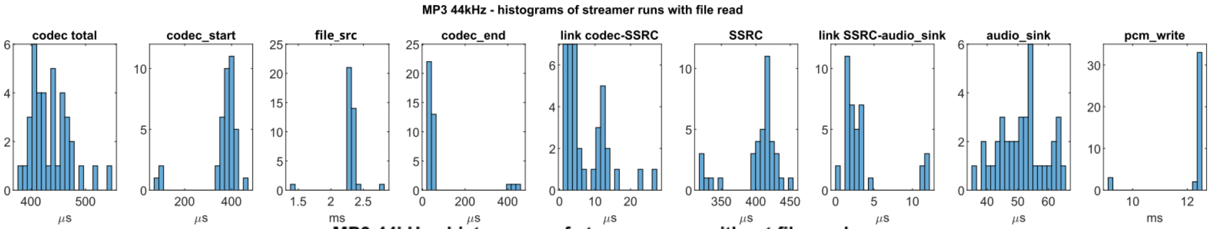
MP3 48 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_|	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.508 ms	403 µs	x	x	x	8 µs	39 µs	3 µs	36 µs	11.326 ms
min	0.407 ms	208 µs	x	x	x	<1 µs	25 µs	<1 µs	21 µs	7.715 ms
max	1.834 ms	563 µs	x	x	x	41 µs	69 µs	16 µs	104 µs	12.941 ms

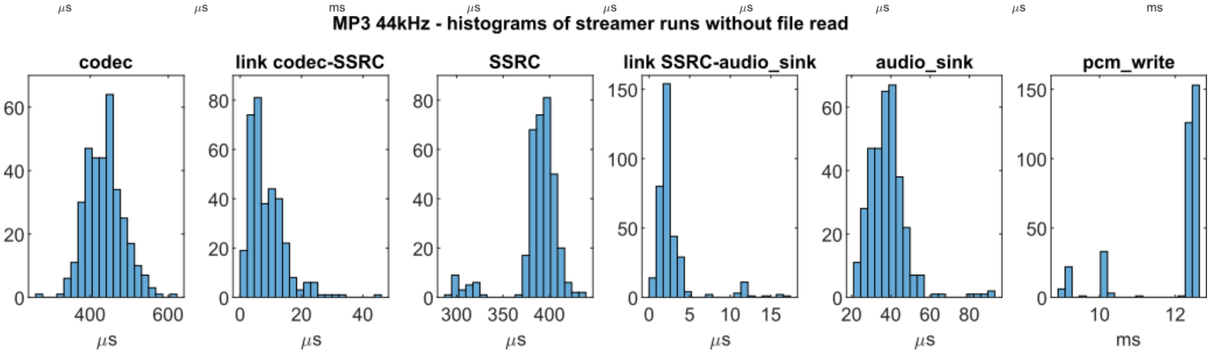




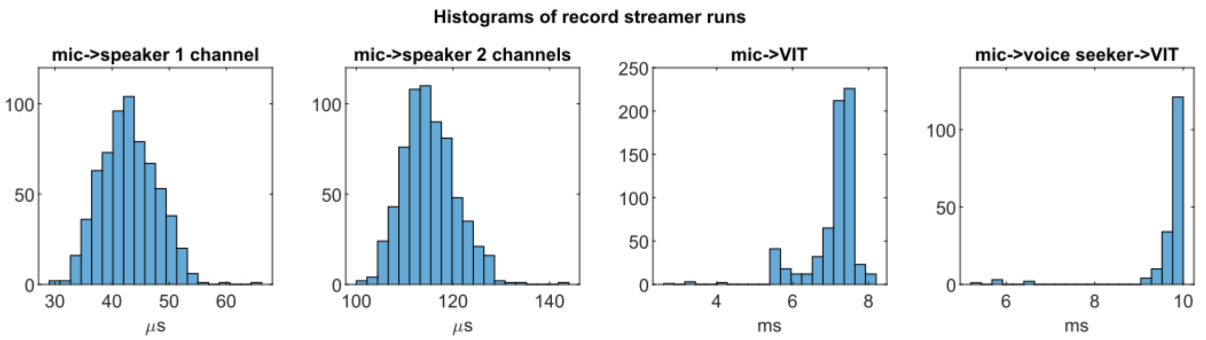
MP3 44 kHz w/ file read	strear	codec total	codec start	file_sı	codec end	link codec-SSRC	SSRC_|	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	3.217 ms	436 µs	367 µs	2.300 ms	66 µs	7 µs	403 µs	3 µs	51 µs	12.188 ms
min	2.329 ms	383 µs	73 µs	1.411 ms	26 µs	2 µs	318 µs	<1 µs	35 µs	9.119 ms
max	3.726 ms	547 µs	464 µs	2.801 ms	441 µs	27 µs	454 µs	12 µs	65 µs	12.529 ms

MP3 44 kHz w/o file read	strear	codec total	codec start	file_s	codec end	link codec-SSRC	SSRC_|	link SSRC-audio_sink	au-dio_sir	pcm_write
mean	0.891 ms	437 µs	x	x	x	9 µs	388 µs	3 µs	38 µs	11.934 ms
min	0.738 ms	268 µs	x	x	x	<1 µs	290 µs	<1 µs	22 µs	8.964 ms
max	1.115 ms	620 µs	x	x	x	45 µs	438 µs	17 µs	92 µs	12.624 ms





Maestro record example Typical execution times of the streamer pipeline and its individual elements for the EVKC-MIMXRT1060 development board are detailed in the following tables. The duration spent on output buffers and reading from the microphone is excluded from traversal measurements. Three measured pipelines are depicted in the figure below. The first involves a loopback from microphone to speaker, supporting both mono and stereo configurations. The second pipeline is a mono voice control setup, comprising microphone and VIT blocks. The final pipeline is a stereo voice control setup, integrating microphone and VIT blocks. The measurement of streamer pipeline run started at the beginning of streamer_process_pipelines():streamer.c and ended in the function streamer_pcm_write(): streamer_pcm.c just before the output buffer.



The individual blocks in the tables are as follows:

- **streamer** - total time of one pipeline run without time on output buffers and without time reading from the microphone
- **audio_src_start** - time on audio src before reading from the microphone
- **audio_src_end** - time on audio src after reading from the microphone
- **pcm_read** - reading from the microphone
- **vit** - time on VIT element
- **audio_sink** - time on audio sink without ouput buffers

- **pcm_write** - time on output buffers
- **link** - time on element links

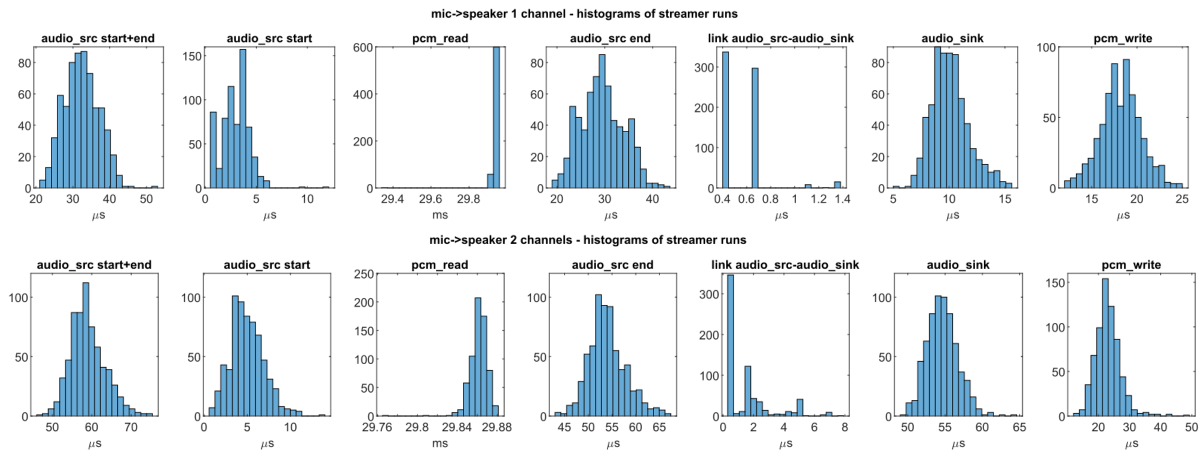
The start times of the time intervals for individual blocks and their respective links were measured by altering the GPIO pin level in the following functions:

- **streamer** - streamer_process_pipelines():streamer.c
- **audio_src** - audiosrc_src_process():audio_src.c
- **pcm_read** - streamer_pcm_read():streamer_pcm.c
- **vit** - vitsink_sink_pad_chain_handler():vit_sink.c
- **audio_sink** - audiosink_sink_pad_chain_handler():audio_sink.c
- **pcm_write** - streamer_pcm_write():streamer_pcm.c
- **link** - pad_push():pad.c

Pipeline Microphone -> Speaker

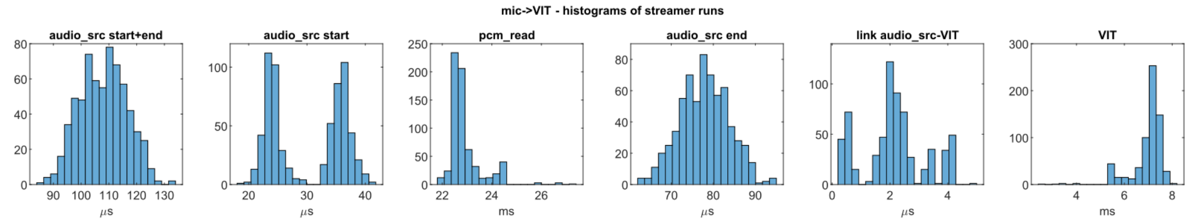
microphone -> speaker mono	stream	au-dio_src_sta	pcm_re	au-dio_src_er	link audio_src-audio_sink	au-dio_sink	pcm_write
mean	43 µs	3 µs	29.938 ms	29 µs	<1 µs	10 µs	18 µs
min	26 µs	<1 µs	29.350 ms	19 µs	<1 µs	5 µs	12 µs
max	72 µs	12 µs	29.957 ms	44 µs	1 µs	15 µs	25 µs

microphone -> speaker stereo	stream	au-dio_src_sta	pcm_re	au-dio_src_er	link audio_src-audio_sink	au-dio_sink	pcm_write
mean	115 µs	5 µs	29.861 ms	54 µs	2 µs	55 µs	23 µs
min	94 µs	<1 µs	29.768 ms	43 µs	<1 µs	50 µs	12 µs
max	154 µs	14 µs	29.880 ms	67 µs	8 µs	65 µs	49 µs



Pipeline Microphone -> VIT

microphone -> VIT	streamer	audio_src_start	pcm_read	audio_src_end	link audio_src-vit	vit
mean	7.380 ms	30 µs	22.624 ms	78 µs	2 µs	7.261 ms
min	2.641 ms	10 µs	2.2265 ms	58 µs	<1 µs	2.559 ms
max	7.780 ms	42 µs	2.7341 ms	94 µs	5 µs	7.624 ms



Maestro on Zephyr

- Based on and tested with Zephyr version, given by tag v4.0.0
- Tested with Zephyr SDK version 16.4
- To see the pre-built documentation, see: README.html. Also see the *documentation section*.

Maestro sample for recording data from microphone to RAM

Description This sample records data from microphone (alias dmic0 in devicetree) and stores them to a buffer in RAM.

Currently one PDM channel with fixed 16 kHz sample rate and 16 bit sample width is supported.

For configuration options, see Kconfig and prj.conf.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro voice detection sample using VIT

Description Records data from microphone (alias dmic0 in devicetree) and detects voice commands from selected language model. Detected commands are printed via UART.

Language model may be changed via Kconfig using CONFIG_MAESTRO_EXAMPLE_VIT_LANGUAGE selection. For other configuration options, see example's Kconfig and prj.conf.

This project requires an NXP board supported by the VIT library.

The example has to be modified if a new board needs to be added. Please create an issue in that case.

User Input/Output

- Input:

  None.

- Output:

  UART Output:

  - List of voice commands the model can detect (printed immediately after start)
  - <Specific voice command> if voice command was detected
  - Demo result: FAIL otherwise

Dependencies

- VIT library: https://www.nxp.com/design/design-center/software/embedded-software/voice-intelligent-technology-wake-word-and-voice-command-engines:VOICE-INTELLIGENT-TECHNOLOGY

Supported platforms Currently tested for:

- RD_RW612_BGA.

Maestro decoder sample

Description Tests and demonstrates decoder functionality in Maestro pipeline.

Supported decoders:

- MP3
- WAV
- AAC
- FLAC
- OPUS with OGG envelop
- (RAW OPUS - TBD)

Data Input:

- Prepared encoded audio data (part of Maestro repository, folder zephyr/audioTracks)
- Prepared decoded audio data (RAW PCM format, part of Maestro repository, folder zephyr/audioTracks)

Function:

1. Loads encoded data into source buffer stored in RAM

2. Decodes audio data using selected decoder and stores data in RAM
3. Compares prepared data with decoded data to check if its the same
4. Prints Demo result: OK or Demo result: FAIL via UART

User Input/Output

- Input:

  None

- Output:

  UART Output

  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Decoder libraries

Configuration

- See prj.conf for user input sections
  - Selecting decoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_DECODER_SELECTED in prj.conf file. When no decoder is selected, default one (WAV) is used instead.
  - System settings should be modified (stack size, heap size) based on selected decoder and system capabilities/requirements in prj.conf.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

Currently tested for:

- RD_RW612_BGA - Working decoders: FLAC, WAV, OPUS OGG

Maestro encoder sample

Description

Tests and demonstrates encoder functionality in Maestro pipeline.

Supported encoders:

- OPUS with OGG envelop - TBD
- RAW OPUS - TBD

Input:

- Prepared decoded audio data (RAW PCM format, part of Maestro repository)
- Prepared encoded audio data (part of Maestro repository)

Function:

1. Loads RAW data into source buffer stored in RAM
2. Encodes audio data using selected encoder and stores data in RAM
3. Compares prepared data with decoded data if same
4. Prints Demo result: OK or Demo result: FAIL via UART

Dependencies

- Audio voice component library (pulled in by Maestro's west), containing Encoder libraries

User Input/Output

Input:

- None

Output:

- UART Output
  - Demo result: OK if everything went OK
  - Demo result: FAIL otherwise

Configuration

- See prj.conf for user input sections
  - Selecting encoder may be done by enabling CONFIG_MAESTRO_EXAMPLE_ENCODER_SELECTED in prj.conf file. When no encoder is selected, default one (OPUS) is used instead.
  - System settings should be modified (stack size, heap size) based on selected encoder and system capabilities/requirements in prj.conf file.
- For other configuration options, see example's Kconfig and prj.conf.

Supported platforms

Currently tested for:

- RD_RW612_BGA - Working encoders: None.

Maestro mem2mem sample

Description

Tests basic memory to memory pipeline.

Function:

1. Moves generated data with fixed size of 256B from memory source to memory sink.
2. Compares copied data to check if they're the same.
3. Returns Demo result: OK or Demo result: FAIL via UART.

- *Maestro environment setup*
- *Build and run Maestro example*
  - *Using command line*
  - *Using MCUXpresso for VS Code*
- *Folder structure*
- *Supported elements and libraries*
- *Examples support*
- *Creating your own example*
- *Documentation*
- *FAQ*

Maestro environment setup Follow these steps to set up a Maestro development environment on your machine.

1. If you haven't already, please follow this guide to set up a Zephyr development environment and its dependencies first:
   - Cmake
   - Python
   - Devicetree compiler
   - West
   - Zephyr SDK bundle
2. Get Maestro. You can pick either of the options listed below. If you need help deciding which option is the best fit for your needs, please see the *FAQ*.
   - Freestanding Maestro - This option pulls in only Maestro's necessary dependencies.

     Run:

     ```
     1. west init -m <maestro repository url> --mr <revision> --mf west-freestanding.yml
     ↪<foldername>
     2. cd <foldername>
     3. west update
     ```

   - Maestro as a Zephyr module

     To include Maestro into Zephyr, update Zephyr's `west.yml` file:

     ```
     projects:
     name: maestro
     url: <maestro repository url>
     revision: <revision with Zephyr support>
     path: modules/audio/maestro
     import: west.yml
     ```

     Then run `west update maestro` command.

Build and run Maestro example These steps will guide you through building and running Maestro samples. You can use either the command line utilizing Zephyr's powerful `west` tool or you can use VS Code's GUI. Detailed steps for both options are listed below.

Using command line See Zephyr's Building, Flashing and Debugging guide if you aren't familiar with it yet.

1. To **build** a project, run:

   ```
   west build -b <board> -d <output build directory> <path to example> -p
   ```

   For example, this compiles VIT example for rd_rw612_bga board:

   ```
   1. cd maestro/zephyr
   2. west build -b rd_rw612_bga -d build samples/vit -p
   ```

2. To **run** a project, run:

   ```
   west flash -d <directory>
   ```

   e.g.:

```
west flash -d build
```

3. To **debug** a project, run:

   ```
   west debug -d <directory>
   ```

   e.g.:

   ```
   west debug -d build
   ```

Using MCUXpresso for VS Code For this you have to have NXP's MCUXpresso for VS Code extension installed.

1. Import your topdir as a repository to MCUXPresso for VS Code:
   - Open the MCUXpresso Extension. In the *Quickstart Panel* click *Import Repository*.
   - In the displayed menu click *LOCAL* tab and select the folder location of your *topdir*.
   - Click *Import*.
   - The repository is successfully added to the *Installed Repositories* view once the import is successful.
2. To import any project from the imported repository:
   - In the *Quickstart Panel* click *Import Example from Repository*.
   - For **Repository** select *your imported* repository.
   - For **Zephyr SDK** the installed Zephyr SDK is selected automatically. If not, select one.
   - For **Board** select your board (*make sure* you've selected the *correct* revision).
   - For **Template** select the folder path to your project.
   - Click the *Create* button.
3. Build the project by clicking the *Build Selected* icon (displayed on hover) in the extension's *Projects* view. After the build, the debug console window displays the memory usage (or compiler errors if any).
4. Debug the project by clicking the *Debug* (play) icon (displayed on hover) in the extension's *Projects* view.
5. The execution will pause. To continue execution click *Continue* on the debug options.
6. In the *SERIAL MONITOR* tab of your console panel, the application prints the Zephyr boot banner during startup and then prints the test results.

Folder structure

```
maestro/
    ...
    zephyr/              All Zephyr related files
        samples/           Sample examples
        tests/           Tests
        audioTracks/       Audio tracks for testing
        doc/             Documentation configuration for Sphinx
        wrappers/          NXP SDK Wrappers
        scripts/         Helper scripts, mostly for testing
        module.yml         Defines module name, Cmake and Kconfig locations
        CMakeList.txt      Defines module's build process
        Kconfig          Defines module's configuration
        osa/             Deprecated. OSA port for Zephyr
        ...
```

Supported elements and libraries Here is the list of all features currently supported in Maestro on Zephyr. Our goal is to support all features in Maestro on Zephyr that are already supported in Maestro on NXP's SDK and to extend them further.

Supported elements:

- Memory source
- Memory sink
- Audio source
- Audio sink
- Process sink
- Decoder
- Encoder

Supported decoders:

- WAV
- MP3
- FLAC
- OPUS OGG
- AAC

Supported encoders:

- OPUS RAW

Supported libraries:

- VIT

Examples support All included examples use UART as output. Examples are located in `zephyr/tests` and `zephyr/samples` directories.

List of included examples:

- *Maestro sample for recording data from microphone to RAM*
- *Maestro voice detection sample using VIT*
- *Maestro encoder sample*
- *Maestro decoder sample*
- *Maestro mem2mem sample*

Examples support for specific boards:

Example	RDRW612BGA	LPCxpresso55s69	MIMXRT1060EVKB	MIMXRT1170EVKB
Record	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
VIT	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED
Encoder	In progress: OPUS RAW	TO BE TESTED	TO BE TESTED	TO BE TESTED
Decoder	YES - WAV, FLAC, OPUS OGG	TO BE TESTED	TO BE TESTED	TO BE TESTED
Mem2mem	YES	TO BE TESTED	TO BE TESTED	TO BE TESTED

Creating your own example There are two ways to create your own example - you can either one of the included examples as a reference or you can create your own example from scratch by hand.

When creating your own example from scratch, set CONFIG_MAESTRO_AUDIO_FRAMEWORK=y in your prj.conf file. Then you can start enabling specific elements by setting CONFIG_MAESTRO_ELEMENT_<NAME>_ENABLE=y.

However, the recommended way to edit config options is to open gui-config (or menuconfig) by calling west build -t guiconfig. Then you can use the graphical interface to interactively turn on/off the features you need.

Documentation Please note, Maestro documentation is under reconstruction. It is currently mixing several tools and formats.

To see the pre-generated Maestro Zephyr documentation, see zephyr/doc/doc/README.html

To generate the Zephyr documentation, go under zephyr/doc folder and execute make html. Sphinx version sphinx-build 8.1.3 must be installed. Open doc/doc/html/README.hml afterwards.

To see Maestro core documentation, go to the Maestro top directory and see README.md.

FAQ

1. Should I choose the freestanding version of Maestro or should integrate it into my west instead?
   - Freestanding version of Maestro pulls in all the dependencies it needs including Zephyr itself.
   - Integrating it as a module is easier if you already have your Zephyr environment set up.

Maestro Audio Framework changelog

2.0.2

- Removed VoiceSeeker support

2.0.1

- Fixed filesrc buffer alignment

2.0.0 (newest)

- Added Zephyr port, see *Zephyr README*.
  - Possible to use standalone version, pulling its own Zephyr and dependencies
  - Possible to import it as a module in your Zephyr project
- Changed build system - newly uses Kconfig and Cmake
- Supports NXP MCUXSDK (previously 2.x)
- Changed folder structure and names to improve readability (description may be found in *README*)
- Removed audio libraries and placed into audio-voice-components repository
- Added libraries are pulled into the build via Kconfig and Cmake

- Changed Maestro library core - minor changes

1.8.0

- New platforms support: MCX-N5XX-EVK, FRDMMCXN236 and RD-RW612-BGA
- Fixed compilation warnings
- Documentation improvements and updates
  - Added section with processing time information
  - Added application state diagrams
- Various updates and fixes

1.7.0

- Removed EAP support for future SDK releases
- Created new API for audio_sink and audio_src to support USB source, sink
- ASRC library integrated
- License changed to BSD 3-Clause
- Improved pipeline creation API
- Fixed compilation warnings in Opus
- Various other improvements and bug fixes

1.6.0

- Up to 2 parallel pipelines supported
- Synchronous Sample Rate Converter support Added
- Various improvements and bug fixes

1.5.0

- Enabled switching from 2 to 4 channel output during processing
- PadReturn type has been replaced by FlowReturn
- Support of AAC, WAV, FLAC decoders
- Renamed eap element to audio_proc element
- Added audio_proc to VIT pipeline to support VoiceSeeker
- Minor bug fixes

1.4.0

- Use Opusfile lib for Ogg Opus decoder
- Refactor code, fix issues found in unit tests
- Various bug fixes

1.3.0

- Make Maestro framework open source (except mp3 and wav decoder)
- Refactor code, remove unused parts, add comments

1.2.0

- Unified buffering in audio source, audio sink
- Various improvements and bug fixes

1.0_rev0

- Initial version of framework with support for Cortex-M7 platforms

1.7.2 VGLite Graphics Driver

IMXRTVGLITEAPIRM

Introduction The VGLite Graphics API (Application Programming Interface) is designed to support 2D vector and 2D raster-based operations for rendering the interactive user interface that may include menus, fonts, curves, and images. The goal is to provide the maximum 2D vector/raster rendering performance, while keeping the memory footprint to the minimum.

Note: This document contains proprietary information of VeriSilicon Holdings Co., Ltd, and Vivante Corporation.

VGLite Graphics API The Vivante VGLite Graphics API is used to control the Vivante vector graphics hardware units that provide accelerated vector and raster operations.

The Vivante VGLite API is developed for use with Vivante GCNanoLiteV, GCNanoUltraV, GCNanoV, GC355, and GC555 hardware. GC355 and GC555 support the Khronos OpenVG 1.1 feature set, while GCNanoLiteV, GCNanoUltraV and GCNanoV have a feature set smaller than that required to pass Khronos OpenVG CTS.

The VGLite API driver V4 is a new design and implementation of the driver (from 2023Q1) to support the new generation 2.5D GPU (GC555), and the previous 2.5D GPU releases (GC255, GC265, GC355). The new V4 driver supports the new and improved VGLite API (version 3.0) and can generate the most CPU-efficient, customized driver build for a specific 2.5D GPU release based on the hardware feature set.

VGLite API supported features include: Porter-Duff Blending, Gradient Controls, Fast Clear, Arbitrary Rotations, Path Filling rules, Path painting, and Pattern Path Filling.

By default, VGLite API driver V4 supports one implicit global application context in a single thread. VGLite V4 driver does not support multithreaded applications, which is not suitable for embedded IoT devices.

Parent topic:*Introduction*

API function group The VGLite Graphics API has been designed to have independent function groups. It is permissible for a user to use only one of the function groups in the VGLite application:

- **Initialization** is used for initializing hardware and software structures
- **Blit API** is used for the raster part of rendering
- **Draw API** is used for 2D vector-based draw operations

Parent topic:*Introduction*

API files The VGLite source code is available as part of the NXP MCUXpresso SDK:

The VGLite graphics API functions are defined in the header file VGLite/inc/vg_lite.h.

All VGLite enumerations and data types are defined in VGLite/inc/vg_lite.h.

Parent topic:*Introduction*

Hardware versions The Vivante VGLite API is compatible with a range of Vivante Vector Graphics IPs including: GCNanoLiteV, GCNanoUltraV, GCNanoV, GC355, and GC555.

Note: A specific hardware version has customized feature set that may limit hardware support for some VGLite API options. The VGLite application can use the vg_lite_query_feature API to query specific VGLite feature availability.

Users can also check the VGLite/VGLite/vg_lite_options.h file which includes CHIPID, REVISION, CID to identify specific HW releases, and the gcFEATURE_VG_* macros to define the feature set for the HW release.

The gcFEATURE_VG_* macro values (except a few SW features) should NOT be changed. Otherwise, the VGLite driver does not function correctly on the specific HW release. Users can change the "SW Features" macro values to disable some software features, unnecessary error checks, or enable VGLite API trace for debug purposes.

.

Parent topic:*Introduction*

Common parameters and error values This chapter provides an overview of the common parameter types and the enumeration used for error reporting.

Common parameter types The VGLite graphics API uses a naming convention scheme wherein definitions are preceded by vg_lite.

Below is the list of types and structures in the driver implementation.

<table>
<tr><th>Nam</th><th>Type-def</th><th>Value</th></tr>
<tr><td>vg_li</td><td>int</td><td>A signed 32-bit integer 0: FALSE; 1: TRUE.</td></tr>
<tr><td>vg_li</td><td>char</td><td>A signed 8-bit integer</td></tr>
<tr><td>vg_li</td><td>un-signe char</td><td>An unsigned 8-bit integer</td></tr>
<tr><td>vg_li</td><td>short</td><td>A signed 16-bit integer</td></tr>
<tr><td>vg_li</td><td>un-signe short</td><td>An unsigned 16-bit integer</td></tr>
<tr><td>vg_li</td><td>int</td><td>A signed 32-bit integer</td></tr>
<tr><td>vg_li</td><td>un-signe int</td><td>An unsigned 32-bit integer</td></tr>
<tr><td>vg_li</td><td>un-signe long long</td><td>An unsigned 64-bit integer</td></tr>
<tr><td>vg_li</td><td>float</td><td>A 32-bit single precision floating point number</td></tr>
<tr><td>vg_li</td><td>dou-ble</td><td>A 64-bit double precision floating point number</td></tr>
<tr><td>vg_li</td><td>char</td><td>A signed 8-bit integer</td></tr>
<tr><td>vg_li</td><td>char*</td><td>A pointer to a character string</td></tr>
<tr><td>vg_li</td><td>void*</td><td>A generic address pointer (void *). On 32-bit OS, it is a 32-bit address pointer. On 64-bit OS, it is a 64-bit address pointer.</td></tr>
<tr><td>vg_li</td><td>void</td><td>The void type</td></tr>
<tr><td>vg_li</td><td>vg_lit</td><td>A 32-bit color value The color value specifies the color used in various functions. The color is formed using 8-bit RGBA channels. The red channel is in the lower 8-bit of the color value, followed by the green and blue channels. The alpha channel is in the upper 8-bit of the color value.
<table>
<tr><th></th><th>31:24</th><th>23:16</th><th>15:8</th><th>7:0</th></tr>
<tr><td>vg_lite_color_t</td><td>A</td><td>B</td><td>G</td><td>R</td></tr>
</table>
For L8 target formats, the RGB color is converted to L8 by using the default ITU-R BT.709 conversion rules.</td></tr>
<tr><td>VG_I</td><td>enum vg_lit</td><td>A signed 8-bit integer coordinate</td></tr>
<tr><td>VG_I</td><td>enum vg_lit</td><td>A signed 16 bit integer coordinate</td></tr>
<tr><td>VG_I</td><td>enum vg_lit</td><td>A signed 32-bit integer coordinate</td></tr>
<tr><td>VG_I</td><td>enum vg_lit</td><td>A 32-bit floating point coordinate</td></tr>
</table>

Parent topic:*Common parameters and error values*

Enumerations for error reporting This section describes enumerations used for error reporting.

vg_lite_error_t **enumeration** Most functions in the API include an error status via the vg_lite_error_tenumeration. API functions return the status of the command and will report VG_LITE_SUCCESS if successful with no errors. Possible error values include the values in the table below. vg_lite_error_tenumeration is used in many functions, including initialization, flush, blit, draw, gradient, and pattern functions.

vg_lite_error_t string values	Description
VG_LITE_GENERIC_IO	Cannot communicate with the kernel driver
VG_LITE_INVALID_ARGUMEN	An invalid argument was specified
VG_LITE_MULTI_THREAD_FA	Multi-thread/tasks fail *(available from June 2020)*
VG_LITE_NO_CONTEXT	No context specified
VG_LITE_NOT_SUPPORT	Function call is not supported. Hardware support is not available.
VG_LITE_OUT_OF_MEMORY	Out of memory (driver heap)
VG_LITE_OUT_OF_RESOURC	Out of resources (OS heap)
VG_LITE_SUCCESS	Successful with no errors
VG_LITE_TIMEOUT	Timeout
VG_LITE_ALREADY_EXISTS	Object exists *(available from August 2021)*
VG_LITE_NOT_ALIGNED	Data alignment error *(available from August 2021)*

Parent topic: Enumerations for error reporting

Parent topic:*Common parameters and error values*

Hardware product and feature information These query functions can be used to identify the product and its key features and to get VGLite driver information.

Enumerations for product and feature queries This section describes enumerations used for product and feature queries.

vg_lite_feature_t **enumeration** The following feature values may be queried for availability in compatible hardware. *(expanded March 2023 to support additional hardware for driver V4)*

Used in information function: vg_lite_query_feature.

vg_lite_feature_t string values	Description
gcFEATURE_BIT_VG_16PIXELS_ALIGN	Require 16 pixels aligned for the input pixel buffer
gcFEATURE_BIT_VG_24BIT	RGB888 or RGBA5658 formats support
gcFEATURE_BIT_VG_24BIT_PLANAR	24-bit planar format support
gcFEATURE_BIT_VG_AYUV_INPUT	AYUV input format support
gcFEATURE_BIT_VG_BORDER_CULLING	Border culling support
gcFEATURE_BIT_VG_COLOR_KEY	Color key support.
gcFEATURE_BIT_VG_COLOR_TRANSFORMATION	Color transform support.
gcFEATURE_BIT_VG_DEC_COMPRESS	DEC compression format output support
gcFEATURE_BIT_VG_DITHER	Dither support
gcFEATURE_BIT_VG_DOUBLE_IMAGE	Support two image source inputs
gcFEATURE_BIT_VG_FLEXA	FLEXA interface support
gcFEATURE_BIT_VG_GAMMA	Gamma support
gcFEATURE_BIT_VG_GAUSSIAN_BLUR	Gaussian blur sampling support
gcFEATURE_BIT_VG_GLOBAL_ALPHA	Global alpha support
gcFEATURE_BIT_VG_HW_PREMULTIPLY	HW supports alpha premultiply for image
gcFEATURE_BIT_VG_IM_DEC_INPUT	DEC compressed format input support
gcFEATURE_BIT_VG_IM_FASTCLEAR	Fast Clear support
gcFEATURE_BIT_VG_IM_INDEX_FORMAT	Index format support for image
gcFEATURE_BIT_VG_IM_INPUT	Blit and draw API support
gcFEATURE_BIT_VG_IM_REPEAT_REFLECT	Image repeat reflect mode support
gcFEATURE_BIT_VG_INDEX_ENDIAN	Index format endian support
gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT	Support for extended linear gradient capabilities

continues on next page

Table 1 – continued from previous page

vg_lite_feature_t string values	Description
gcFEATURE_BIT_VG_LVGL_SUPPORT	LVGL blend mode support
gcFEATURE_BIT_VG_MASK	Mask support
gcFEATURE_BIT_VG_MIRROR	Mirror support
gcFEATURE_BIT_VG_NEW_BLEND_MODE	New blend mode DARKEN/LIGHTEN support
gcFEATURE_BIT_VG_NEW_IMAGE_INDEX	New CLUT image index support
gcFEATURE_BIT_VG_PARALLEL_PATHS	New parallel path HW support
gcFEATURE_BIT_VG_PE_CLEAR	Pixel engine clear support
gcFEATURE_BIT_VG_PIXEL_MATRIX	Pixel matrix support
gcFEATURE_BIT_VG_QUALITY_8X	8x anti-aliasing path support
gcFEATURE_BIT_VG_RADIAL_GRADIENT	Radial gradient support
gcFEATURE_BIT_VG_RECTANGLE_TILED_OUT	Rectangle tiled output support
gcFEATURE_BIT_VG_RGBA2_FORMAT	RGBA2222 format support
gcFEATURE_BIT_VG_RGBA8_ETC2_EAC	ETC2/EAC compressed image format support
gcFEATURE_BIT_VG_SCISSOR	Scissor support
gcFEATURE_BIT_VG_SRC_PREMULTIPLIED	Source image alpha premultiplied
gcFEATURE_BIT_VG_STENCIL	Stencil image mode support
gcFEATURE_BIT_VG_STRIPE_MODE	Stripe mode support
gcFEATURE_BIT_VG_TESSELLATION_TILED_OUT	Tessellation tiled output support
gcFEATURE_BIT_VG_USE_DST	Read destination pixel support
gcFEATURE_BIT_VG_YUV_INPUT	YUV input format support
gcFEATURE_BIT_VG_YUV_OUTPUT	YUV format output support
gcFEATURE_BIT_VG_YUV_TILED_INPUT	YUV tiled input format support
gcFEATURE_BIT_VG_YUY2_INPUT	YUY2 input format support

Parent topic: Enumerations for product and feature queries

Parent topic:*Hardware product and feature information*

Structures for product and feature queries This section describes structures used for product and feature queries.

`vg_lite_info_t` **structure** This structure is used to query VGLite driver information.

Used in function: `vg_lite_get_info_t`.

vg_lite_info_t member	Type	Description
api_version	vg_lite_uint32_t	VGLite API version
header_version	vg_lite_uint32_t	VGLite header version
release_version	vg_lite_uint32_t	VGLite driver release version
reserved	vg_lite_uint32_t	Reserved for future use

Parent topic: Structures for product and feature queries

Parent topic:*Hardware product and feature information*

Functions for product and feature queries This section describes functions used for product and feature queries.

`vg_lite_get_product_info` **Description:**

This function is used to identify the VGLite-compatible product.

Syntax:

```
uint32_t vg_lite_get_product_info (
      char            *name,
      uint32_t         *chip_id,
      uint32_t         *chip_rev
);
```

Parameters:

Name	Description
name	A character array to store the name of the chip.
chip_id	Stores an ID number for the chip.
chip_rev	Stores a revision number for the chip.

Parent topic: Functions for product and feature queries

vg_lite_get_info **Description:**

This function is used to query the VGLite driver information.

Syntax:

```
vg_lite_error_t vg_lite_get_info (
    vg_lite_info_t              *info
);
```

Parameters:

Name	Description
info	Points to the VGLite driver information structure, which includes the API version, header version, and release version

Parent topic: Functions for product and feature queries

vg_lite_get_register **Description:**

This function can be used to read a GPU AHB register value given the AHB byte address of a register. Refer to the appropriate Vivante GPU AHB register specification documents for register descriptions. The value range of AHB accessible addresses for VGLite cores is usually 0x0 to 0x1FF and 0xA00 to 0xA7F.

Syntax:

```
vg_lite_error_t vg_lite_get_register (
      vg_lite_uint32_t       address,
      vg_lite_uint32_t       *result
);
```

Parameters:

Name	Description
address	Byte Address of the register which value you want.
*result	The registers value.

Parent topic: Functions for product and feature queries vg_lite_query_feature **Description:**

This function is used to query if a specific feature is available.

Syntax:

```
vg_lite_uint32_t vg_lite_query_feature (
    vg_lite_feature_t          feature
);
```

Parameters:

Name	Description
feature	Feature to be queried, as detailed in enum vg_lite_feature_t

Returns:

The feature is either not supported (0) or supported (1).

Parent topic: Functions for product and feature queries

vg_lite_get_mem_size **Description:**

This function queries whether there is any remaining allocated contiguous video memory. *(available from June 2020)*

Syntax:

```
vg_lite_error_t vg_lite_get_mem_size(
    vg_lite_uint32_t            *size
);
```

Parameters:

Name	Description
size	Pointer to the remaining allocated contiguous video memory.

Returns:

Returns VG_LITE_SUCCESS if the query is successful and memory is available. Returns VG_LITE_NO_CONTEXT if the driver is not initialized or there is no available memory.

Parent topic: Functions for product and feature queries

Parent topic:*Hardware product and feature information*

API control Before calling any VGLite API function, the application must initialize the VGLite implicit (global) context by calling *vg_lite_init()*, which will fill in a features table, reset the fast-clear buffer, reset the compositing target buffer and allocate the command and tessellation buffers.

The VGLite driver only supports one current context and one thread to issue commands to the Vivante Vector Graphics hardware. The VGLite driver does not support multiple concurrent contexts running simultaneously in multiple threads/processes, as the VGLite kernel driver does not support context switching. A VGLite application can only use a single context at any time to issue commands to the Vivante Vector Graphics hardware. If a VGLite application must switch contexts, vg_lite_close() must be called to close the current context in the current thread, then vg_lite_init() can be called to initialize a new context either in the current thread or from another thread/process.

Context initialization and control functions This section provides an overview of the context initialization and control functions.

vg_lite_init **Description:**

This function initializes the memory and data structures needed for VGLite draw/blit functions, by allocating memory for the command buffer and a tessellation buffer of the specified size.

GC555 has a newly designed hardware tessellation module that requires less memory for the tessellation buffer than GC355 and GNanoLite-V. Specifically, the GC555 required tessellation buffer size is "buffer_height * 128 byte". vg_lite_init API can simply be called with the render buffer "width" and "height" as the input parameters for GC555. This results in the best path to tessellation performance.

GC355 and GCNanoLiteV hardware tessellation module requires a tessellation buffer with size "buffer_height * buffer_width * 8 byte". If system memory is limited, the application can define a smaller tessellation window based on the amount of memory available. GPU hardware can process the entire render buffer path tessellation in multiple passes with the tessellation window sliding across the render buffer. The multi-pass path tessellation with the smaller tessellation window has a certain performance overhead.

The minimum tessellation window that can be used is 16x16. If tess_height or tess_width is less than 0 in vg_lite_init API, then no path tessellation buffer is created and path drawing APIs do not work, only blit APIs can be used after vg_lite_init.

If this would be the first context that accesses the hardware, the hardware is turned on and initialized. If a new context must be initialized, vg_lite_close must be called to close the current context. Otherwise, vg_lite_init will return an error.

Syntax:

```
vg_lite_error_t vg_lite_init (
    vg_lite_int32_t            tess_width,
    vg_lite_int32_t            tess_height

);
```

Parameters:

Name	Description
tess_v	Width of tessellation window. Maximum cannot be greater than render buffer width. If less than or equal to 0, then no tessellation buffer is created, in which case only blit APIs can be used afterward.
tess_h	Height of tessellation window. Maximum cannot be greater than render buffer height. If less than or equal to 0, then no tessellation buffer is created, in which case blit APIs can be used afterward.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions vg_lite_close **Description:**

This function deallocates all the resources and free up all the memory that was initialized earlier by the vg_lite_init function. It will also turn OFF the hardware automatically if this was the only active context.

Syntax:

```
vg_lite_error_t vg_lite_close (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_flush **Description:**

This function explicitly submits the command buffer to the GPU without waiting for it to complete. *(From Dec 2019, return type is vg_lite_error_t, previously was void.)*

Syntax:

```
vg_lite_error_t vg_lite_flush (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the flush is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_finish **Description:**

This function explicitly submits the command buffer to the GPU and waits for it to complete.

Syntax:

```
vg_lite_error_t vg_lite_finish (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_frame_delimiter **Description:**

This function sets a flag for GPU to signal the completion of current frame. A vg_lite_finish is called by default within this API. The enum VG_LITE_FRAME_END_FLAG is the only value that can be set by flag parameter.

Syntax:

```
vg_lite_error_t vg_lite_frame_delimiter (
    vg_lite_frame_flag_t flag
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_command_buffer_size **Description:**

This function is optional. If used, call it before vg_lite_init if you want to change the command buffer size. *(available from March 2020)*

Syntax:

```
vg_lite_error_t vg_lite_set_command_buffer_size (
    vg_lite_uint32_t            size
);
```

Parameters:

Name	Description
size	Size of the VGLite Command buffer. Default is 64K.

Returns:

Returns VG_LITE_SUCCESS if the flush is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_command_buffer **Description:**

This function sets a user-defined external memory buffer (physical, 64-byte aligned) as the VGLite command buffer. By default, the VGLite driver allocates a static command buffer internally. Thus, it is not necessary for an application to allocate and set the command buffer. This function is only used for devices where an application needs to allocate the command buffer dynamically. *(from December 2021)*

Syntax:

```
vg_lite_error_t vg_lite_set_command_buffer (
    vg_lite_uint32_t            physical,
    vg_lite_uint32_t            size
);
```

Parameters:

Name	Description
physical	The physical address of a memory buffer. The address must be 64-byte aligned.
size	The size of memory buffer. The size must be 128-byte aligned.

Returns:

Returns VG_LITE_SUCCESS if the command buffer set is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_tess_buffer **Description:**

This function specifies a memory buffer from an application as the VGLite driver's tessellation buffer. By default, the VGLite driver allocates a static tessellation buffer internally. Thus, it is not necessary for an application to allocate and set the tessellation buffer. This function is only used for devices where the application needs to allocate the tessellation buffer dynamically. *(from December 2021)*

Syntax:

```
vg_lite_error_t vg_lite_set_tess_buffer (
    vg_lite_uint32_t            physical,
    vg_lite_uint32_t            size
);
```

Parameters:

Name	Description
physical	The physical address of a tessellation buffer. The address must be 64-byte aligned.
size	The size of tessellation buffer. tessellation buffer size = target buffer's height * 128B.

Returns:

Returns VG_LITE_SUCCESS if the tessellation buffer set is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

vg_lite_set_memory_pool **Description:**

This function sets the specific memory pool from which certain type of buffers, VG_LITE_COMMAND_BUFFER, VG_LITE_TESSELLATION_BUFFER, or VG_LITE_RENDER_BUFFER, should be allocated. By default, all types of buffers are allocated from VG_LITE_MEMORY_POOL_1. This API must be called before vg_lite_init() for setting VG_LITE_COMMAND_BUFFER or VG_LITE_TESSELLATION_BUFFER memory pools. This API can be called anytime for VG_LITE_RENDER_BUFFER to affect the following vg_lite_allocate() calls.*(from December 2023)*

Syntax:

```
vg_lite_error_t vg_lite_set_memory_pool (
    vg_lite_buffer_type_t         type,
    vg_lite_memory_pool_t         pool
);
```

Parameters:

Nam	Description
type	The buffer type (VG_LITE_COMMAND_BUFFER, VG_LITE_TESSELLATION_BUFFER, or VG_LITE_RENDER_BUFFER) to be allocated from memory pool.
pool	The memory pool (VG_LITE_MEMORY_POOL_1, VG_LITE_MEMORY_POOL_2) from which the buffer type should be allocated.

Returns:

Returns VG_LITE_SUCCESS if the memory pool set is successful. See vg_lite_error_t enumeration for other return codes.

Parent topic: Context initialization and control functions

Parent topic:*API control*

Pixel buffers This chapter provides an overview of the pixel buffer alignment, cache, internal representation, enumerations, structures, and functions.

Pixel buffer alignment The VGLite hardware requires the pixel buffer start address and stride to be properly byte-aligned to work correctly. The start address and stride alignment requirement for a pixel buffer depends on the specific pixel format, and gcFEATURE_VG_16PIXELS_ALIGNED value (0/1) in vg_lite_options.h file.

Parent topic:*Pixel buffers*

Pixel cache The Vivante Imaging Engine (IM) includes two fully associative caches. Each cache has 8 lines. Each line has 64 bytes. In this case, one cache line can hold either a 4x4-pixel tile or a 16x1-pixel row.

Parent topic:*Pixel buffers*

Internal representation For non-32-bit color formats, each pixel is extended to 32 bits as follows:

If the source and destination formats have the same color format, but differ in the number of bits per color channel, the source channel is multiplied by (2d- 1)/(2s– 1) and is rounded to the nearest integer, where:

- **d** is the number of bits in the destination channel
- **s** is the number of bits in the source channel

Example: a b11111 5-bit source channel gets converted to an 8-bit destination b11111000.

The YUV formats are internally converted to RGB. The pixel selection is unified for all formats by using the LSB of the coordinate.

Parent topic:*Pixel buffers*

Pixel buffer enumerations This section provides an overview of the pixel buffer enumerations.

vg_lite_buffer_format_t enumeration This enumeration specifies the color format to use for a buffer. This applies to both image and Render Target. Formats include supported swizzles for RGB. For YUV swizzles, use the related values and parameters in vg_lite_swizzle_t.

The application shall use the vg_lite_query_feature API to determine support for some hardware-specific formats. For example, related vg_lite_feature_t enum values include gcFEATURE_BIT_VG_RGBA2_FORMAT and gcFEATURE_BIT_VG_IM_INDEX_FORMAT.

(Alignment columns refined March and Sept 2023)

Used in structure: vg_lite_buffer_t.

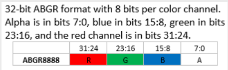
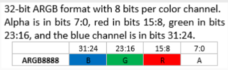
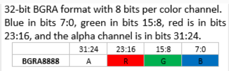
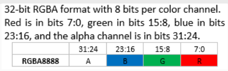
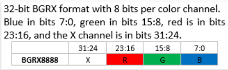
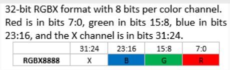
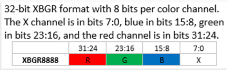
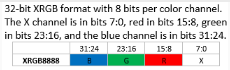
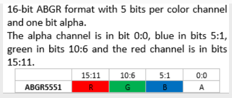
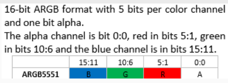
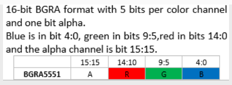
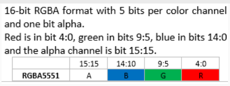
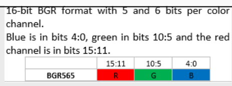
See also vg_lite_blit, vg_lite_clear, vg_lite_draw.

Attention: OpenVG VGImageFormat Note: The bits for each color channel are stored within a machine word from MSB to LSB in the order indicated by the pixel format name. This is the opposite of the original VG_LITE_* formats that are ordered from LSB to MSB. The formats with the same organization are listed in the same row as their VG_Lite counterparts.

Attention: Original VGLite API Image Format Note: The bits for each color channel are stored within a machine word from LSB to MSB in the order indicated by the pixel format name. This is the opposite of the OPENVG VG_* formats that are ordered from MSB to LSB.

The following codes, as also used in OpenVG 1.1 Specification Table 11, are used for format description:

- A - Alpha channel
- B - Blue color channel
- G - Green color channel
- R - Red color channel
- X - Uninterpreted padding byte or bit
- L - Grayscale
- BW - 1-bit black and white
- l - Linear color space
- s - Non-linear (sRGB) color space
- PRE - Alpha values are premultiplied

vg_lite_buffer_format_t String Value	Description	Supported as source	Supported as destination	Start address/ Stride alignment: bytes
VG_LITE_ABGR8888 VG_sRGBA_8888 VG_sRGBA_8888_PRE VG_lRGBA_8888 VG_lRGBA_8888_PRE	 32-bit ABGR format with 8 bits per color channel. Alpha is in bits 7:0, blue in bits 15:8, green in bits 23:16, and the red channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_ARGB8888 VG_sBGRA_8888 VG_sBGRA_8888_PRE VG_lBGRA_8888 VG_lBGRA_8888_PRE	 32-bit ARGB format with 8 bits per color channel. Alpha is in bits 7:0, red in bits 15:8, green in bits 23:16, and the blue channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_BGRA8888 VG_sARGB_8888 VG_sARGB_8888_PRE VG_lARGB_8888 VG_lARGB_8888_PRE	 32-bit BGRA format with 8 bits per color channel. Blue in bits 7:0, green in bits 15:8, red is in bits 23:16, and the alpha channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_RGBA8888 VG_sABGR_8888 VG_sABGR_8888_PRE VG_lABGR_8888 VG_lABGR_8888_PRE	 32-bit RGBA format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16, and the alpha channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_BGRX8888 VG_sXRGB_8888 VG_lXRGB_8888	 32-bit BGRX format with 8 bits per color channel. Blue in bits 7:0, green in bits 15:8, red is in bits 23:16, and the X channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_RGBX8888 VG_sXBGR_8888 VG_lXBGR_8888	 32-bit RGBX format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16, and the X channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_XBGR8888 RGBX VG_sRGBX_8888 VG_lRGBX_8888	 32-bit XBGR format with 8 bits per color channel. The X channel is in bits 7:0, blue in bits 15:8, green in bits 23:16, and the red channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_XRGB8888 VG_sBGRX_8888 VG_lBGRX_8888	 32-bit XRGB format with 8 bits per color channel. The X channel is in bits 7:0, red in bits 15:8, green in bits 23:16, and the blue channel is in bits 31:24.	Yes	Yes	Start 4B / Stride 64B
VG_LITE_ABGR1555 VG_sRGBA_5551	 16-bit ABGR format with 5 bits per color channel and one bit alpha. The alpha channel is in bit 0:0, blue in bits 5:1, green in bits 10:6 and the red channel is in bits 15:11.	Yes	Yes	Start 4B / Stride 32B
VG_LITE_ARGB1555 VG_sBGRA_5551	 16-bit ARGB format with 5 bits per color channel and one bit alpha. The alpha channel is bit 0:0, red in bits 5:1, green in bits 10:6 and the blue channel is in bits 15:11.	Yes	Yes	Start 4B / Stride 32B
VG_LITE_BGRA5551 VG_sARGB_1555	 16-bit BGRA format with 5 bits per color channel and one bit alpha. Blue is in bit 4:0, green in bits 9:5,red in bits 14:0 and the alpha channel is bit 15:15.	Yes	Yes	Start 4B / Stride 32B
VG_LITE_RGBA5551 VG_sABGR_1555	 16-bit RGBA format with 5 bits per color channel and one bit alpha. Red is in bit 4:0, green in bits 9:5, blue in bits 14:0 and the alpha channel is bit 15:15.	Yes	Yes	Start 4B / Stride 32B
VG_LITE_BGR565 VG_sRGB_565	 16-bit BGR format with 5 and 6 bits per color channel. Blue is in bits 4:0, green in bits 10:5 and the red channel is in bits 15:11.	Yes	Yes	Start 4B / Stride 32B
	16-bit RGB format with 5 or 6 bits per color channel.			

Hardware-dependent formats for vg_lite_buffer_format_t	Description	Supported as source	Supported as destination	Alignment (bytes)
VG_LITE_ABGR2222	8-bit BGRA format with 2 bits per color channel. Alpha is in bits 1:0, blue in bits 3:2, green in bits 5:4 and the red channel is in bits 7:6. ABGR2222: 7:6 R, 5:4 G, 3:2 B, 1:0 A	Yes	Yes	Start 4B / Stride 16B
VG_LITE_ARGB2222	8-bit BGRA format with 2 bits per color channel. Alpha is in bits 1:0, red in bits 3:2,green in bits 5:4 and the blue channel is in bits 7:6. ARGB2222: 7:6 B, 5:4 G, 3:2 R, 1:0 A	Yes	Yes	Start 4B / Stride 16B
VG_LITE_BGRA2222	8-bit BGRA format with 2 bits per color channel. Blue is in bits 1:0, green in bits 3:2, red in bits 5:4 and the alpha channel is in bits 7:6. BGRA2222: 7:6 A, 5:4 R, 3:2 G, 1:0 B	Yes	Yes	Start 4B / Stride 16B
VG_LITE_RGBA2222	8-bit RGBA format with 2 bits per color channel. Red is in bits 1:0, green in bits 3:2, blue in bits 5:4 and the alpha channel is in bits 7:6. RGBA2222: 7:6 A, 5:4 B, 3:2 G, 1:0 R	Yes	No	8B
VG_LITE_INDEX_1	1-bit index format	Yes	No	8B
VG_LITE_INDEX_2	2-bit index format	Yes	No	both 8B
VG_LITE_INDEX_4	4-bit index format	Yes	No	both 8B
VG_LITE_INDEX_8	8-bit index format	Yes	No	both 16B
VG_LITE_NV12_TILED	Supertiled (8x8 pixels), planar YUV format, 96-bit for 4 pixels. Y plane is 32 bits for 4 pixels and is organized in 64 pixel super tiles (8x8 Y); UV plane is 64 bits for 4 pixels. Pixels are organized in super tiles are (4x4 UV pairs), available for Source IMAGE only on the supporting hardware. Y Buffer / Y Buffer: 31:24 Y3, 23:16 Y2, 15:8 Y1, 7:0 Y0, … UV Buffer: 31:24 V1 V3 …, 23:16 U1 U3, 15:8 V0 V2, 7:0 U0 U2	Yes	No	Y: both 16 Bytes UV: both 8 Bytes
VG_LITE_ANV12_TILED	Pixel organization as NV12_TILED but with an Alpha Buffer is also supertiled, available for Source IMAGE only on the supporting hardware. Alpha Buffer: 31:24 A3, 23:16 A2, 15:8 A1, 7:0 A0, … Y Buffer / Y Buffer: 31:24 Y3, 23:16 Y2, 15:8 Y1, 7:0 Y0, … UV Buffer: 31:24 V1 V3 …, 23:16 U1 U3, 15:8 V0 V2, 7:0 U0 U2	Yes	No	A, Y: both 16 Bytes UV: both 8 Bytes
VG_LITE_AYUY2_TILED	Supertiled (8x8) and packed YUV format a with separate tiled Alpha Buffer. Y0 is in bits 7:0 and V is in bits 31:23, available for Source IMAGE only on the supporting hardware. Alpha Buffer: 31:24 A3, 23:16 A2, 15:8 A1, 7:0 A0, … YUY2 Buffer: 31:24 V0 V2 V4, 23:16 Y1 Y3 Y5, 15:8 U0 U2 U4, 7:0 Y0 Y2 Y4	Yes	No	both 32B
VG_LITE_RGB888	24-bit RGB format with 8 bits per color channel. Red is in bits 7:0, green in bits 15:8, blue in bits 23:16. RGB888: 23:16 B, 15:8 G, 7:0 R	Yes	Yes	Start 4B / Stride 32B
VG_LITE_BGR888	24-bit RGB format with 8 bits per color channel. Blue is in bits 7:0, green in bits 15:8, red in bits 23:16. BGR888: 23:16 R, 15:8 G, 7:0 B		Yes	Yes
VG_LITE_ARGB8565	24-bit RGBA format with 4 and 5 bits per color channel. The alpha channel is in bit 7:0, red in bits 12:8, green in bits 18:13 and the blue in bits 23:19. ARGB8565: 23:19 B, 18:13 G, 12:8 R, 7:0 A		Yes	Yes
VG_LITE_BGRA5658	24-bit RGBA format with 4 and 5 bits per color channel. Blue is in bits 4:0, green in bits 10:5, red in bits 15:11,alpha channel is in bit 23:16. BGRA5658: 23:16 A, 15:11 R, 10:5 G, 4:0 B	Yes	Yes	Start 4B / Stride 32B

Parent topic: Pixel buffer enumerations

Image buffer alignment requirement The image (or source) buffer alignment requirement depends on the specific pixel format, and some gcFEATURE_*_ALIGNED defines in the vg_lite_options.h file.

Image format	Bits per pixel	Source tile mode	Start address alignment requirement in bytes	Stri
VG_LITE_INDEX1	1	linear	8B	2B
VG_LITE_INDEX1	1	tile	8B	1B
VG_LITE_INDEX2	2	linear	8B	4B
VG_LITE_INDEX2	2	tile	8B	1B
VG_LITE_INDEX4	4	linear	8B	8B
VG_LITE_INDEX4	4	tile	8B	2B
VG_LITE_INDEX8	8	linear	16B	16B
VG_LITE_INDEX8	8	tile	16B	4B
VG_LITE_A4	4	linear	8B	8B
VG_LITE_A4	4	tile	8B	2B
VG_LITE_A8	8	linear	16B	16B
VG_LITE_A8	8	tile	16B	4B
VG_LITE_L8	8	linear	16B	16B
VG_LITE_L8	8	tile	16B	4B
VG_LITE_ARGB2222	8	linear	16B	16B
VG_LITE_ARGB2222	8	tile	16B	4B
VG_LITE_RGB565	16	linear	32B	32B
VG_LITE_RGB565	16	tile	32B	8B
VG_LITE_ARGB1555	16	linear	32B	32B
VG_LITE_ARGB1555	16	tile	32B	8B
VG_LITE_ARGB4444	16	linear	32B	32B
VG_LITE_ARGB4444	16	tile	32B	8B
VG_LITE_ARGB8888	32	linear	64B	64B
VG_LITE_ARGB8888	32	tile	64B	16B
VG_LITE_XRGB8888	32	linear	64B	64B
VG_LITE_XRGB8888	32	tile	64B	16B
VG_LITE_ARGB8565	24	linear	64B	48B
VG_LITE_ARGB8565	24	tile	64B	12B
VG_LITE_RGB888	24	linear	64B	48B
VG_LITE_RGB888	24	tile	64B	12B
VG_LITE_YUY2/UYVY	16	linear	32B	32B
VG_LITE_YUY2/UYVY	16	tile	32B	8B
VG_LITE_NV12	12	linear	Y: 32B UV: 32B	Y: 3
VG_LITE_YV12	12	linear	Y: 32B U: 16B V: 16B	Y: 3
VG_LITE_NV16	16	linear	Y: 32B UV: 32B	Y: 3
VG_LITE_YV16	16	linear	Y: 32B U: 16B V: 16B	Y: 3
VG_LITE_YV24	24	linear	Y: 32B U: 32B V: 32B	Y: 3
VG_LITE_ETC2	8	tile	16B	4B

Note:

1. The values in the table reflect the alignment requirements of the data in memory. The stride of ARGB8888 / ARGB8565 is seen as 4Byte per pixel when configuring the hardware.
2. For tile mode, the stride is still the byte size of a row of pixels in the buffer instead of 4 rows.
3. When DECNano function is enabled for the buffer, the total buffer size need align to 64Byte*compression rate for ARGB8888 or XRGB8888 format, align to 48Byte*compress rate for RGB888 format.

Additional Alignment Requirement

1. Buffer starting address must be 16 pixel-byte-size aligned, that is 8 bit-per-pixel format buffer must be 16 bytes aligned; 16 bit-per-pixel format buffer must be 32 bytes aligned; 24 and 32 bit-per-pixel format buffer must be 64 bytes aligned.
2. For linear mode buffer, the buffer stride must be 16 pixel-byte-size aligned.
3. For tile mode buffer, buffer width and height must be 4 pixel aligned so buffer width and height end at tile boundary.

Parent topic: Pixel buffer enumerations

Destination buffer alignment requirement The destination (or render target) buffer alignment requirement depends on the specific pixel format, and some gcFEATURE_*_ALIGNED defines in the vg_lite_options.h file.

Target format	Bits per pixel	Target tile mode	VG tile mode	Start address alignment requirement i
VG_LITE_A8	8	linear	linear	4B
VG_LITE_A8	8	linear	tile	64B
VG_LITE_A8	8	tile	linear	64B
VG_LITE_A8	8	tile	tile	64B
VG_LITE_L8	8	linear	linear	4B
VG_LITE_L8	8	linear	tile	64B
VG_LITE_L8	8	tile	linear	64B
VG_LITE_L8	8	tile	tile	64B
VG_LITE_ARGB2222	8	linear	linear	4B
VG_LITE_ARGB2222	8	linear	tile	64B
VG_LITE_ARGB2222	8	tile	linear	64B
VG_LITE_ARGB2222	8	tile	tile	64B
VG_LITE_RGB565	16	linear	linear	4B
VG_LITE_RGB565	16	linear	tile	64B
VG_LITE_RGB565	16	tile	linear	64B
VG_LITE_RGB565	16	tile	tile	64B
VG_LITE_ARGB1555	16	linear	linear	4B
VG_LITE_ARGB1555	16	linear	tile	64B
VG_LITE_ARGB1555	16	tile	linear	64B
VG_LITE_ARGB1555	16	tile	tile	64B
VG_LITE_ARGB4444	16	linear	linear	4B
VG_LITE_ARGB4444	16	linear	tile	64B
VG_LITE_ARGB4444	16	tile	linear	64B
VG_LITE_ARGB4444	16	tile	tile	64B
VG_LITE_ARGB8888	32	linear	linear	4B
VG_LITE_ARGB8888	32	linear	tile	64B
VG_LITE_ARGB8888	32	tile	linear	64B
VG_LITE_ARGB8888	32	tile	tile	64B
VG_LITE_XRGB8888	32	linear	linear	4B
VG_LITE_XRGB8888	32	linear	tile	64B
VG_LITE_XRGB8888	32	tile	linear	64B
VG_LITE_XRGB8888	32	tile	tile	64B
VG_LITE_ARGB8565	24	linear	linear	64B
VG_LITE_ARGB8565	24	linear	tile	64B
VG_LITE_ARGB8565	24	tile	linear	64B
VG_LITE_ARGB8565	24	tile	tile	64B
VG_LITE_RGB888	24	linear	linear	64B
VG_LITE_RGB888	24	linear	tile	64B
VG_LITE_RGB888	24	tile	linear	64B
VG_LITE_RGB888	24	tile	tile	64B

Note:

1. The values in the table reflect the alignment requirements of pixel data in memory. The stride of ARGB8888/ARGB8565 is seen as 4 Bytes per pixel when configuring the hardware.
2. For tile mode, the buffer stride is still the byte size of a row of pixels instead of 4 rows of pixels.
3. For PE clear function, the clear size must align to 48 Bytes for the RGB888 or ARGB8565 format.
4. For PE clear function with DECNano enabled, the clear size must align to 48 Bytes for RGB888, align to 64 Bytes for ARGB8888 or XRGB8888.
5. If the DECNano function is enabled for the buffer, the target buffer start address needs to align to 64 Bytes.
6. If the DECNano function is enabled for the buffer, the total buffer size needs to align to a 64-byte compression rate for ARGB8888 or XRGB8888 format and align to a 48 Byte*compression rate for RGB888 format.

Additional Alignment Requirement

1. Buffer starting address must be at least 4-byte aligned. Buffer stride must be at least one pixel size aligned.
2. Buffer starting address must be 64-byte aligned for 24 bit-per-pixel format, or tile mode, or DECNano enabled.
3. Buffer height must be 4-pixel aligned for tile mode buffer.
4. For tile mode buffer, the buffer stride must be 16-byte aligned for non-24bit-per-pixel formats. So, 8 bits-per-pixel format buffer width must be 16-pixel aligned; 16 bits-per-pixel format buffer width must be 8-pixel aligned; 32 bit-per-pixel format buffer width must be 4 pixel aligned.
5. For tile mode buffer, the buffer stride must be 12-byte aligned for 24 bits-per-pixel formats, that is, the buffer width must be 4-pixel aligned.
6. For PE clear function, the clear size must align to 48 Bytes for 24-bits-per-pixel formats.
7. For PE clear function with DECNano enabled, the clear size must align to 48 Bytes for 24 bits-per-pixel formats and align to 64 Bytes for 32 bits-per-pixel formats.
8. If source buffer tile mode is different from destination buffer tile mode, buffer starting address must be 64 Byte aligned, buffer stride must be 64 Byte aligned for non-24 bits-per-pixel formats, buffer stride must be 48-Byte aligned for 24 bits-per-pixel formats.

VGLite hardware requires the raster image width to be a multiple of 16 pixels for linear gradient and radial gradient operations. This requirement applies to all image formats. Therefore, the user must pad an arbitrary image width to a multiple of 16 pixels for VGLite linear gradient and radial gradient APIs.

Parent topic: Pixel buffer enumerations

vg_lite_buffer_layout_t **enumeration** Specifies the buffer data layout in memory.

Used in structure: vg_lite_buffer.

vg_lite_buffer_layout_t String Value	Description
VG_LITE_LINEAR	Linear (scanline) layout.
VG_LITE_TILED	Data is organized in 4x4 pixel tiles. **Note:** for this layout, the buffer start address and stride must be 64-byte aligned

Parent topic: Pixel buffer enumerations

vg_lite_compress_mode_t **enumeration** Specifies the DECNano comprssion mode. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_compress_mode_t string value	Description
VG_LITE_DEC_DISABLE	Disable compression.
VG_LITE_DEC_NON_SAMPLE	compression ratio is 1.6 for ARGB8888, 2.0 for XRGB8888
VG_LITE_DEC_HSAMPLE	compression ratio is 2.0 for ARGB8888, 2.6 for XRGB8888
VG_LITE_DEC_HV_SAMPLE	compression ratio is 2.6 for ARGB8888, 4.0 for XRGB8888

Parent topic: Pixel buffer enumerations

vg_lite_gamma_conversion_t **enumeration** Specifies the gamma conversion mode *(from Sept 2022)*

Used in function: vg_lite_set_gamma.

vg_lite_gamma_conversion_t string value	Description
VG_LITE_GAMMA_NO_CONVERSION	Leave the color as it is.
VG_LITE_GAMMA_LINEAR	Convert from sRGB to linear.
VG_LITE_GAMMA_NON_LINEAR	Convert from linear to sRGB space.

Parent topic: Pixel buffer enumerations

vg_lite_index_endian_t **enumeration** Specifies the endian order parsing mode for index formats *(from March 2023)*.

Used in structure: vg_lite_buffer_t.

vg_lite_index_endi string value	Description
VG_LITE_INDEX	Parse the index pixel from low to high, when using index1, the parsing order is bit0~bit7. when using index2, the parsing order is bit0:1,bit2:3,bit4:5.bit6:7. when using index4, the parsing order is bit0:3,bit4:7.
VG_LITE_INDEX	Parse the index pixel from low to high, when using index1, the parsing order is bit7~bit0. when using index2, the parsing order is bit7:6,bit5:4,bit3:2.bit1:0. when using index4, the parsing order is bit4:7,bit0:3.

Parent topic: Pixel buffer enumerations

vg_lite_image_mode_t **enumeration** Specifies how an image is rendered onto a buffer *(prior to Sept 2022 name was* vg_lite_buffer_image_mode_t*).*

Used in structure: vg_lite_buffer_t.

vg_lite_image_mode_t string value	Description
VG_LITE_ZERO	
VG_LITE_NORMAL_IMAGE_MODE	Image drawn with blending mode
VG_LITE_MULTIPLY_IMAGE_MODE	Image is multiplied with paint color
VG_LITE_STENCIL_MODE	
VG_LITE_NONE_IMAGE_MODE	Image input is ignored.
VG_LITE_RECOLOR_MODE	

Parent topic: Pixel buffer enumerations

vg_lite_map_flag_t **enumeration** Specifies whether mapping is for user memory or the DMA buffer *(from March 2023).*

Used in function: vg_lite_map.

vg_lite_map_flag_t string value	Description
VG_LITE_MAP_USER_MEMORY	Mapping is for user memory.
VG_LITE_MAP_DMABUF	Mapping is for the DMA buffer.

Parent topic: Pixel buffer enumerations

vg_lite_paint_type_t **enumeration** Specifies paint type *(from May 2023).*

Used in structure: vg_lite_buffer_t.

vg_lite_paint_type_t string value	Description
VG_LITE_PAINT_ZERO	
VG_LITE_PAINT_COLOR	Color
VG_LITE_PAINT_LINEAR_GRADIENT	Linear Gradient
VG_LITE_PAINT_RADIAL_GRADIENT	Radial Gradient
VG_LITE_PAINT_PATTERN	Pattern

Parent topic: Pixel buffer enumerations

vg_lite_transparency_t **enumeration** Specifies the transparency mode for a buffer *(prior to Sept 2022 name was vg_lite_buffer_transparency_mode_t).*

Used in structure:vg_lite_buffer.

vg_lite_transparency_t string value	Description
VG_LITE_IMAGE_OPAQUE	Opaque image: all image pixels are copied to the VG PE for rasterization
VG_LITE_IMAGE_TRANSPARENT	Transparent image: only the non-transparent image pixels are copied to the VG PE. **Note:** This mode is only valid when IMAGE_MODE (vg_lite_image_mode_t) is either VG_LITE_NORMAL_IMAGE_MODE or VG_LITE_MULTIPLY_IMAGE_MODE.

Parent topic: Pixel buffer enumerations

vg_lite_swizzle_t enumeration This enumeration specifies the swizzle for the UV components of YUV data.

Used in structure: vg_lite_yuvinfo.

vg_lite_swizzle_t string value	Description
VG_LITE_SWIZZLE_UV	U in lower bits, V in upper bits
VG_LITE_SWIZZLE_VU	V in lower bits, U in upper bits

Parent topic: Pixel buffer enumerations

vg_lite_yuv2rgb_t enumeration This enumeration specifies the standard for conversion of YUV data to RGB data.

Used in structure: vg_lite_yuvinfo.

vg_lite_yuv2rgb_t string value	Description
VG_LITE_YUV601	YUV Converting with ITC.BT-601 standard
VG_LITE_YUV709	YUV Converting with ITC.BT-709 standard

Parent topic: Pixel buffer enumerations

Parent topic:*Pixel buffers*

Pixel buffer structures This section provides an overview on the pixel buffer structures.

vg_lite_buffer_t structure This structure defines the buffer layout for a VGLite image or memory data.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in init functions: vg_lite_allocate, vg_lite_free, vg_lite_upload_buffer, vg_lite_map, vg_lite_unmap.

Used in blit functions:vg_lite_blit, vg_lite_blit_rect, vg_lite_clear, vg_lite_create_masklayer, vg_lite_fill_masklayer, vg_lite_blend_masklayer, vg_lite_set_masklayer, vg_lite_render_masklayer, vg_lite_destroy_masklayer

Used in draw functions: vg_lite_draw, vg_lite_draw_pattern, vg_lite_draw_grad, vg_lite_draw_radial_grad

vg_lite_buffer_t member	Type	Description
width	vg_lite_int32_t	Width of buffer in pixels
height	vg_lite_int32_t	Height of buffer in pixels
stride	vg_lite_int32_t	Stride in bytes
tiled	vg_lite_buffer_layou	Linear or tiled format for buffer enum
format	vg_lite_buffer_form	color format enum
handle	vg_lite_pointer	memory handle
memory	vg_lite_pointer	pointer to the start address of the memory
address	vg_lite_uint32_t	GPU address
yuv	vg_lite_yuvinfo_t	YUV format info struct
image_mode	vg_lite_image_mod	Blit image mode enum
transparency_mode	*vg_lite_transparenc*	Image transparency mode enum
fc_buffer[3]	vg_lite_fc_buffer_t	Three (3) fast clear buffers, reserved YUV format *(from March 2023)*
compress_mode	vg_lite_compress_m	Compression mode *(from March 2023)*
index_endian	vg_lite_index_endia	Big/Little Endian setting for index formats *(from March 2023)*
paintType	*vg_lite_paint_type_t*	Paint type enum *(from May 2023)*
fc_enable	vg_lite_int8_t	Enable Image fast clear *(moved from Aug 2023)*
scissor_layer	vg_lite_int8_t	Get paintcolor from different paint types *(from Aug 2023)*
premulitplied	vg_lite_int8_t	The RGB pixel values are alpha-premultipled *(from Aug 2023)*

Parent topic: Pixel buffer structures

vg_lite_fc_buffer_t **structure** This structure defines the organization of a fast clear buffer. *(from March 2023)*

Used in structure: vg_lite_buffer_t.

vg_lite_fc_buffer_t members	Type	Description
width	vg_lite_int32_t	Width of buffer in pixels
height	vg_lite_int32_t	Height of buffer in pixels
stride	vg_lite_int32_t	Stride in bytes
handle	vg_lite_pointer	memory handle as allocated by the VGLite kernel
memory	vg_lite_pointer	logical pointer to the start address of the memory for the CPU
address	vg_lite_uint32_	address to the buffer's memory for the GPU hardware
color	vg_lite_uint32_	The fast clear color value

Parent topic: Pixel buffer structures

vg_lite_yuvinfo_t **structure** This structure defines the organization of VGLite YUV data.

Used in structure: vg_lite_buffer_t.

vg_lite_yuvinfo_t member	Type	Description
swizzle	*vg_lite_swizzle_t*	UV swizzle enum
yuv2rgb	*vg_lite_yuv2rgb_*	YUV conversion standard enum
‘uv_planar	vg_lite_uint32_t	UV (U) planar address for GPU, generated by driver
v__planar	vg_lite_uint32_t	V planar address for GPU, generated by driver
‘alpha_planar	vg_lite_uint32_t	Alpha planar address for GPU, generated by driver
‘uv_stride	vg_lite_uint32_t	UV (U) stride in bytes
‘v_stride	vg_lite_uint32_t	V planar stride in bytes
alpha__stride	vg_lite_uint32_t	Alpha stride in bytes
‘uv_height	vg_lite_uint32_t	UV (U) height in pixels
‘v_height	vg_lite_uint32_t	V stride in bytes
uv__memory	vg_lite_pointer	Logical pointer to the UV (U) planar memory
‘v_memory	vg_lite_pointer	Logical pointer to the V planar memory
uv__handle	vg_lite_pointer	Memory handle of the UV (U) planar, generated by the driver
v__handle	vg_lite_pointer	Memory handle of the V planar, generated by the driver

Parent topic: Pixel buffer structures

Parent topic:*Pixel buffers*

Pixel buffer functions This section provides an overview of the pixel buffer functions.

vg__lite__allocate **function** **Description:**

This function is used to allocate a buffer before it is used in either blit or draw functions.

To allow the hardware to access some memory, such as a source image or target buffer, you must first allocate the memory. The supplied vg_lite_buffer_t structure must be initialized with the size (width and height) and format of the requested buffer. If the stride is set to zero, then this function fills it in. The only input parameter to this function is the pointer to the buffer structure. If the structure has all the information needed, then appropriate memory is allocated for the buffer.

This function calls the kernel to allocate the memory. The kernel fills in the memory handle, logical address, and hardware addresses in the vg_lite_buffer_t structure.

Alignment note:

Vivante GPUs have an alignment requirement of 64 bytes. However, to meet the alignment requirements of the Vivante display controller, the VGLite driver sets the render target buffer alignment to 128 bytes. For source image buffer alignment requirements, see the alignment notes available in Table 1.

The vg_lite_buffer_format_t value descriptions:

Syntax:

```
vg__lite__error__t vg__lite__allocate (
        vg__lite__buffer__t        *buffer
);
```

Parameters:

Nam	Description
buffe	Pointer to the buffer that holds the size and format of the buffer being allocated. Either the memory or address field must be set to a non-zero value to map either a logical or physical address into hardware accessible memory.

Returns:

- VG_LITE_SUCCESS if the contiguous buffer was allocated successfully.
- VG_LITE_OUT_OF_RESOURCES if there is insufficient memory in the host OS heap for the buffer.
- VG_LITE_OUT_OF_MEMORY if allocation of a contiguous buffer failed.

Parent topic: Pixel buffer functions

vg_lite_free function

Description:

This function is used to deallocate the buffer that was previously allocated. It frees up the memory for that buffer.

Syntax:

```
vg_lite_error_t vg_lite_free (
        vg_lite_buffer_t        *buffer
);
```

Parameters:

Name	Description
buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_allocate() function.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_upload_buffer function

Description:

The function uploads the pixel data to a GPU memory buffer object. The format of the data (pixel) to be uploaded must match the format defined for the buffer object. The input data memory buffer should contain enough data to be uploaded to the GPU buffer pointed by the input parameter buffer.

Note: Vivante Vector Graphics IP only uses data[0] and stride[0] as it does not support planar YUV formats..

Syntax:

```
vg_lite_error_t vg_lite_upload_buffer (
    vg_lite_buffer_t            *buffer,
    vg_lite_uint8_t             *data[3],
    vg_lite_uint32_t            stride[3]
);
```

Parameters:

Name	Description
buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_allocate() function
data[3]	Pointer to pixel data. For the YUV format, there may be up to 3 pointers.
stride[3]	Stride for the pixel data

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_map function

Description:

This function is used to map the memory appropriately for a particular buffer. For some operating systems, it is used to get proper translation to the physical or logical address of the buffer needed by the GPU.

To use a frame buffer directly as a target buffer:

- Wrap a vg_lite_buffer_t structure around the buffer
- Call the kernel to map the supplied logical or physical address into hardware accessible memory

For example, if you know the logical address of the frame buffer, set the memory field of the vg_lite_buffer_t structure with that address and call this function. If you know the physical address, set the memory field to NULL and program the address field with the physical address.

Syntax:

```
vg_lite_error_t vg_lite_map (
   vg_lite_buffer_t          *buffer,
   vg_lite_map_flag_t         flag,
   int32_t                   fd
   );
```

Parameters:

Name	Description
*buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_allocate() function
flag	Enumerate the vg_lite_map_flag_t value that specifies whether the mapping is for user memory or DMA buffer. *(from March 2023)*
fd	File descriptor for dma_buf if the flag is VG_LITE_MAP_DMABUF. Otherwise, this parameter is ignored. *(from March 2023)*

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_unmap function

Description:

This function unmaps the buffer and frees any memory resources allocated by a previous call to the vg_lite_map() function.

Syntax:

```
vg_lite_error_t vg_lite_unmap (
    vg_lite_buffer_t        *buffer
);
```

Parameters:

Name	Description
buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_map() function

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_flush_mapped_buffer function

Description:

This function flushes the CPU cache for the mapped buffer to make sure the buffer contents are written to GPU memory.

Syntax:

```
vg_lite_error_t vg_lite_flush_mapped_buffer (
    vg_lite_buffer_t          *buffer
);
```

Parameters:

Name	Description
*buffer	Pointer to a buffer structure that was filled in by calling the vg_lite_map() function

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_set_CLUT function

Description:

This function sets the Color Lookup Table (CLUT) in the context state for index color image. Once the CLUT is set (Not NULL), the image pixel color for index format image rendering is obtained from the Color Lookup Table (CLUT) according to the pixel's color index value.

Note: Available only for IP with Indexed color support..

Syntax:

```
vg_lite_error_t vg_lite_set_CLUT (
  vg_lite_uint32_t                    count,
  vg_lite_uint32_t                    *colors
);
```

Parameters:

Nam	Description
coun	This is the count of the colors in the color look-up table: - For INDEX_1, there can be up to 2 colors in the table - For INDEX_2, there can be up to 4 colors in the table - For INDEX_4, there can be up to 16 colors in the table - For INDEX_8, there can be up to 256 colors in the table
*col-ors	The Color Lookup Table (CLUT) pointed by "colors" will be stored in the context and programmed to the command buffer when needed. The CLUT will not take effect until the command buffer is submitted to HW. The color is in ARGB format with A located in the upper bits. **Note:** The VGLite driver does not validate the CLUT contents from the application.

Returns:

VG_LITE_SUCCESS as no checking is done.

Parent topic: Pixel buffer functions

vg_lite_enable_dither **function** **Description:**

This function is used to enable the dither function. Dither is turned off by default. The application can use the VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_DITHER) to determine HW support for dither.

Syntax:

```
vg_lite_error_t vg_lite_enable_dither (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_disable_dither **function** **Description:**

This function is used to disable the dither function. Dither is turned off by default.

Syntax:

```
vg_lite_error_t vg_lite_disable_dither (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

vg_lite_set_gamma **function** **Description:**

This function sets a gamma value.

Application can use the VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_GAMMA) to determine HW support for gamma.

Syntax:

```
vg_lite_error_t vg_lite_set_gamma (
    vg_lite_gamma_conversion_t     gamma_value
);
```

Parameters:

Name	Description
gamma_value	Sets a gamma value. See enum vg_lite_gamma_conversion_t.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Pixel buffer functions

Parent topic:*Pixel buffers*

Matrices This part of the API provides matrix controls.

Note: All the transformations in the driver/API are actually the final plane/surface coordinate system. There is no transformation of different coordinate systems with VGLite.

Matrix control float parameter type

Name	Typedef	Value
vg_lite_float_t	float	A single-precision floating-point number
vg_lite_pixel_matrix_t [20]	vg_lite_float_t	Pixel transform matrix m[20], which transforms each pixel as follows: |a'| |m0 m1 m2 m3 m4 | |a| |r'| |m5 m6 m7 m8 m9 | |r| |g'| = |m10 m11 m12 m13 m14|.|g| |b'| |m15 m16 m17 m18 m19| |b| |1 | |0 0 0 0 1 | |1|

Parent topic:*Matrices*

Matrix control structures This section provides an overview of the graphic transformation matrix control structures.

vg_lite_matrix_t structure This structure defines a 3x3 floating point matrix.

Used in structures: vg_lite_linear_gradient_t, vg_lite_radial_gradient_t.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw, vg_lite_draw_gradient, vg_lite_draw_radial_gradient, vg_lite_draw_pattern, vg_lite_identity, vg_lite_scale, vg_lite_translate.

vg_lite_matrix_t member	Type	Description
m[3][3]	vg_lite_float_t	3x3 matrix, in [row] [column] order

Parent topic: Matrix control structures

vg_lite_pixel_channel_enable_t structure This structure provides enable disable flags for hardware pixel channels A,R,G,B.

Used in function: vg_lite_set_pixel_matrix_t.

vg_lite_pixel_channel_enable_t members	Type	Description
enable_a	vg_lite_uint8_t	Enable A channel
enable_b	vg_lite_uint8_t	Enable B channel
enable_g	vg_lite_uint8_t	Enable G channel
enable_r	vg_lite_uint8_t	Enable R channel

Parent topic: Matrix control structures

Parent topic:*Matrices*

Matrix control functions This section provides an overview of the matrix control functions.

vg_lite_identity function **Description:**

This function loads an identity matrix into a matrix variable.

Syntax:

```
vg_lite_error_t vg_lite_identity (
    vg_lite_matrix_t            *matrix,
);
```

Parameters:

Name	Description
*matrix	Pointer to the vg_lite_matrix_t structure that will be loaded with an identity matrix.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_set_pixel_matrix function **Description:**

This function sets up a pixel transform matrix m[20] which transforms each pixel as follows:

```
|a'|    |m0  m1  m2  m3  m4 |  |a|
|r'|    |m5  m6  m7  m8  m9 |  |r|
|g'| =  |m10 m11 m12 m13 m14|. |g|
|b'|    |m15 m16 m17 m18 m19|  |b|
|1 |    |0   0   0   0   1  |  |1|
```

The pixel transform for the A, R, G, B channels can be enabled/disabled individually with the channel parameter.

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_PIXEL_MATRIX) to determine HW support for gaussian blur.

Syntax:

```
vg_lite_error_t vg_lite_set_pixel_matrix (
    vg_lite_pixel_matrix_t           matrix,
    vg_lite_pixel_channel_enable_t   *channel
);
```

Parameters:

Name	Description
*matrix	Specifies the vg_lite_pixel_matrix_t pixel transform matrix that will be loaded.
*channel	Pointer to the vg_lite_pixel_channel_enable_t structure used to enable/disable individual channels.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_rotate function

Description:

This function rotates a matrix a specified number of degrees.

Syntax:

```
vg_lite_error_t vg_lite_rotate (
    vg_lite_float_t          degrees,
    vg_lite_matrix_t         *matrix
);
```

Parameters:

Name	Description
degrees	Number of degrees to rotate the matrix. Positive numbers rotate clockwise.The coordinates for the transformation are given in the surface coordinate system (top-to-bottom orientation). Rotations with positive angles are in the clockwise direction.
*matrix	Pointer to the vg_lite_matrix_t structure that has to be rotated

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_scale function **Description:**

This function scales a matrix in both horizontal and vertical directions.

Syntax:

```
vg_lite_error_t vg_lite_scale (
    vg_lite_float_t            scale_x,
    vg_lite_float_t            scale_y,
    vg_lite_matrix_t            *matrix
);
```

Parameters:

Name	Description
scale_x	Horizontal scale
scale_y	Vertical scale
matrix	Pointer to the vg_lite_matrix_t structure that will be scaled.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

vg_lite_translate function **Description:**

This function translates a matrix to a new location.

Syntax:

```
vg_lite_error_t vg_lite_translate (
    vg_lite_float_t            x,
    vg_lite_float_t            y,
    vg_lite_matrix_t            *matrix
);
```

Parameters:

Name	Description
x	X location of the transformation.
y	Y location of the transformation.
matrix	Pointer to the vg_lite_matrix_t structure that will be translated.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Matrix control functions

Parent topic:*Matrices*

Blits for compositing and blending This part of the API performs the hardware accelerated blit operations.

Compositing rules describes how two areas are combined to form a single area. Blending rules describes how combining the colors of the overlapping areas are combined. VGLite supports two blending operations and a subset of the Porter-Duff operations [PD84]. The Porter-Duff operators assume that the pixels have the alpha associated (premultiplied), it means that the pixels are premultiplied prior to the blending operation. GC555, GC355, and some GCNanoUltraV hardware support alpha premultiply for RGB image, but GCNanoLiteV does not.

The source image is copied to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction.

- The blit function can be used with or without the blend mode.
- The blit function can be used with or without specifying any color value.
- The blit function can be used for color conversion with an identity matrix and appropriate formats specified for the source and the destination buffers. In this case, do not specify blend mode and color value.

Blit enumerations This section gives details on blit enumerations.

vg_lite_blend_t **enumeration** This enumeration defines the blending modes supported by some VGLite API functions. S and D represent source and destination non-premultiplied RGB color channels. Sa and Da represent the source and destination alpha channels. SP and DP represent source and destination alpha-premultiplied RGB color channels (SP = S*Sa, DP = D*Da).

Note: VG_LITE_BLEND_*_LVGL modes are supported on all VG cores. On VG cores that do not support gcFEATURE_BIT_VG_LVGL_SUPPORT, the LVGL blend modes are supported by a combination of software and hardware operations. OPENVG_BLEND_* modes can only be supported on GC355 and GC555 cores.

Used in blit functions: vg_lite_blit, vg_lite_blit2, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_blend_t String Values	Description
VG_LITE_BLEND_NONE	**S, no blending** Non-premultiplied
VG_LITE_BLEND_SRC_OVER	**S + D * (1 - Sa)** Non-premultiplied
VG_LITE_BLEND_DST_OVER	**S * (1 – Da) + D** Non-premultiplied
VG_LITE_BLEND_SRC_IN	**S * Da** Non-premultiplied
VG_LITE_BLEND_DST_IN	**D * Sa** Non-premultiplied
VG_LITE_BLEND_MULTIPLY	**S * (1 - Da) + D * (1 - Sa) + S * D** Non-premultiplied
VG_LITE_BLEND_SCREEN	**S + D - S * D** Non-premultiplied
VG_LITE_BLEND_DARKEN	**min(SRC_OVER, DST_OVER)** Non-premultiplied
VG_LITE_BLEND_LIGHTEN	**max(SRC_OVER, DST_OVER)** Non-premultiplied
VG_LITE_BLEND_ADDITIVE	**S + D** Non-premultiplied
VG_LITE_BLEND_SUBTRACT	**D * (1 - Sa)** Non-premultiplied
VG_LITE_BLEND_NORMAL_LVGL	**S * Sa + D * (1 - Sa)** Non-premultiplied (from March 2023)
VG_LITE_BLEND_ADDITIVE_LVGL	**(S + D) * Sa + D * (1 - Sa)** Non-premultiplied (from March 2023)
VG_LITE_BLEND_SUBTRACT_LVGL	**(S - D) * Sa + D * (1 - Sa)** Non-premultiplied (from March 2023)
VG_LITE_BLEND_MULTIPLY_LVGL	**(S * D) * Sa + D * (1 - Sa)** Non-premultiplied (from March 2023)
OpenVG Porter-Duff Blend String Values	*(from Aug 2023)*
OPENVG_BLEND_NONE	**SP, no blending** Premultiplied
OPENVG_BLEND_SRC_OVER	**(SP + DP * (1 - Sa)) / (Sa + Da * (1 - Sa))** Premultiplied
OPENVG_BLEND_DST_OVER	**(SP * (1 - Da) + DP) / (Sa * (1 - Da) + Da)** Premultiplied
OPENVG_BLEND_SRC_IN	**(SP * Da) / (Sa * Da)** Premultiplied
OPENVG_BLEND_DST_IN	**(DP * Sa) / (Sa * Da)** Premultiplied
OPENVG_BLEND_MULTIPLY	**(SP*DP + SP*(1 - Da) + DP*(1 - Sa)) / (Sa + Da*(1 - Sa))** Premultiplied
OPENVG_BLEND_SCREEN	**(SP + DP - (SP*DP)) / (Sa + Da*(1 - Sa))** Premultiplied
OPENVG_BLEND_DARKEN	**min(SRC_OVER, DST_OVER)** Premultiplied
OPENVG_BLEND_LIGHTEN	**max(SRC_OVER, DST_OVER)** Premultiplied
OPENVG_BLEND_ADDITIVE	**(SP + DP) / (Sa + Da)** Premultiplied

Parent topic: Blit enumerations

`vg_lite_color_t` **parameter** The common parameter `vg_lite_color_t` is described in Table 1.

Parent topic: Blit enumerations

`vg_lite_color_transform_t` **structure** Specifies the pixel color_transform values for scale and bias.

Used in functions: `vg_lite_set_color_transform`.

vg_lite_color_transform_t members	Type	Description
a_scale	vg_lite_float_t	Scale value for alpha.
a_bias	vg_lite_float_t	Bias value for alpha.
r_scale	vg_lite_float_t	Scale value for red.
r_bias	vg_lite_float_t	Bias value for red.
g_scale	vg_lite_float_t	Scale value for green.
g_bias	vg_lite_float_t	Bias value for green.
b_scale	vg_lite_float_t	Scale value for blue.
b_bias	vg_lite_float_t	Bias value for blue.

Parent topic: Blit enumerations

vg_lite_filter_t **enumeration** Specifies the sample-filtering mode in VGLite blit and draw APIs.

Used in blit functions: vg_lite_blit, vg_lite_blit_rect.

Used in draw functions: vg_lite_draw_radial_gradient, vg_lite_draw_pattern.

vg_lite_filter_t string values	Description
VG_LITE_FILTER_POIN	Fetch only the nearest image pixel
VG_LITE_FILTER_LINEA	Use linear interpolation along a horizontal line
VG_LITE_FILTER_BI_LI	Use a 2x2 box around the image pixel and perform an interpolation
VG_LITE_FILTER_GAUS	Perform 3x3 gaussian blur with the convolution for image pixel. *(from March 2023)*

Parent topic: Blit enumerations

vg_lite_color_transform_t **structure** Specifies the pixel color_transform values for scale and bias.

Used in functions: vg_lite_set_color_transform.

vg_lite_color_transform_t members	Type	Description
a_scale	vg_lite_float_t	Scale value for alpha.
a_bias	vg_lite_float_t	Bias value for alpha.
r_scale	vg_lite_float_t	Scale value for red.
r_bias	vg_lite_float_t	Bias value for red.
g_scale	vg_lite_float_t	Scale value for green.
g_bias	vg_lite_float_t	Bias value for green.
b_scale	vg_lite_float_t	Scale value for blue.
b_bias	vg_lite_float_t	Bias value for blue.

Parent topic: Blit enumerations

vg_lite_mask_operation_t **enumeration** Specifies the mask operation mode in VGLite blit APIs.

Used in functions: vg_lite_blend_masklayer, vg_lite_render_masklayer.

vg_lite_mask_o string values	Description
VG_LITE_CLI	This operation sets all mask values in the region of interest to 0, ignoring the new mask layer.
VG_LITE_FIL	This operation sets all mask values in the region of interest to 1, ignoring the new mask layer.
VG_LITE_SE1	This operation copies values in the region of interest from the new mask layer, overwriting the previous mask values.
VG_LITE_UN.	This operation replaces the previous mask in the region of interest by its union with the new mask layer. The resulting values are always greater than or equal to their previous value.
VG_LITE_INT	This operation replaces the previous mask in the region of interest by its intersection with the new mask layer. The resulting mask values are always less than or equal to their previous value.
VG_LITE_SUI	This operation subtracts the new mask from the previous mask and replaces the previous mask in the region of interest by the resulting mask. The resulting values are always less than or equal to their previous value.

Parent topic: Blit enumerations

vg_lite_orientation_t **enumeration** Specifies the mirror orientation in VGLite blit APIs.

Used in functions: vg_lite_set_mirror.

vg_lite_orientation_t string values	Description
VG_LITE_ORIENTATION_TOP_BOTT	Target output orientation is from top to bottom (default).
VG_LITE_ORIENTATION_BOTTOM_T	Target output orientation is from bottom to top.

Parent topic: Blit enumerations

vg_lite_param_type_t **enumeration** Specifies the parameter type in VGLite blit APIs.

Used in functions: vg_lite_get_parameter.

vg_lite_param_type_t string value	Description
VG_LITE_GPU_IDLE_STATE	The count must be 1 for GPU idle state TRUE or FALSE.
VG_LITE_SCISSOR_RECT	The count must be 4n for x, y, right, bottom.

Parent topic: Blit enumerations

Parent topic:*Blits for compositing and blending*

Blit structures This section provides details about blit structures.

vg_lite_buffer_t **structure** Defined under the "Pixel buffer structures" section (see vg_lite_buffer_t structure).

Parent topic: Blit structures

vg_lite_color_key_t **structure** A "color key" have two sections, where each section contains R,G,B channels, which are noted as high_rgb and low_rgb respectively. *(from April 2022)*

When the enable value is true, the color key specified is effective and the alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are in range [low_rgb, high_rgb]. After the color key is used in the current frame, if the color key is not needed for the next frame, it should be disabled before the next frame.

Used in structure: vg_lite_color_key4_t

vg_lite_color_key_t members	Type	Description
enable	vg_lite_uint8	When set (true), this color key is enabled
low_r	vg_lite_uint8	The R channel of low_rgb
low_g	vg_lite_uint8	The G channel of low_rgb
low_b	vg_lite_uint8	The B channel of low_rgb
alpha	vg_lite_uint8	The alpha value to replace the destination pixel alpha channel value with
high_r	vg_lite_uint8	The R channel of high_rgb
high_g	vg_lite_uint8	The G channel of high_rgb
high_b	vg_lite_uint8	The B channel of high_rgb

Parent topic: Blit structures

vg_lite_color_key4_t **structure** The priority order is: color_key_0 > color_key_1 > color_key_2 > color_key_3. *(from April 2022)*

Used in blit function: vg_lite_set_color_key

vg_lite_color_key4_t members	Type	Description
color_key_0		high_rgb_0, low_rgb_0, alpha_0, enable_0
color_key_1		high_rgb_1, low_rgb_1, alpha_1, enable_1
color_key_2		high_rgb_2, low_rgb_2, alpha_2, enable_2
color_key_3		high_rgb_3, low_rgb_3, alpha_3, enable_3

Parent topic: Blit structures

vg_lite_matrix_t **structure** Defined under the "Matrix control structures" section (see vg_lite_matrix_t structure).

Parent topic: Blit structures

vg_lite_path_t **structure** Defined under the "Vector path structures" section (see vg_lite_path_t structure).

Parent topic: Blit structures

vg_lite_rectangle_t **structure** This structure defines a rectangle by using coordinates.

Used in blit function: vg_lite_clear.

vg_lite_rectangle_t member	Type	Description
x	vg_lite_int32_t	X origin of rectangle, left coordinate in pixels
y	vg_lite_int32_t	Y origin of rectangle, top coordinate in pixels
width	vg_lite_int32_t	X Width of rectangle in pixels
height	vg_lite_int32_t	Y Height of rectangle in pixels

Parent topic: Blit structures

vg_lite_point_t structure This structure defines a 2D point *(from March 2021)*.

Used in structure: vg_lite_point4_t.

vg_lite_point_t member	Type	Description
X	vg_lite_int32_t	X value of coordinate
Y	vg_lite_int32_t	Y value of coordinate

Parent topic: Blit structures

vg_lite_point4_t structure This structure defines four 2D points that form a polygon. The points are defined by structure vg_lite_point_t. *(from March 2021)*

vg_lite_point4_t member	Type	Description
vg_lite_point_t[4]	vg_lite_int32_t each	a set of four points

Parent topic: Blit structures

vg_lite_float_point_t structure This structure defines a 2D float point *(from March 2024)*.

Used in structure: vg_lite_float_point4_t.

vg_lite_float_point_t members	**Type**	**Description**
x	vg_lite_float_t	X value of coordinate
y	vg_lite_float_t	Y value of coordinate

Parent topic: Blit structures

vg_lite_float_point4_t structure This structure defines four 2D float points that form a polygon. The points are defined by structure vg_lite_float_point_t. *(from March 2024)*

Used in blit function: vg_lite_get_transform_matrix.

vg_lite_float_point4_t members	**Type**	**Description**
vg_lite_float_point[4]	vg_lite_float_t each	a set of four points

Parent topic: Blit structures

Parent topic:*Blits for compositing and blending*

Blit functions This section provides an overview on blit functions.

vg_lite_blit function **Description:**

This is the blit function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth, especially with a rotation matrix. VGLite path rendering can be used to achieve high-quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Note:

- The blit function can be used with or without the blend function (*vg_lite_blend_t*)
- The blit function can be used with or without specifying a foreground color value (*vg_lite_color_t*)
- The blit function can be used for color conversion with an identity matrix and appropriate formats specified for the source and the destination buffers. In this case, do not specify blend mode and color value.

Syntax:

```
vg_lite_error_t vg_lite_blit (
    vg_lite_buffer_t        *target,
    vg_lite_buffer_t        *source,
    vg_lite_matrix_t        *matrix,
    vg_lite_blend_t         blend,
    vg_lite_color_t         color,
    vg_lite_filter_t        filter
);
```

Parameters:

Nam	Description
*target	Points to the vg_lite_buffer_t structure, which defines the destination buffer. See Image Source Alignment Requirement for valid destination color formats for the blit functions.
*sou	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
*matrix	Points to a vg_lite_matrix_t structure that defines the transformation matrix of source pixels into the target. If the matrix is NULL, then an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
blend	Specifies one of the enum vg_lite_blend_t values for hardware-supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). **Note:** If the matrix parameter is specified with rotation or perspective, and the blend parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN; then, the VGLite driver overwrites the application setting for the blit operation as follows: - If gcFEATURE_BIT_VG_BORDER_CULLING (*vg_lite_feature_t*) is supported, then Transparency mode is always set to TRANSPARENT- If gcFEATURE_BIT_VG_BORDER_CULLING (*vg_lite_feature_t*) is not supported, then Blend mode is always set to VG_LITE_BLEND_SRC_OVER. It happens due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mixed color gets multiplied with each source pixel before blending happens. If you don't need a mix color, set the color parameter to 0.**Note:** this parameter has no effect if the source vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_blit2 function

Description:

This is the blit function for use with two sources. The blit2 operation is performed using two source buffers and one destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Source0 and Source1 are first blended according to the blend mode with a specific transformation matrix for each image. Source1 is used as the source while Source0 is used as the dest and is directly output to the render target buffer.

The specified matrices can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth, especially with a rotation matrix. VGLite path rendering can be used to achieve high-quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_DOUBLE_IMAGE) to determine HW support for double image.

Note:

- The vg_lite_blit function can be used for color conversion for Source0 or Source1 before merging sources with vg_lite_blit2.

Syntax:

```
vg_lite_error_t vg_lite_blit2 (
    vg_lite_buffer_t            *target,
```

(continues on next page)

(continued from previous page)

```
    vg_lite_buffer_t            *source0,
    vg_lite_buffer_t            *source1,
    vg_lite_matrix_t             *matrix0,
    vg_lite_matrix_t             *matrix1,
    vg_lite_blend_t             blend,
    vg_lite_filter_t            filter
);
```

Parameters:

Nam	Description
*target	Points to the vg_lite_buffer_t structure, which defines the destination buffer. See Alignment notes for valid destination color formats for the blit functions
*sou *sou	Points to the vg_lite_buffer_t structure for the source0 and source1 buffers. All color formats available in the vg_lite_buffer_format_t‘ enum are valid source formats for the blit functions.
*mat *matrix1	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix0 for the source0 pixels and matrix1 for the source1 pixels. If matrix0 and matrix1 are both NULL, the identity matrix is assumed, meaning the blending result of Source0 and Source1 is copied directly on the target at location(0,0).
blenc	Specifies one of the enum vg_lite_blend_t values for hardware-supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). Note: If the "matrix" parameter is specified with rotation or perspective, and the "blend" parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN, the VGLite driver overwrites the application's setting for the BLIT operation as follows: - If gcFEATURE_BIT_VG_BORDER_CULLING (*vg_lite_feature_t*) is supported, the transparency mode will always be set to TRANSPARENT. - If gcFEATURE_BIT_VG_BORDER_CULLING (*vg_lite_feature_t*) is not supported, the blend mode will always be set to VG_LITE_BLEND_SRC_OVER. This is due to some limitations in the VGLite hardware.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_blit_rect **function** **Description:**

This is the blit rectangle function. The blit operation is performed using a source and a destination buffer. The source and destination buffer structures are defined using the vg_lite_buffer_t structure. Blit copies a source image to the destination window with a specified matrix that can include translation, rotation, scaling, and perspective correction. Note that vg_lite_buffer_t does not support coverage sample anti-aliasing so the destination buffer edge may not be smooth, especially with a rotation matrix. VGLite path rendering can be used to achieve high-quality coverage sample anti-aliasing (16X, 8X, 4X) rendering effect.

Note:

- The blit_rect function can be used with or without the blend function (vg_lite_blend_t).
- The blit_rect function can be used with or without specifying any color value (vg_lite_color_t).

- The blit_rect function can be used for color conversion with an identity matrix and appropriate formats specified for the source and destination buffers. In this case, do not specify blend mode and color value.
- The vg_lite_blit_rect rectangle start origin point is always (0,0) for hardware versions prior to GCNanoLiteV 1311p that do not support a non-zero rectangle origin.

Syntax:

```
vg_lite_error_t vg_lite_blit_rect (
    vg_lite_buffer_t            *target,
    vg_lite_buffer_t            *source,
    vg_lite_rectangle_t         *rect,
    vg_lite_matrix_t            *matrix,
    vg_lite_blend_t             blend,
    vg_lite_color_t             color,
    vg_lite_filter_t            filter
);
```

Parameters:

Nam	Description
*target	Points to the vg_lite_buffer_t structure that defines the destination buffer.
*sou	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit_rect function.
*rect	Specifies the rectangle area of the source image to blit. rect[0]/[1]/[2]/[3] are x, y, width, and height of the source rectangle respectively. **Note:** Non-zero source origins are supported.
*matrix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of source pixels into the target. If the matrix is NULL, then an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
blenc	Specifies one of the enum vg_lite_blend_t values for hardware-supported blend modes to be applied to each image pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0). **Note:** If the matrix parameter is specified with rotation or perspective, and the blend parameter is specified as VG_LITE_BLEND_NONE, VG_LITE_BLEND_SRC_IN, or VG_LITE_BLEND_DST_IN; then, the VGLite driver overwrites the application setting for the blit operation as follows: - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is supported, then Transparency mode is always set to TRANSPARENT - If gcFEATURE_BIT_VG_BORDER_CULLING (vg_lite_feature_t) is not supported, then Blend mode is always set to VG_LITE_BLEND_SRC_OVER. It happens due to some limitations in the VGLite hardware.
color	If non-zero, this color value is used as a mix color. The mixed color gets multiplied with each source pixel before blending happens. If you do not need a mix color, then set the color parameter to 0. **Note:** This parameter has no effect if the source vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_copy_image **function** **Description:**

This API copied a pixel rectangle with dimension (width, height) from source buffer to destination buffer. The source image pixel (sx *+ i,*sy + *j*) is copied to the destination image pixel (dx *+ i,*dy + *j*), for *0 □ i <*width and *0 □ j <*height. Pixels whose source or destination lie outside the bounds of the respective image are ignored. Pixel format conversion is applied as needed.

No pre-multiply, transformation, blending, filtering operations are applied to the pixel copy.

Syntax:

```
vg_lite_error_t vg_lite_copy_image (
    vg_lite_buffer_t            *target,
    vg_lite_buffer_t            *source,
    vg_lite_int32_t             sx,
    vg_lite_int32_t             sy,
    vg_lite_int32_t             dx,
    vg_lite_int32_t             dy,
    vg_lite_int32_t             width,
    vg_lite_int32_t             height
);
```

Parameters:

Nam	Description
*tar-get	Points to the vg_lite_buffer_t structure that defines the destination buffer.
*sour	Points to the vg_lite_buffer_t structure for the source buffer. All color formats available in the vg_lite_buffer_format_t enum are valid source formats for the blit function.
sx, sy	Pixel coordinates of the lower-left corner of a pixel rectangle within the source buffer.
dx, dy	Pixel coordinates of the lower-left corner of a pixel rectangle within the target buffer.
widtl	Width of the copied pixel rectangle.
heigh	Height of the copied pixel rectangle.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_get_transform_matrix **function** **Description:**

This function generates a 3x3 homogenous transform matrix from 4 float point source coordinates and 4 float point target coordinates. *(from March 2021)*

Syntax:

```
vg_lite_error_t vg_lite_get_transform_matrix (
    vg_lite_float_point4_t       src,
    vg_lite_float_point4_t       dst,
    vg_lite_matrix_t            *mat
);
```

Parameters:

Name	Description
src	Pointer to the four 2D points that form a source polygon
dst	Pointer to the four 2D points that form a destination polygon
mat	Output parameter, pointer to a 3x3 homogenous matrix that transforms the source polygon to a destination polygon.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_clear **function** **Description:**

This function performs the clear operation, clearing/filling the specified buffer (entire buffer or partial rectangle in a buffer) with an explicit color.

Syntax:

```
vg_lite_error_t vg_lite_clear (
    vg_lite_buffer_t            *target,
    vg_lite_rectangle_t          *rect,
    vg_lite_color_t             color
);
```

Parameters:

Name	Description
*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for the clear function.
*rect	Pointer to the vg_lite_rectangle_tstructure that specifies the area to be filled. If the rectangle is NULL, the entire target buffer is filled with the specified color.
color	Clear color, as specified in the vg_lite_color_t enum that is the color value to use for filling the buffer. If the buffer is in L8 format, the RGBA color is converted into a luminance value.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit functions

vg_lite_set_color_key **function** **Description:**

This function sets a color key. Color key can be used for blit or for draw pattern operations. *(from April 2022)*

A "color key" have two sections, where each section contains R,G,B channels which are noted as high_rgb and low_rgb respectively.

When the vg_lite_color_key_t structure value enable is true, the color key specified is effective and the alpha value is used to replace the alpha channel of the destination pixel when its RGB channels are within range [low_rgb, high_rgb]. After the color key is used in the current frame, if the color key is not needed for the next frame, it should be disabled before the next frame.

Hardware support for color key is not available for GCNanoLiteV. Application can use VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_COLOR_KEY) to determine HW support for color key.

Syntax:

```
vg_lite_error_t vg_lite_set_color_key (
    vg_lite_color_key4_t           colorkey
);
```

Parameters:

Parameter	Description
colorkey	Color keying parameters as defined by vg_lite_color_key4_t.

Here are 4 groups of color key states:

- color_key_0, high_rgb_0, low_rgb_0, alpha_0, enable_0
- color_key_1, high_rgb_1, low_rgb_1, alpha_1, enable_1
- color_key_2, high_rgb_2, low_rgb_2, alpha_2, enable_2
- color_key_3, high_rgb_3, low_rgb_3, alpha_3, enable_3

The priority order of these states is:

color_key_0 > color_key_1 > color_key_2 > color_key_3.

Returns:

VG_LITE_SUCCESS if successful. VG_LITE_NOT_SUPPORT if color key is not supported in hardware.

Parent topic: Blit functions

vg_lite_gaussian_filter **function**

Description:

This function sets 3x3 gaussian blur weighted values to filter an image pixel. *(from March 2023)*

The parameters w0, w1, w2 define a 3x3 gaussian blur weight matrix as:

```
|   w2    w1    w2 |
|   w1    w0    w1 |
|   w2    w1    w2 |
```

The sum of the 9 kernel weights must be 1.0 to avoid convolution overflow (w0 + 4*w1 + 4*w2 = 1.0).

The 3x3 weight matrix applies to a 3x3 pixel block:

```
| pixel[i-1][j-1]     pixel[i][j-1]       pixel[i+1][j-1]|
| pixel[i-1][j]       pixel[i][j]         pixel[i+1][j]  |
| pixel[i-1][j+1]     pixel[i][j+1]       pixel[i+1][j+1]|
```

With the following dot product equation:

```
color[i][j] = w2*pixel[i-1][j-1] + w1*pixel[i][j-1] + w2*pixel[i+1][j-1]
            + w1*pixel[i-1][j]   + w0*pixel[i][j]   + w1*pixel[i+1][j]
            + w2*pixel[i-1][j+1] + w1*pixel[i][j+1] + w2*pixel[i+1][j+1];
```

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_GAUSSIAN_BLUR) to determine HW support for gaussian blur.

Syntax:

```
vg_lite_error_t vg_lite_gaussian_filter (
    vg_lite_float_t             w0
    vg_lite_float_t             w1
    vg_lite_float_t             w2
);
```

Parameters:

Parameter	Description

w0, w1, w2	w0, w1, w2 define a 3x3 gaussian blur weighted matrix as:
	`| w2 w1 w2 |`
	`| w1 w0 w1 |`
	`| w2 w1 w2 |`

Returns:

VG_LITE_SUCCESS if successful. Otherwise, VG_LITE_NOT_SUPPORT if gaussian blur is not supported in hardware.

Parent topic: Blit functions

Parent topic:*Blits for compositing and blending*

Blit/Draw extended functions The following BLIT or DRAW-related functions typically require GC355 or GC555 hardware and are not available for all Vivante Vector Graphics hardware configurations.

Applications can use the VGLite API vg_lite_query_feature to determine HW support for the related functionality.

vg_lite_get_parameter function **Description:**

This function returns the selected VGLite / GPU states to the application.

(from Aug 2023)

Syntax:

```
vg_lite_error_t vg_lite_get_parameter (
    vg_lite_param_type_t          type,
    vg_lite_int32_t              count,
    vg_lite_pointer              params
);
```

Parameters:

Parameter	Description
type	The parameter type to be queried (VG_LITE_GPU_IDLE_STATE, VG_LITE_SCISSOR_RECT)
count	The number of returned parameters
params	The pointer to the array of returned parameters

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_enable_scissor **function**

Description:

This function enables scissor rectangle operation for the rectangle regions defined by vg_lite_scissor_rects API. *(from March 2020, modified August 2020, requires GC355 or GC555 hardware)*

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_SCISSOR) to determine HW support for scissoring. Support is available with GC355 and GC555.

Syntax:

```
vg_lite_error_t vg_lite_enable_scissor (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_disable_scissor **function**

Description:

This function disables scissor operation for the rectangle regions defined by the vg_lite_scissor_rects API. *(from March 2020, modified August 2020, requires GC355 or GC555 hardware)*.

Syntax:

```
vg_lite_error_t vg_lite_disable_scissor (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_scissor_rects **function**

Description:

This function defines scissor rectangle regions on the hardware mask layer. But the scissor function is enable/disabled by vg_lite_enable_scissor and vg_lite_disable_scissor APIs. *(from August 2022, requires GC355 or GC555 hardware)*.

Syntax:

```
vg_lite_error_t vg_lite_scissor_rects (
    vg_lite_buffer_t              *target,
    vg_lite_uint32_t              nums,
    vg_lite_rectangle_t           rect[]
);
```

Parameters:

Parameter	Description
target	Target render buffer that has the scissor mask layer.
nums	Number of scissor rectangles.
rect[]	The scissor rectangle array.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_scissor function

Description:

This is a legacy scissor API function that can be used to set a single scissor rectangle for the render target. This scissor API is supported by a different hardware mechanism other than the mask layer and it has better performance than the mask layer scissor function.

This API is not enabled/disabled by vg_lite_enable_scissor and vg_lite_disable_scissor APIs. Instead, the vg_lite_set_scissor API calls with a valid scissor rectangle input (x, y, right, bottom) enables the scissor function by default. The vg_lite_set_scissor API call with input parameter (-1, -1, -1, -1) disables the scissor function. *(requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_scissor (
    vg_lite_int32_t               x,
    vg_lite_int32_t               y,
    vg_lite_int32_t               right,
    vg_lite_int32_t               bottom
);
```

Parameters:

Parameter	Description
x	X Origin of rectangle, left coordinate in pixels
Y	Y Origin of rectangle, top coordinate in pixels
right	X rightmost pixel of the rectangle
bottom	Y bottom pixel of the rectangle

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_disable_color_transform **function** **Description:**

This function is used to disable color transformation. By default, the color transform is turned off. *(from Sept 2022, only for GC355 and GC555 hardware)*

Applications can use the VGLite API vg_lite_query_feature(gcFEATURE_BIT_VG_COLOR_TRANSFORMATION)‘ to determine HW support for color transformation. Support is available with GC355 and GC555.

Syntax:

```
vg_lite_error_t vg_lite_disable_color_transform (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_enable_color_transform **function** **Description:**

This function is used to enable color transformation. By default, the color transform is turned off. *(from Sept 2022, only for GC355 and GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_enable_color_transform (
);
```

Parameters: None

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_color_transform **function** **Description:**

This function is used to set pixel scale and bias values for color transformation for each pixel channel. *(from August 2022, only for GC355 and GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_color_transform (
    vg_lite_color_transform_t     *values
);
```

Parameters:

Parameter	Description
*values	Pointer to the color transformation values to set. See enum vg_lite_color_transform_t.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_enable_masklayer function

Description:

This function controls the availability of mask functionality. The mask is turned off by default. *(from August - Sept mber 2022, requires GC555 hardware)*

Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_MASK) to determine HW support for mask. The blit and draw mask functions below require GC555 hardware support. These functions were introduced in August 2022 and the syntax or name was further refined in September 2022.

Syntax:

```
vg_lite_error_t vg_lite_enable_masklayer (
    void
  );
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_disable_masklayer function

Description:

This function controls the availability of mask functionality. The mask is turned off by default. *(from August -September 2022, requires GC555 hardware, prior to Sept 2022 name was vg_lite_disable_mask_layer)*

Syntax:

```
vg_lite_error_t vg_lite_disable_masklayer (
    void
);
```

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_create_masklayer function

Description:

This function creates a mask layer with the specified width and height. The mask format defaults to A8 and the default mask value is 255. *(from August 2022-September, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_create_masklayer (
    vg_lite_buffer_t            *masklayer,
    vg_lite_uint32_t            width,
    vg_lite_uint32_t            height
);
```

Parameters:

Parameter	Description
*masklayer	Points to the address of the buffer of the mask layer to be created.
width	Mask layer width (in pixels).
height	Mask layer height (in pixels).

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_blend_masklayer function

Description:

This function blends the specified area of the source mask layer with the destination mask layer according to an vg_lite_mask_operation_t enumeration value, to create a blended destination mask layer. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_blend_masklayer (
    vg_lite_buffer_t                *dst_masklayer,
    vg_lite_buffer_t                *src_masklayer,
    vg_lite_mask_operation             operation,
    vg_lite_rectangle_t              *rect,
);
```

Parameters:

Parameter	Description
*dst_masklay	Points to the address of the buffer of the destination mask layer.
*src_masklay	Points to the address of the buffer of the source mask layer.
operation	Blending mode to be applied to each image pixel, as defined by the enum *vg_lite_mask_operation_t*.
*rect	The rectangle area (x, y, width, height) of the blend operation.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_masklayer function

Description:

This function sets the given mask layer to the hardware. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_masklayer (
    vg_lite_buffer_t               *masklayer
);
```

Parameters:

Parameter	Description
*masklayer	Points to the address of the buffer of the mask layer to be set.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_render_masklayer function **Description:**

This function draws the mask layer according to the specified path, color, and matrix information. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_render_masklayer (
    vg_lite_buffer_t              *masklayer,
    vg_lite_mask_operation         operation,
    vg_lite_path_t                *path,
    vg_lite_fill_t                fill_rule,
    vg_lite_color_t                color,
    vg_lite_matrix_t              *matrix
);
```

Parameters:

Parameter	Description
*mas	Points to the address of the buffer of the destination mask layer.
operation	Blending mode to be applied to each image pixel, as defined by the enum vg_lite_mask_operation_t
*patl	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw. Refer to Vector path opcodes for plotting paths in this document for opcode detail.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
color	Specifies the color vg_lite_color_t RGBA value to be applied to each pixel drawn by the path.
*matrix	Points to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed, meaning the source is copied directly on the target at 0,0 location. which is usually a bad idea since the path can be anything.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_destroy_masklayer function **Description:**

This function is used to free a mask layer. *(from August-September 2022, requires GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_destroy_masklayer (
    vg_lite_buffer_t              masklayer
);
```

Parameters:

Parameter	Description
*masklayer	Points to the address of the buffer of the mask layer to be destroyed.

Returns:

Returns VG_LITE_SUCCESS if the function is successful. See vg_lite_error_t enum for other return codes.

Parent topic: Blit/Draw extended functions

vg_lite_set_mirror **function** **Description:**

This function is used to control mirror functionality. By default, the mirror is turned off and the default output orientation is from top to bottom. *(from August 2022, only for GC555 hardware)*

Application can use VGLite API [vg_lite_query_feature](vg_lite_query_feature_function.md) (gcFEATURE_BIT_VG_MIRROR) to determine HW support for mirror. Mirror functions require GC555 hardware.

Syntax:

```
vg_lite_error_t vg_lite_set_mirror (
    vg_lite_orientation_t          orientation
);
```

Parameters:

Parameter	Description
orientation	The orientation mode as defined by the enum *vg_lite_orientation_t*.‘

Returns:

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if not supported.

Parent topic: Blit/Draw extended functions

vg_lite_source_global_alpha **function** **Description:**

This function sets the image/source global alpha and return a status error code. *(from June 2021, requires GCNanoUltraV or GC555 hardware)*

Application can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_GLOBAL_ALPHA) to determine HW support for global alpha. The global alpha BLIT-related functions require GC-NanoUltraV or GC555 hardware.

Syntax:

```
vg_lite_error_t vg_lite_source_global_alpha (
    vg_lite_global_alpha_t         alpha_mode,
    vg_lite_uint8_t                alpha_value
);
```

Parameters:

Parameter	Description
alpha_mode	Global alpha mode value. See enum vg_lite_global_alpha_t.
alpha_value	The image/source global alpha value to set.

Returns:

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if global alpha is not supported.

Parent topic: Blit/Draw extended functions

vg_lite_dest_global_alpha function

Description:

This function sets the destination global alpha and returns a status error code. *(from June 2021, requires GCNanoUltraV or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_dest_global_alpha (
    vg_lite_global_alpha_t         alpha_mode,
    vg_lite_uint8_t                alpha_value
);
```

Parameters:

Parameter	Description
alpha_mode	Global alpha mode value. See enum vg_lite_global_alpha_t.
alpha_value	The destination global alpha value to set.

Returns:

VG_LITE_SUCCESS or VG_LITE_NOT_SUPPORT if global alpha is not supported.

Parent topic: Blit/Draw extended functions

Parent topic:*Blits for compositing and blending*

Vector path control

This chapter provides an overview of the vector path enumerations, structures, functions, and opcodes for plotting paths.

Vector path enumerations

This section provides an overview of vector path enumerations.

vg_lite_format_t enumeration

Values for vg_lite_format_t enum are defined in Table 1.

If vg_lite_format_t	Path data alignment in the array should be:
VG_LITE_S8	8 bit
VG_LITE_S16	2 bytes
VG_LITE_S32	4 bytes

Parent topic: Vector path enumerations

vg_lite_quality_t enumeration

Specifies the level of hardware assisted anti-aliasing.

Used in structure: vg_lite_path_t.

Used in function: vg_lite_init_path, vg_lite_init_arc_path.

vg_lite_qua string values	Description
VG_LITE_	High quality: 16x coverage sample anti-aliasing
VG_LITE_	Upper quality: 8x coverage sample anti-aliasing. Use vg_lite_query_feature to determine availability of 8x CSAA (feature enum value gcFEATURE_BIT_VG_QUALITY_8X.(deprecated from June 2020, available with supported hardware from August 2022).
VG_LITE_	Medium quality: 4x coverage sample anti-aliasing
VG_LITE_	Low quality: No anti-aliasing

Parent topic: Vector path enumerations

Parent topic:*Vector path control*

Vector path structures

This section provides an overview of vector path structures.

vg_lite_hw_memory structure

This structure gets the memory allocation information recorded by the kernel.

Used in structure: vg_lite_path_t.

vg_lite_hw_men member	Type	Description
handle	vg_lite_p	GPU memory object handle
memory	vg_lite_p	Logical memory address
address	vg_lite_u	GPU memory address
bytes	vg_lite_u	Size of memory
property	vg_lite_u	Bit 0 is used for path upload: - 0: Disable path data uploading (always embedded into command buffer) - 1: Enable auto path data uploading

Parent topic: Vector path structures

vg_lite_path_t structure

This structure describes VGLite path data.

Path data is made of op codes and coordinates. The format for op codes is always VG_LITE_S8. For more details on opcodes, see Vector path opcodes for plotting paths.

Used in init functions: vg_lite_init_path, vg_lite_init_arc_path, vg_lite_upload_path, vg_lite_clear_path, vg_lite_append_path.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_path_t members	Type	Description
bounding_box[4]	vg_lite_float_	bounding box for path [0] left [1] top [2] right [3] bottom
quality	vg_lite_quality	enum for quality hint for the path, anti-aliasing level
format	vg_lite_format	enum for coordinate format
uploaded	vg_lite_hw_me	struct with path data that has been uploaded into GPU addressable memory
path_length	vg_lite_uint32	number of bytes in the path
path	vg_lite_pointe	pointer to path data
path_changed	vg_lite_int8_t	0: not changed; 1: changed.
pdata_internal	vg_lite_int8_t	0: path data memory is allocated by application; 1: path data memory is allocated by driver.
path_type	vg_lite_path_t	The draw path type as specified in enum vg_lite_path_type_t. *(added for stroke control, from March 2022)*
*stroke	vg_lite_stroke_	As defined by structure vg_lite_stroke_t *(added for stroke control, from March 2022)*
stroke_path	vg_lite_pointe	Pointer to the physical description of the stroke path. *(added for stroke control, from March 2022)*
stroke_size	vg_lite_uint32	Number of bytes in the stroke path data. *(added for stroke control, from March 2022)*
stroke_color	vg_lite_color_t	The stroke path fill color. *(from Sept 2022)*
add_end	vg_lite_int8_t	Flag that add end_path in driver *(from March 2023)*

Special notes for path objects:

- Endianness has no impact, as it is aligned against the boundaries
- Multiple contiguous opcodes should be packed by the size of the specified data format. For example, by 2 bytes for VG_LITE_S16 or by 4 bytes for VG_LITE_S32.

For example, because opcodes are 8-bit (1-byte), 16-bit (2-byte), or 32-bit (4-byte) data types:

```
...
<opcode1_that_needs_data>
<align_to_data_size>
<data_for_opcode1>
<opcode2_that_doesnt_need_data>
<align_to_data_size>
<opcode3_that_needs_data>
<align_to_data_size>
<data_for_opcode3>
...
```

- Path data in the array should always be 1-, 2-, or 4-byte aligned, depending on the format:

For example, for 32-bit (4-byte) data types:

```
...
<opcode1_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode1>
<opcode2_that_doesnt_need_data>
<pad to 4 bytes>
<opcode3_that_needs_data>
<pad to 4 bytes>
<4 byte data_for_opcode3>
...
```

Parent topic: Vector path structures

Parent topic: *Vector path control*

Vector path functions When using a small tessellation window and depending on a path's size, a path might be uploaded to the hardware multiple times because the hardware scanline convert path with the provided tessellation window size, so VGLite path rendering performance might go down. That is why it is preferable to set the tessellation buffer size to the most common path size, for example if you only render 24-pt fonts, you can set the tessellation buffer to be 24x24.

All the RGBA color formats available in the vg_lite_buffer_format_t are supported as the destination buffer for the draw function.

vg_lite_get_path_length function **Description:**

This function calculates the path command buffer length (in bytes).

The application is responsible for allocating a buffer according to the buffer length calculated with this function. Then, the buffer is used by the path as a command buffer. The VGLite driver does not allocate the path command buffer.

Syntax:

```
vg_lite_uint32_t vg_lite_get_path_length (
    vg_lite_uint8_t             *opcode,
    vg_lite_uint32_t            count,
    vg_lite_format_t            format
);
```

Parameters:

Parameter	Description
*opcode	Pointer to the opcode array to use to construct the path. *(*opcode from March 2023)*
count	The opcode count
format	The coordinate data format. All formats available in the vg_lite_format_t enum are valid formats for this function.

Returns:

Returns the command buffer length in bytes.

Parent topic: Vector path functions

vg_lite_append_path function **Description:**

This function assembles the command buffer for the path. The command buffer is allocated by the application and assigned to the path. This function makes the final GPU command buffer for the path based on the input opcodes (cmd) and coordinates (data). The application is responsible for allocating a buffer large enough for the path*. (from Jan 2022, returns a vg_lite_error_t status code)*

Syntax:

```
vg_lite_error_t vg_lite_append_path (
    vg_lite_path_t              *path
    vg_lite_uint8_t             *opcode,
    vg_lite_pointer             data,
    vg_lite_uint32_t            seg_count
```

(continues on next page)

(continued from previous page)

```
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t structure with the path definition.
*opcode	Pointer to the opcode array to use to construct the path. *(*opcode from March 2023)*
data	Pointer to the coordinate data array to use to construct the path
seg_count	The opcode count

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

vg_lite_init_path function

Description:

This function initializes a path definition with specified values. *(From Dec 2019 returns vg_lite_error_t, previous was void.)*

Syntax:

```
vg_lite_error_t vg_lite_init_path (
    vg_lite_path_t              *path,
    vg_lite_format_t            format,
    vg_lite_quality_t           quality,
    vg_lite_uint32_t            length,
    vg_lite_pointer             *data,
    vg_lite_float_t             min_x,
    vg_lite_float_t             min_y,
    vg_lite_float_t             max_x,
    vg_lite_float_t             max_y
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t structure for the path object to be initialized with the member values specified.
format	The coordinate data format. All formats available in the vg_lite_format_t enum are valid formats for this function.
quality	The quality for the path object. All formats available in the vg_lite_quality_t enum are valid formats for this function.
length	The length of the path data (in bytes)
*data	Pointer to path data
min_x min_y max_x max_y	Minimum and maximum x and y values specifying the bounding box of the path

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

vg_lite_init_arc_path function

Description:

This function initializes an arc path definition with specified values. *(from February 2021)*

Syntax:

```
vg_lite_error_t vg_lite_init_arc_path (
    vg_lite_path_t              *path,
    vg_lite_format_t            format,
    vg_lite_quality_t           quality,
    vg_lite_uint32_t            length,
    vg_lite_pointer             *data,
    vg_lite_float_t             min_x,
    vg_lite_float_t             min_y,
    vg_lite_float_t             max_x,
    vg_lite_float_t             max_y
);
```

Parameters:

Parameter	Function
*path	Pointer to the vg_lite_path_t structure for the path object to be initialized with the member values specified.
format	The coordinate data format. The vg_lite_format_t enum value should be FP32.
quality	The quality for the path object. All formats available in the vg_lite_quality_t enum are valid formats for this function.
length	The length of the path data (in bytes).
*data	Pointer to path data.
min_x min_y max_x max_y	Minimum and maximum x and y values specifying the bounding box of the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

vg_lite_upload_path function

Description:

This function is used to upload a path to GPU memory.

In normal cases, the VGLite driver will copy any path data into a command buffer structure during runtime. This does take some time if there are many paths to be rendered. Also, in an embedded system the path data won't change - so it makes sense to upload the path data into GPU memory in such a form that the GPU can directly access it. This function will signal the driver to allocate a buffer that will contain the path data and the required command buffer header and footer data for the GPU to access the data directly. Call vg_lite_clear_path to free this buffer after the path is used.

Syntax:

```
vg_lite_error_t vg_lite_upload_path (
    vg_lite_path_t              *path
);
```

Parameters:

Parameter	Description
*path	Pointer to a vg_lite_path_t structure that contains the path to be uploaded.

Returns:

VG_LITE_OUT_OF_MEMORY if not enough GPU memory is available for buffer allocation.

Parent topic: Vector path functions

vg_lite_clear_path function

Description:

This function will clear and reset path member values. If the path has been uploaded, it frees the GPU memory allocated when uploading the path. *(From Dec 2019 returns vg_lite_error_t, previous was void.)*

.

Syntax:

```
vg_lite_error_t vg_lite_clear_path (
   vg_lite_path_t           *path
);
```

Parameters:

Parameter	Description
*path	Pointer to the vg_lite_path_t path definition to be cleared.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Vector path functions

Parent topic:*Vector path control*

Vector path opcodes for plotting paths

The following opcodes are path drawing commands available for vector path data.

A path operation is submitted to the GPU as [Opcode | Coordinates]. The operation code is stored as a VG_LITE_S8 while the coordinates are specified via vg_lite_format_t.

Op-code	Arguments	Description
0x00	None	**VLC_OP_END**. Finish tessellation. Close any open path.
0x01	None	**VLC_OP_CLOSE**. For VGLite driver internal use only. Application should not use this OP directly.
0x02	(x, y)	**VLC_OP_MOVE**. Move to the given vertex. Close any open path. $start_x = x$ $start_y = y$
0x03	(□x, □y)	**VLC_OP_MOVE_REL**. Move to the given relative point. Close any open path. $start_x = start_x + \Delta x$ $start_y = start_y + \Delta y$
0x04	(x, y)	**VLC_OP_LINE**. Draw a line to the given point. $Line(start_x, start_y, x, y)$ $start_x = x$ $start_y = y$
0x05	(□x, □y)	**VLC_OP_LINE_REL**. Draw a line to the given relative point. $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Line(start_x, start_y, x, y)$ $start_x = x$ $start_y = y$
0x06	(cx, cy) (x, y)	**VLC_OP_QUAD**. Draw a quadratic Bezier curve to the given end point using the specified control point. $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
0x07	(□cx, □cy) (□x, □y)	**VLC_OP_QUAD_REL**. Draw a quadratic Bezier curve to the given relative end point using the specified relative control point. $cx = start_x + \Delta cx$ $cy = start_y + \Delta cy$ $x = start_x + \Delta x$ $y = start_y + \Delta y$ $Quad(start_x, start_y, cx, cy, x, y)$ $start_x = x$ $start_y = y$
		VLC_OP_CUBIC. Draw a cubic Bezier curve to the given end point using the specified control points. $Cubic(start_x, start_y, cx_1, cy_1, cx_2, cy_2, x, y)$ $start_x = x$

Parent topic:*Vector path control*

Vector-dased draw operations This part of the API performs the hardware accelerated draw operations.

Draw and gradient enumerations This section provides an overview of draw and gradient enumerations.

vg_lite_blend_t **enumeration** This enumeration is defined under the "Blit enumerations" section (see vg_lite_blend_t enumeration).

Parent topic: Draw and gradient enumerations

vg_lite_color_t **parameter** The common parameter vg_lite_color_t is described in Common parameter types.

Parent topic: Draw and gradient enumerations

vg_lite_fill_t **enumeration** This enumeration is used to specify the fill rule to use. For drawing any path, the hardware supports both non-zero and odd-even fill rules.

To determine whether any point is contained inside an object, imagine drawing a line from that point out to infinity in any direction such that the line does not cross any vertex of the path. For each edge that is crossed by the line, add 1 to the counter if the edge is crossed from left to right, as seen by an observer walking across the line towards infinity, and subtract 1 if the edge crossed from right to left. In this way, each region of the plane will receive an integer value.

The non-zero fill rule says that a point is inside the shape if the resulting sum is not equal to zero. The even/odd rule says that a point is inside the shape if the resulting sum is odd, regardless of sign.

Used in function: vg_lite_render_masklayer.

Used in draw functions: vg_lite_draw, vg_lite_draw_grad, vg_lite_draw_radial_grad, vg_lite_draw_pattern.

vg_lite_fill_t values	string	Description
VG_LITE_FILL_NON_ZEI		Non-zero fill rule. A pixel is drawn if it crosses at least one path pixel.
VG_LITE_FILL_EVEN_OI		Even-odd fill rule. A pixel is drawn if it crosses an odd number of path pixels.

Parent topic: Draw and gradient enumerations

vg_lite_filter_t **enumeration** This enum is defined under the "Blit enumerations" section (see vg_lite_filter_t enumeration).

Parent topic: Draw and gradient enumerations vg_lite_gradient_spreadmode_t **enumeration** vg_lite_gradient_spreadmode_t enum is defined to match OpenVG enum VGColorRampSpreadMode *(from March 2023, replaces*vg_lite_radial_gradient_spreadmode*, requires GC355/GC555 hardware)*

The application may only define stops with offsets between 0 and 1. Spread modes define how the given set of stops are repeated or extended in order to define interpolated color values for arbitrary input values outside the [0,1] range.

Used in structure: vg_lite_radial_gradient_t.

vg_lite_gradient_spreadı String Values	Description
VG_LITE_GRADIENT_S	The current fill color is used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_GRADIENT_S	Colors defined at 0 and 1 are used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_GRADIENT_S	Color values defined between 0 and 1 are repeated indefinitely in both directions.
VG_LITE_GRADIENT_S	Color values defined between 0 and 1 are repeated indefinitely in both directions but with alternate copies of the range reversed.

Parent topic: Draw and gradient enumerations

vg_lite_pattern_mode_t **enumeration** Defines how the region outside the image pattern is filled for the path.

Used in function: vg_lite_draw_gradient, vg_lite_draw_pattern.

vg_lite_pattern_ string values	Description
VG_LITE_PAT	Pixels outside the bounds of the source image should be taken as the color.
VG_LITE_PAT	Pixels outside the bounds of the source image should be taken as having the same color as the closest edge pixel. The color of the pattern border is expanded to fill the region outside the pattern.
VG_LITE_PAT	Pixels outside the bounds of the source image should be repeated indefinitely in all directions. *(from March 2023)*
VG_LITE_PAT	Pixels outside the bounds of the source image should be reflected indefinitely in all directions. *(from March 2023)*

Parent topic: Draw and gradient enumerations

vg_lite_radial_gradient_spreadmode_t **enumeration** *(Deprecated March 2023)* use vg_lite_gradient_spreadmode_t. Defines the radial gradient padding mode. *(from Nov 2020, requires GC355 hardware)*

Used in structure: vg_lite_radial_gradient_t.

vg_lite_radial_gradient_sprea String Values	Description
VG_LITE_RADIAL_GRADI = 0	The current fill color is used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_RADIAL_GRADI	Colors defined at 0 and 1 are used for all stop values less than 0 or greater than 1 respectively.
VG_LITE_RADIAL_GRADI	Color values defined between 0 and 1 are repeated indefinitely in both directions.
VG_LITE_RADIAL_GRADI	Color values defined between 0 and 1 are repeated indefinitely in both directions but with alternate copies of the range reversed.

Parent topic: Draw and gradient enumerations

Parent topic:*Vector-dased draw operations*

Draw and gradient structures This section provides an overview of the draw and gradient structures.

vg_lite_buffer_t **structure** This structure is defined under the "Pixel buffer structures" section (see vg_lite_buffer_t structure).

Parent topic: Draw and gradient structures

vg_lite_color_ramp_t **structure** This structure defines the stops for the radial gradient. The five parameters provide the offset and color for the stop. Each stop is defined by a set of floating point values which specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255].*(from November 2020, requires GC355 hardware)*

The define for the max number of radial gradient stops is #define MAX_COLOR_RAMP_STOPS256.

Used in radial gradient structure: vg_lite_radial_gradient_t.

vg_lite_color_ramp_t members	Type	Description
stop	vg_lite_float_t	Offset value for the color stop
red	vg_lite_float_t	Red color channel value for the color stop
green	vg_lite_float_t	Green color channel value for the color stop
blue	vg_lite_float_t	Blue color channel value for the color stop
alpha	vg_lite_float_t	Alpha color channel value for the color stop

Parent topic: Draw and gradient structures

vg_lite_linear_gradient_t **structure** This structure defines the organization of a linear gradient in VGLite data. The linear gradient is applied to filling a path. It generates a 256x1 image according to the specified settings.

Used in init and draw functions: vg_lite_init_grad, vg_lite_set_grad, vg_lite_update_grad, vg_lite_get_grad_matrix, vg_lite_clear_grad, vg_lite_draw_grad.

vg_lite_linear_gradient_t constants	Type	Description
VLC_MAX_GRADIENT_STOP	vg_lite_int32_	Constant. Maximum number of gradient colors = 16.
vg_lite_linear_gradient_t members		
colors [VLC_MAX_GRADIENT_STOF	vg_lite_uint32	Color array for the gradient
count	vg_lite_uint32	Number of colors
stops [VLC_MAX_GRADIENT_STOF	vg_lite_uint32	Number of color stops, from 0 to 255
matrix	vg_lite_matrix	Struct for the matrix to transform the gradient color ramp
image	vg_lite_buffer_	Image object struct to represent the color ramp

Parent topic: Draw and gradient structures

vg_lite_ext_linear_gradient **structure** This structure defines the organization of the extended parameters possible for a linear gradient *(from April 2022)*.

Used in functions: vg_lite_draw_linear_grad.

vg_lite_ext_linear_gradient members	Type	Description
count	vg_lite_uint32_t	Count of colors, up to 256.
matrix	vg_lite_matrix_t	The matrix to transform the gradient.
image	vg_lite_buffer_t	The image for rendering as gradient pattern.
linear_grad	vg_lite_linear_gra	Linear gradient parameters. Includes center point, focal point and radius.
ramp_length	vg_lite_uint32_t	Color ramp length for gradient paints provided to the driver.
color_ramp[VLC_MAX_C(	vg_lite_color_ram	Color ramp parameter for gradient paints provided to the driver.
converted_length	vg_lite_uint32_t	Converted internal color ramp length.
converted_ramp[VLC_MAX_(	vg_lite_color_ram	Converted internal color ramp.
pre-multiplied	vg_lite_uint8_t	If this value is set to 1, the color value of color_ramp will be multiplied by the alpha value of color_ramp.
spread_mode	vg_lite_radial_gra	The spread mode that is applied to the pixels out of the image after transformed.

|

Parent topic: Draw and gradient structures

vg_lite_linear_gradient_parameter **structure** This structure defines a radial direction for a linear gradient. *(from April 2022)*

Line0 connects point (X0, Y0) to point (X1, Y1) and represents the radial direction of the linear gradient.

Line1 is a line perpendicular to line0 which passes through point (X0, Y0).

Line2 is a line perpendicular to line0 which passes through point (X1, Y1)

The linear gradient paint is applied at the intersection of the path fill area and the plane starting from line 1 and ending at line 2.

Used in structure: vg_lite_ext_linear_gradient.

Used in functions: vg_lite_set_linear_grad.

vg_lite_linear_gradient_parameter_t members	Type	Description
X0	vg_lite_float_	X origin of linear gradient radial direction.
Y0	vg_lite_float_	Y origin of linear gradient radial direction.
X1	vg_lite_float_	X end point of linear gradient radial direction.
Y1	vg_lite_float_	Y end point of linear gradient radial direction.

Parent topic: Draw and gradient structures

vg_lite_matrix_t structure This structure is defined under the "Matrix control structures" section (see vg_lite_matrix_t structure).

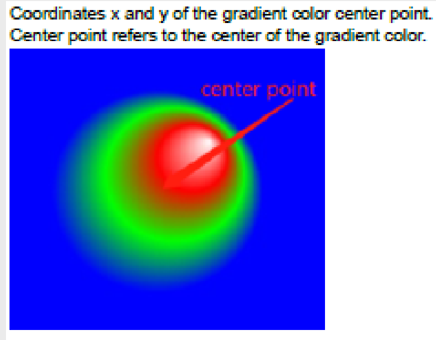
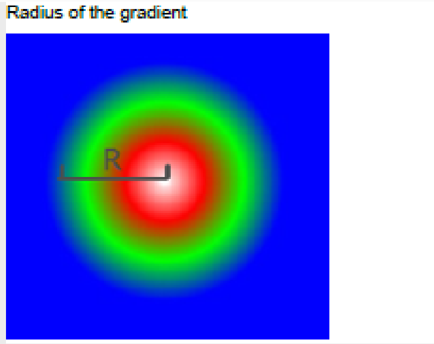
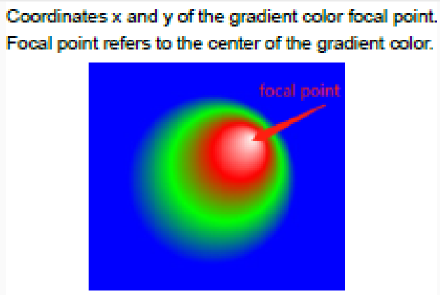
Parent topic: Draw and gradient structures

vg_lite_path_t structure This structure is defined under the "Vector path structures" section (see vg_lite_path_t structure).

Parent topic: Draw and gradient structures

vg_lite_radial_gradient_parameter_t structure This structure defines the gradient radius and the X and Y coordinates for the center and focal points of the gradient *(from November 2020, requires GC355 or GC555 hardware)*.

Used in radial gradient structure: vg_lite_radial_gradient_t.

vg_lite_radial_gradient_parameter_t member	Type	Description
cx	vg_lite_float_t	
cy	vg_lite_float_t	
r	vg_lite_float_t	
fx	vg_lite_float_t	
fy	vg_lite_float_t	

Parent topic: Draw and gradient structures

vg_lite_radial_gradient_t **structure** This structure defines the application of the radial gradient to fill a path. *(from November 2020, requires GC355 or GC555 hardware).*

Used in radial gradient functions: vg_lite_draw_grad, vg_lite_set_radial_grad, vg_lite_update_radial_grad, vg_lite_get_radial_grad, vg_lite_clear_radial_grad.

vg_lite_radial_gradient_t member	Type	Description
count	vg_lite_uint32_t	Count of colors, up to 256
matrix	vg_lite_matrix_t	Structure that specifies the transform matrix for the gradient
image	vg_lite_buffer_t	Structure that specifies the image for rendering as a gradient pattern
radial_grad	vg_lite_radial_grad	Structure that specifies the location of the gradient's center point (cx, cy), focal point(fx, fy) and radius(r)
ramp_length	vg_lite_uint32_t	Color ramp parameters for gradient paints provided to the driver
color_ramp[VLC_MAX_C	vg_lite_color_ram	Structure that specifies the color ramp
converted_length	vg_lite_uint32_t	Converted internal color ramp.
converted_ramp[VLC_MAX_(	vg_lite_color_ram	Structure that specifies the internal color ramp
pre_multiplied	vg_lite_uint32_t	If this value is set to 1, the color value of color_ramp will be multiplied by the alpha value of color_ramp.
spread_mode	vg_lite_radial_grad	Enum that specifies the tiling mode, which is applied to the pixels out of the image after transformation

Parent topic: Draw and gradient structures

Parent topic:*Vector-dased draw operations*

Draw functions This section provides an overview of the draw functions.

vg_lite_draw **function** **Description:**

This function performs a hardware accelerated 2D vector draw operation.

The size of the tessellation buffer can be specified at initialization and it is aligned with the minimum hardware alignment requirements of the kernel. Specifying a smaller size for tessellation buffer allocates less memory but reduces performance. Because the hardware walks the target with the provided tessellation window size, a path may be sent to the hardware multiple times. It is a good practice to set the tessellation buffer size to the most common path size. For example, if all you do is render up to 24-point fonts, you can set the tessellation buffer to 24x24.

Note:

- All the color formats available in the vg_lite_buffer_format_t enum are supported as the destination buffer for the draw function
- The hardware does not support strokes; they must be converted to paths before you use them in the draw API

Syntax:

```
vg_lite_error_t vg_lite_draw (
      vg_lite_buffer_t          *target,
      vg_lite_path_t            *path,
      vg_lite_fill_t           fill_rule,
      vg_lite_matrix_t          *matrix,
      vg_lite_blend_t           blend,
      vg_lite_color_t           color
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for the draw function.
*patl	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw. See opcode details in Vector path opcodes for plotting paths.
fill_ı	Specifies the vg_lite_fill_t enum value for the fill rule for the path
*matrix	Pointer to a vg_lite_matrix_t structure that defines the *affine* transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed. **Note:** Non-affine transformations are not supported by vg_lite_draw; therefore, a perspective transformation matrix might have unexpected effects on path rendering.
blenc	Select one of the hardware-supported blend modes in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
color	The color applied to each pixel drawn by the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

vg_lite_draw_grad function

Description:

This function is used to fill a path with a linear gradient according to the specified fill rules. The specified path is transformed according to the selected matrix and is filled with the specified color gradient.

Syntax:

```
vg_lite_error_t vg_lite_draw_grad (
    vg_lite_buffer_t            *target,
    vg_lite_path_t              *path,
    vg_lite_fill_t              fill_rule,
    vg_lite_matrix_t            *matrix,
    vg_lite_linear_gradient_t   *grad,
    vg_lite_blend_t             blend
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw and fill with the linear gradient. See opcode details in Vector path opcodes for plotting paths.
fill_ru	Specifies the vg_lite_fill_t enum value for the fill rule for the path
*matrix	Pointer to the vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed; however, this option is not preferable.
*grad	Pointer to the vg_lite_linear_gradient_t structure that contains the values to be used to fill the path.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

vg_lite_draw_radial_grad function

Description:

This function is used to fill a path with a radial gradient according to the specified fill rules. The specified path is transformed according to the selected matrix and is filled with the radial color gradient. The application can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_RADIAL_GRADIENT) to determine HW support for radial gradient.

Syntax:

```
vg_lite_error_t vg_lite_draw_radial_grad (
    vg_lite_buffer_t              *target,
    vg_lite_path_t                *path,
    vg_lite_fill_t                fill_rule,
    vg_lite_matrix_t              *path_matrix,
    vg_lite_radial_gradient_t    *grad,
    vg_lite_color_t               paint_color,
    vg_lite_blend_t               blend,
    vg_lite_filter_t              filter
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw for and fill with the radial gradient. See opcode details in Vector path opcodes for plotting paths.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path
*path	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed; however, his option is not preferable.
*grad	Pointer to the vg_lite_radial_gradient_t structure that contains the values to be used to fill the path. **Note:** grad->image.image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE .
paint_	Specifies the paint color vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL set by the function vg_lite_set_radial_grad. When pixels are out of the image after transformation, paint_color is applied to them. For details, see vg_lite_radial_gradient_spreadmode_t.
blend	Specifies the blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

vg_lite_draw_pattern function

Description:

This function fills a path with an image pattern. The path is transformed according to the specified matrix and is filled with the transformed image pattern.

Syntax:

```
vg_lite_error_t vg_lite_draw_pattern (
    vg_lite_buffer_t            *target,
    vg_lite_path_t              *path,
    vg_lite_fill_t              fill_rule,
    vg_lite_matrix_t            *path_matrix,
    vg_lite_buffer_t            *pattern_image,
    vg_lite_matrix_t            *pattern_matrix,
    vg_lite_blend_t             blend,
    vg_lite_pattern_mode_t      pattern_mode,
    vg_lite_color_t             pattern_color,
    vg_lite_color_t             color,
    vg_lite_filter_t            filter
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure for the destination buffer. All color formats available in the vg_lite_buffer_format_t enum are valid destination formats for this draw function.
*patl	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw. See opcode details in Vector path opcodes for plotting paths
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*patl	Pointer to the vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the source pixels into the target. If the matrix is NULL, an identity matrix is assumed, meaning the source is copied directly onto the target at 0,0 location.
*pattern_	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed.
*pattern_	Pointer to the vg_lite_buffer_t structure that describes the source of the image pattern
	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the source pixels into the target. If the matrix is NULL, an identity matrix is assumed, which means that the source is copied directly at 0,0 location on the target.
blenc	Specifies one of the vg_lite_blend_t enum values for hardware-supported blend modes to be applied to each drawn pixel in the image. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
pattern_	Specifies the vg_lite_pattern_mode_t value that defines how the region outside the image pattern is to be filled.
pattern_	Specifies a 32bpp ARGB color (vg_lite_color_t) to be applied to the fill outside the image pattern area when the pattern_mode value is VG_LITE_PATTERN_COLOR. *(from Dec 2019, type now vg_lite_color_t, previously was uint32_t)*
color	Specifies a 32bpp ARGB color (vg_lite_color_t) to be applied as a mix color. If non-zero, the mix color value gets multiplied with each source pixel before blending happens. If a mix color is not needed, set the color parameter to 0 *(from May 2023)*. **Note:** This parameter has no effect if the pattern image vg_lite_buffer_t structure member image_mode is set to VG_LITE_ZERO or VG_LITE_NORMAL_IMAGE_MODE.
filter	Specifies the filter type. All formats available in the vg_lite_filter_t enum are valid formats for this function. A value of zero (0) indicates VG_LITE_FILTER_POINT.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Draw functions

Parent topic:*Vector-dased draw operations*

Linear gradient initialization and control functions This part of the API performs linear gradient operations.

A color gradient (color progression, color ramp) is a smooth transition between a set of colors (color stops) that is done along a line (linear, or axial color gradient) or radially, along concentric circles (radial color gradient). The color transition is done by linear interpolation between two consecutive color stops.

Note: VGLite supports linear color gradients for GCNanoLiteV and GCNanoUltraV. Both linear and radial gradients are supported with GC355 and GC555.

vg_lite_init_grad **function** **Description:**

This function initializes the internal buffer for the linear gradient object with default settings for rendering.

Syntax:

```
vg_lite_error_t vg_lite_init_grad (
    vg_lite_linear_gradient_t *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure, which defines the gradient to be initialized. Default values are used.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

vg_lite_clear_grad function

Description:

This function is used to clear the values of a linear gradient object and free up the memory of the image buffer.

Syntax:

```
vg_lite_error_t vg_lite_clear_grad (
    vg_lite_linear_gradient_t       *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure that is to be cleared

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

vg_lite_set_grad function

Description:

This function is used to set values for the members of the vg_lite_linear_gradient_t structure.

Note: The vg_lite_set_grad API adopts the following rules to set the default gradient colors if the input parameters are incomplete or invalid:

- If no valid stops have been specified (for example, due to an empty input array, out-of-range or out-of-order stops), a stop at 0 with (R, G, B, A) color (0.0, 0.0, 0.0, 1.0) (opaque black) and a stop at 1 with color (1.0, 1.0, 1.0, 1.0) (opaque white) are implicitly defined
- If at least one valid stop has been specified, but none has been defined with an offset of 0, then an implicit stop is added with an offset of 0 and the same color as the first user-defined stop

- If at least one valid stop has been specified, but none has been defined with an offset of 1, then an implicit stop is added with an offset of 1 and the same color as the last user-defined stop

Syntax:

```
vg_lite_error_t vg_lite_set_grad (
      vg_lite_linear_gradient_t       *grad,
      uint32_t                        count,
      uint32_t                        *colors,
      uint32_t                        *stops
     );
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure to be set
count	The number of colors in the linear gradient. The maximum color stop count is defined by VLC_MAX_GRAD which is 16.
*colors	Specifies the color array for the gradient stops. The color is in ARGB8888 format with alpha in the upper byte.
*stops	Pointer to the gradient stop offset

Returns:

Always returns VG_LITE_SUCCESS.

Parent topic: Linear gradient initialization and control functions

vg_lite_get_grad_matrix function

Description:

This function is used to get a pointer to the transformation matrix of the gradient object. It allows an application to manipulate the matrix to facilitate correct rendering of the gradient path.

Syntax:

```
vg_lite_error_t vg_lite_get_grad_matrix (
      vg_lite_linear_gradient_t        *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure, which contains the matrix to be retrieved

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

vg_lite_update_grad function

Description:

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_linear_gradient_t object has an image buffer, which is used to render the gradient pattern. The image buffer is created or updated with the corresponding gradient parameters.

Syntax:

```
vg_lite_error_t vg_lite_update_grad (
      vg_lite_linear_gradient_t      *grad
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_t structure, which contains the update values to be used for the object to be rendered

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient initialization and control functions

Parent topic:*Vector-dased draw operations*

Linear gradient extended functions

The following functions are available only with IP that includes hardware support for extended linear gradient capabilities, such as GC355 and GC555. These functions are not available with GCNanoLiteV, GCNanoUltraV, or GCNanoV. Applications can use VGLite API vg_lite_query_feature (gcFEATURE_BIT_VG_LINEAR_GRADIENT_EXT) to determine HW support for linear gradient.

vg_lite_set_linear_grad function

Description:

This function is used to set the values that define the linear gradient. *(from April 2022)*

Syntax:

```
vg_lite_error_t vg_lite_set_linear_grad (
    vg_lite_ext_linear_gradient_t            *grad,
    vg_lite_uint32_t                         count,
    vg_lite_color_ramp_t                     *color_ramp,
    vg_lite_linear_gradient_parameter_t      grad_param,
    vg_lite_radial_gradient_spreadmode_t     spread_mode,
    vg_lite_uint8_t                          pre_mult
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure that is to be set.
count	Count of the colors in the gradient. The maximum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is set to 256.
*colors	It is the array of stops for the linear gradient. The number of parameters for each stop is 5, and gives the offset and color of the stop. Each stop is defined by a floating-point *offset* value and four floating-point values containing the sRGBA color and alpha value associated with each stop, in the form of a non-premultiplied (R, G, B, alpha) quad. The range of all parameters is [0,1].
grad_	Gradient parameters as specified in the structure vg_lite_linear_gradient_parameter_t.
sprea	The fill mode is applied to the pixels out of the paint after transformation. Uses the same spread mode enumeration types as radial gradient. For details, see vg_lite_radial_gradient_spreadmode_t enum.
pre_	This parameter controls whether color and alpha values are interpolated in premultiplied or non-premultiplied form.

Returns:

Returns VG_LITE_INVALID_ARGUMENTS to indicate the parameters are wrong.

Parent topic: Linear gradient extended functions

vg_lite_get_linear_grad_matrix function **Description:**

This function returns a pointer to an extended linear gradient object's matrix.*(from March 2023)*.

Syntax:

```
vg_lite_matrix_t* vg_lite_get_linear_grad_matrix (
    vg_lite_ext_linear_gradient_t    *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure.

Returns:

Returns a pointer to vg_lite_matrix_t for the specified extended linear gradient.

Parent topic: Linear gradient extended functions

vg_lite_draw_linear_grad function **Description:**

This function returns a pointer to an extended linear gradient object's matrix.*(from March 2023)*.

Syntax:

```
vg_lite_error_t vg_lite_draw_linear_grad (
    vg_lite_buffer_t                 *target,
    vg_lite_path_t                   *path,
    vg_lite_fill_t                  fill_rule,
```

(continues on next page)

(continued from previous page)

```
    vg_lite_matrix_t                *path_matrix,
    vg_lite_ext_linear_gradient_t   *grad,
    vg_lite_color_t                 paint_color,
    vg_lite_blend_t                 blend,
    vg_lite_filter_t                filter
);
```

Parameters:

Parameter	Description
*target	Pointer to the vg_lite_buffer_t structure containing data describing the target path.
*path	Pointer to the vg_lite_path_t structure containing path data that describes the path to draw for the linear gradient. Refer to Vector path opcodes for plotting paths in this document for opcode detail.
fill_r	Specifies the vg_lite_fill_t enum value for the fill rule for the path.
*path	Pointer to a vg_lite_matrix_t structure that defines the 3x3 transformation matrix of the path. If the matrix is NULL, an identity matrix is assumed; however, this option is not preferable.
*grad	Pointer to the vg_lite_ext_linear_gradient_t structure that contains the values to be used to fill the path. **Note:** grad->image.image_mode does not support VG_LITE_MULTIPLY_IMAGE_MODE.
paint	Specifies the paint color vg_lite_color_t RGBA value to be applied by VG_LITE_RADIAL_GRADIENT_SPREAD_FILL, set by function vg_lite_set_linear_grad. When pixels are out of the image after transformation, this paint_color is applied to them. For details, see enum vg_lite_radial_gradient_spreadmode_t.
blend	Specifies blend mode in the vg_lite_blend_t enum to be applied to each drawn pixel. If no blending is required, set this value to VG_LITE_BLEND_NONE (0).
filter	Specified the filter mode vg_lite_filter_t enum value to be applied to each drawn pixel. If no filtering is required, set this value to VG_LITE_BLEND_POINT (0).

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient extended functions

vg_lite_update_linear_grad function

Description:

This function is used to update or generate the corresponding image object to render *(from April 2022)*.

The vg_lite_ext_linear_gradient_t object has an image buffer that is used to render the linear gradient paint. The image buffer is created/updated according to the specified grad parameters.

Syntax:

```
vg_lite_error_t vg_lite_update_linear_grad (
    vg_lite_ext_linear_gradient_t   *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_ext_t structure that is to be updated or created.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient extended functions

vg_lite_clear_linear_grad function

Description:

This function is used to clear the linear gradient object. This resets the grad members and free the image buffer's memory *(from April 2022)*.

Syntax:

```
vg_lite_error_t vg_lite_clear_linear_grad (
    vg_lite_ext_linear_gradient_t     *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_linear_gradient_ext_t structure that is to be cleared.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Linear gradient extended functions

Parent topic:*Vector-dased draw operations*

Radial gradient functions initialization and control functions

The following functions are available only with IP that supports radial gradients, such as GC355 and GC555. These functions are not available with GCNanoLiteV, or GCNanoUltraV or GCNanoV.

Note: There is no init function required for radial gradients. Buffer initialization is done through the vg_lite_update_radial_grad function. *(from Nov 2020, requires GC355 or GC555 hardware)*

vg_lite_set_radial_grad function

Description:

This function is used to set the values for the radial linear gradient definition. *(from November 2020, requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_set_radial_grad (
    vg_lite_radial_gradient_t                 *grad,
    vg_lite_uint32_t                          count,
    vg_lite_color_ramp_t                      *color_ramp,
    vg_lite_radial_gradient_parameter_t       grad_param,
    vg_lite_radial_gradient_spreadmode_t      spread_mode,
    vg_lite_uint8_t                           pre_mult
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure for the radial gradient that has to be set
coun	The number of color stops in the gradient. The maximum color stop count is defined by MAX_COLOR_RAMP_STOPS, which is currently 256.
*colc	Pointer to the vg_lite_color_ramp_t structure that defines the stops for the radial gradient. The five parameters provide the offset and color for each stop. Each stop is defined by a set of floating point values that specify the offset and the sRGBA color and alpha values. Color channel values are in the form of a non-premultiplied (R, G, B, alpha) quad. All parameters are in the range of [0,1]. The red, green, blue, alpha value of [0, 1] is mapped to an 8-bit pixel value [0, 255].
grad_	The radial gradient parameters are supplied as a vector of 5 floats. Parameters (cx, cy) specify the center point, parameters (fx, fy) specify the focal point, and r specifies the radius. See structure vg_lite_radial_gradient_parameter_t.
sprea	The tiling mode that is applied to pixels out of the paint after transformation. See enum vg_lite_radial_gradient_spreadmode_t.
pre_	Controls whether color and alpha values are interpolated in premultiplied or non-premultiplied form. If this value is set to 1, the color value of vgColorRamp is multipled by the alpha value of vgColorRamp.

Returns:

Returns VG_LITE_INVALID_ARGUMENTS to indicate that the parameters are wrong.

Parent topic: Radial gradient functions initialization and control functions

vg_lite_update_radial_grad function

Description:

This function is used to update or generate values for an image object that is going to be rendered. The vg_lite_radial_gradient_t object has an image buffer that is used to render the gradient pattern. The image buffer will be created or updated with the corresponding gradient parameters. *(from November 2020, requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_update_radial_grad (
    vg_lite_radial_gradient_t    *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure, which contains the updated values to be used for the object to be rendered

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Radial gradient functions initialization and control functions

vg_lite_get_radial_grad_matrix function **Description:**

This function is used to get a pointer to the radial gradient object's transformation matrix. This allows an application to manipulate the matrix to facilitate correct rendering of the gradient path*. (from Nov 2020, requires GC355 or GC555 hardware).*

Syntax:

```
vg_lite_error_t vg_lite_get_radial_grad_matrix (
    vg_lite_radial_gradient_t     *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure, which contains the matrix to be retrieved

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Radial gradient functions initialization and control functions

vg_lite_clear_rad_grad function **Description:**

This function is used to clear the values of a radial gradient object and free the image buffer's memory*. (from Nov 2020, requires GC355 or GC555 hardware)*

Syntax:

```
vg_lite_error_t vg_lite_clear_radial_grad (
    vg_lite_radial_gradient_t    *grad,
);
```

Parameters:

Parameter	Description
*grad	Pointer to the vg_lite_radial_gradient_t structure which is to be cleared

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Radial gradient functions initialization and control functions

Parent topic:*Vector-dased draw operations*

Stroke operations

This part of the API performs stroke operations. *(from March 2022)*

Stroke enumerations

This section gives details on stroke enumerations.

vg_lite_cap_style_t **enumeration** Defines the style of cap at the end of a stroke *(from March 2022)*.

Used in structure: vg_lite_stroke_t.

Used in function: vg_lite_set_stroke.

vg_lite_cap values	Description
VG_LITE_	The *butt* end cap style terminates each segment with a line perpendicular to the tangent at each endpoint.
VG_LITE_	The *round* end cap style appends a semicircle with a diameter equal to the line width centered around each endpoint.
VG_LITE_	The *square* end cap style appends a rectangle with two sides of length equal to the line width perpendicular to the tangent, and two sides of length equal to half the line width parallel to the tangent, at each endpoint.

Parent topic: Stroke enumerations

vg_lite_path_type_t **enumeration** Defines the type of draw path *(from March 2022)*.

Used in structure: vg_lite_path_t, vg_lite_stroke_t.

Used in function: vg_lite_set_path_type.

vg_lite_path_type_t string values	Description
VG_LITE_DRAW_FILL_PATH	Draw path is fill.
VG_LITE_DRAW_STROKE_PATH	Draw path is stroke.
VG_LITE_DRAW_FILL_STROKE_PATH	Draw path is both fill and stroke.

Parent topic: Stroke enumerations

vg_lite_join_style_t **enumeration** Defines the type of styles available for line joints. *(from March 2022)*

Used in structure: vg_lite_stroke_t.

Used in function: vg_lite_set_stroke.

vg_lite_joi string values	Description
VG_LITE_	The *miter* join style appends a trapezoid with one vertex at the intersection point of the two original lines, two adjacent vertices at the outer endpoints of the two “thickened” lines and a fourth vertex at the extrapolated intersection point of the outer perimeters of the two “thickened” lines.
VG_LITE_	The *round* join style appends a wedge-shaped portion of a circle, centered at the intersection point of the two original lines, having a radius equal to half the line width.
VG_LITE_	The *bevel* type join style appends a triangle with two vertices at the outer endpoints of the two “thickened” lines and a third vertex at the intersection point of the two original lines.

Parent topic: Stroke enumerations

Parent topic:*Stroke operations*

Stroke structures This section gives details on stroke structures.

vg_lite_path_t **structure** Defined under Vector Path Structures - vg_lite_path_t structure.

(additional members added for stroke from March 2022)

Parent topic: Stroke structures

vg_lite_path_list_t **structure** The structure vg_lite_path_list_ptr points to the vg_lite_path_list structure that provides divided path data according to MOVE/MOVE_REL. *(from Aug 2023)*

Used (vg_lite_path_list_ptr) in structures: vg_lite_stroke_t.

vg_lite_path_list_t members	Type	Description
path_points	vg_lite_path_point_ptr	
path_end	vg_lite_path_point_ptr	
point_count	vg_lite_uint32_t	
next	vg_lite_path_list_ptr	
closed	vg_lite_uint8_t	

Parent topic: Stroke structures

vg_lite_path_point_t **structure** The structure vg_lite_path_point_ptr points to the vg_lite_path_point structure which provides path detail *(from March 2022)*

Used (vg_lite_path_point_ptr) in structures: vg_lite_path_point_t, vg_lite_stroke_conversion. vg_lite_sub_path_t.

vg_lite_path_point_t members	Type	Description
x	vg_lite_float_t	X coordinate
y	vg_lite_float_t	Y coordinate
flatten_flag	vg_lite_uint8_t	Flatten flag for flattened path
curve_type	vg_lite_uint8_t	Curve type for the stroke path
tangentX	vg_lite_float_t	X tangent (Note: #define centerX tangent)
tangentY	vg_lite_float_t	Y tangent (Note: #define centerX tangent)
length	vg_lite_float_t	Line length
prev	vg_lite_path_point_pt	Pointer to the previous point node

Parent topic: Stroke structures

vg_lite_stroke_t **structure** The structure provides stroke parameters and pointers to temp storage for a stroke sub path. Refer to the function vg_lite_set_stroke parameter descriptions for additional description for some members. *(from March 2022)*

Used in structure: vg_lite_path_t.

vg_lite_stroke_t members	Type	Description
cap_style	vg_lite_cap_style_t	Stroke cap style
join_style	vg_lite_join_style_t	Stroke joint style
line_width	vg_lite_float_t	Stroke line width
miter_limit	vg_lite_float_t	Stroke miter limit
*dash_pattern	vg_lite_float_t	Pointer to stroke dash pattern
pattern_count	vg_lite_uint32_t	Number of dash pattern repetitions
dash_phase	vg_lite_float_t	Stroke dash phrase
dash_length	vg_lite_float_t	Stroke dash initial length
dash_index	vg_lite_uint32_t	Stroke dash initial index
half_width	vg_lite_float_t	Half line width
pattern_length	vg_lite_float_t	Total length of stroke dash patterns.
miter_square	vg_lite_float_t	For fast checking
path_points	vg_lite_path_point_ptr	Temp storage for stroke sub path
path_end	vg_lite_path_point_ptr	Temp storage for stroke sub path
point_count	unint32_t	Temp storage for stroke sub path
left_point	vg_lite_path_point_ptr	Temp storage for stroke sub path
right_pont	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_points	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_end	vg_lite_path_point_ptr	Temp storage for stroke sub path
stroke_count	vg_lite_uint32_t	Temp storage for stroke sub path
path_list_divide	vg_lite_path_list_ptr	Divide stroke path according to move or move_rel for
cur_list	vg_lite_path_list_ptr	Pointer to current divided path data. *(from Aug 2023)*
add_end	vg_lite_uint8_t	Flag that adds end_path in driver *(from Aug 2023)*
dash_reset	vg_lite_uint8_t	*(from Aug 2023)*
stroke_paths	vg_lite_sub_path_ptr	
last_stroke	vg_lite_sub_path_ptr	
swing_handling	vg_lite_uint32_t	
swing_deltax	vg_lite_float_t	
swing_deltay	vg_lite_float_t	
swing_start	vg_lite_path_point_ptr	
swing_stroke	vg_lite_path_point_ptr	
swing_length	vg_lite_float_t	
swing_centlen	vg_lite_float_t	
swing_count	vg_lite_uint32_t	
need_swing	vg_lite_uint8_t	
swing_ccw	vg_lite_uint8_t	
stroke_length	vg_lite_float_t	
stroke_size	vg_lite_uint32_t	
fattened	vg_lite_uint8_t	The stroke line is a fat line.
closed	vg_lite_uint8_t	

Parent topic: Stroke structures

vg_lite_sub_path_t **structure** The structure vg_lite_sub_path_ptr points to the vg_lite_sub_path structure that provides sub path detail and a pointer to the next sub path. *(from March 2022)*

Used in structure: vg_lite_stroke_conversion.

vg_lite_path_point_t members	Type	Description
next	vg_lite_sub_path_ptr	Pointer to the next sub path
point_count	vg_lite_uint32_t	Number of points in the sub path
point_list	vg_lite_path_point_pt	Pointer to the point list.
end_point	vg_lite_path_point_pt	Pointer to the last point.
closed	vg_lite_uint8_t	Indicates whether or not the path is closed.
length	vg_lite_float_t	Length of the sub path.

Parent topic: Stroke structures

Parent topic:*Stroke operations*

Stroke functions

All return vg_lite_error_t status.

vg_lite_set_path_type function

Description:

This function sets the path type*. (from March 2022)*

Syntax:

```
vg_lite_error_t vg_lite_set_path_type (
    vg_lite_path_t              *path,
    vg_lite_path_type_t          path_type
);
```

Parameters:

Parameter	Description
*path	Pointer to the **vg_lite_path_t** structure that describes the vector path.
path_type	Pointer to a vg_lite_path_type_t structure that describes the path type.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Stroke functions

vg_lite_set_stroke function

Description:

This function uses input parameters to set stroke attributes *(from March 2022)*.

Syntax:

```
vg_lite_error_t vg_lite_set_stroke (
    vg_lite_path_t              *path,
    vg_lite_cap_style_t          cap_style,
    vg_lite_join_style_t         join_style,
    vg_lite_float_t             line_width,
    vg_lite_float_t             miter_limit,
    vg_lite_float_t             *dash_pattern,
    vg_lite_uint32_t             pattern_count,
    vg_lite_float_t             dash_phase,
    vg_lite_color_t              color
```

(continues on next page)

(continued from previous page)

```
);
```

Parameters:

Parameter	Description
*patl	Pointer to the vg_lite_path_t structure that describes the path.
cap_	The end cap style is defined by the vg_lite_cap_style_t enum.
join_	The line join style defined by the vg_lite_join_style_t enum.
line_	The line width of the stroke path. A line width less than or equal to 0 prevents stroking from taking place.
miter	When stroking using the Miter stroke vg_lite_join_style_t, the miter length (that is, the length between the intersection points of the inner and outer perimeters of the two "fattened" lines) is compared to the product of the user-set miter limit and the line width. If the miter length exceeds this product, the Miter join is not drawn and a Bevel join is substituted. **Note**: Miter limit values less than 1 are silently clamped to 1.
*dasl	Pointer to a dash pattern that consists of a sequence of lengths of alternating "on" and "off" dash segments. The first value of the dash array defines the length, in user coordinates, of the first "on" dash segment. The second value defines the length of the following "off" segment. Each subsequent pair of values defines one "on" and one "off" segment. **Note**: If the dash pattern has an odd number of elements, the final element is ignored.
pattern_	The count of dash on/off segments.
dash_	Defines the starting point in the dash pattern that is associated with the start of the first segment of the path. For example, if the dash pattern is [10 20 30 40] and the dash phase is 35, the path is stroked with an "on" segment of length 25 (skipping the first "on" segment of length 10, the following "off" segment of length 20, and the first 5 units of the next "on" segment), followed by an "off" segment of length 40. The pattern is then repeated from the beginning, with an "on" segment of length 10, an "off" segment of length 20, an "on" segment of length 30.
color	The stroke color.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Stroke functions

vg_lite_update_stroke **function** **Description:**

This function uses the path and stroke attributes as specified with the function vg_lite_set_stroke to update the stroke path's parameters and generate stroke path data . *(from March 2022)*

Syntax:

```
vg_lite_error_t vg_lite_update_stroke (
    vg_lite_path_t              *path,
);
```

Parameters:

Parameter	Description
*path	Pointer to the **vg_lite_path_t** structure that describes the path.

Returns:

Returns VG_LITE_SUCCESS if successful. See vg_lite_error_t enum for other return codes.

Parent topic: Stroke functions

Parent topic:*Stroke operations*

Deprecated and renamed APIs The following functions are deprecated and are either obsolete or replaced by a more efficient implementation. Their use is discouraged and will produce unpredictable behaviors.

The names of some functions, enums and structures were modified during code refinements in 2022Q3. If the parameters did not change, the deprecated syntax detail is not provided below. Changes to enums and structs are not mentioned here, instead refer to the item itself.

Deprecated or renamed API	Recommended replacement API	Source file	Date deprecated
vg_lite_perspective	n/a	vg_lite.h	August 2022
vg_lite_set_dither	vg_lite_enable_dither vg_lite_disable_dither	vg_lite.h	August 2022
vg_lite_append_path	vg_lite_path_append	vg_lite.h	Sept 2022
vg_lite_path_calc_length	vg_lite_get_path_length	vg_lite.h	Sept 2022
vg_lite_set_image_global_alpha	vg_lite_set_source_global_alpha	vg_lite.h	Sept 2022
vg_lite_dest_global_alpha	vg_lite_set_dest_global_alpha	vg_lite.h	Sept 2022
vg_lite_mem_avail	vg_lite_get_mem_size	vg_lite.h	Sept 2022
vg_lite_enable_premultiply	n/a	vg_lite.h	Dec 2022
vg_lite_disable_premultiply	n/a	vg_lite.h	Dec 2022
vg_lite_set_premultiply	n/a	vg_lite.h	Aug 2023
vg_lite_radial_gradient_spreadmo enum	vg_lite_gradient_spreadmode_t enum	vg_lite.h	March 2023
API Name Refinement	***(no change to parameters)***		
vg_lite_buffer_upload	vg_lite_upload_buffer_	vg_lite.h	Sept 2022
vg_lite_*mask*	*most vg_lite_*mask_layer*	vg_lite.h	Sept 2022
vg_lite_*_grad	vg_lite_*_gradient *(parameters unchanged)*	vg_lite.h	Sept 2022
vg_lite_*_radial_grad*	vg_lite_*_rad_grad*	vg_lite.h	Sept 2022
vg_lite_buffer_image_mode_t	vg_lite_image_mode_t	vg_lite.h	Sept 2022
vg_lite_transparency_mode_t	vg_lite_transparency_t	vg_lite.h	Sept 2022
vg_lite_set_update_stroke	vg_lite_update_stroke	vg_lite.h	Sept 2022
vg_lite_set_draw_path_type	vg_lite_set_path_type	vg_lite.h	Sept 2022

Deprecated vg_lite syntax Syntax for deprecated functions is provided below for reference.

Note: This list does not include items renamed during code refinement of Sept 2022.

vg_lite_perspective ***(deprecated)*** **Syntax:**

```
void vg_lite_perspective (
    vg_lite_float_t            px,
    vg_lite_float_t            py,
    vg_lite_matrix_t           *matrix
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_set_dither ***(deprecated)*** **Syntax:**

```
vg_lite_error_t vg_lite_set_dither (
    int               enable
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_enable_premultiply ***(deprecated)*** **Syntax:**

```
vg_lite_error_t vg_lite_enable_premultiply (
    void
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_disable_premultiply ***(deprecated)*** **Syntax:**

```
vg_lite_error_t vg_lite_disable_premultiply (
    void
);
```

Parent topic: Deprecated vg_lite syntax

vg_lite_set_premultiply ***(deprecated)*** **Syntax:**

```
vg_lite_error_t vg_lite_set_premultiply (
   vg_lite_uint8_t              src_premult,
   vg_lite_uint8_t              dst_premult,
);
```

Parent topic: Deprecated vg_lite syntax

Parent topic:*Deprecated and renamed APIs*

VGLite API version 2.0 to 3.0 migration guide The VGLite API version 3.0 is not fully compatible with VGLite API version 2.0. VGLite API version 3.0 includes some new API functions for the new features in the latest VG GPU like GC555. Some VGLite API version 2.0 function interfaces are changed in API version 3.0. So, the existing VGLite API version 2.0 applications must be modified to compile and run properly with the VGLite API version 3.0 driver. This chapter provides guidance for migrating VGLite API version 2.0 applications to VGLite API version 3.0.

VGLite API name changes in API version 3.0 Some original VGLite API names are changed in API version 3.0 for API naming consistency. In the VGLite API version 3.0 header file vg_lite.h, a set of API name macros are defined for the equivalent API names between API version 3.0 and API version 2.0, so it is not necessary to modify the VGLite API function names in API version 2.0 applications for the application to compile and run with the API version 3.0 driver.

The list of equivalent VGLite API functions between API version 3.0 and API version 2.0 is shown below. These API functions' parameters are the same between API version 3.0 and API version 2.0.

```
/* API name defines for backward compatibility to VGLite 2.0 APIs */
#define vg_lite_buffer_upload                    vg_lite_upload_buffer
#define vg_lite_path_append                      vg_lite_append_path
#define vg_lite_path_calc_length                 vg_lite_get_path_length
#define vg_lite_set_ts_buffer                    vg_lite_set_tess_buffer
#define vg_lite_set_draw_path_type               vg_lite_set_path_type
#define vg_lite_create_mask_layer                vg_lite_create_masklayer
#define vg_lite_fill_mask_layer                  vg_lite_fill_masklayer
#define vg_lite_blend_mask_layer                 vg_lite_blend_masklayer
#define vg_lite_generate_mask_layer_by_path      vg_lite_render_masklayer
#define vg_lite_set_mask_layer                   vg_lite_set_masklayer
#define vg_lite_destroy_mask_layer               vg_lite_destroy_masklayer
#define vg_lite_enable_mask                      vg_lite_enable_masklayer
#define vg_lite_enable_color_transformation      vg_lite_enable_color_transform
#define vg_lite_set_color_transformation         vg_lite_set_color_transform
#define vg_lite_set_image_global_alpha           vg_lite_source_global_alpha
#define vg_lite_set_dest_global_alpha            vg_lite_dest_global_alpha
#define vg_lite_clear_rad_grad                   vg_lite_clear_radial_grad
#define vg_lite_update_rad_grad                  vg_lite_update_radial_grad
#define vg_lite_get_rad_grad_matrix              vg_lite_get_radial_grad_matrix
#define vg_lite_set_rad_grad                     vg_lite_set_radial_grad
#define vg_lite_draw_linear_gradient             vg_lite_draw_linear_grad
#define vg_lite_draw_radial_gradient             vg_lite_draw_radial_grad
#define vg_lite_draw_gradient                    vg_lite_draw_grad
#define vg_lite_mem_avail                        vg_lite_get_mem_size
#define vg_lite_set_update_stroke                vg_lite_update_stroke
```

The list of equivalent VGLite API structures and enumerations is shown below:

```
#define vg_lite_buffer_image_mode_t              vg_lite_image_mode_t
#define vg_lite_draw_path_type_t                 vg_lite_path_type_t
#define vg_lite_linear_gradient_ext_t            vg_lite_ext_linear_gradient_t
#define vg_lite_buffer_transparency_mode_t       vg_lite_transparency_t
```

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_set_scissor API interface change The VGLite API vg_lite_set_scissor() function name is not changed in API version 3.0, but the API parameters are defined differently in API version 3.0.

In VGLite API version 3.0, the vg_lite_set_scissor() function is defined as:

```
/* Set and enable a scissor rectangle for render target. */
vg_lite_error_t vg_lite_set_scissor(vg_lite_int32_t x, vg_lite_int32_t y,
                vg_lite_int32_t right, vg_lite_int32_t bottom);
```

In VGLite API version 2.0, the vg_lite_set_scissor() function is defined as:

```
vg_lite_error_t vg_lite_set_scissor(int32_t x, int32_t y, int32_t width, int32_t height);
```

So, the vg_lite_set_scissor() API parameters "width" and "height" in the VGLite API version 2.0 application must be changed to "right" x-coordinate value and "bottom" y-coordinate value.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_map API interface change The VGLite API vg_lite_map() function name is not changed in API version 3.0, but the API parameters are defined differently in API version 3.0.

In VGLite API version 3.0, the vg_lite_map() function is defined as:

```
/* Map a buffer into hardware accessible address space. */
vg_lite_error_t vg_lite_map(vg_lite_buffer_t *buffer, vg_lite_map_flag_t flag, int32_t fd);
```

In VGLite API version 2.0, the vg_lite_map() function is defined as:

```
vg_lite_error_t vg_lite_map(vg_lite_buffer_t *buffer);
```

So, vg_lite_map() in VGLite API version 3.0 API requires two extra parameters "flag" and "fd", which can simply be set as vg_lite_map (buffer, 0, 0) in applications.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_enable_scissor / vg_lite_disable_scissor API The VGLite API vg_lite_enable_scissor() and vg_lite_disable_scissor() functions are valid only for vg_lite_scissor_rects() API. They have no effect for vg_lite_set_scissor() in VGLite API version 3.0.

Although the behavior of vg_lite_enable_scissor() and vg_lite_disable_scissor() is changed in VGLite API version 3.0, there is no need to change these functions in VGLite API version 2.0 applications to work with the VGLite API version 3.0 driver.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_draw_pattern API interface change The VGLite API vg_lite_draw_pattern() function name is not changed in API version 3.0, but the API parameters are defined differently in API version 3.0.

In VGLite API version 3.0, the vg_lite_draw_pattern() function is defined as:

```
/* Draw a path that is filled by a transformed image pattern. */
vg_lite_error_t vg_lite_draw_pattern(vg_lite_buffer_t *target,
                                     vg_lite_path_t *path,
                                     vg_lite_fill_t fill_rule,
                                     vg_lite_matrix_t *path_matrix,
                                     vg_lite_buffer_t *pattern_image,
                                     vg_lite_matrix_t *pattern_matrix,
                                     vg_lite_blend_t blend,
                                     vg_lite_pattern_mode_t pattern_mode,
                                     vg_lite_color_t  pattern_color,
                                     vg_lite_color_t  color,
                                     vg_lite_filter_t filter);
```

Compared to the VGLite API version 2.0 vg_lite_draw_pattern() function, "color" is a new additional parameter. It specifies a 32bpp ARGB color (vg_lite_color_t) to be applied as a mix color. If nonzero, the mix color value gets multiplied with each source pixel before blending happens. If a mix color is not needed, set the color parameter to 0.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

[New] vg_lite_copy_image in VGLite API version 3.0 The new API vg_lite_copy_image() is added in VGLite API version 3.0 to support the OpenVG vgCopyImage API, which performs a pixel rectangle copy without pixel transformation, blending, filtering operations.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

vg_lite_set_dither API is deprecated in API version 3.0 The original API version 2.0 function vg_lite_set_dither(int enable) API is removed from API version 3.0, it is replaced with two new APIs for dither enable/disable:

```
/* Enable dither function. Dither is OFF by default. */
vg_lite_error_t vg_lite_enable_dither();
/* Disable dither function. Dither is OFF by default. */
vg_lite_error_t vg_lite_disable_dither();
```

Therefore, the vg_lite_set_dither(enable) function in the VGLite API version 2.0 application must be replaced with vg_lite_enable_dither() or vg_lite_disable_dither() to work with the VGLite API version 3.0 driver.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

Deprecated VGLite API version 2.0 functions The VGLite API vg_lite_perspective(), vg_lite_enable_premultiply(), vg_lite_disable_premultiply() functions are removed from API version 3.0. These API functions must be deleted from a VGLite API version 2.0 application to work with the VGLite API version 3.0 driver.

In VGLite API version 3.0, the color premultiply setting is defined by the vg_lite_blend_t enumeration to replace the original vg_lite_enable_premultiply() and vg_lite_disable_premultiply() APIs.

- VG_LITE_BLEND_* enumeration values in vg_lite_blend_t define non-premultiplied blending modes.
- OPEVG_BLEND_* enumeration values in vg_lite_blend_t define premultiplied Porter-Duff blending modes.

So, the VGLite API version 3.0 application can set different blending modes to get the desired premultiplied/non-premultiplied blending result.

Parent topic:*VGLite API version 2.0 to 3.0 migration guide*

Revision history

Document ID	Release date	Description
IMX Rev. 1.2	17 January 2025	The document is updated to correspond to the API version 3.0
IMX Rev. 1.1	22 September 2022	- Paragraph 4.1.1 Updated *Table 3 - vg_lite_feature_t* enumeration. - Paragraph 6.6 Added documentation for new API *vg_lite_set_dither* - Paragraph 8.2 Blit structures- Added documentation for new data structure *vg_lite_color_key_t* -; added documentation for new data structure *vg_lite_color_key4_t* - Paragraph 8.3.1, *vg_lite_blit* function- added note related to HW limitation on RT500 platform - Paragraph 8.3.2, *vg_lite_blit_rect* function -added note related to HW limitation on RT500 platforms - Paragraph 8.3.3, *vg_lite_get_transform_matrix* function- adjusted function description, adjusted function parameters description - Paragraph 8.3, blit functions- added documentation for new API *vg_lite_set_color_key* - Paragraph 8.4.1, *vg_lite_enable_premultiply* function- added note about limited support on specific platforms - Paragraph 8.4.2, *vg_lite_disable_premultiply* function- added note about limited support on specific platforms - Paragraph 10.1.3, *vg_lite_fill_t* enumeration- added note about crossing points buffer limitation - Paragraph 10.2, draw and gradient structures- added documentation for new data structure *vg_lite_gradient_parameter_t - done*- added documentation for new data structure *vg_lite_gradient_ext_t*- Paragraph 10.3, draw functions- added documentation for new API *vg_lite_draw_linear_gradient*- Paragraph 10, vector-Based Draw Operations - added new paragraph 10.5 *Extended linear gradient initialization and control functions*; added documentation for new API *vg_lite_set_linear_gradient*; added documentation for new API *vg_lite_get_linear_grad_matrix*; added documentation for new API *vg_lite_update_linear_grad*; Added documentation for new API *vg_lite_clear_linear_grad* - Paragraph 10.5, Radial gradient functions - adjusted paragraph title - Added new Chapter *Stroke Operations* - Chapter *Platform-Specific Features* -updated *Table 41* - Platform-specific VGLite features
IMX Rev. 1	27 January 2022	*Introduction* Added i.MX RT1160 to the list of NXP devices that support VGLite graphics API vg_lite_error_t enumeration Updated Table 1 vg_lite_feature_t enumerationUpdated Table 1 *API control*
IMX Rev. 0	22 February 2021	Initial release

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

1.8 Wireless

1.8.1 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 25.12.00 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 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) TX and RX sup
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) 4k TX and RX su
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC) 1x1 20 MHz and 40 MHz
802.11n - High throughput	HT Beamformee (explicit)
802.11ac - Very high throughput	2.4 GHz band supported channel bandwidth: 20MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac - Very high throughput	5 GHz band supported channel bandwidth: 80 MHz
802.11ac - Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac - Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9) - 1x1
802.11ac - Very high throughput	MU-MIMO Beamformee (Explicit and Implicit)
802.11ac - Very high throughput	RTS/CTS with BW signaling
802.11ac - Very high throughput	Operation mode notification
802.11ac - Very high throughput	Backward compatibility with non-VHT devices
802.11ac - Very high throughput	TX VHT MCS rate adaptation
802.11ac - Very high throughput	Low density parity check (LDPC)
802.11ax - High efficiency	2.4 GHz band supported channel bandwidth: 20MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 20 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidth: 40 MHz
802.11ax - High efficiency	5 GHz band supported channel bandwidths: 80 MHz
802.11ax - High efficiency	OFDMA (UL/DL, 106 RU)
802.11ax - High efficiency	OFDMA (UL/DL, 484 RU)
802.11ax - High efficiency	1024 QAM
802.11ax - High efficiency	Target wake time (TWT)
802.11ax - High efficiency	256 QAM modulation – MCS8 and MCS9
802.11ax - High efficiency	1024 QAM modulation – MCS10 and MCS11, 2.4 GHz

Table 5 – continued from p

Features	Sub features
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
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Opensource WPA Supplicant Support
802.11i security	WPA2-PSK AES | WPA Supplicant
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | WPA S
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 Methods3 | WPA Supplicant
802.11i security	WPA2-Enterprise Mixed Mode3 | WPA Supplicant
802.11i security	WPA3-Enterprise3 (Suite-B) |National Security Algorithm (C
802.11i security	802.11w - PMF (Protected Management Frames) | WPA Sup
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 Enterprise3
Power save mode	Deep sleep
Power save mode	IEEE power save
Power save mode	Host sleep/WoWLAN (inband)3
Power save mode	Host sleep/WoWLAN (outband)3
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 connect3
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	TKIP1
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

Table 5 – continued from p

Features	Sub features
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)3
General features	Ambient Motion Index (AMI)3
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)3
General features	802.11az3

Parent topic: Wi-Fi radio

[1] As per Wi-Fi specification, connecting in TKIP security in non 802.11n mode is allowed.

[2] Support available in host-base supplicant.

[3] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

[4] Read more about NCP feature in *References*. [5] To enable the feature, CONFIG_ECSA = 1 must be defined in wifi_config.h (does not apply to RW610 and RW612).

AP mode

Features	Sub features
802.11n - High throughput	2.4 GHz band operation supported channel bandwidth: 20 M
802.11n - High throughput	2.4 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11n - High throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11n - High throughput	Short/long guard interval (400 ns/800 ns)
802.11n - High throughput	Data rates up to 72 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	Data rates up to 150 Mbit/s (MCS 0 to MCS 7)
802.11n - High throughput	1 spatial stream (1x1)
802.11n - High throughput	HT protection mechanisms
802.11n - High throughput	Aggregated MAC protocol data unit (AMPDU) Rx support
802.11n - High throughput	Aggregated MAC service data unit (AMSDU) -4k RX support
802.11n - High throughput	Max client support (up to 8 devices)
802.11n - High throughput	TX MCS rate adaptation (BGN)
802.11n - High throughput	RX low density parity check (LDPC)
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 20 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 40 MHz
802.11ac – Very high throughput	5 GHz band supported channel bandwidth: 80MHz
802.11ac – Very high throughput	Short/long guard interval (400ns/800ns)
802.11ac – Very high throughput	Data rates up to 86.7 Mbps (MCS0 to MCS 8)
802.11ac – Very high throughput	Data rates up to 433.3 Mbps (MCS 0 to MCS 9)
802.11ac – Very high throughput	Single user- Aggregated MAC protocol data unit (SU-AMPDU
802.11ac – Very high throughput	RTS/CTS with BW signaling
802.11ac – Very high throughput	Backward compatibility with non-VHT devices
802.11ac – Very high throughput	TX VHT MCS rate adaptation
802.11ac – Very high throughput	MU-MIMO Beamformee (explicit and implicit)
802.11ac – Very high throughput	Operation mode notification

Table 6 – continued from prev

Features	Sub features
802.11ax – High efficiency	2.4 GHz band operation (20 MHz channel bandwidth)
802.11ax – High efficiency	2.4 GHz band operation (40 MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (20MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (40MHz channel bandwidth)
802.11ax – High efficiency	5 GHz band operation (80 MHz channel bandwidth)
802.11d	802.11d - Regulatory domain/operating class/country info
802.11e -QoS	EDCA [enhanced distributed channel access] / WMM (wirele
802.11i security	Hostapd Support
802.11i security	WPA2-PSK AES | hostapd
802.11i security	WPA3-SAE (Simultaneous Authentication of Equals) | Hosta
802.11i security	WPA2+WPA3 PSK Mixed Mode (WPA3 Transition Mode) | H
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 Mode1 | Hostapd
802.11i security	WPA3-Enterprise (Suite-B)1 |National Security Algorithm (C
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) mode2
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 diversity1	Y	Y	Y	Y	Y	Y

Parent topic: Wi-Fi radio

[1] Feature not enabled by default in the SDK. Refer to *Feature enable and memory impact* for the macro to enable the feature and the impact on the memory when enabling the feature.

Parent topic:*Features*

Wi-Fi Generic features

Features	Sub features	88W898	IW416	IW611/IW6	RW610/RW6	IW610	AW611
Generic	Firmware download (parallel)1	Y	Y	Y	N	N	Y
Generic	Secure boot	N	N	Y	Y	Y	Y
Generic	Kconfig memory optimizer3	Y	Y	Y	Y	Y	Y
Generic	Firmware Compression2	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 [3] Refer to *10*.

Wi-Fi direct/P2P

Features	Sub features	88W898	IW416	IW611/IW6	RW610/RW6	IW610	AW6113
P2P basic functionality1	P2P Auto GO	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P GO	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P GC	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Persistent Group	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Invitation	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Device Discovery	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P Provision Discovery	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P simultaneous GO + STA	Y	Y	Y	Y	Y	Y
P2P basic functionality1	P2P simultaneous GC + uAP	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	IW41	IW611/	RW610/	IW61	AW611
General features	Bluetooth Class 1.5 and Class 2 support	Y	Y	Y	N	N	Y
General features	Scatternet support	Y	Y	Y	N	N	Y
General features	Maximum of seven simultaneous ACL connections – Central links	Y	Y	Y	N	N	Y
General features	Automatic packet type selection	Y	Y	Y	N	N	Y
General features	Bluetooth - 2.1 to 5.0 specification support	Y	Y	Y	N	N	Y
General features	Low power sniff	Y	Y	Y	N	N	Y
General features	Deep sleep using out-of-band	Y	Y	N	N	N	N
General features	Wake on Bluetooth (SoC to host)	Y	Y	Y	N	N	Y
General features	Independent reset (in-band)1	Y	Y	Y	Y	N	Y
General features	Independent reset (out-band)1	Y	Y	N	N	N	N
General features	Firmware download (parallel)1	Y	Y	N	N	N	N
General features	RF test mode	Y	Y	Y	N	N	Y
Bluetooth packet type supported	ACL (DM1, DH1, DM3, DH3, DM5, DH5, 2-DH1, 2-DH3, 2-DH5, 3-DH1, 3-DH3, 3-DH5)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	SCO (HV1, HV3)	Y	Y	Y	N	N	Y
Bluetooth packet type supported	eSCO (EV3, EV4, EV5, 2EV3, 3EV3, 2EV5, 3EV5)	Y	Y	Y	N	N	Y
Bluetooth profiles supported	A2DP source/sink	Y	Y	Y	N	N	Y
Bluetooth profiles supported	AVRCP target/controller	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HFP Dev/AG	Y	Y	Y	N	N	Y
Bluetooth profiles supported	OPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	SPP server/client	Y	Y	Y	N	N	Y
Bluetooth profiles supported	HID target/device	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM NBS central/peripheral	Y	Y	Y	N	N	Y
Bluetooth audio features	PCM WBS central/peripheral	Y	Y	Y	N	N	Y

Parent topic: Bluetooth radio

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

Bluetooth LE

Features	Sub features
Generic features	Maximum 16 Bluetooth LE connections (central role)
Generic features	Deep sleep using out-of-band
Generic features	Wake on Bluetooth LE (SoC to Host)
Generic features	RF Test mode
Bluetooth profile support	Bluetooth LE GATT
Bluetooth profile support	Bluetooth LE HID over GATT
Bluetooth profile support	Bluetooth LE GAP
Bluetooth LE 4.0 support	Low Energy physical layer
Bluetooth LE 4.0 support	Low Energy link layer
Bluetooth LE 4.0 support	Enhancements to HCI for Low Energy
Bluetooth LE 4.0 support	Low energy direct test mode
Bluetooth 4.1 support	Low duty cycle directed advertising
Bluetooth 4.1 support	Bluetooth LE dual mode topology
Bluetooth 4.1 support	Bluetooth LE privacy v1.1
Bluetooth 4.1 support	Bluetooth LE link layer topology
Bluetooth 4.2 support	Bluetooth LE secure connection
Bluetooth 4.2 support	Bluetooth LE link layer privacy v1.2
Bluetooth 4.2 support	Bluetooth LE data length extension
Bluetooth 4.2 support	Link layer extended scanner filter policies
Bluetooth 5.0 support	Bluetooth LE 2 Mbps support
Bluetooth 5.0 support	High duty cycle directed advertising
Bluetooth 5.0 support	Low Energy advertising extension
Bluetooth 5.0 support	Low Energy long range
Bluetooth 5.0 support	Low Energy periodic advertisement
Bluetooth 5.2 support	Low Energy power control
Bluetooth LE audio support1 2	Isochronous channel
Bluetooth LE audio support1 2	Broadcast LE Audio BIS source
Bluetooth LE audio support1 2	Broadcast LE Audio BIS sink
Bluetooth LE audio support1 2	Broadcast LE Audio BIG Validation
Bluetooth LE audio support1 2	Broadcast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support1 2	Broadcast LE Audio framed mode
Bluetooth LE audio support1 2	Broadcast LE Audio unframed mode
Bluetooth LE audio support1 2	Broadcast LE Audio sequential packing
Bluetooth LE audio support1 2	Broadcast LE Audio: Mono and Stereo
Bluetooth LE audio support1 2	Broadcast LE Audio BIS encrypted audio
Bluetooth LE audio support1 2	Broadcast LE Audio BIS unencrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio CIS source
Bluetooth LE audio support1 2	Unicast LE Audio CIS sink
Bluetooth LE audio support1 2	Unicast LE Audio CIG validation
Bluetooth LE audio support1 2	Unicast LE Audio CIS synchronization
Bluetooth LE audio support1 2	Unicast LE Audio Phy: 1M/2M/ coded
Bluetooth LE audio support1 2	Unicast LE Audio framed mode
Bluetooth LE audio support1 2	Unicast LE Audio unframed mode
Bluetooth LE audio support1 2	Unicast LE Audio sequential packing
Bluetooth LE audio support1 2	Unicast LE Audio: mono and stereo
Bluetooth LE audio support1 2	Unicast LE Audio CIS encrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio CIS unencrypted audio
Bluetooth LE audio support1 2	Unicast LE Audio TX/RX and bidirectional traffic

Table 7 – continued from prev

Features	Sub features
Bluetooth LE audio support1 2	ISO interval for LE Audio: 7.5ms 10ms 20ms 30ms
Bluetooth LE audio support1 2	Sampling frequency for LE Audio: 8kHz 16kHz 24kHz, 32kH
Bluetooth LE audio support1 2	LE Audio Auracast use cases: Auracast streaming 2 BISes
Bluetooth LE audio support1 2	LE Audio Unicast use cases: Unicast streaming 2 CISes
Bluetooth LE audio support1 2	LE Audio Unicast Use cases: Unicast streaming 4 CISes
Bluetooth LE audio support1 2	A2DP + Auracast/Unicast Bridge use cases – CIS/BIS
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth LE coexistence
BCA TDM Coexistence mode (shared antenna)	AP + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	STA + Bluetooth + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth LE coexistence
BCA TDM coexistence mode (separate antenna)	AP + Bluetooth + Bluetooth LE coexistence

Note: Details of the tested Bluetooth LE Audio use cases:

- Number of streams:
  - 1-CIG | upto 4-CIS with 1 LE ACL (for 4-CIS: execute only mono UCs, SDU Int: 10ms)
  - 1-CIG | upto 4-CIS with 4 separate LE ACL (for 4-CIS: SDU Size= Max 100 Oct, PHY=2M, RTN=1, SDU Int: 10ms only) (execute only mono UCs for 4-CIS)
  - 1-BIG | upto 4-BIS (for 4-BIS: execute only mono UCs, SDU Int: 10ms only)
- PHY: 2M and 1M
- Audio mode: mono (for 1 to 4 streams) and stereo (for 1 stream)
- Packing: sequential and interleaved
- Bit rate: maximum 96kbps
  - For 1-CIG with upto 3-CIS: maximum bit rate 96kbps
  - For 1-CIG with 4-CIS: maximum bit rate 80kbps
  - For 1-BIG with 4-BIS: maximum bit rate 80kbps
  - For 2-CIG cases: maximum bit rate 80kbps
- Mode: unframed mode
- 48_5 and 48_6 mono and stereo configurations are not supported.

Details of the tested Bluetooth coexistence (Bluetooth + Bluetooth LE Audio) use cases:

- Bluetooth + Bluetooth LE Audio
- A2DP + Bluetooth LE Audio bridging support
- A2DP sink link (central) -> LEA 2-CIS (SDU Int: 10ms only | A2DP only with SBC Codec | PHY: 2M)

Parent topic: Bluetooth radio

[1] Experimental feature intended for evaluation/early development only and not production. Incomplete mandatory certification.

[2] LE audio feature is supported for standalone scenarios only and not for BR/EDR and Wi-Fi coexistence scenarios such as LE audio + BR/EDR link or LE audio + Wi-Fi link. From the perspective

of NXP Edgefast Bluetooth host stack, LE audio feature can be disabled by the CONFIG_BT_AUDIO macro without impact on any other features. LE audio feature can be tested by the user, using their own supported host stack.

Parent topic:*Features*

802.15.4 radio

Features	Sub features	IW612	IW610	RW612
General features	Spinel over SPI	Y	N	N
General features	OpenThread RCP Mode implementing Thread1.3	Y	N	N
General features	802.15.4-2015 MAC/PHY as required by Thread 1.3	Y	Y	Y
General features	OpenThread Border Router (OTBR) v1.1	Y	Y	Y
General features	Direct/indirect transmission with/without ACK	Y	Y	Y
General features	802.15.4 CSL parent feature implementation	Y	Y	Y
General features	Enhanced Frame Pending	Y	Y	Y
General features	Enhanced keep alive	Y	Y	Y
General features	Router	Y	Y	Y
General features	Leader	Y	Y	Y
General features	Router Eligible End Device (REED)	Y	Y	Y
General features	End Device (FED, MED)	Y	Y	Y
Zigbee features	Coordinator	N	N	Y
Zigbee features	Router	N	N	Y
Zigbee features	End Device (RX ON)	N	N	Y
Zigbee features	R23	N	N	Y
Zigbee features	OTA Client	N	N	Y
Zigbee features	OTA server	N	N	Y
Matter features	Matter over Wi-Fi	Y	N	N
Matter features	Matter over Thread	Y	N	Y

Parent topic:*Features*

Coexistence

Wi-Fi and Bluetooth/802.15.4 coexistence

Features	Sub features	IW6´	IW6´	RW612
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	STA + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Mobile AP + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + Bluetooth LE + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth LE2	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Bluetooth + Bluetooth LE	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth + OpenThread + Wi-Fi	Y	N	N
BCA_TDM separate antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Bluetooth LE + OpenThread + Wi-Fi	Y	Y	Y
BCA_TDM separate antenna1 (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 antenna1 (lower and higher isolation) 1x1 Wi-Fi, (Bluetooth and 802.15.4 shared)	Single antenna configuration	Y	Y	Y
BCA_TDM separate antenna1 (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
AMI	CONFIG_CSI_AMI3	Flash - 2032K, RAM - 772K
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_WMM1	Flash - 10K, RAM - 57K
802.11mc	CONFIG_11MC CONFIG_CSI CONFIG_WLS_CSI_PROC2 CONFIG_11AZ	Flash: 52.78KB, RAM : 121.1KB
802.11az	CONFIG_11MC CONFIG_CSI[2] CONFIG_WLS_CSI_PROC2 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.

[1] The macro is not used for IW416.

[2] Prerequisite macros for 802.11mc and 802.11az features

[3] Enable PRINTF_FLOAT_ENABLE only for MCUXpresso IDE and specifically for the RT1060-EVKC and RT1170-EVKB platforms

- Go to project properties > C/C++ Build > Settings > Preprocessor.
- Add PRINTF_FLOAT_ENABLE=1

88W8987 release notes

Package information

- SDK version: 25.12.00

Parent topic:*88W8987 release notes*

Version information

- Wireless SoC: 88W8987
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p153.9
  - 16 - Major revision
  - 92 - Feature pack
  - 21 - Release version
  - p153.9 - 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
Open security	52	52	60	63
WPA2-AES	50	51	60	62
WPA3-SAE	50	51	60	62

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
Open security	124	119	228	235
WPA2-AES	118	107	228	204
WPA3-SAE	114	107	229	203

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
Open security	47	48	57	60
WPA2-AES	46	49	57	60
WPA3-SAE	47	49	57	60

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
Open security	127	141	227	217
WPA2-AES	124	127	227	198
WPA3-SAE	125	127	227	173

Parent topic: Wi-Fi throughput

Parent topic:*88W8987 release notes*

EU conformance tests

- EU Adaptivity test - EN 300 328 v2.1.1 (for 2.4 GHz)
- EU Adaptivity test - EN 301 893 v2.1.1 (for 5 GHz)

Parent topic:*88W8987 release notes*

Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p64.1 to 16.91.21.p82

Component	Description
Wi-Fi	WPA3-R3 enabled APUT beacons does not have RSNXE when configured in H2E mode-Associated event is received even when connecting using wrong password WFA APUT Low iperf TCP/UDP Tx throughput with Realtek station

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p82 to 16.91.21.p91.6

Component	Description
Wi-Fi	In wrong password scenario, After updating new password the phone is not able to connect with DUTAP

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p91.6 to 16.91.21.p124

Component	Description
Wi-Fi	Cloud keep alive packets not seen after DUT enters host sleep. DUT is sending QOS null packets even in host sleep

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p124 to 16.91.21.p133

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133 to 16.91.21.p142.5

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

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

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

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

Firmware version: From 16.92.21.p151.7 to 16.92.21.p153.5

Component	Description
Wi-Fi	Added P2P Persistance and P2P Invitation

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.5 to 16.92.21.p153.6

Component	Description
Wi-Fi	Enabled mbedtls 3.x

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: 25.12.00

Parent topic:*IW416 release notes*

Version information

- Wireless SoC: IW416
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 16.92.21.p153.9
  - 16 - Major revision
  - 92 - Feature pack

  - 21 - Release version
  - p153.9 - 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
Open security	44	47	59	59
WPA2-AES	39	43	58	55
WPA3-SAE	39	45	57	53

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
Open security	72	59	95	87
WPA2-AES	69	58	116	92
WPA3-SAE	57	58	115	91

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
Open security	43	48	59	59
WPA2-AES	42	48	56	60
WPA3-SAE	42	47	57	58

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
Open security	68	64	118	96
WPA2-AES	65	59	117	96
WPA3-SAE	69	59	118	96

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
Open security	41	45	52	54
WPA2-AES	42	45	53	53
WPA3-SAE	45	42	53	53

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
Open security	62	70	123	90
WPA2-AES	61	65	117	90
WPA3-SAE	61	65	118	87

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
Open security	44	45	58	57
WPA2-AES	42	45	55	56
WPA3-SAE	43	45	57	56

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
Open security	75	85	118	100
WPA2-AES	77	68	118	100
WPA3-SAE	77	69	118	100

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

Component	Description
Wi-Fi	DUT STA getting rebooted after 15~20 iterations of 11R-Command based roaming0xa4 command timeout after several hours of stress test

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.91.21.p133.2 to 16.91.21.p142.5

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

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

Component	Description
Wi-Fi	Samsung S24 Ultra and Google Pixel 7 mobiles having Android 14 are not able connect to the DUTAP with WPA3 SAE security.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p151.7 to 16.92.21.p153.5

Component	Description
Wi-Fi	The DUT encounters a command response timeout during the execution of the wlan-info command following UDP traffic tests.
Wi-Fi	Random hang issue seen when using wlan-p2p-find/stop in succession

Parent topic: Bug fixes and/or feature enhancements

Firmware version: From 16.92.21.p153.5 to 16.92.21.p153.6

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*IW416 release notes*

Known issues

Component	Description
Coex	Wi-Fi connection in 2.4GHz is not stable, observed deauthentication within 10sec.

Parent topic:*IW416 release notes*

IW611/IW612 release notes

Note: The IW611/IW612 support is enabled in i.MX RT1170 EVKB and i.MX RT1060 EVKC.

Package information

- SDK version: 25.12.00

Parent topic:*IW611/IW612 release notes*

Version information

- Wireless SoC: IW611/IW612
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.3.p27.10
  - 18 - Major revision
  - 99 - Feature pack
  - 3 - Release version
  - p27.10 - Patch number

Parent topic:*IW611/IW612 release notes*

Host platform

- i.MX RT1170 EVKB and i.MX RT1060 EVKC Platforms running FreeRTOS
- Host interfaces
  - Wi-Fi over SDIO (SDIO 2.0 support, SDIO clock frequency: 50 MHz)
  - Bluetooth/Bluetooth LE over UART
  - 802.15.4 over SPI (IW612 only)
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*IW611/IW612 release notes*

Wi-Fi and Bluetooth certification

The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD
- STA | WPA3 SAE (R3)
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *6*.

Note: This release supports STAUT only certifications.

Parent topic: Wi-Fi and Bluetooth certification

Bluetooth controller certification

QDID: refer to *4*.

Note: QDID upgrade to Bluetooth Core Specification Version 5.4 is in progress.

Parent topic: Wi-Fi and Bluetooth certification

Parent topic:*IW611/IW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: IW612 Murata (Module: 2EL M.2) with EVK-MIMXRT1060 EVKC platform
- DUT Power Source: External power supply
- Client: Apple MacBook Air
- Channel: 6 | 36
- Wi-Fi application: wifi_wpa_supplicant
- Compiler used to build application: armgcc
- Compiler Version gcc-arm-none-eabi-13.2
- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *2*

The throughput numbers are captured with default configurations using *wifi_wpa_supplicant* sample application.

Parent topic: Wi-Fi throughput

iPerf host configuration and impact on throughput {#iperf_host_configuration_and_impact_on_throughpu

To get the highest throughput, the throughput values shown in STA throughput and Mobile AP throughput are measured with the maximum values of the default host configuration macros. STA and AP throughput captured with the minimum values of the host configuration macros shows the throughput numbers obtained when using the minimum values of the host configuration macros. The macro values are defined in *lwipopts.h* file.

The table below lists the minimum and maximum values of the host configuration macros.

Values of the host configuration macros

Parameter	Maximum value	Minimum value
TCPIP_MBOX_SIZE	96	32
DEFAULT_RAW_RECVMBOX_SIZE	32	12
DEFAULT_UDP_RECVMBOX_SIZE	64	12
DEFAULT_TCP_RECVMBOX_SIZE	64	12
TCP_MSS	1460	536
TCP_SND_BUF	24 * TCP_MSS	2 * TCP_MSS
MEM_SIZE	319160	41,080
TCP_WND	15 * TCP_MSS	10 * TCP_MSS
MEMP_NUM_PBUF	20	10
MEMP_NUM_TCP_SEG	96	12
MEMP_NUM_TCPIP_MSG_INPKT	80	16
MEMP_NUM_TCPIP_MSG_API	80	8
MEMP_NUM_NETBUF	32	16

STA and AP throughput captured with the minimum values of the host configuration macros {#sta_and_ap_throughput_captured_with_the_minimum_values_of_the_host_configuration_macr

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	7	18	111	124
WPA2-AES	7	18	110	124
WPA3-SAE	6	18	110	124

Mobile AP mode throughput - HE Mode | 5 GHz Band | 80 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open Security	2	19	93	127
WPA2-AES	2	19	105	126
WPA3-SAE	2	19	104	132

Parent topic:iPerf host configuration and impact on throughput

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	52	51	64	63
WPA2-AES	51	50	62	62
WPA3-SAE	51	50	63	61

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
Open security	79	85	118	131
WPA2-AES	78	84	118	129
WPA3-SAE	78	83	118	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
Open security	50	52	63	64
WPA2-AES	49	51	63	63
WPA3-SAE	49	51	63	63

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
Open security	77	86	118	133
WPA2-AES	76	86	118	132
WPA3-SAE	79	86	118	132

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	56	59	76	76
WPA2-AES	56	59	74	75
WPA3-SAE	56	59	76	75

STA mode throughput - VHT Mode | 2.4 GHz Band | 40 MHz (VHT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	74	92	162	170
WPA2-AES	74	90	160	169
WPA3-SAE	71	91	161	171

STA mode throughput - VHT 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	43	57	76	78
WPA2-AES	42	57	75	77
WPA3-SAE	43	57	75	77

STA mode throughput - VHT 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
Open security	88	95	118	177
WPA2-AES	87	94	118	175
WPA3-SAE	91	94	118	175

STA mode throughput - VHT 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
Open security	121	102	118	200
WPA2-AES	121	103	118	200
WPA3-SAE	121	103	118	200

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	78	64	117	105
WPA2-AES	78	67	117	104
WPA3-SAE	79	65	117	97

STA mode throughput - HE Mode | 2.4 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	95	91	118	199
WPA2-AES	93	90	118	200
WPA3-SAE	91	87	118	199

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	76	66	118	127
WPA2-AES	75	68	118	125
WPA3-SAE	75	68	118	126

STA mode throughput - HE Mode | 5 GHz Band | 40 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	105	69	118	200
WPA2-AES	104	70	118	200
WPA3-SAE	104	70	118	200

STA mode throughput - HE Mode | 5 GHz Band | 80 MHz (HE)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
Open security	125	73	118	200
WPA2-AES	123	76	118	200
WPA3-SAE	123	76	118	200

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
Open security	51	54	61	60
WPA2-AES	50	55	61	60
WPA3-SAE	51	54	61	60

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
Open security	85	107	118	124
WPA2-AES	86	101	118	126
WPA3-SAE	84	102	118	126

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
Open security	51	43	63	60
WPA2-AES	50	43	62	60
WPA3-SAE	50	43	63	60

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
Open security	89	115	118	128
WPA2-AES	88	110	118	128
WPA3-SAE	88	115	118	128

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
Open security	58	66	76	72
WPA2-AES	58	65	75	72
WPA3-SAE	58	65	75	72

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
Open security	103	141	135	168
WPA2-AES	102	134	137	167
WPA3-SAE	102	134	139	167

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
Open security	137	180	182	218
WPA2-AES	130	174	181	218
WPA3-SAE	136	175	182	218

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
Open security	53	66	85	120
WPA2-AES	52	65	83	116
WPA3-SAE	52	65	83	118

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
Open security	86	100	133	132
WPA2-AES	83	100	135	134
WPA3-SAE	86	100	136	134

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
Open security	54	65	82	83
WPA2-AES	58	65	82	82
WPA3-SAE	58	65	81	81

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
Open security	104	141	151	170
WPA2-AES	102	137	151	170
WPA3-SAE	103	136	150	170

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	138	180	189	219
WPA2-AES	135	175	190	218
WPA3-SAE	135	175	192	218

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

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

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

Component	Description
-	NA

Parent topic: Bug fixes and/or feature enhancements

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

Firmware version: 18.99.3.p25.11 to 18.99.3.p26.10

Component	Description
Wi-Fi	During legacy roaming when the "Link Lost" observed the DUTSTA fails to roam
Wi-Fi	During the automated testing of the channel performance, a system hang can occur, with the error message ".sdio_drv_write failed".

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.3.p26.10 to 18.99.3.p27.1

Component	Description
Wi-Fi	Enabled mbedtls 3.x

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: 25.12.00

Parent topic:*RW610/RW612 release notes*

Version information

- Wi-Fi firmware version: 18.99.6.p50
  - rw61x_sb_wifi_a2.bin for A2
  - 18 - Major revision
  - 99 - Feature pack
  - 6 - Release version
  - p50 - Patch number
- Bluetooth LE firmware version: 18.25.6.p50
  - rw61x_sb_ble_a2.bin for A2
  - 18 - Major revision
  - 25 - Feature pack
  - 6 - Release version
  - p50 - Patch number
- 802.15.4 and Bluetooth LE (up to core 4.1) firmware version: 18.34.6.p50
  - rw61x_sb_ble_15d4_combo_a2.bin for A2
  - 18 - Major revision
  - 34 - Feature pack
  - 6 - Release version
  - p50 - Patch number

Parent topic:*RW610/RW612 release notes*

Host platform

- RW610/RW612 platform running FreeRTOS
- Test tools
  - iPerf (version 2.1.9)

Parent topic:*RW610/RW612 release notes*

Wireless certification The Wi-Fi and Bluetooth certification is obtained with the following combinations.

WFA certifications

- STA | 802.11n
- STA | PMF
- STA | FFD
- STA | SVD

- STA | WPA3 SAE (R3)
- STA | 802.11ac
- STA | 802.11ax
- STA | QTT

Refer to *1*.

Note: This release supports STAUT only certifications.

Parent topic: Wireless certification

Bluetooth LE controller certification

QDID: Refer to *4*.

Parent topic: Wireless certification

Thread

Thread group: refer to *7*.

Product Name: NXP RW612 Wireless MCU with Integrated Tri-Radio

Thread version: V1.3.0

CID #: 13A109

Parent topic: Wireless certification

Matter

RW612 certification: refer to *8*.

Certificate ID: CSA23C36MAT41746-24

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi

RW610 certification: refer to *9*.

Certificate ID: CSA23C43MAT41753-50

Device type: Root Node, Thermostat

Transport: Matter over Wi-Fi and Matter over Thread

Parent topic: Wireless certification

Parent topic:*RW610/RW612 release notes*

Wi-Fi throughput

Throughput test setup

- Environment: Shield Room - Over the Air
- Access Point: Asus AX88u
- DUT: RW610/RW612
- External Client: Intel AX210
- Channel: 6 | 36
- Wi-Fi application: wifi_cli
- Compiler used to build application: armgcc
- Compiler version gcc-arm-none-eabi-13.2

- iPerf commands used in test:

TCP TX

```
iperf -c <remote_ip> -t 60
```

TCP RX

```
iperf -s
```

UDP TX

```
iperf -c <remote_ip> -t 60 -u -B <local_ip> -b 120
```

Note: The default rate is 100 Mbps.

UDP RX

```
iperf -s -u -B <local_ip>
```

Note: Read more about the throughput test setup and topology in *3*.

Parent topic: Wi-Fi throughput

STA throughput

External AP: Asus AX88u

STA mode throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	38	38	62	62
WPA2-AES	37	37	61	63
WPA3-SAE	37	37	60	61

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	39	64	64
WPA2-AES	37	38	62	64
WPA3-SAE	39	38	62	64

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	41	41	75	74
WPA2-AES	41	41	73	74
WPA3-SAE	40	41	72	73

STA mode throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	42	42	76	76
WPA2-AES	42	41	75	75
WPA3-SAE	42	41	75	74

STA mode throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	45	97	99
WPA2-AES	43	44	96	98
WPA3-SAE	42	44	97	98

STA mode throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	47	47	100	103
WPA2-AES	45	46	100	101
WPA3-SAE	47	46	100	101

Parent topic: Wi-Fi throughput

Mobile AP throughput

External client: Apple MacBook Air

Mobile AP throughput - BGN Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	39	39	62	62
WPA2-AES	39	39	61	61
WPA3-SAE	38	39	61	61

Mobile AP throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	40	40	63	63
WPA2-AES	39	39	62	61
WPA3-SAE	39	39	62	61

Mobile AP throughput - VHT Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	43	43	73	73
WPA2-AES	43	42	72	72
WPA3-SAE	43	42	73	72

Mobile AP throughput - VHT Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	44	44	74	74
WPA2-AES	43	43	74	74
WPA3-SAE	43	43	74	74

Mobile AP throughput - HE Mode | 2.4 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	48	48	95	96
WPA2-AES	47	47	98	95
WPA3-SAE	47	47	97	95

Mobile AP throughput - HE Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	49	49	96	97
WPA2-AES	48	48	101	97
WPA3-SAE	48	48	101	97

Parent topic: Wi-Fi throughput

Parent topic:*RW610/RW612 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p34 to 18.99.6.p40

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

Firmware version: 18.99.6.p40 to 18.99.6.p46

Component	Description
Wi-Fi	Fails to establish a persistent connection when the device attempts to reinvoke the second stored Persistent Group
Bluetooth	NCP cannot work after flash uart bins for both host and device side

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.6.p46 to 18.99.6.p47

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*RW610/RW612 release notes*

Known issues

Component	Description
Wi-Fi	—
Bluetooth LE	—
Zigbee	-
Coex	-

Parent topic:*RW610/RW612 release notes*

IW610 release notes

Package information

- SDK version: 25.12.00

Parent topic:*IW610 release notes*

Version information

- Wireless SoC: IW610
- Wi-Fi and Bluetooth/Bluetooth LE firmware version: 18.99.5.p86
  - 18 - Major revision
  - 99 - Feature pack
  - 5 - Release version
  - p86 - 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.

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	37	37	60	62
WPA2-AES	36	37	59	61
WPA3-SAE	36	37	59	61

STA mode throughput - AN Mode | 5 GHz Band | 20 MHz

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	35	40	64	65
WPA2-AES	34	39	62	64
WPA3-SAE	35	39	77	76

STA mode throughput - VHT Mode | 2.4 GHz Band | 20 MHz (HT)

Protocol	TCP (Mbit/s)	TCP (Mbit/s)	UDP (Mbit/s)	UDP (Mbit/s)
Direction	TX	RX	TX	RX
OpenSecurity	41	40	72	72
WPA2-AES	40	40	72	72
WPA3-SAE	40	40	72	71

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	38	42	77	76
WPA2-AES	37	41	75	75
WPA3-SAE	37	40	75	75

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	45	44	93	96
WPA2-AES	43	43	93	95
WPA3-SAE	44	43	93	96

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	42	46	94	100
WPA2-AES	42	45	94	101
WPA3-SAE	41	45	94	101

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
Open security	48	44	61	61
WPA2-AES	47	43	59	59
WPA3-SAE	47	43	59	59

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
Open security	49	46	64	63
WPA2-AES	48	45	62	61
WPA3-SAE	48	45	62	61

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
Open security	54	50	73	73
WPA2-AES	53	49	73	72
WPA3-SAE	52	49	73	72

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
Open security	54	51	71	70
WPA2-AES	53	50	71	70
WPA3-SAE	52	50	71	70

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
Open security	59	56	93	90
WPA2-AES	57	53	94	84
WPA3-SAE	57	53	94	84

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
Open security	61	58	96	91
WPA2-AES	59	56	98	85
WPA3-SAE	59	55	98	85

Parent topic: Wi-Fi throughput

Parent topic:*IW610 release notes*

Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p66 to 18.99.5.p76

Component	Description
Wi-Fi	The P2P client connection fails when an attempt is made to connect after the P2P Group Owner (P2P-GO) has been stopped.

Parent topic: Bug fixes and/or feature enhancements

Firmware version: 18.99.5.p76 to 18.99.5.p79

Component	Description
Wi-Fi	Enabled mbedtls 3.x

Parent topic: Bug fixes and/or feature enhancements

Parent topic:*IW610 release notes*

Known issues

Component	Description
NA	

Parent topic:*IW610 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
SAE	Simultaneous authentication of equals
STA	Station

continues on next page

Table 8 – continued from previous page

Abbreviation	Definition
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)
10. Application note - AN14634 – Kconfig Memory Optimizer (link)

1.8.2 EdgeFast Bluetooth

Currently we provide pdf version of those documentation, later release may convert the pdf documentation to markdown for better review and aligned format.

- EdgeFast BT PAL API Reference Manual pdf.

MCUXpressoSDK EdgeFast Bluetooth Protocol Abstraction

Introduction This document provides an overview of the EdgeFast Bluetooth Protocol Abstraction Layer stack software based on FreeRTOS OS on the NXP board with variant wireless module chipsets. This document covers hardware setup, build, and usage of the provided demo applications.

Stack API Reference EdgeFast Bluetooth Protocol Abstraction Layer is a wrapper layer on top of the bluetooth host stack. Zephyr Bluetooth host stack API is used as the basis of the EdgeFast Bluetooth Protocol Abstraction Layer with some enhancement on A2DP/SPP/HFP.

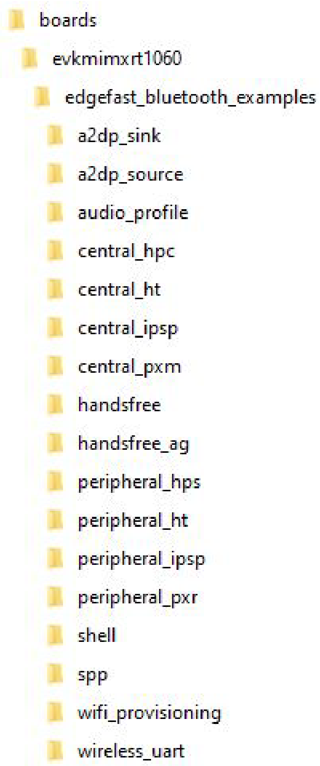
The APIs of the EdgeFast Bluetooth Protocol Abstraction Layer host stack are described in the EdgeFast Bluetooth Protocol Abstraction Layer RM document.

Note: The online document of the Zephyr Bluetooth Host stack is available here: https://docs.zephyrproject.org/latest/reference/bluetooth/index.html.

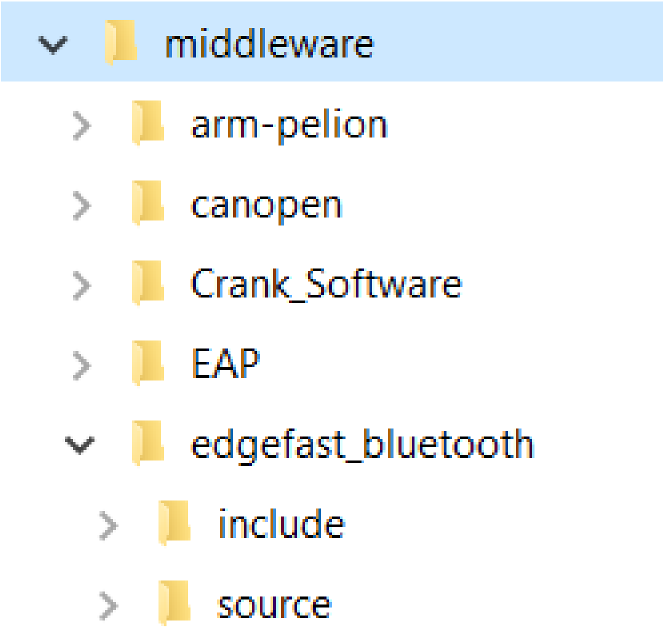
Parent topic:*Introduction*

Overview The EdgeFast Bluetooth Protocol Abstraction Layer host stack software is built based on MCUXpresso SDK. The following chapter uses RT1060 as an example, other boards have similar folder structure and corresponding document.

Folder structure The following figure shows the EdgeFast Bluetooth examples folder structure.



The following figure shows the EdgeFast Bluetooth Protocol Abstraction Layer host stack folder structure.



The following table provides information regarding the structure and description.

|**Folder**

|**Description**

| |———|————| |*boards/*

CMSIS/

devices/

docs/

middleware/

rtos/

tools/

|MCUXpresso SDK directory. Refer to Chapter 5

Release contents of MCUXpresso SDK Release Notes at *root/docs/ MCUXpresso SDK Release Notes for EVK-MIMXRT1060.pdf* to *know the details*

| |*boards/<board>/wireless/edgefast_bluetooth_examples*

|EdgeFast Bluetooth Protocol Abstraction Layer host stack example projects| |*middleware/wireless/edgefast_bluetooth*

|EdgeFast Bluetooth Protocol Abstraction Layer host stack source code

|

The EdgeFast Bluetooth folder includes two subfolders:

- **include**: This subfolder includes EdgeFast Bluetooth Protocol Abstraction Layer host stack headers.
- **source**: This subfolder includes EdgeFast Bluetooth Protocol Abstraction Layer host stack source code based on the Ethermind Bluetooth host stack APIs.

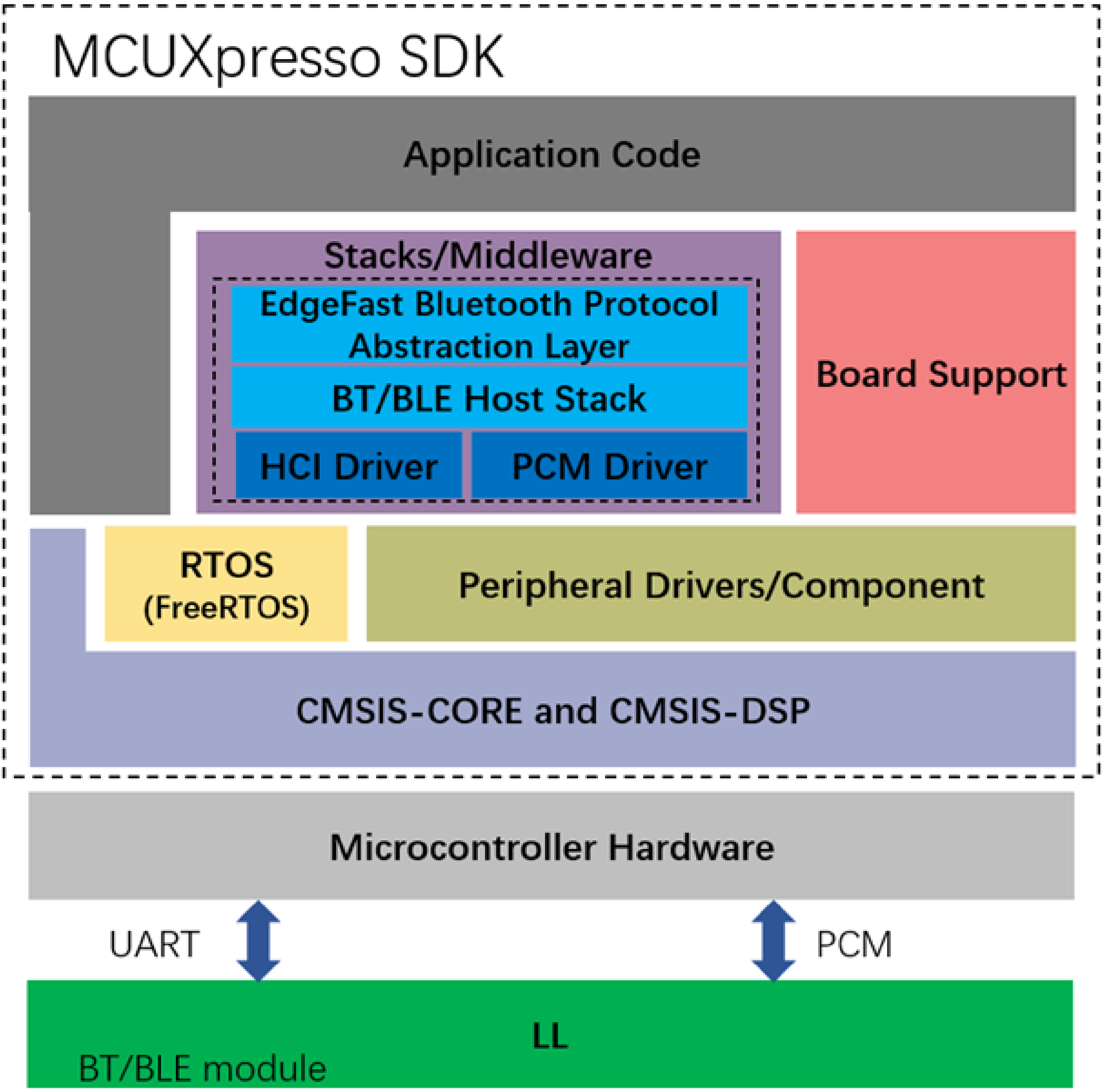
Parent topic:*Overview*

Architecture The figure Architecture of EdgeFast Bluetooth Protocol Abstraction Layer demo in MCUXpresso SDK below shows that the EdgeFast Bluetooth Protocol Abstraction Layer host stack is integrated into the MCUXpresso SDK as a middleware component. It leverages the RTOS, the board support, the peripheral driver/component, and other components in the MCUXpresso SDK. The Bluetooth application is built on top of the EdgeFast Bluetooth Protocol Abstraction Layer host stack and supports different peripheral features, Bluetooth features, and different RTOSes required by the user.

MCUXpresso SDK has the dual-chip architecture defined by EdgeFast Bluetooth Protocol Abstraction Layer project, the Bluetooth application code, and the EdgeFast Bluetooth Protocol Abstraction Layer host stack running on the reference board. For example, MIMXRT1060-EVK and the Linker Layer (LL) run on the Bluetooth modules like AW-AM457-USD, Murata Type 1XK, and Murata Type 1ZM and has single-chip architecture. Bluetooth Host stack and LL runs on the same chip, and communicate with Internal Communication Unit (IMU).

The communication between the host stack and the LL is implemented via the standard HCI UART interface and PCM interface for voice, or the IMU interface.

For details about the different components in MCUXpresso SDK, see *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf*. For details on possible hardware rework requirements, see the hardware rework guide document of the relative board. For example, Hardware Rework Guide for EdgeFast BT



PAL.

Parent topic:*Overview*

Features This section provides an overview of Bluetooth features, toolchain support, and RTOS support.

Bluetooth features

- Bluetooth 5.0 compliant
- Protocol support
  - L2CAP, GAP, GATT, RFCOMM, SDP, and SM

    Note: The Enhanced Attribute (EATT) protocol is not supported in the current version. However, the support will be available in a future version.
- Classic profile
  - SPP, A2DP, and HFP
- LE profile
  - HTP, PXP, IPSP, HPS
- Integrated the Fatfs based on USB Host MSD in SDK
- Digital Audio Interface including PCM interface for HFP

Parent topic: Features

Toolchain support

- IAR Embedded Workbench for ARM®
- MCUXpresso IDE
- Keil® MDK/µVision
- Makefiles support with GCC from Arm Embedded

Note: For details on IDE Development tools version details, see Section 3, Development tools in MCUXpresso SDK Release Notes (document MCUXSDKMIMXRT106XRN). The Release Notes document is available at *root/docs/ MCUXpresso SDK Release Notes for EVK-MIMXRT1060.pdf*.

Parent topic: Features

RTOS support

- FreeRTOSTMOS

Note: The FreeRTOS static allocation feature is required by Edgefast Bluetooth. The macro configSUPPORT_STATIC_ALLOCATION needs to be set to enable this feature.

Parent topic: Features

Parent topic:*Overview*

Examples list

- The following examples are provided. Not all the examples are implemented on all the boards. See the board package for a list of the implemented examples.

- **central_hpc (central http proxy service client)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for HPS Server and programs a set of characteristics that configures a Hyper Text Transfer Protocol (HTTP) request, initiates request, and reads the response once connected.
- **central_ht (central health thermometer)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for health thermometer sensor and reports the die temperature readings once connected.
- **central_ipsp (central Internet protocol support profile)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes connection to the peripheral with the strongest signal. The application specifically looks for IPSP Service and communicates between the devices that support IPSP. Once connected, the communication is done using IPv6 packets over the Bluetooth Low Energy transport.
- **central_pxm (central proximity monitor)**: Demonstrates a basic Bluetooth Low Energy Central role functionality. The application scans for other Bluetooth Low Energy devices and establishes a connection to the peripheral with the strongest signal. The application specifically looks for Proximity Reporter.
- **peripheral beacon**: Demonstrates the Bluetooth Low Energy Peripheral role, This application implements types of beacon applications.
  * **beacon**: Demonstrates the Bluetooth Low Energy Broadcaster role functionality by advertising Company Identifier, Beacon Identifier, UUID, A, B, C, RSSI.
  * **Eddystone**: The Eddystone Configuration Service runs as a GATT service on the beacon while it is connectable and allows configuration of the advertised data, the broadcast power levels, and the advertising intervals.
  * **iBeacon**: Demonstrates the Bluetooth Low Energy Broadcaster role functionality by advertising an Apple iBeacon.
- **peripheral_hps (peripheral http proxy service)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the HTTP Proxy GATT Service.
- **peripheral_ht (peripheral health thermometer)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the HT (Health Thermometer) GATT Service. Once a device connects, it generates dummy temperature values.
- **peripheral_ipsp (peripheral Internet protocol support profile)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the Internet Protocol Support GATT Service.
- **peripheral_pxr (peripheral proximity reporter)**: Demonstrates the Bluetooth Low Energy Peripheral role. The application specifically exposes the Proximity Reporter (including LLS, IAS, and TPS) GATT Service.
- **wireless uart**: The application automatically starts advertising the wireless uart service and connects to the wireless uart service after the role switch. The wireless UART service is a custom service that implements a custom writable ASCII Char characteristic (UUID: 01ff0101-ba5e-f4ee-5ca1-eb1e5e4b1ce0) that holds the character written by the peer device.
- **spp (serial prot profile)**: Application demonstrates the use of the SPP feature.
- **handsfree**: Application demonstrating usage of the Hands-free Profile (HFP) feature.

- **handsfree_ag**: Application demonstrating usage of the Hands-free Profile Audio Gateway (HFP-AG) feature.
- **a2dp_sink**: Application demonstrating how to use the a2dp sink feature.
- **a2dp_source**: Application demonstrating how to use the a2dp source feature.
- **audio_profile**: Demonstrates the following functions.
  * There are five parts working in the demo: AWS cloud, Android app, audio demo (running on RT1060), U-disk, and Bluetooth headset.
  * With an app running on the smartphone (Android phone), the end user connects to the AWS cloud and controls the audio demo running on the RT1060 EVK board through AWS cloud. Some operations like play, play next, and pause are used to control the media play functionalities.
  * Audio demo running on the RT1060 EVK board connects to the AWS through WiFi. A connection establishes between the RT1060 EVK board and a Bluetooth headset. To get the media resource (mp3 files) from the U-disk, an HS USB host is enabled, and a U-disk with mp3 files is connected to RT1060 EVK board via the USB port. The audio demo searches the root directory of the U-disk for the music files (only mp3 files are supported) and uploads the song file list to AWS. The song list is shown in the app running on the smartphone. The music can then be played out via the Bluetooth headset once end user controls the app to play the mp3 file.
- **wifi_provisioning**: Demonstrates the WiFi provisioning service that safely sends credential from phone to device over Bluetooth low energy. By default, AWS Wi-Fi provisioning demo starts advertising if the Wi-Fi access point (AP) is not configured and waits for the Wi-Fi AP configuration. After connecting to the Android APK, the demo executes the request from cellphone and sends the response. When the Wi-Fi AP is configured, the Shadow demo connects to the AWS via Wi-Fi and publishes the configured Wi-Fi AP information.
- **shell**: Shell application demonstrating the shell mode of the simplified Adapter APIs.

Parent topic:*Overview*

Hardware For dual-chip implementation, the Bluetooth demo runs on a (reference board) along with the ported EdgeFast Bluetooth Protocol Abstraction Layer API host stack. The Linker Layer (LL) runs on a wireless module. A standard UART HCI and PCM is used to communicate between the two boards, the IMU is used to communicate in between. The Bluetooth host and controller stack run on different boards. The demo hardware requires two different boards; a development board for host stack and application and a wireless module adapter board for controller running. For example, the evkmimxrt1060 and uSD-15x15 Adapter Board for AW-AM457-uSD board, or any of the supported Murata modules with the Murata uSD-M.2 adapter. For details on the board hardware requirement and board setting, see the following documents. For one-chip implementation, the Bluetooth demo, EdgeFast Bluetooth Protocol Abstraction Layer API host stack, and LL run on one chip and they communicate with IMU.

- Hardware rework guide document of the relative board, Hardware Rework Guide for MIMXRT1060-EVK and AW-AM457-uSD, or Hardware Interconnection Guide for i.MX RT EVKs and Murata M.2 modules.
- Readme file of the examples.

Reference boards list

- MIMXRT1170: For details, see the quick start guide of this reference board (MIMXRT1170).
- MIMXRT685-EVK: For details, see the quick start guide of this reference board (MIMXRT685-EVK).

- MIMXRT595-EVK: For details, see the quick start guide of this reference board. (MIMXRT595-EVK).
- MIMXRT1050-EVKB: For details, see the quick start guide of this reference board (MIMXRT1050-EVKB).

Parent topic:*Hardware*

Dual-chip wireless module list

Module	HCI
uSD-15x15 Adapter Board for AW-AM457-uSD	UART
uSD-15x15 Adapter Board for AW-CM358-uSD	UART
uSD-15x15 Adapter Board for AW-AM510-uSD	UART
AW-CM358MA	UART
AW-CM510MA	UART
K32W061	UART
Murata uSD-M.2 Adapter (LBEE0ZZ1WE-uSD-M2) and Embedded Artists 1ZM M.2 Module (EAR00364)	UART
Murata uSD-M.2 Adapter (LBEE0ZZ1WE-uSD-M2) and Embedded Artists 1XK M.2 Module (EAR00385)	UART

For details on AzureWave module, see the quick start guide of this reference board AW-AM457-uSD, AW-CM358-uSD, AW-CM358MA, AW-AM510-uSD, AW-CM510MA, and K32W061.

For Murata documentation, refer to the Quick Start Guide and User Guide here.

Note: The boards and wireless module lists are not random combination. For the wireless module support list of specific board, see the readme.txt of each example.

Parent topic:*Hardware*

Demo This topic lists the steps to run a demo application using IAR, steps to run a demo application using MCUXpresso IDE, and steps to download LL firmware from the reference board. The following chapter uses RT1060 and peripheral_ht as an example.

Before you run the example, see the readme.txt in current the peripheral_ht directory and the Hardware Rework Guide for EdgeFast BT PAL document to set the jumper and connect the wireless module with development board.

The uSD type wireless module is similar to the Development board connector in the Run an IAR example section. If the module is M2 type, connect the module to the onboard M2 interface.

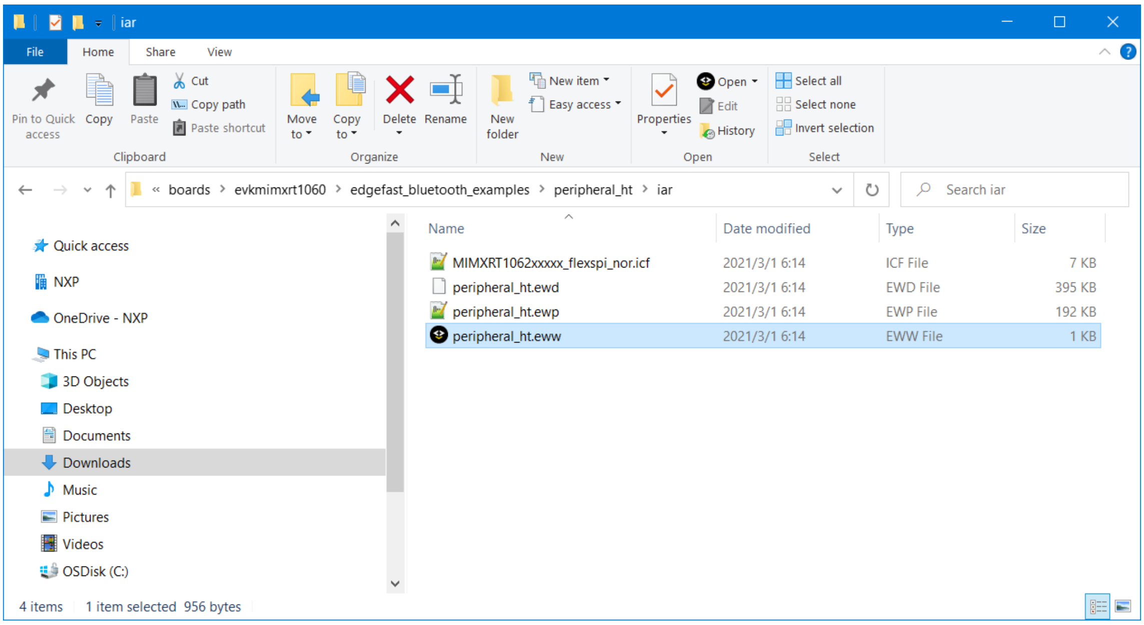
Run a demo application using IAR This document uses EVKRT1060 EdgeFast Bluetooth Protocol Abstraction Layer API example to describe the steps to open a project, build an example, and run a project. For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide*(document MCUXSDKGSUG) at*root/docs/Getting Started with MCUXpresso SDK.pdf*.

Open an IAR example For the IAR Embedded Workbench, unpack the contents of the archive to a folder on a local drive.

1. The example projects are available at:

   <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/iar

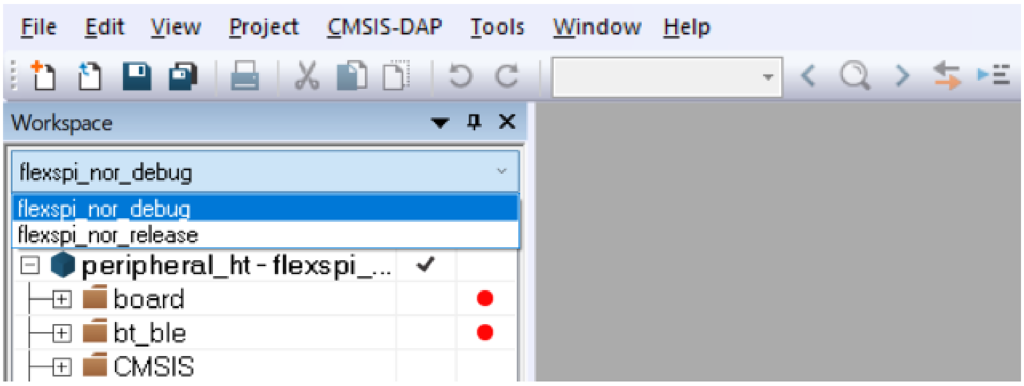
2. Open the IAR workspace file. For example, the highlighted *.eww format file



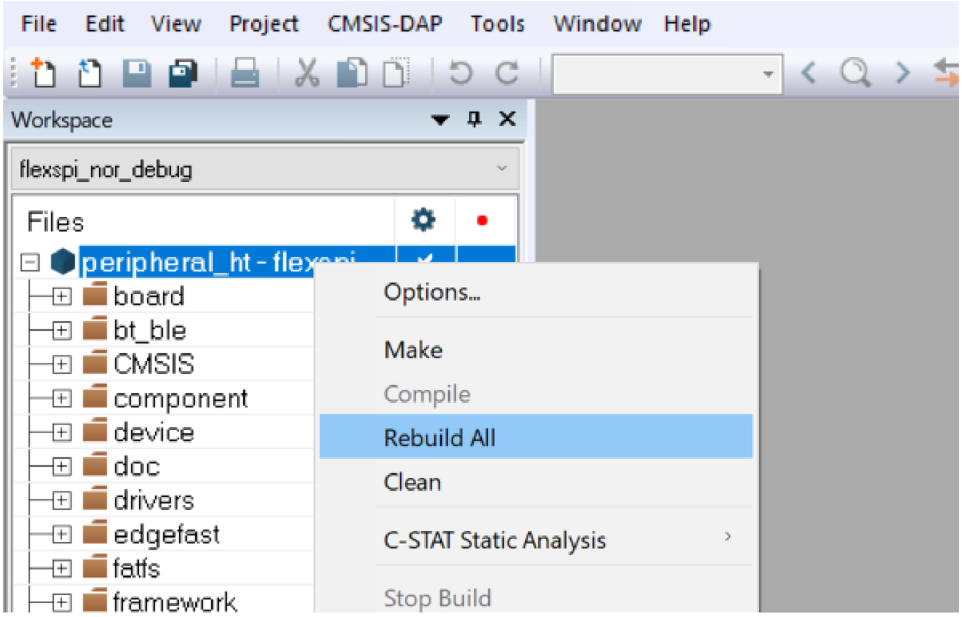
Parent topic: Run a demo application using IAR

Build an IAR example

1. Select flexspi_nor_debug or flexspi_nor_release configurations from the drop-down selector above the project tree in the workspace.



2. Build the EdgeFast Bluetooth Protocol Abstraction Layer project.

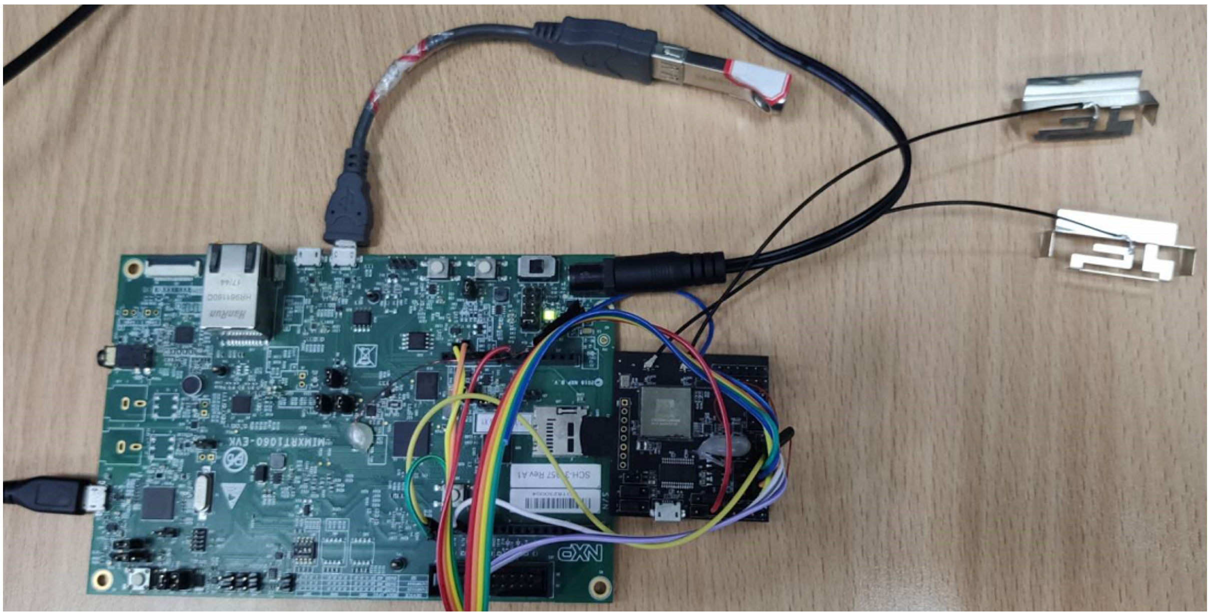


Note: Wireless module does not have flash hardware and requires 512 KB image loaded from board (such as RT1060) on system startup. The 512 KB image is kept on RT1060 side and only flexspi_nor target is supported for Bluetooth examples. Other targets are not supported because memory size limit.

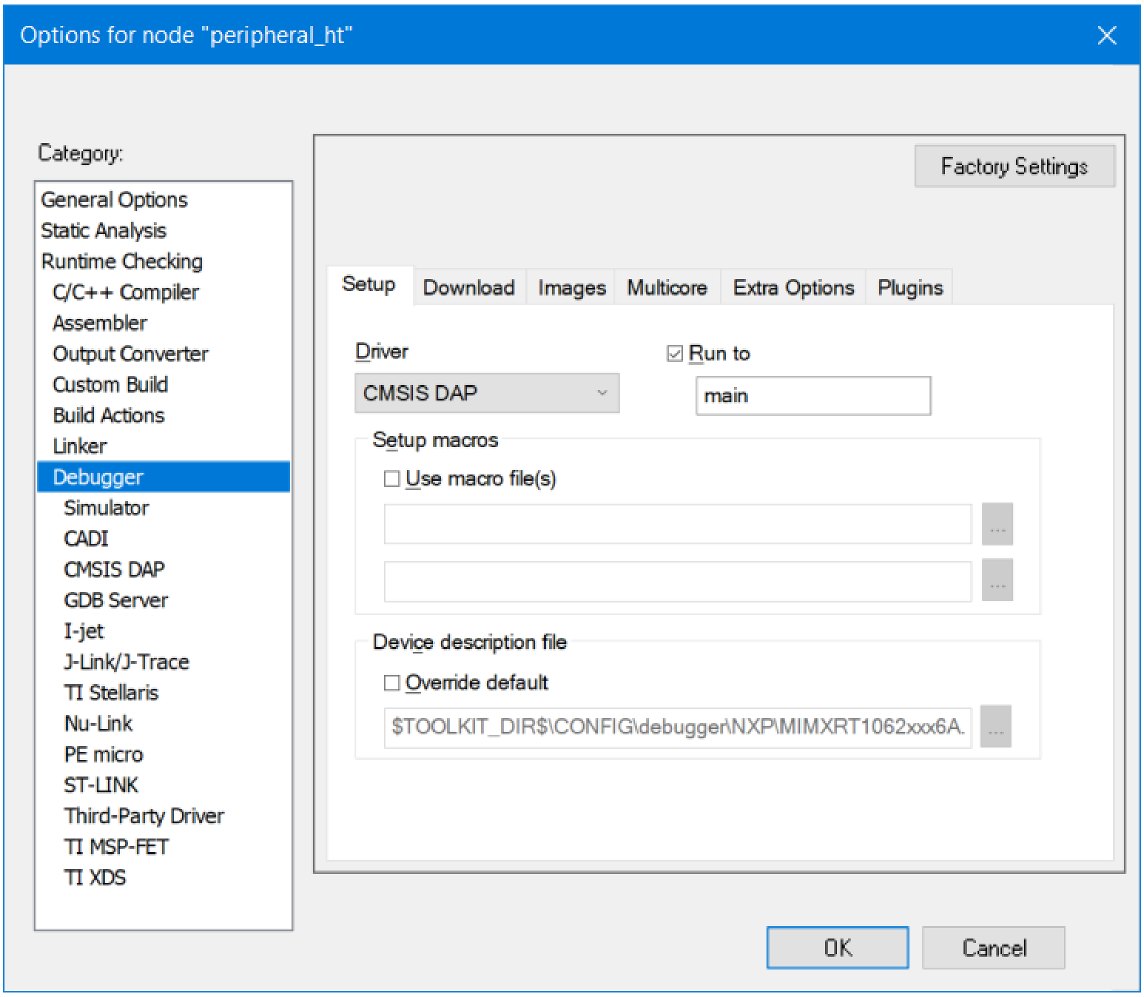
Parent topic: Run a demo application using IAR

Run an IAR example This document uses the peripheral_ht as an example to describe the steps to run an example. For details on other projects and compilers, see the readme file in the corresponding example directory.

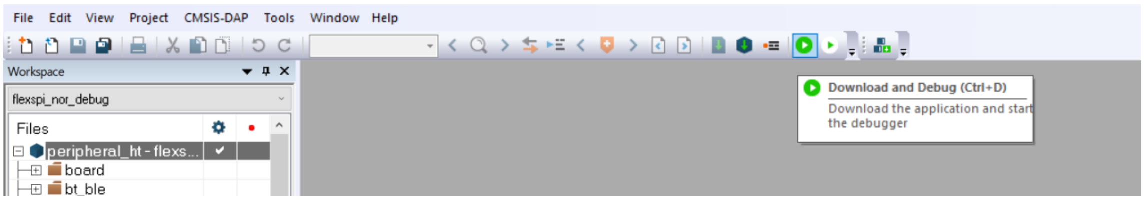
The following figure shows the connection of RT1060 and the uSD wireless module.



1. Connect the USB debug console port to PC. For example, connect J14 of EVKRT1060 to the PC.
2. Connect a 5 V power source to the J1 jack in the Wireless module board.
3. Make the appropriate debugger settings in the project options window, as shown in the figure below.



4. Click the **Download and Debug** button to flash the executable onto the board, as shown in the following figure. After the download is complete, if you must test the function of HFP, stop IAR debugging, and then connect the PCM interface. Reset the target board by manually.



5. Linker layer (LL) Firmware running in wireless module loads from EVKRT1060 by SDIO interface, so need take a bit time to download the LL firmware, “Initialize AW-AM457-uSD Driver” prints in the debug console. For example, it depends on the firmware. For details, see readme.txt.

Note: The projects are configured to use “CMSIS DAP” as the default debugger. Ensure that the OpenSDA chip of the board contains a CMSIS. DAP firmware or that the debugger selection corresponds to the physical interface used to interface to the board.

Parent topic: Run a demo application using IAR

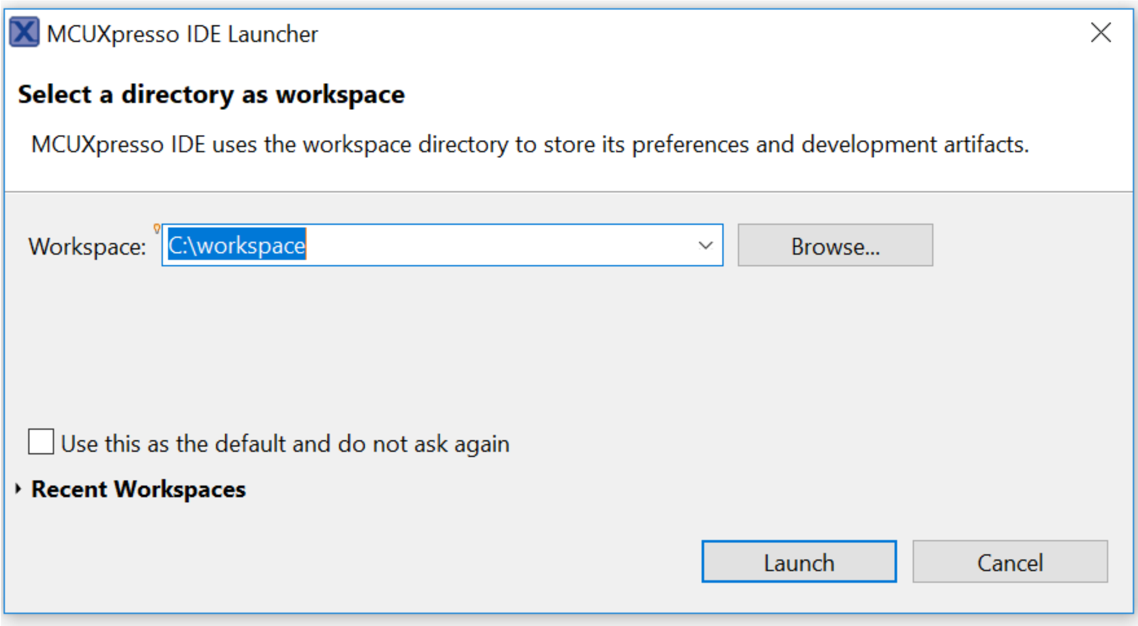
Parent topic:*Demo*

Run a demo application using MCUXpresso IDE This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MCUXpresso IDE.

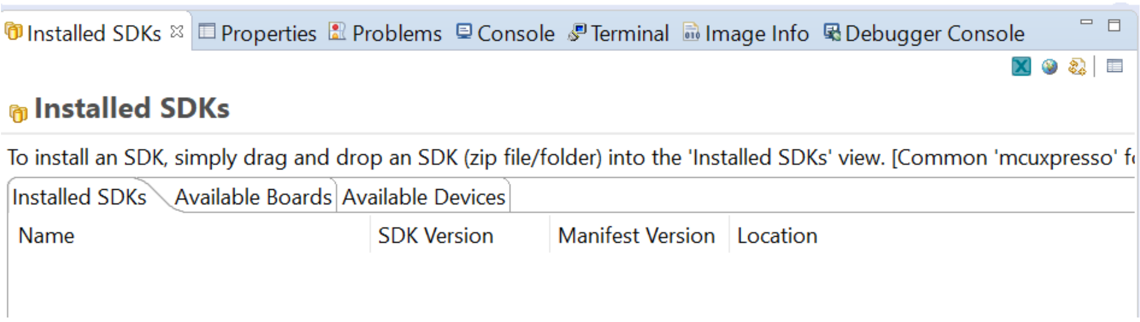
For details, see Section 3 in *Getting Started with MCUXpresso SDK User's Guide* (document MCUXSDKGSUG) at *root/docs/Getting Started with MCUXpresso SDK.pdf* and refer to the readme file in the corresponding demo's directory.

Open an MCUXpresso IDE example

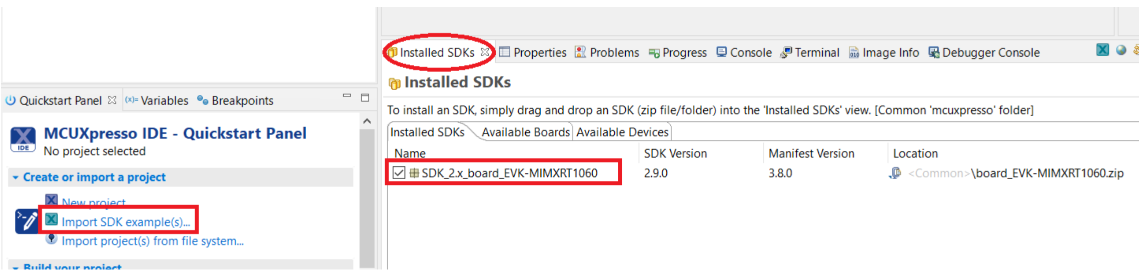
1. Open MCUXpresso IDE and open an existing or a new workspace location.



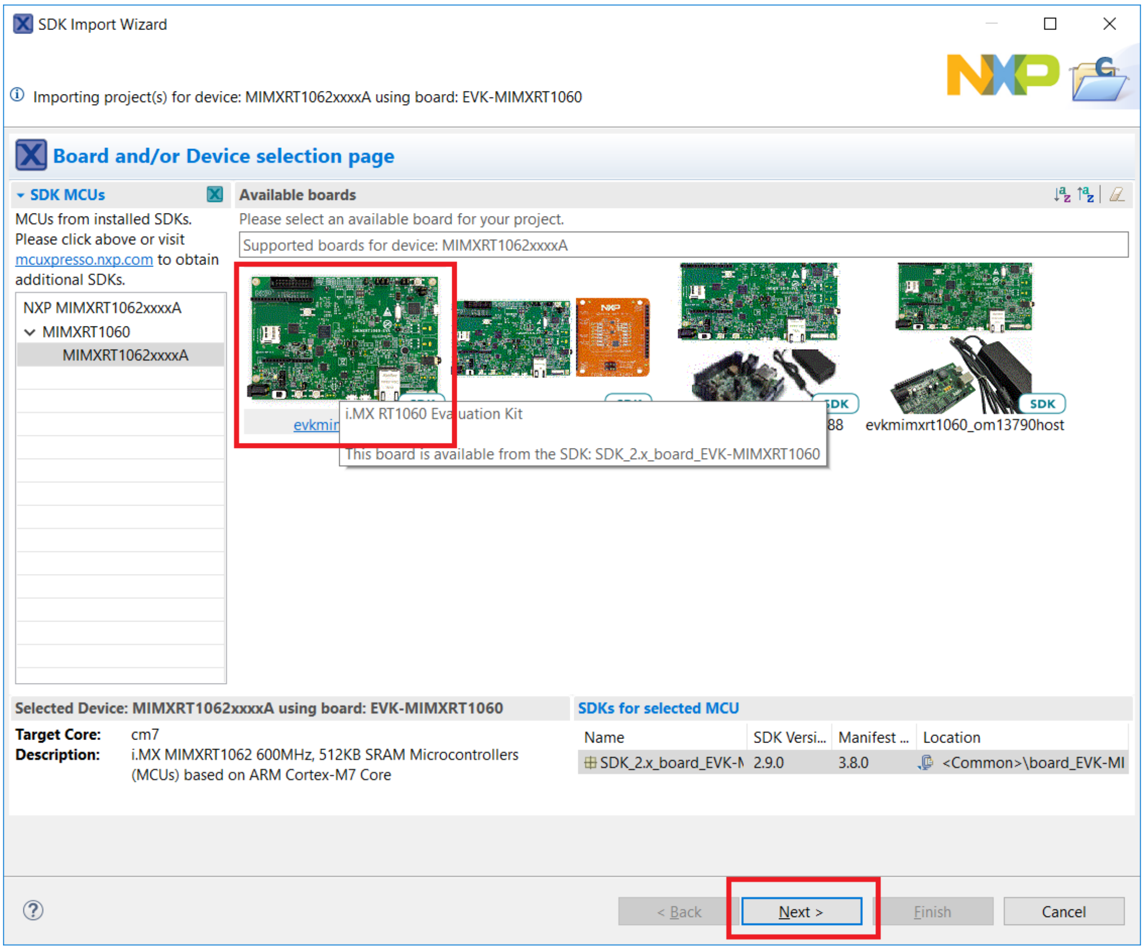
2. Drag and drop the package archive into the MCUXpresso Installed SDKs area in the lower right of the main window.



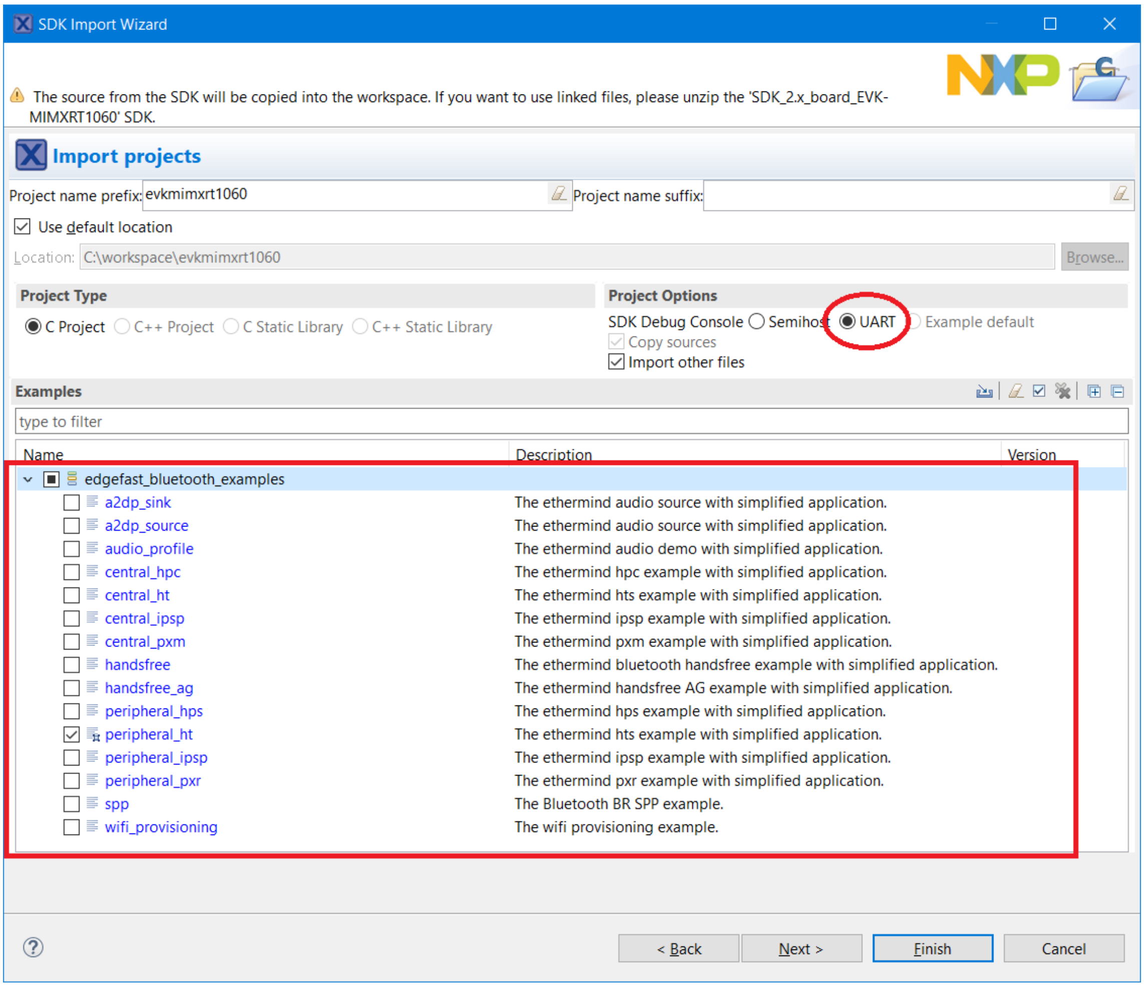
3. After the SDK is loaded successfully, select the **Import the SDK examples(s)...** to add examples to your workspace.



4. Select the evkmimxrt1060 board and click the **Next** button to select the desired example(s).



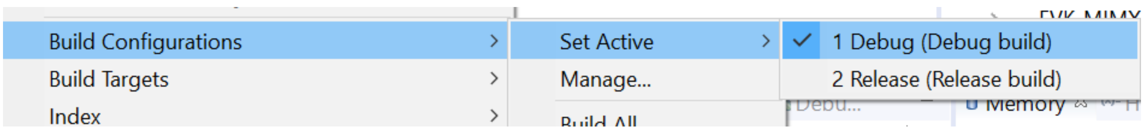
5. Select the evkmimxrt1060 board EdgeFast Bluetooth example. For example, peripheral_ht.
6. Ensure to change SDK debug console from **Semihost** to **UART**.
7. Click **Finish**.



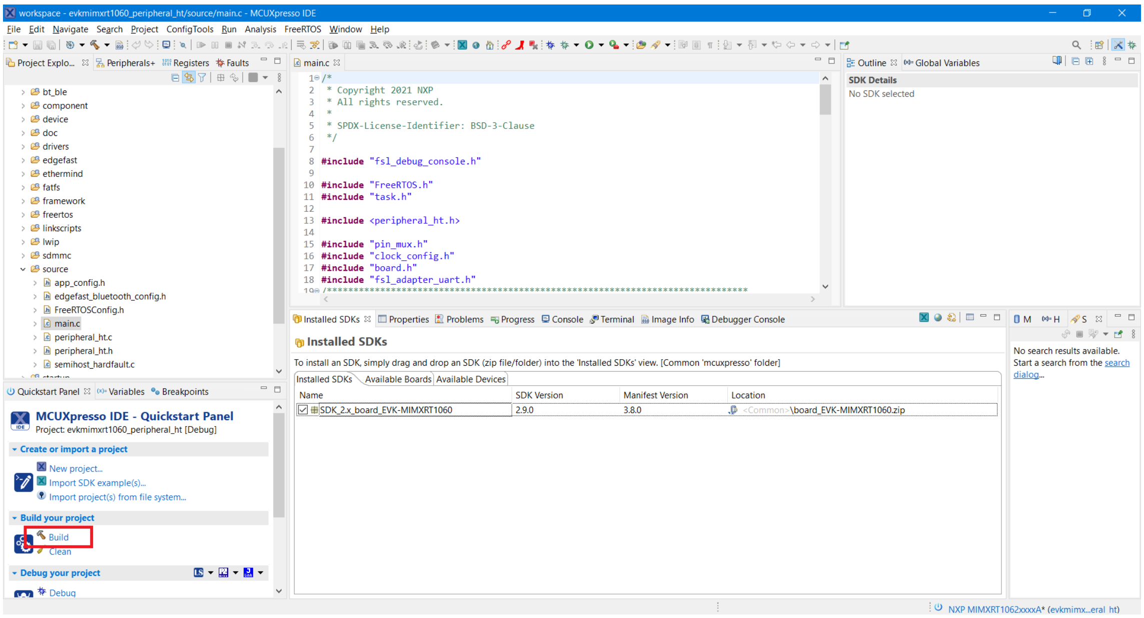
Parent topic: Run a demo application using MCUXpresso IDE

Build an MCUXpresso IDE example

1. Select desired target for your project.



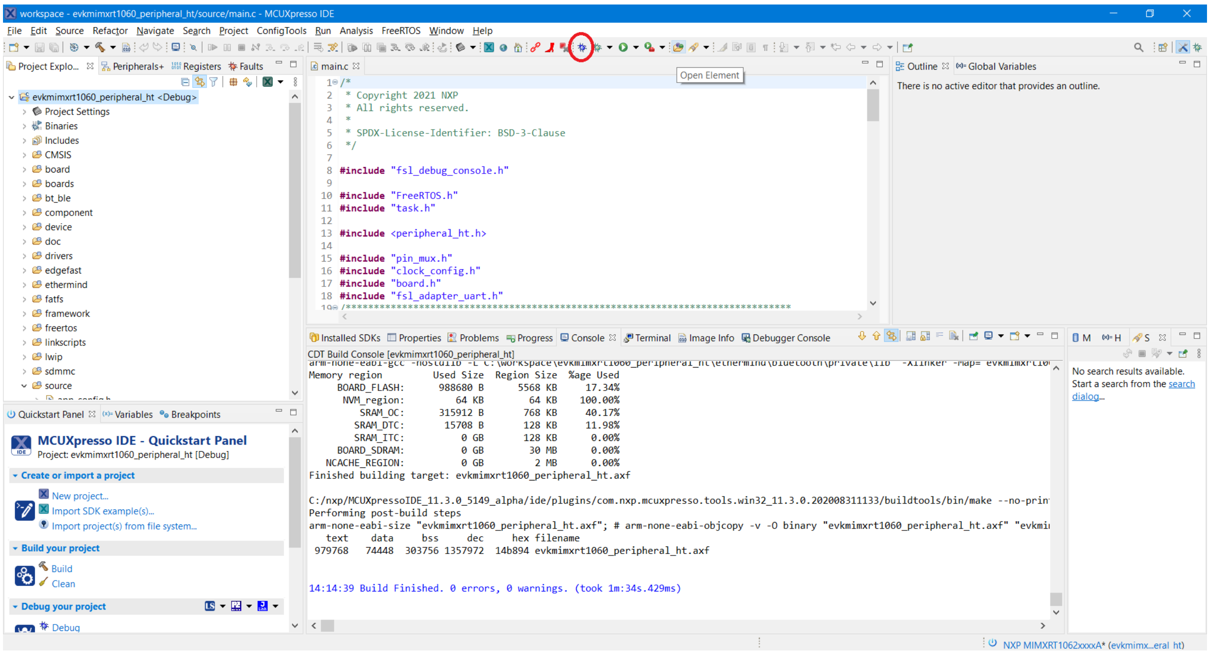
2. Build MCUXpresso IDE EdgeFast Bluetooth Protocol Abstraction Layer project.



Parent topic: Run a demo application using MCUXpresso IDE

Run an MCUXpresso IDE example For MCUXpresso IDE project running, all steps are similar to Run an IAR example except the steps of downloading image from compiler.

To download MCUXpresso IDE image to board, click the **Debug** button to download the executable file onto the board.



Parent topic: Run a demo application using MCUXpresso IDE

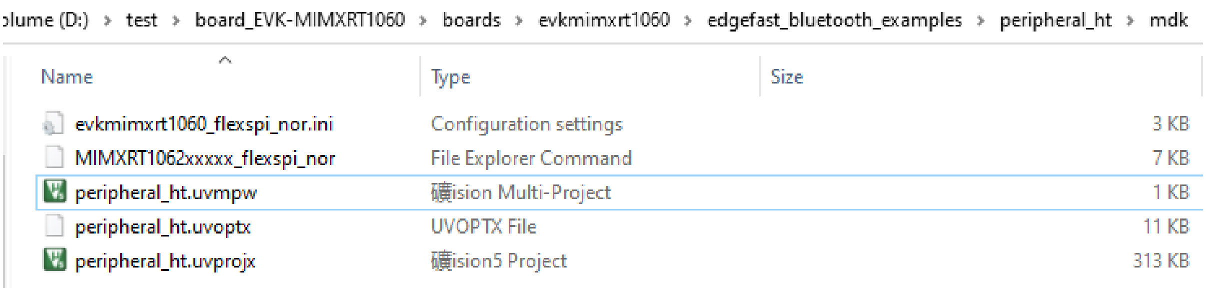
Parent topic:*Demo*

Run a demo application using MDK This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MDK.

For details, see the related section in the Getting Started with MCUXpresso SDK User's Guide (document: MCUXSDKGSUG) in the directory *root/docs/* and the readme file in the corresponding demo's directory.

Open an MDK project For the IAR Embedded Workbench, unpack the contents of the archive to a folder on a local drive.

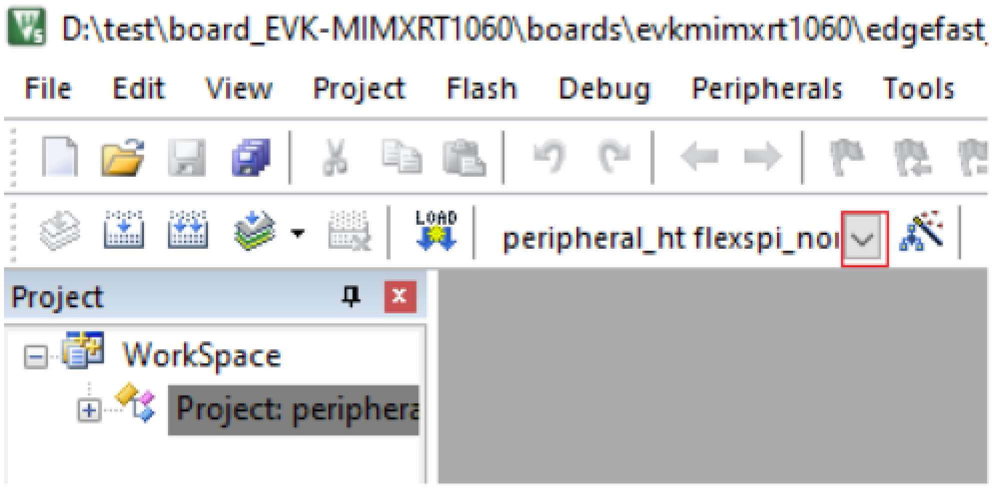
1. The example projects are available at: <root>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/mdk.
2. Open the mdk workspace file. For example, the highlighted *.uvmpw format file.



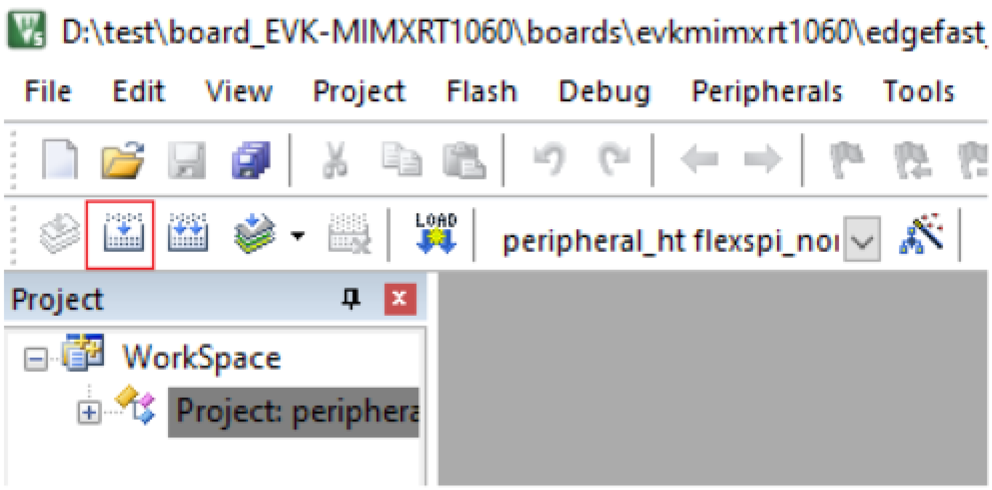
Parent topic: Run a demo application using MDK

Build an MDK example To build an MDK example:

1. Select *flexspi_nor_debug* or *flexspi_nor_release* configurations from the drop-down selector above the project tree in the workspace.



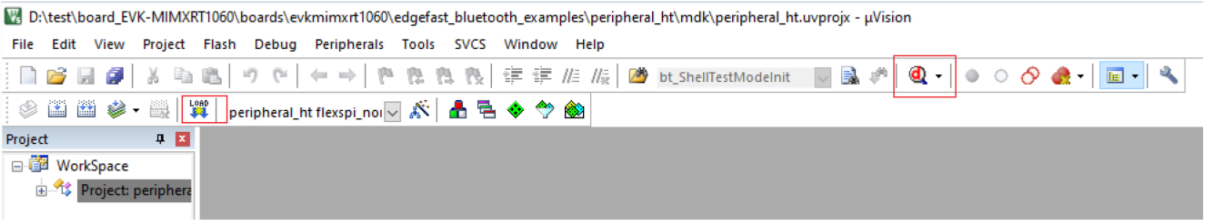
2. Click the highlighted icon to build the EdgeFast Bluetooth Protocol Abstraction Layer project.



Parent topic: Run a demo application using MDK

Run an MDK example For MDK project running, all steps are similar to Run an IAR example except the steps of downloading image from compiler.

To download the MDK image to the board, click the **Debug** button. The executable file downloads to the board.



Parent topic: Run a demo application using MDK

Parent topic:*Demo*

Run a demo application using Arm GCC This document uses peripheral_ht example to describe the steps to open a project, build an example, and run a project on MDK.

For details, see the related section in *Getting Started with MCUXpresso SDK User's Guide* (document: MCUXSDKGSUG) at *root/docs/* and the readme file in the corresponding demo's directory.

Setup tool chains See the section "Run a demo using Arm GCC" of getting start document. For example, *Getting Started with MCUXpresso SDK for MIMXRT1160-EVK*.

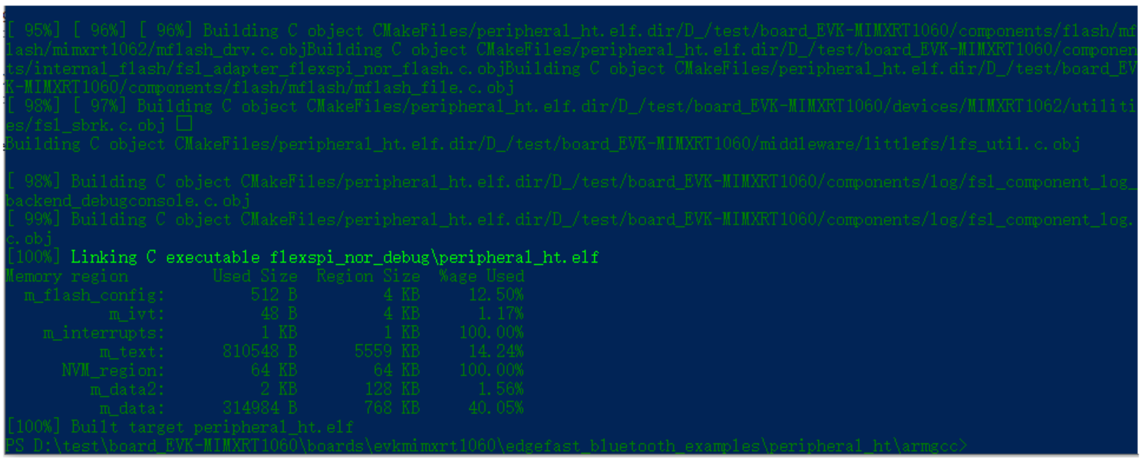
Parent topic: Run a demo application using Arm GCC

Build a GCC example To build a GCC example:

1. Change the directory to the project directory: <install_dir>\boards\evkmimxrt1060\edgefast_bluetooth_examples\peripheral_ht\armgcc.
2. Run the build script.

   For windows, the script is build_flexspi_nor_debug.bat/ build_flexspi_nor_release.bat.

   The build output is shown in the following figure.



Parent topic: Run a demo application using Arm GCC

Run a GCC example Refer to the section "Run a demo using Arm GCC" of the getting start document. For example, see Getting Started with MCUXpresso SDK for MIMXRT1060-EVK. The peripheral_ht.elf is the target to download.

Parent topic: Run a demo application using Arm GCC

Parent topic:*Demo*

Download Linker Layer firmware from the reference board Download the Linker Layer (LL) Firmware from Reference board EVKRT1060 by SDIO interface before running the Bluetooth Controller stack. The LL download is necessary because wireless module does not support flash.

Parent topic:*Demo*

Change board-specific parameters There are some board-specific parameters that can be changed in the application layer for EdgeFast BT PAL.

Change HCI UART parameters Since the controller can support different baud rates, the demo provides an interface with configurable baud rates. The function*controller_hci_uart_get_configuration* is used to get HCI UART parameters, including the instance, default baud rate, which depends on the controller, running baud rate which defined by macro BOARD_BT_UART_BAUDRATE and so on. If this function returns '0' and the running baud rate is inconsistent with the default baud rate, EdgeFast BT PAL switches the baud rate of the controller to the running baud rate.

Parent topic: Change board-specific parameters

Change USB Host stack parameters Since the board supports multiple USB ports, the demo provides a configurable interface for USB Host stack. The function*USB_HostGetConfiguration* received the instance of USB for EdgeFast BT PAL. For the case where there is a USBPHY, the demo configures the properties of the PHY through*USB_HostPhyGetConfiguration*.

Note: There are series of hex bytes printed on the console after the wireless module resets. However, it does not impact the EdgeFast BT PAL application running.

Parent topic: Change board-specific parameters

Parent topic:*Demo*

Known issues This section provides a list of known issues in the release package.

Notes This section provides a list of notes to use EdgeFast Bluetooth stack

- the follow configuration items related to resource needs more attention
  - CONFIG_BT_MAX_CONN The max connections that can be created.
  - CONFIG_BT_MAX_PAIRED The max supported paired devices.
  - CONFIG_BT_BUF_EVT_RX_COUNT The max received hci events and acl data packets at one time if the sys work queue task is blocked. One example is: when LE connection is created and HCI_LE_Enhanced_Connection_Complete is received, the sys work queue task is busy with processing the HCI_LE_Enhanced_Connection_Complete. If the received hci events exceed CONFIG_BT_BUF_EVT_RX_COUNT, it may leads potential issue, please increase value of the macro.
- All the EdgeFast Bluetooth API should be called only after EdgeFast Bluetooth is initialized.
- Don't send HCI cmd from the sys work queue task or any stack's callbacks.

EdgeFast BT PAL configuration documentation **CONFIG_BT_BUF_RESERVE**

Buffer reserved length, suggested value is 8.

CONFIG_BT_SNOOP

Whether enable bt snoop feature, 0 - disable, 1 - enable.

CONFIG_BT_HCI_CMD_COUNT

Number of HCI command buffers, ranging from 2 to 64. Number of buffers available for HCI commands Range 2 to 64 is valid.

CONFIG_BT_RX_BUF_COUNT

Number of HCI RX buffers, ranging from 2 to 255. Number of buffers available for incoming ACL packets or HCI events from the controller Range 2 to 255 is valid.

CONFIG_BT_RX_BUF_LEN

Maximum supported HCI RX buffer length, ranging from 73 to 2000. Maximum data size for each HCI RX buffer. This size includes everything starting with the ACL or HCI event headers. Note that buffer sizes are always rounded up to the nearest multiple of 4, so if this Kconfig value is something else then there is some wasted space. The minimum of 73 has been taken for LE SC which has an L2CAP MTU of 65 bytes. On top of this, The L2CAP header (4 bytes) and the ACL header (also 4 bytes) which yields 73 bytes. Range is 73 to 2000.

CONFIG_BT_HCI_RESERVE

Reserve buffer size for user. Headroom that the driver needs for sending and receiving buffers. Add a new 'default' entry for each new driver.

CONFIG_BT_DISCARDABLE_BUF_COUNT

Number of discardable event buffers, if the macro is set to 0, disable this feature, if greater than 0, this feature is enabled. Number of buffers in a separate buffer pool for events which the HCI driver considers discardable. Examples of such events could be , for example, Advertising Reports. The benefit of having such a pool means that if there is a heavy inflow of such events it does not cause the allocation for other critical events to block and may even eliminate deadlocks in some cases.

CONFIG_BT_DISCARDABLE_BUF_SIZE

Size of discardable event buffers, ranging from 45 to 257. Size of buffers in the separate discardable event buffer pool. The minimum size is set based on the Advertising Report. Setting the buffer can save memory if with size set differently from that of the CONFIG_BT_RX_BUF_LEN. range is 45 to 257.

CONFIG_BT_HCI_TX_STACK_SIZE

HCI TX task stack size needed for executing bt_send with specified driver, should be no less than 512.

CONFIG_BT_HCI_TX_PRIO

HCI TX task priority.

CONFIG_BT_RX_STACK_SIZE

Size of the receiving thread stack. This is the context from which all event callbacks to the application occur. The default value is sufficient for basic operation, but if the application needs to do advanced things in its callbacks that require extra stack space, this value can be increased to accommodate for that.

CONFIG_BT_RX_PRIO

RX task priority.

CONFIG_BT_PERIPHERAL

Peripheral Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Peripheral role support.

CONFIG_BT_BROADCASTER

Broadcaster Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Broadcaster role support.

CONFIG_BT_EXT_ADV

Extended Advertising and Scanning support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this to enable Extended Advertising API support. This enables support for advertising with multiple advertising sets, extended advertising data, and advertising on LE Coded PHY. It enables support for receiving extended advertising data as a scanner, including support for advertising data over the LE coded PHY. It enables establishing connections over LE Coded PHY.

CONFIG_BT_CENTRAL

Central Role support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Select this for LE Central role support.

CONFIG_BT_WHITELIST

Enable whitelist support. This option enables the whitelist API. This takes advantage of the whitelisting feature of a Bluetooth LE controller. The whitelist is a global list and the same whitelist is used by both scanner and advertiser. The whitelist cannot be modified while it is in use. An Advertiser can whitelist which peers can connect or request scan response data. A scanner can whitelist advertiser for which it generates advertising reports. Connections can be established automatically for whitelisted peers.

This option deprecates the bt_le_set_auto_conn API in favor of the bt_conn_create_aute_le API.

CONFIG_BT_DEVICE_NAME

Bluetooth device name. Name can be up to 248 bytes long (excluding NULL termination). Can be empty string.

CONFIG_BT_DEVICE_APPEARANCE

Bluetooth device appearance. For the list of possible values, see the link: www.bluetooth.com/specifications/assigned-numbers.

CONFIG_BT_DEVICE_NAME_DYNAMIC

Allow to set Bluetooth device name on runtime. Enabling this option allows for runtime configuration of Bluetooth device name.

CONFIG_BT_ID_MAX

Maximum number of local identities, range 1 to 10 is valid. Maximum number of supported local identity addresses. For most products, this is safe to leave as the default value (1). Range 1 to 10 is valid.

CONFIG_BT_CONN

Connection enablement, if the macro is set to 0, feature is disabled, if 1, feature is enabled.

CONFIG_BT_MAX_CONN

it is the max connection supported by host stack. Maximum number of simultaneous Bluetooth connections supported.

CONFIG_BT_HCI_ACL_FLOW_CONTROL

Controller to host ACL flow control support. Enable support for throttling ACL buffers from the controller to the host. This is useful when the host and controller are on separate cores, since it ensures that we do not run out of incoming ACL buffers.

CONFIG_BT_PHY_UPDATE

PHY Update, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable support for Bluetooth 5.0 PHY Update Procedure.

CONFIG_BT_DATA_LEN_UPDATE

Data Length Update. If the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable support for Bluetooth v4.2 LE Data Length Update procedure.

CONFIG_BT_CREATE_CONN_TIMEOUT

Timeout for pending LE Create Connection command in seconds.

CONFIG_BT_CONN_PARAM_UPDATE_TIMEOUT

Peripheral connection parameter update timeout in milliseconds, range 1 to 65535 is valid. The value is a timeout used by peripheral device to wait until it starts the connection parameters update procedure to change default connection parameters. The default value is set to 5s, to comply with BT protocol specification: Core 4.2 Vol 3, Part C, 9.3.12.2 Range 1 to 65535 is valid.

CONFIG_BT_CONN_TX_MAX

Maximum number of pending TX buffers. Maximum number of pending TX buffers that have not yet been acknowledged by the controller.

CONFIG_BT_REMOTE_INFO

Enable application access to remote information. Enable application access to the remote information available in the stack. The remote information is retrieved once a connection has been established and the application is notified when this information is available through the remote_version_available connection callback.

CONFIG_BT_REMOTE_VERSION

Enable fetching of remote version. Enable this to get access to the remote version in the Controller and in the host through bt_conn_get_info(). The fields in question can be then found in the bt_conn_info struct.

CONFIG_BT_SMP_SC_ONLY

Secure Connections Only Mode. This option enables support for Secure Connection Only Mode. In this mode device shall only use Security Mode 1 Level 4 with exception for services that only require Security Mode 1 Level 1 (no security). Security Mode 1 Level 4 stands for authenticated LE Secure Connections pairing with encryption. Enabling this option disables legacy pairing.

CONFIG_BT_SMP_OOB_LEGACY_PAIR_ONLY

Force Out of Band Legacy pairing. This option disables Legacy and LE SC pairing and forces legacy OOB.

CONFIG_BT_SMP_DISABLE_LEGACY_JW_PASSKEY

Forbid usage of insecure legacy pairing methods. This option disables Just Works and Passkey legacy pairing methods to increase security.

CONFIG_BT_PRIVACY

Privacy Feature, if the macro is set to 0, feature is disabled, if 1, feature is enabled. Enable local Privacy Feature support. This makes it possible to use Resolvable Private Addresses (RPAs).

CONFIG_BT_ECC

Enable ECDH key generation support. This option adds support for ECDH HCI commands.

CONFIG_BT_TINYCRYPT_ECC

Use TinyCrypt library for ECDH. If this option is used to set TinyCrypt library which is used for emulating the ECDH HCI commands and events needed by e.g. LE Secure Connections. In builds including the Bluetooth LE host, if don't set the controller crypto which is used for ECDH and if the controller doesn't support the required HCI commands the LE Secure Connections support will be disabled. In builds including the HCI Raw interface and the Bluetooth LE controller, this option injects support for the 2 HCI commands required for LE Secure Connections so that hosts can make use of those. The option defaults to enabled for a combined build with Zephyr's own controller, since it does not have any special ECC support itself (at least not currently).

CONFIG_BT_TINYCRYPT_ECC_PRIORITY

Thread priority of ECC Task.

CONFIG_BT_HCI_ECC_STACK_SIZE

Thread stack size of ECC Task.

CONFIG_BT_RPA

Bluetooth Resolvable Private Address (RPA)

CONFIG_BT_RPA_TIMEOUT

Resolvable Private Address timeout, defaults to 900 seconds. This option defines how often resolvable private address is rotated. Value is provided in seconds and defaults to 900 seconds (15 minutes).

CONFIG_BT_SIGNING

Data signing support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables data signing which is used for transferring authenticated data in an unencrypted connection.

CONFIG_BT_SMP_APP_PAIRING_ACCEPT

Accept or reject pairing initiative. When receiving pairing request or pairing response queries, the application shall either accept proceeding with pairing or not. This is for pairing over SMP and does not affect SSP, which will continue pairing without querying the application. The application can return an error code, which is translated into an SMP return value if the pairing is not allowed.

CONFIG_BT_SMP_ALLOW_UNAUTH_OVERWRITE

Allow unauthenticated pairing for paired device. This option allows all unauthenticated pairing attempts made by the peer where an unauthenticated bond already exists. This would enable cases where an attacker could copy the peer device address to connect and start an unauthenticated pairing procedure to replace the existing bond. When this option is disabled in order to create a new bond the old bond must be explicitly deleted with bt_unpair.

CONFIG_BT_FIXED_PASSKEY

Use a fixed passkey for pairing, set passkey to fixed or not. With this option enabled, the application will be able to call the bt_passkey_set() API to set a fixed passkey. If set, the pairing_confim() callback will be called for all incoming pairings.

CONFIG_BT_BONDABLE

Bondable Mode, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for Bondable Mode. In this mode, Bonding flag in AuthReq of SMP Pairing Request/Response is set indicating the support for this mode.

CONFIG_BT_BONDING_REQUIRED

Always require bonding. When this option is enabled remote devices are required to always set the bondable flag in their pairing request. Any other kind of requests will be rejected.

CONFIG_BT_SMP_ENFORCE_MITM

Enforce MITM protection, if the macro is set to 0, feature is disabled, if 1, feature is enabled. With this option enabled, the Security Manager is set MITM option in the Authentication Requirements Flags whenever local IO Capabilities allow the generated key to be authenticated.

CONFIG_BT_OOB_DATA_FIXED

Use a fixed random number for LESC OOB pairing. With this option enabled, the application will be able to perform LESC pairing with OOB data that consists of fixed random number and confirm value. This option should only be enabled for debugging and should never be used in production.

CONFIG_BT_KEYS_OVERWRITE_OLDEST

Overwrite oldest keys with new ones if key storage is full. With this option enabled, if a pairing attempt occurs and the key storage is full, then the oldest keys in storage will be removed to free space for the new pairing keys.

CONFIG_BT_HOST_CCM

Enable host side AES-CCM module. Enables the software-based AES-CCM engine in the host. Will use the controller's AES encryption functions if available, or BT_HOST_CRYPTO otherwise.

CONFIG_BT_L2CAP_RX_MTU

Maximum supported L2CAP MTU for incoming data, if CONFIG_BT_SMP is set, range is 65 to 1300, otherwise range is 23 to 1300. Maximum size of each incoming L2CAP PDU. Range is 23 to 1300 range is 65 to 1300 for CONFIG_BT_SMP.

CONFIG_BT_L2CAP_TX_BUF_COUNT

Number of buffers available for outgoing L2CAP packets, ranging from 2 to 255. Range is 2 to 255.

CONFIG_BT_L2CAP_TX_FRAG_COUNT

Number of L2CAP TX fragment buffers, ranging from 0 to 255. Number of buffers available for fragments of TX buffers.

Warning: Setting this to 0 means that the application must ensure that queued TX buffers never need to be fragmented, that is the controller's buffer size is large enough. If this is not ensured, and there are no dedicated fragment buffers, a deadlock may occur. In most cases the default value of 2 is a safe bet. Range is 0 to 255.

CONFIG_BT_L2CAP_TX_MTU

Maximum supported L2CAP MTU for L2CAP TX buffers, if CONFIG_BT_SMP is set, the range is 65 to 2000. Otherwise, range is 23 to 2000. Range is 23 to 2000. Range is 65 to 2000 for CONFIG_BT_SMP.

CONFIG_BT_L2CAP_DYNAMIC_CHANNEL

L2CAP Dynamic Channel support. This option enables support for LE Connection oriented Channels, allowing the creation of dynamic L2CAP Channels.

CONFIG_BT_L2CAP_DYNAMIC_CHANNEL

L2CAP Dynamic Channel support. This option enables support for LE Connection oriented Channels, allowing the creation of dynamic L2CAP Channels.

Bluetooth BR/EDR support [EXPERIMENTAL] This option enables Bluetooth BR/EDR support.

CONFIG_BT_ATT_PREPARE_COUNT

Number of ATT prepares write buffers, if the macro is set to 0, feature is disabled, if greater than 1, feature is enabled. Number of buffers available for ATT prepares write, setting this to 0 disables GATT long/reliable writes.

CONFIG_BT_ATT_TX_MAX

Maximum number of queued outgoing ATT PDUs. Number of ATT PDUs that can be at a single moment queued for transmission. If the application tries to send more than this amount the calls blocks until an existing queued PDU gets sent. Range is 1 to CONFIG_BT_L2CAP_TX_BUF_COUNT.

CONFIG_BT_GATT_SERVICE_CHANGED

GATT Service Changed support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the service changed characteristic.

CONFIG_BT_GATT_DYNAMIC_DB

GATT dynamic database support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables registering/unregistering services at runtime.

CONFIG_BT_GATT_CACHING

GATT Caching support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for GATT Caching. When enabled the stack registers Client Supported Features and Database Hash characteristics which is used by clients to detect if anything has changed on the GATT database.

CONFIG_BT_GATT_CLIENT

GATT client support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the GATT Client role.

CONFIG_BT_GATT_READ_MULTIPLE

GATT Read Multiple Characteristic. Values support, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables support for the GATT Read Multiple Characteristic Values procedure.

CONFIG_BT_GAP_AUTO_UPDATE_CONN_PARAMS

Automatic Update of Connection Parameters, if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option, if enabled, allows automatically sending request for connection parameters update after GAP recommended 5 seconds of connection as peripheral.

CONFIG_BT_GAP_PERIPHERAL_PREF_PARAMS

Configure peripheral preferred connection parameters. This configures peripheral preferred connection parameters. Enabling this option results in adding PPCP characteristic in GAP. If disabled it is up to application to set expected connection parameters.

CONFIG_BT_MAX_PAIRED

Maximum number of paired devices. Maximum number of paired Bluetooth devices. The minimum (and default) number is 1.

CONFIG_BT_MAX_SCO_CONN

Maximum number of simultaneous SCO connections. Maximum number of simultaneous Bluetooth synchronous connections supported. The minimum (and default) number is 1. Range 1 to 3 is valid.

CONFIG_BT_RFCOMM

Bluetooth RFCOMM protocol support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth RFCOMM support.

CONFIG_BT_RFCOMM_L2CAP_MTU

L2CAP MTU for RFCOMM frames. Maximum size of L2CAP PDU for RFCOMM frames.

CONFIG_BT_HFP_HF

Bluetooth Handsfree profile HF Role support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth HF support.

CONFIG_BT_AVDTP

Bluetooth AVDTP protocol support [EXPERIMENTAL], if the macro is set to 0, feature is disabled, if 1, feature is enabled. This option enables Bluetooth AVDTP support.

CONFIG_BT_A2DP

Bluetooth A2DP Profile [EXPERIMENTAL]. This option enables the A2DP profile.

CONFIG_BT_A2DP_SOURCE

Bluetooth A2DP profile source function. This option enables the A2DP profile Source function.

CONFIG_BT_A2DP_SINK

Bluetooth A2DP profile sink function. This option enables the A2DP profile Sink function.

CONFIG_BT_A2DP_TASK_PRIORITY

Bluetooth A2DP profile task priority. This option sets the task priority. The task is used to process the streamer data and retry command.

CONFIG_BT_A2DP_TASK_STACK_SIZE

Bluetooth A2DP profile task stack size. This option sets the task stack size.

CONFIG_BT_PAGE_TIMEOUT

Bluetooth Page Timeout. This option sets the page timeout value. Value is selected as (N * 0.625) ms.

CONFIG_BT_DIS_MODEL

Model name. The device model inside Device Information Service.

CONFIG_BT_DIS_MANUF

Manufacturer name. The device manufacturer inside Device Information Service.

CONFIG_BT_DIS_PNP

Enable PnP_ID characteristic. Enable PnP_ID characteristic in Device Information Service.

CONFIG_BT_DIS_PNP_VID_SRC

Vendor ID source, range 1 - 2. The Vendor ID Source field designates which organization assigned the value used in the Vendor ID field value. The possible values are:

- 1 Bluetooth SIG, the Vendor ID was assigned by the Bluetooth SIG
- 2 USB IF, the Vendor ID was assigned by the USB IF

CONFIG_BT_DIS_PNP_VID

Vendor ID, range 0 - 0xFFFF. The Vendor ID field is intended to uniquely identify the vendor of the device. This field is used in conjunction with Vendor ID Source field, which determines which organization assigned the Vendor ID field value. Note: The Bluetooth Special Interest Group assigns Device ID Vendor ID, and the USB Implementers Forum assigns Vendor IDs, either of which can be used for the Vendor ID field value. Device providers should procure the Vendor ID from the USB Implementers Forum or the Company Identifier from the Bluetooth SIG.

CONFIG_BT_DIS_PNP_PID

Product ID, range 0 - 0xFFFF. The Product ID field is intended to distinguish between different products made by the vendor identified with the Vendor ID field. The vendors themselves manage Product ID field values.

CONFIG_BT_DIS_PNP_VER

Product Version, range 0 - 0xFFFF. The Product Version field is a numeric expression identifying the device release number in Binary-Coded Decimal. This is a vendor-assigned value, which defines the version of the product identified by the Vendor ID and Product ID fields. This field is intended to differentiate between versions of products with identical Vendor IDs and Product IDs. The value of the field value is 0xJJMN for version JJ.M.N (JJ - major version number, M - minor version number, N - subminor version number); For example, version 2.1.3 is represented with value 0x0213 and version 2.0.0 is represented with a value of 0x0200. When upward-compatible changes are made to the device, it is recommended that the minor version number be incremented. If incompatible changes are made to the device. It is recommended that the major version number is incremented. The subminor version is incremented for bug fixes.

CONFIG_BT_DIS_SERIAL_NUMBER

Enable DIS Serial number characteristic, 1 - enable, 0 - disable. Enable Serial Number characteristic in Device Information Service.

CONFIG_BT_DIS_SERIAL_NUMBER_STR

Serial Number. Serial Number characteristic string in Device Information Service.

CONFIG_BT_DIS_FW_REV

Enable DIS Firmware Revision characteristic, 1 - enable, 0 - disable. Enable Firmware Revision characteristic in Device Information Service.

CONFIG_BT_DIS_FW_REV_STR

Firmware revision. Firmware Revision characteristic String in Device Information Service.

CONFIG_BT_DIS_HW_REV

Enable DIS Hardware Revision characteristic, 1 - enable, 0 - disable. Enable Hardware Revision characteristic in Device Information Service.

CONFIG_BT_DIS_HW_REV_STR

Hardware revision. Hardware Revision characteristic String in Device Information Service.

CONFIG_BT_DIS_SW_REV

Enable DIS Software Revision characteristic, 1 - enable, 0 - disable. Enable Software Revision characteristic in Device Information Service.

CONFIG_BT_DIS_SW_REV_STR

Software revision Software revision characteristic String in Device Information Service.

CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE

System work queue stack size.

CONFIG_SYSTEM_WORKQUEUE_PRIORITY

System work queue priority.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_TYPE

HCI transport interface type.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_INSTANCE

HCI transport interface instance number.

CONFIG_BT_HCI_TRANSPORT_INTERFACE_SPEED

HCI transport interface rate. Configures the interface speed, for example, the default interface is h4, the speed to 115200

CONFIG_BT_HCI_TRANSPORT_TX_THREAD

Whether enable HCI transport TX thread.

CONFIG_BT_HCI_TRANSPORT_RX_THREAD

Whether enable HCI transport RX thread.

CONFIG_BT_HCI_TRANSPORT_RX_STACK_SIZE

HCI transport RX thread stack size.

CONFIG_BT_HCI_TRANSPORT_TX_STACK_SIZE

HCI transport TX thread stack size.

CONFIG_BT_HCI_TRANSPORT_TX_PRIO

HCI transport TX thread priority.

CONFIG_BT_HCI_TRANSPORT_RX_PRIO

HCI transport RX thread priority.

CONFIG_BT_MSG_QUEUE_COUNT

Message number in message queue.

Rework Guide for EdgeFast Bluetooth Protocol Abstraction Layer

Hardware Rework Guide for MIMXRT1170-EVKB and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

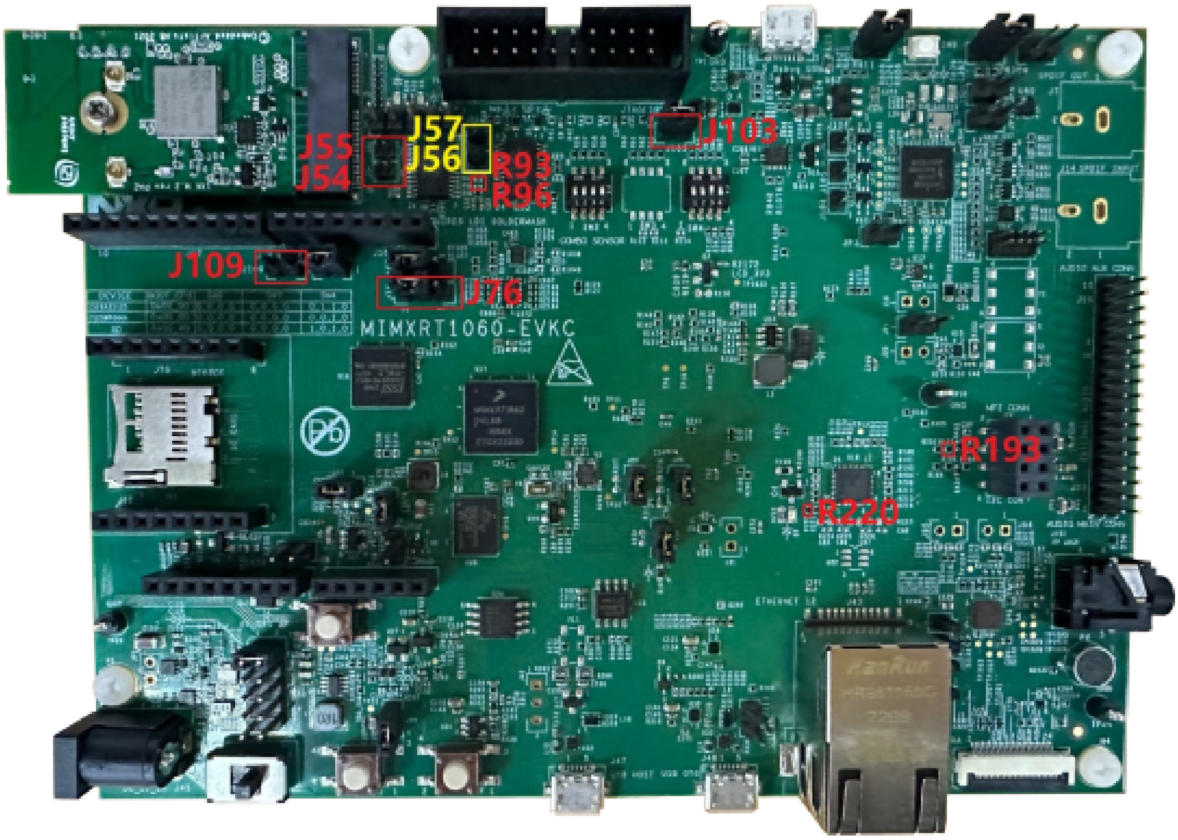
- **HCI UART rework**

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

- **PCM interface rework**

1. Remove J54 and J55, connect J56 and J57.
2. Remove R220.
3. Connect J103.

Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1170-EVKB and Murata 2EL M.2 Module Hardware Rework Guide for MIMXRT1170-EVKB and Murata

2EL M.2 Module

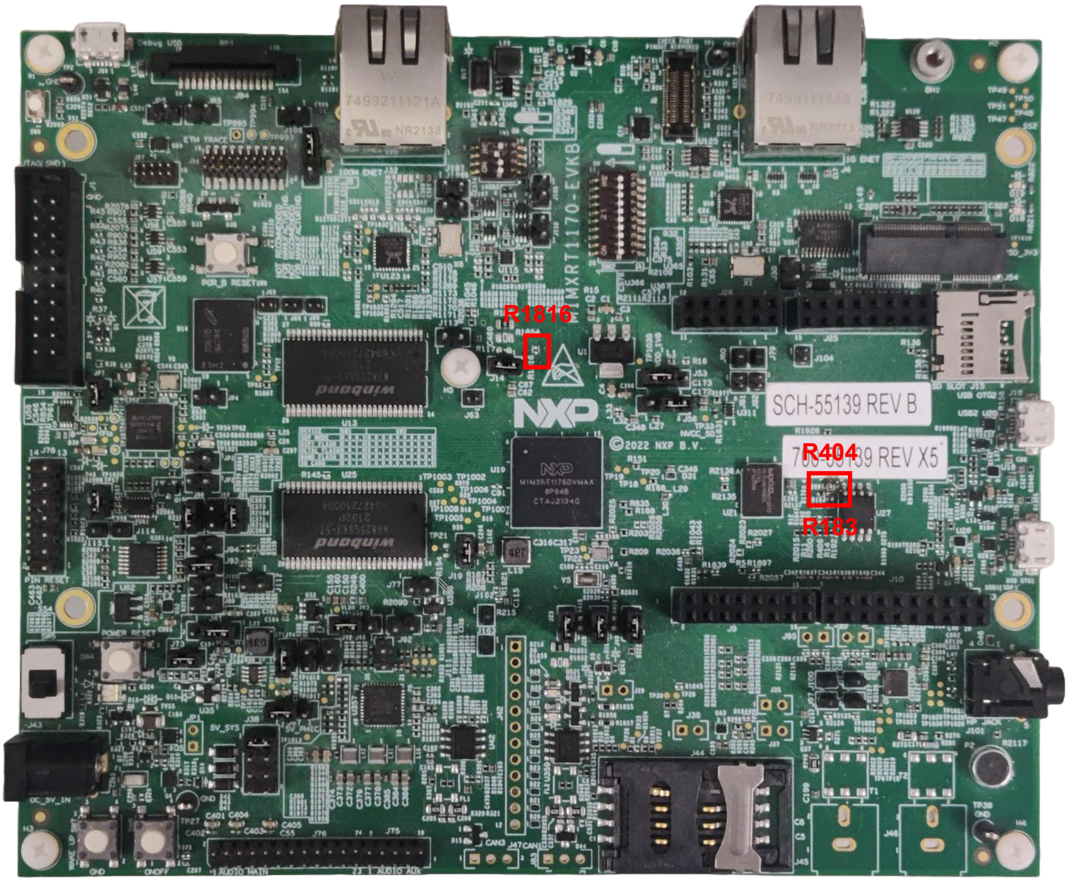
This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1170-EVKB and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

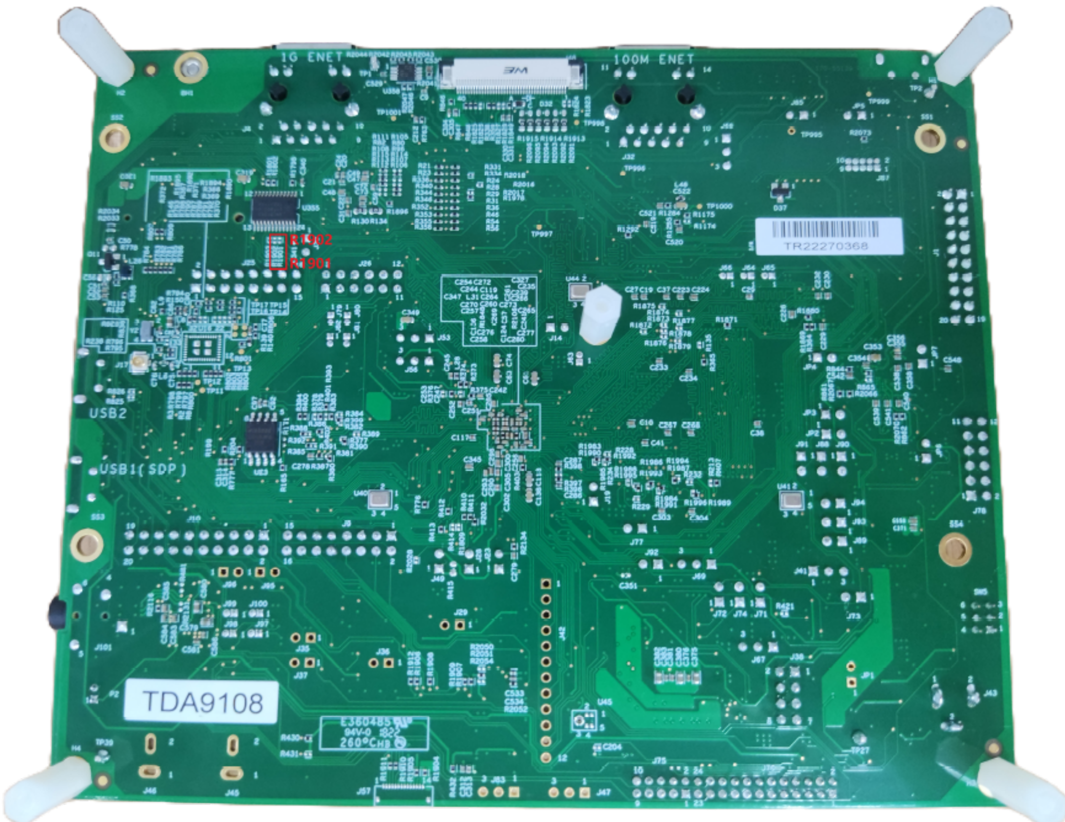
The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

Hardware rework

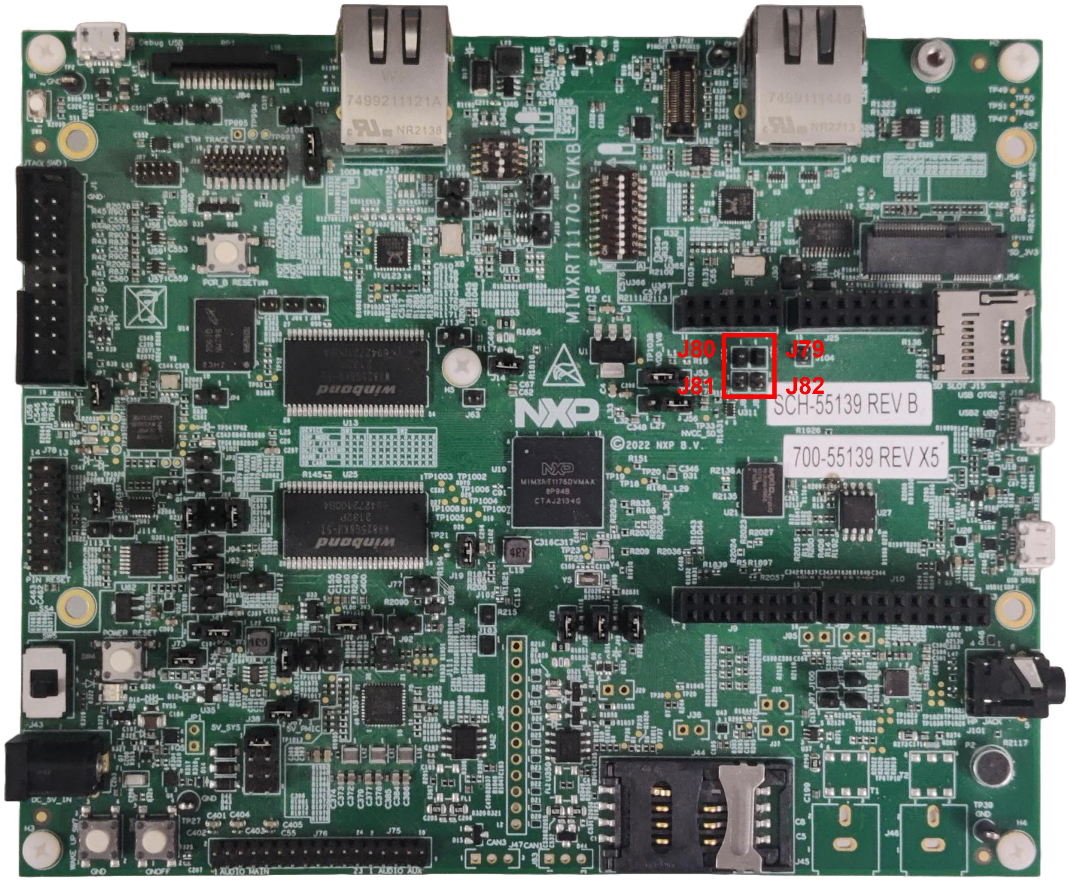
- **HCI UART rework**
  1. Remove resistors R183 and R1816.
  2. Solder 0 ohm resistor to R404, R1901, and R1902.

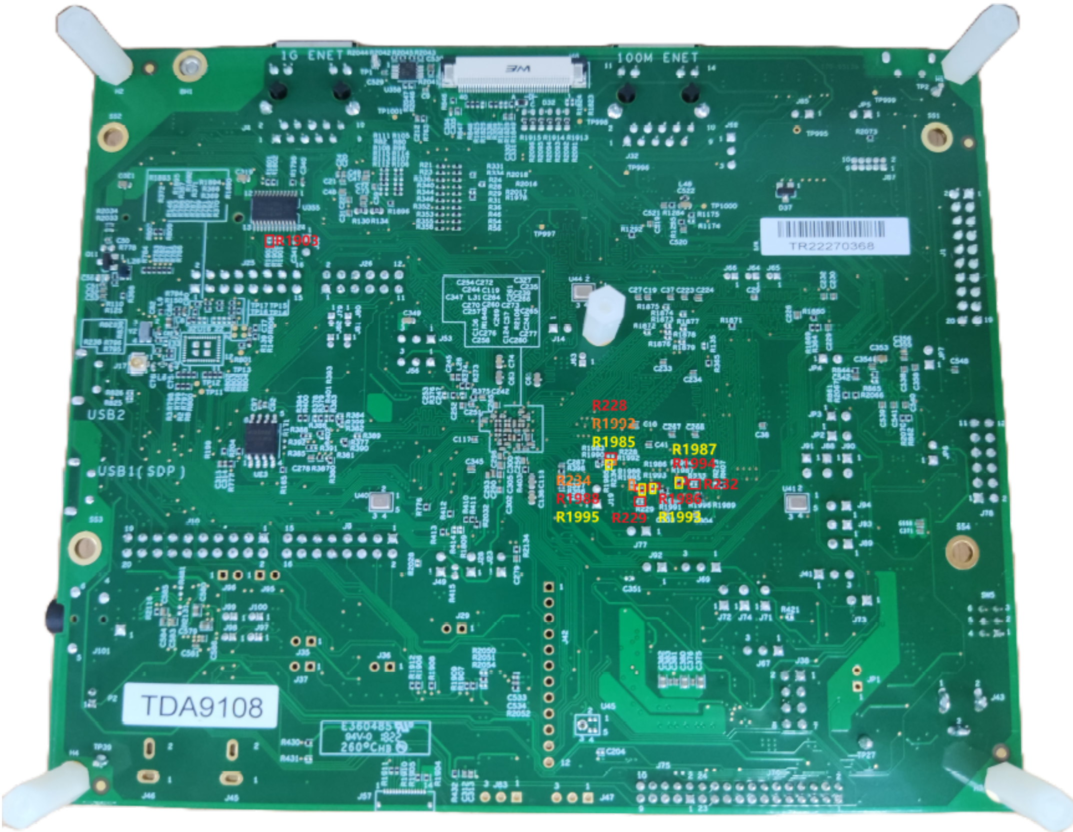




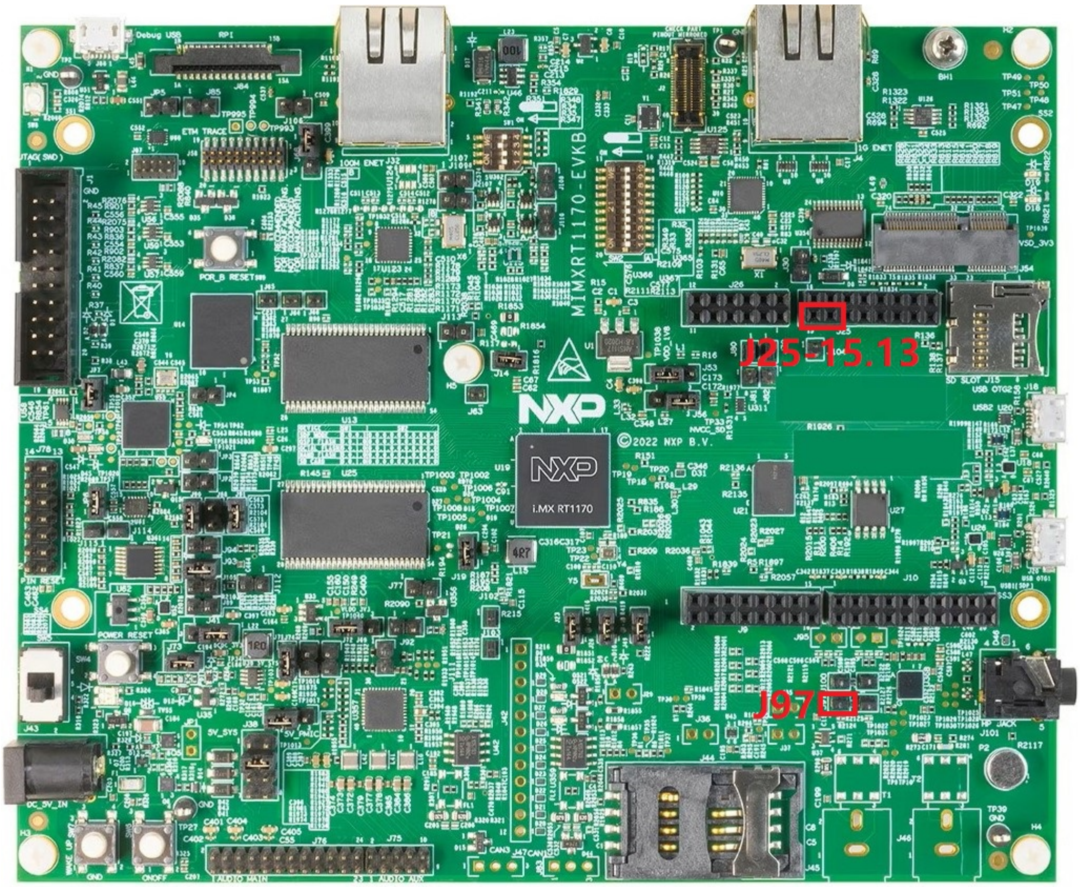
- **PCM interface rework**
  1. Disconnect header J79 and J80.

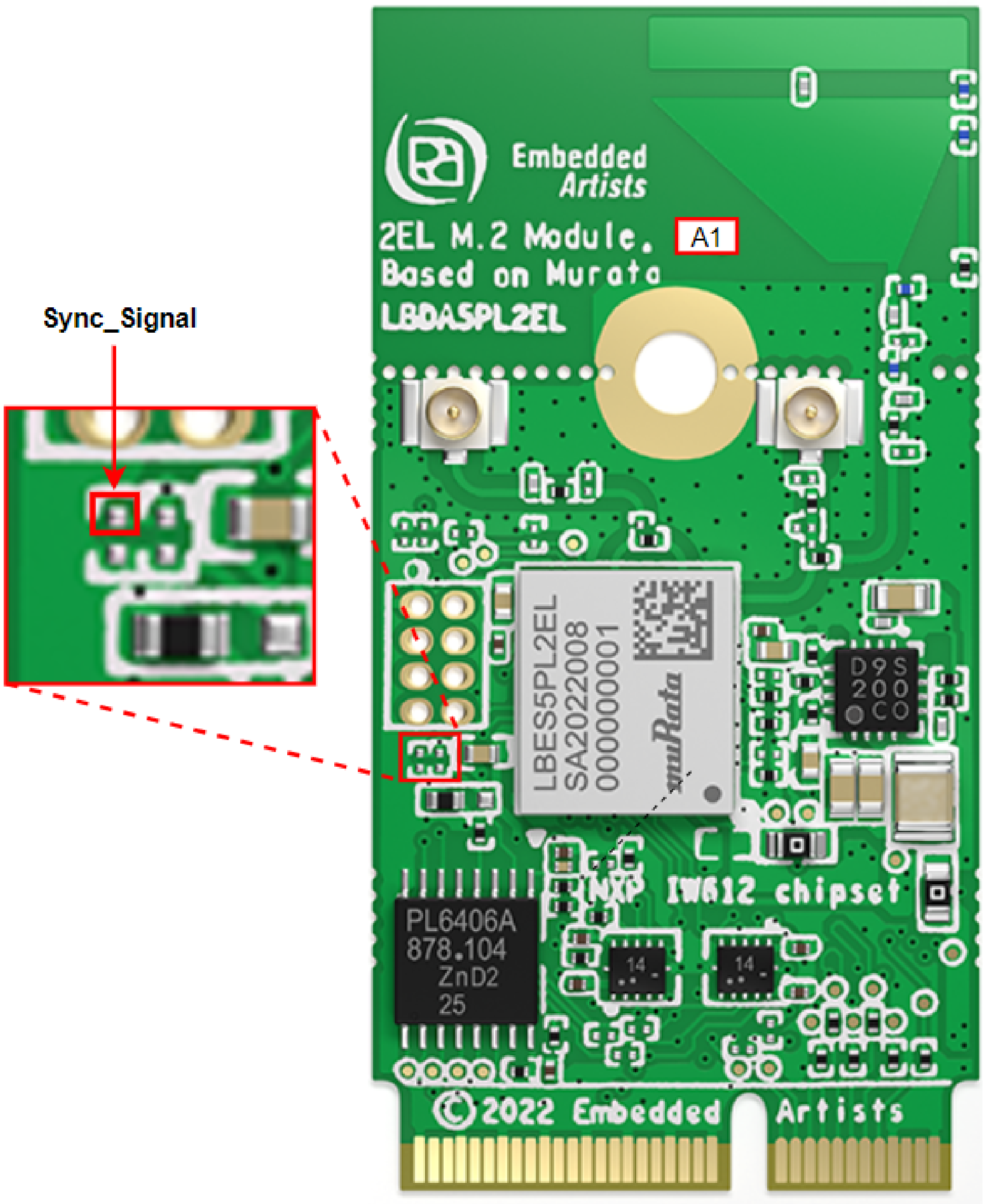
2. Connect header J81 and J82.
3. Remove resistors R1985, R1986, R1987, R1988, R1992, R1993, R1994, and R1995.
4. Solder 0 ohm resistor to R228, R229, R232, R234, and R1903.





- LE Audio Synchronization interface rework (only used on sink side)
  1. Connect J25-15 with J97.
  2. Connect J25-13 with 2EL's GPIO_27


J25-15.13
J97



Parent topic:*Hardware Rework Guide for MIMXRT1170-EVKB and Murata 2EL M.2 Adapter*

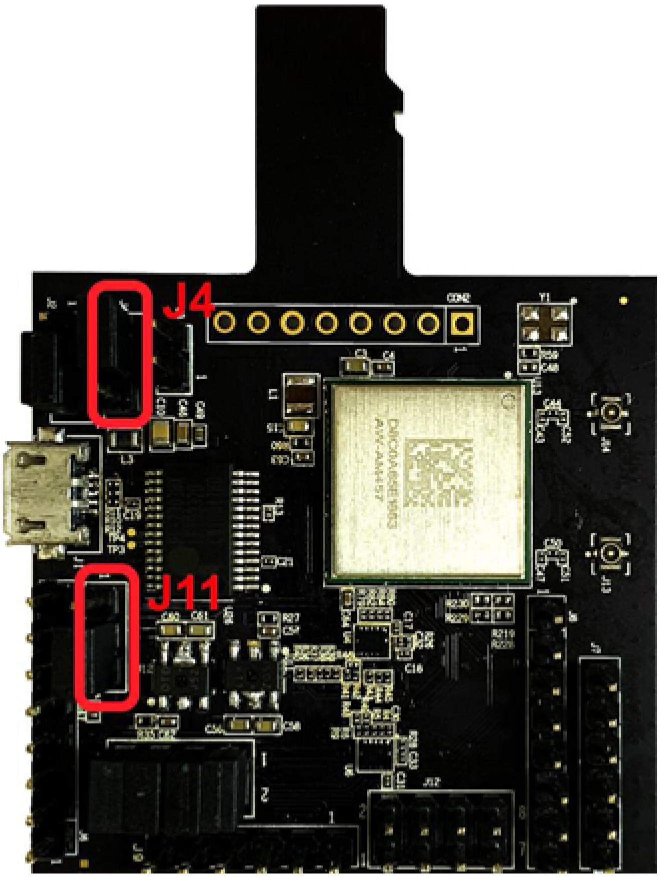
Hardware Rework Guide for MIMXRT685-EVK and AW-AM457-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-AM457-uSD. The AW-AM457-uSD user guide is available here.The hardware rework has one part:

- HCI UART rework

Hardware rework **HCI UART rework**

- R398 move from 1-2 to 2-3
- JP12 2-3
- Connect the pins of two boards as the following table.

Pin Name	AW-AM457-uSD	i.MX RT685	PIN NAME	GPIONAME of i.MX RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND



Jumper Settings:

- Connect J4[2-3] for VIO 3.3 V supply
- Connect J11[2-3] for VIO_SD 3.3 V supply

PCM interface rework

Connect the pins of two boards as the following table.

Pin Name	AW-AM457-uSD	i.MX RT685	PIN NAME of I.MX RT685	GPIONAME of I.MX RT685
PCM_IN	J9 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J9 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J9 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J9 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J9 (pin 6)	J29 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-AM457-uSD*

Hardware Rework Guide for MIMXRT685-EVK and AW-CM358-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-CM358-uSD. The AW-CM358-uSD user guide is available here. The hardware rework has one part:

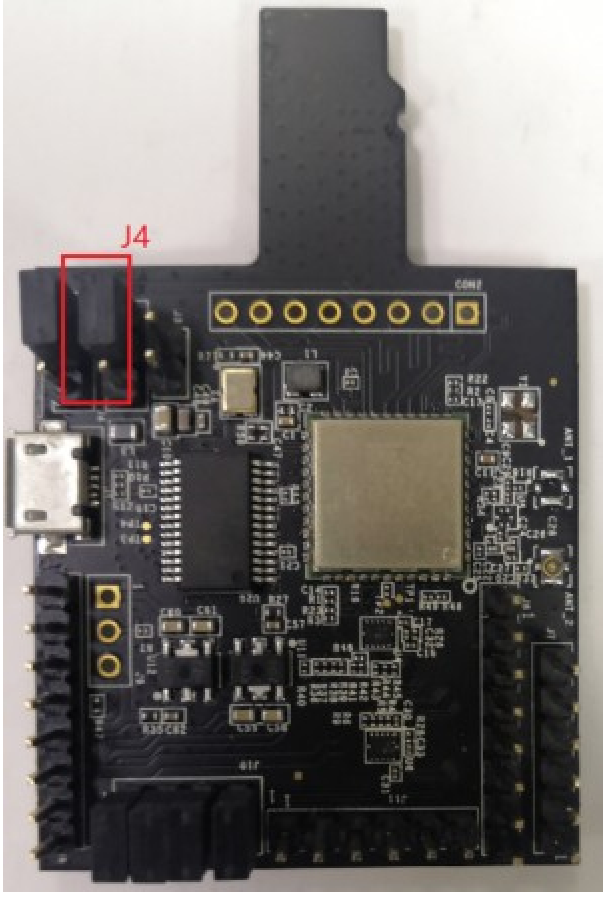
- HCI UART rework

Hardware rework **HCI UART rework**

R398 move from 1-2 to 2-3.

Connect the pins of two boards as the following table.

Pin Name	AW-CM358-USD	i.MXRT685	PIN NAME	GPIONAME of RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND



Jumper Setting:

Connect J4[1-2] for VIO 1.8 V supply.

PCM interface rework

Connect the pins of two boards as the following table.

Pin Name	AW-CM358-USD	i.MX RT685	PIN NAME of RT685	GPIONAME of RT685
PCM_IN	J11 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J11 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J11 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J11 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J11 (pin 5)	J29 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-CM358-uSD*

Hardware Rework Guide for MIMXRT685-EVK and AW-AM510-uSD This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and AW-AM510-uSD. The AW-AM510-uSD user guide is available here. The hardware rework has one part:

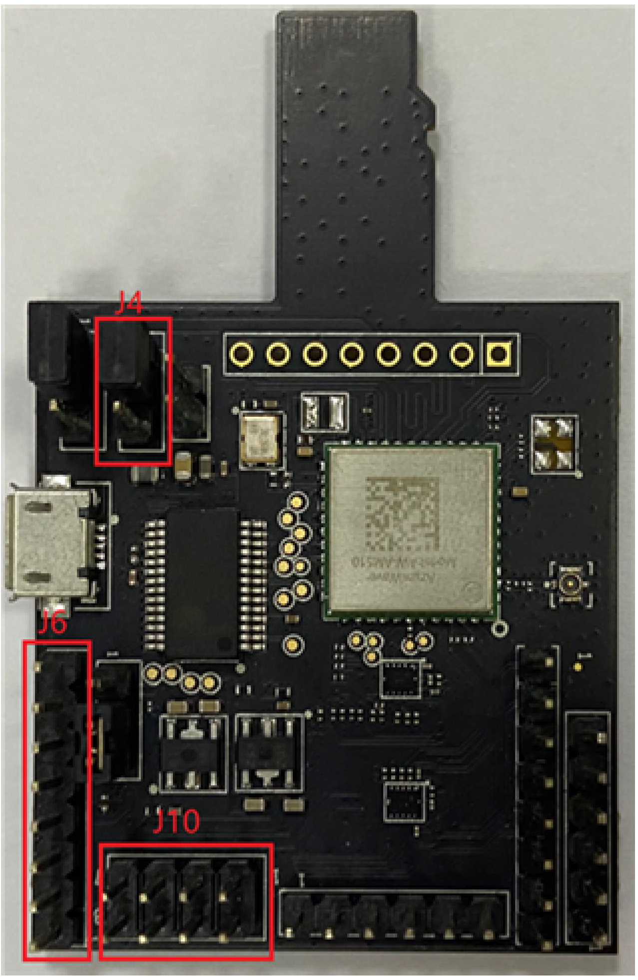
- HCI UART rework

Hardware rework

- **HCI UART rework**

Connect the pins of two boards as the following table.

Pin Name	AW-AM510-uSD	i.MXRT685	PIN NAME	GPIO NAME of RT685
UART_TXD	J10 (pin 4)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
UART_RXD	J10 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
UART_RTS	J10 (pin 6)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
UART_CTS	J10 (pin 8)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1
GND	J6 (pin 7)	J29 (pin 6)	GND	GND



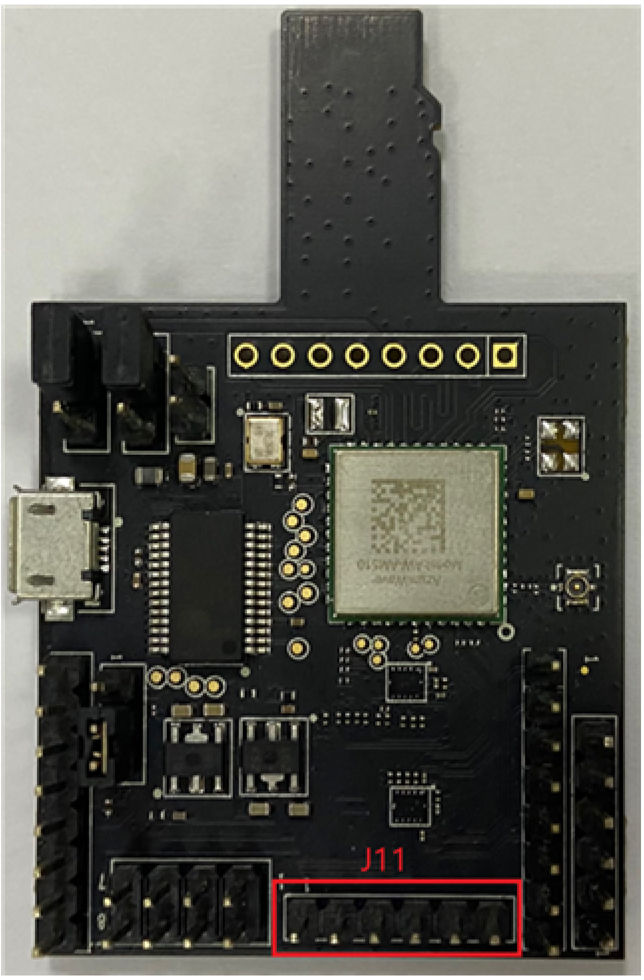
Jumper Setting:

- Connect J4[2-3] for VIO 3.3 V supply

- **PCM interface rework**

Connect the pins of two boards as the following table.

PIN NAME	AW-AM510-USD	i.MX RT685	PIN NAME of RT685	GPIONAME of RT685
PCM_IN	J11 (pin 1)	J47 (pin 7)	I2S2_TXD	FC2_RXD_SDA_MOSI_DATA
PCM_OUT	J11 (pin 2)	J28 (pin 4)	I2S5_RXD	FC5_RXD_SDA_MOSI_DATA
PCM_SYNC	J11 (pin 3)	J28 (pin 5)	I2S5_WS	FC5_TXD_SCL_MISO_WS
PCM_CLK	J11 (pin 4)	J28 (pin 6)	I2S5_SCK	FC5_SCK
GND	J11 (pin 6)	J29 (pin 7)	GND	GND



Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and AW-AM510-uSD*

Hardware Rework Guide for MIMXRT685-EVK and Murata uSD-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT685-EVK board and the Murata uSD-M.2 adapter. For details on the Murata uSD-M.2 Adapter, see Murata's uSD-M.2 webpage.

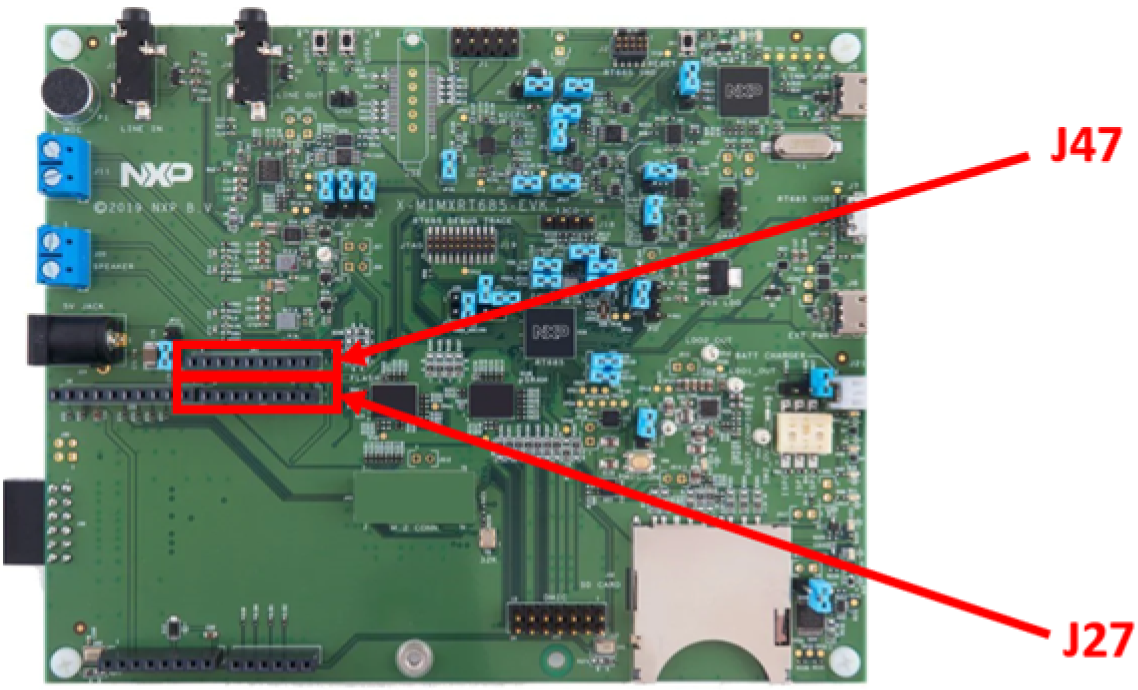
The hardware rework has one part:

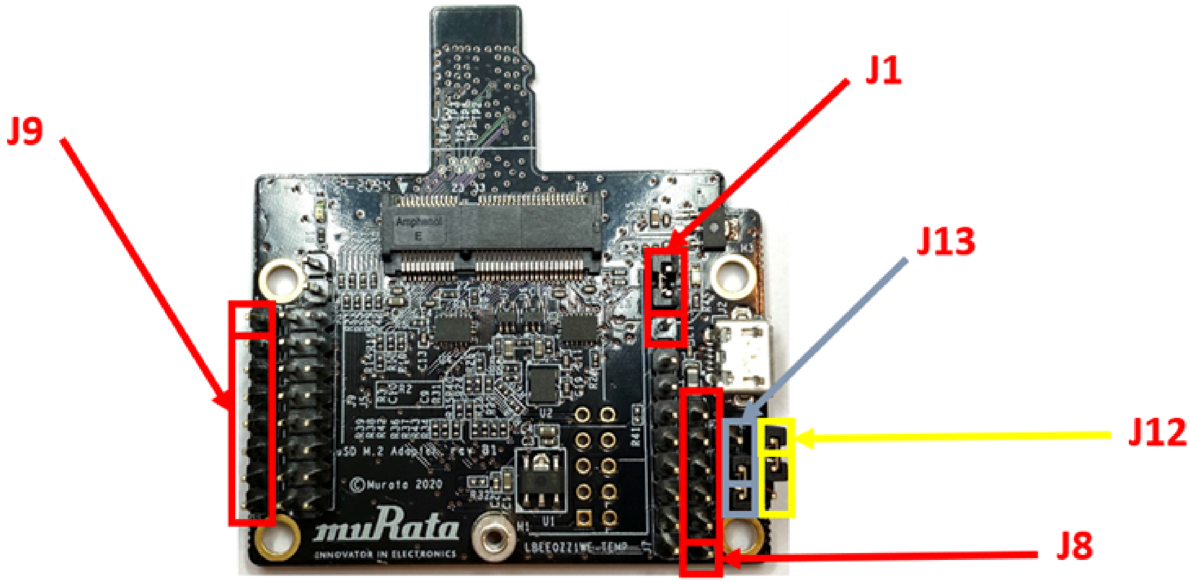
- HCI UART rework

Hardware rework **HCI UART rework :**

- JP12 2-3
- Connect the pins of two boards as the following table using jumper cables included in Murata's uSD-M.2 Adapter kit.

Pin name	uSD-M.2 adapter pin	i.MX RT685 pin	Pin name of RT685	GPIO name of RT685
BT_UART_TXD_HO	J9 (pin 1)	J27 (pin 1)	USART4_RXD	FC4_RXD_SDA_MOSI_DATA
BT_UART_RXD_HO	J9 (pin 2)	J27 (pin 2)	USART4_TXD	FC4_TXD_SCL_MISO_WS
BT_UART_RTS_HO:	J8 (pin 3)	J47 (pin 9)	USART4_CTS	FC4_CTS_SDA_SSEL0
BT_UART_CTS_HO:	J8 (pin 4)	J27 (pin 5)	USART4_RTS	FC4_RTS_SCL_SSEL1





Murata uSD-M.2 jumper settings:

- Both J12 and J13 = 1-2 (WLAN-SDIO = 1.8 V; and BT-UART and WLAN/BT-CTRL = 3.3 V)
- J1 = 2-3 (3.3 V from uSD connector)

Parent topic:*Hardware Rework Guide for MIMXRT685-EVK and Murata uSD-M.2 Adapter*

Hardware Rework Guide for MIMXRT685-AUD-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the Edgefast Bluetooth PAL on the NXP i.MX MIMXRT685-AUD-EVK board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has one part:

- HCI UART rework

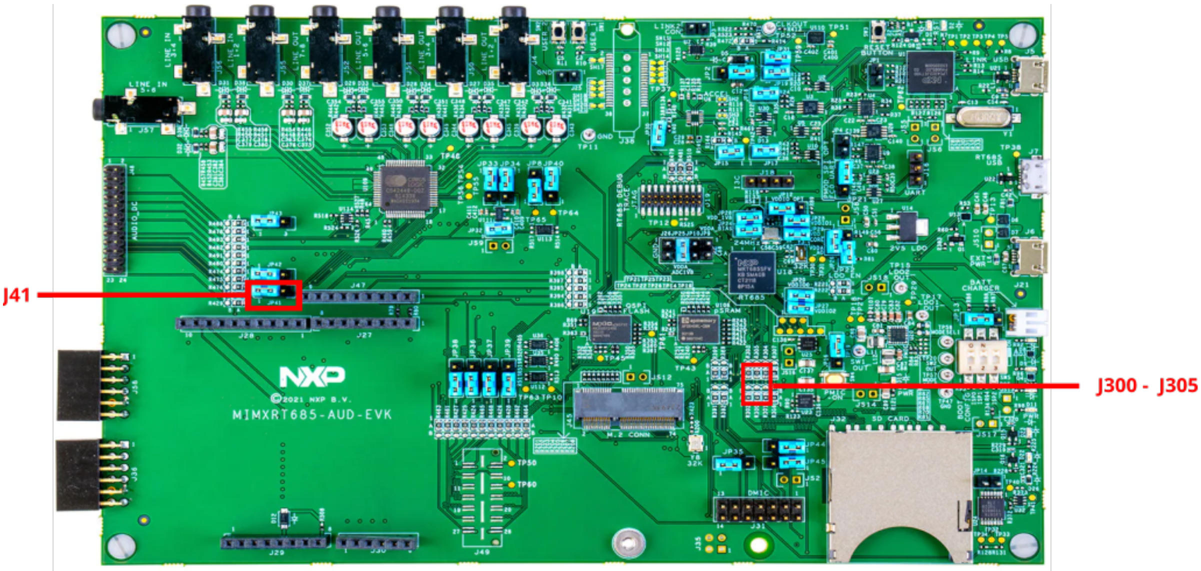
Hardware rework

HCI UART rework:

Mount R300~R305 A-B

Jumper Setting:

- Connect JP41[2-3]



Parent topic: *Hardware Rework Guide for MIMXRT685-AUD-EVK and Murata M.2 Module*

Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 2EL M.2 Module

This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL for low power feature on the NXP i.MX MIMXRT595-EVK board and the Murata's 2EL - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

The hardware rework has three parts:

- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework

- **Debug console serial rework**

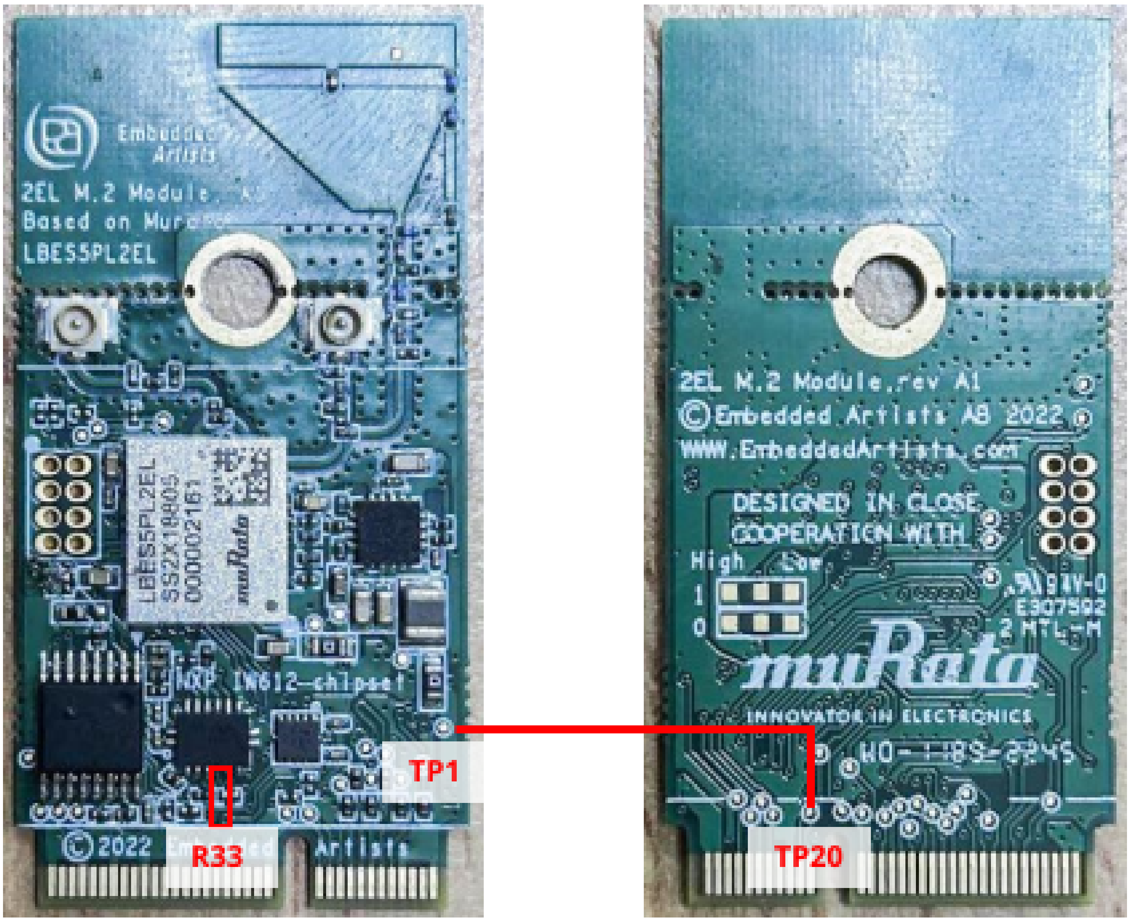
  For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

- **Host wake-up controller pin rework**:

  For details, refer Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module.

- **Controller wake-up host pin rework:**

1. Remove resistors R709 on MIMXRT595-EVK,
2. Solder 0K ohm resistor on R33 of Murata 2EL M.2 Module
3. Solder 10K ohm resistor on the Murata 2EL M.2 Module between TP1 and TP20.



Parent topic: *Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 2EL M.2 Module*

Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

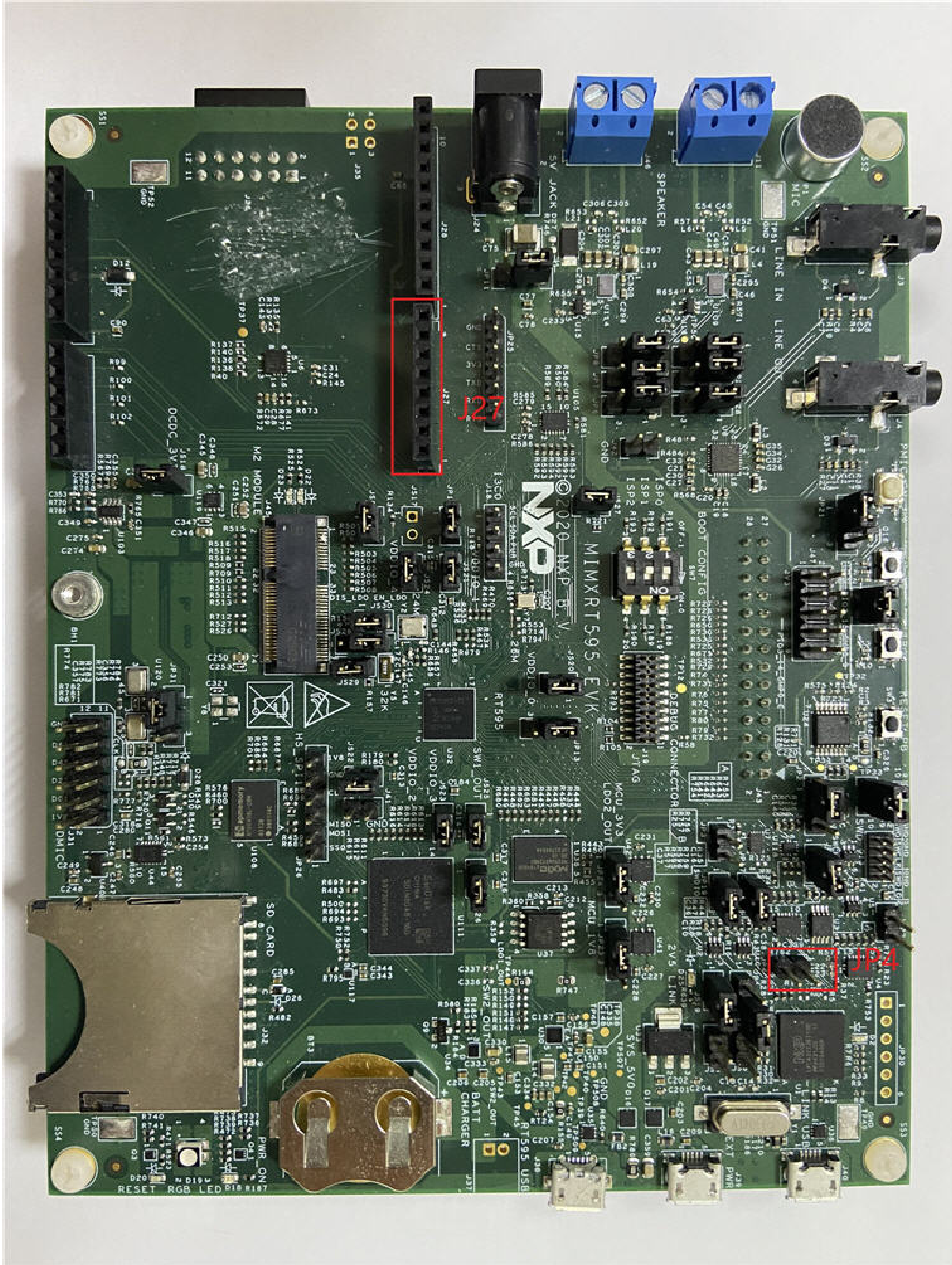
The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

No special rework is required, except the following to enable the debug port.

- JP4 1-2.
- J27 1 - TX of USB to serial converter
- J27 2 - RX of USB to serial converter



Parent topic:*Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*

Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL for low power feature on the NXP i.MX MIMXRT595-EVK board and the Murata's 1XK - direct M.2 connection to Embedded Artists EAR00385 (1XK) M.2 modules.

The hardware rework has three parts:

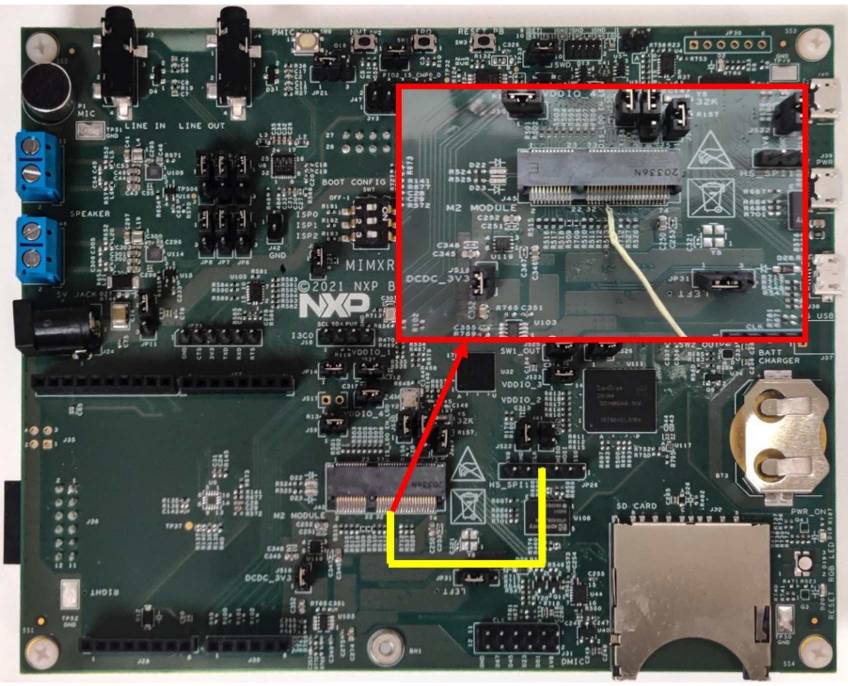
- Debug console serial rework
- Host wake-up controller pin rework (H2C)
- Controller wake-up host pin rework (C2H)

Hardware rework **Debug console serial rework:**

For details, refer *Hardware Rework Guide for MIMXRT595-EVK and Murata M.2 Module*.

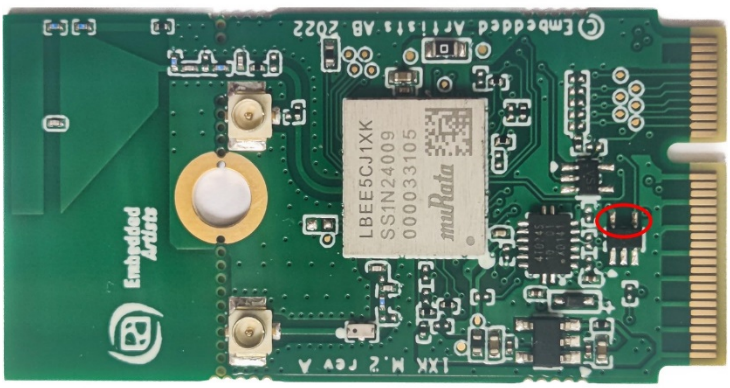
Host wake-up controller pin rework:

Connect M.2 (pin 42) to JP26 (pin 4) with a wire.



Controller wake-up host pin rework:

1. Remove resistors R709 on MIMXRT595-EVK.
2. Solder 10K ohm resistor on the Murata 1XK M.2 Module at the location shown in the following figure.



Parent topic:*Hardware Rework Guide for Low Power Feature on MIMXRT595-EVK and Murata 1XK M.2 Module*

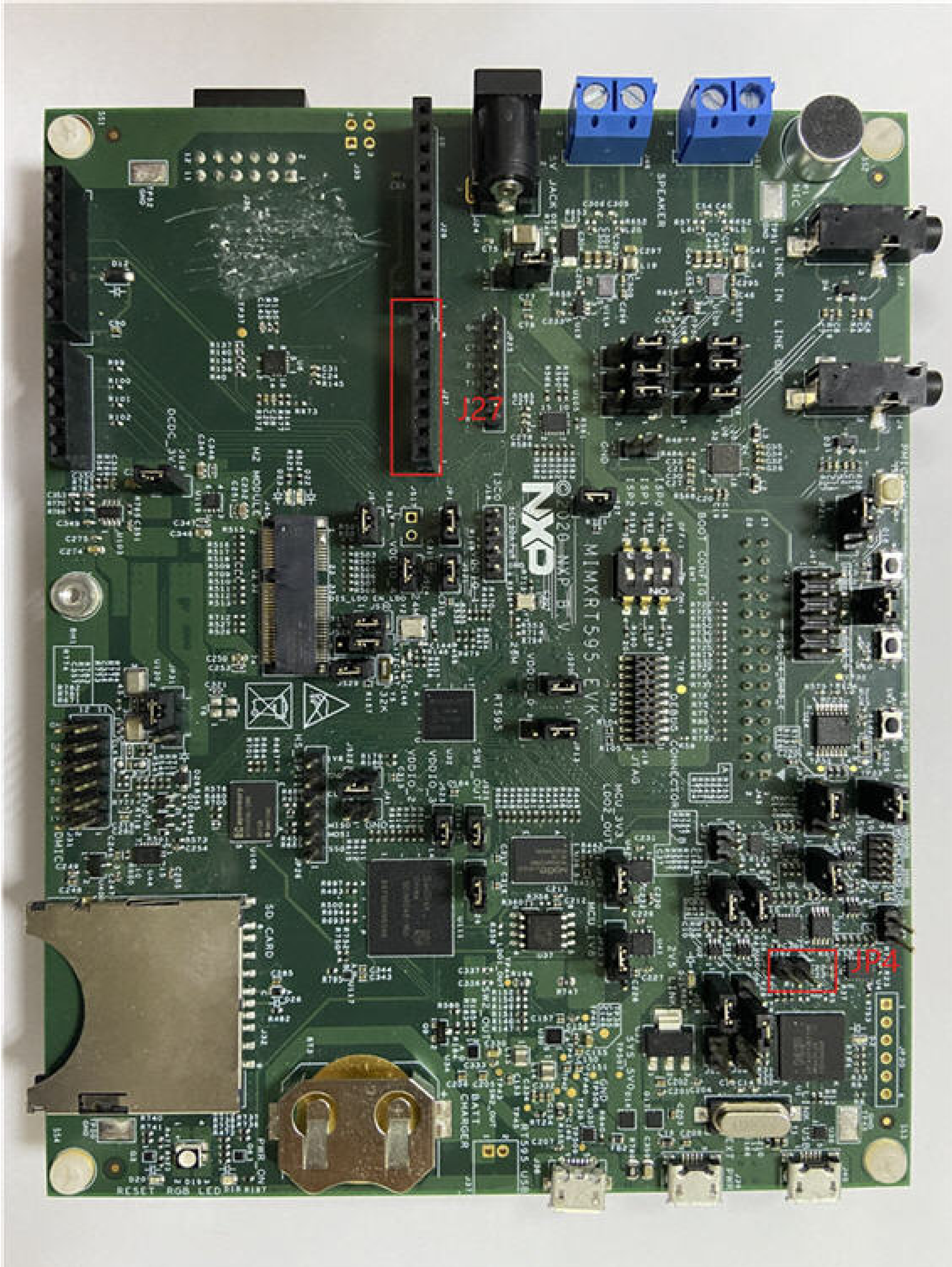
Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT595-EVK board and AW-AM510MA. The AW-AM510MA user guide is available here. The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

No special rework is required, except the following to enable the debug port.

- Connect J39 with external power.
- Connect JP4 1-2.
- J27 1 — TX of USB to serial converter.
- J27 2 — RX of USB to serial converter.



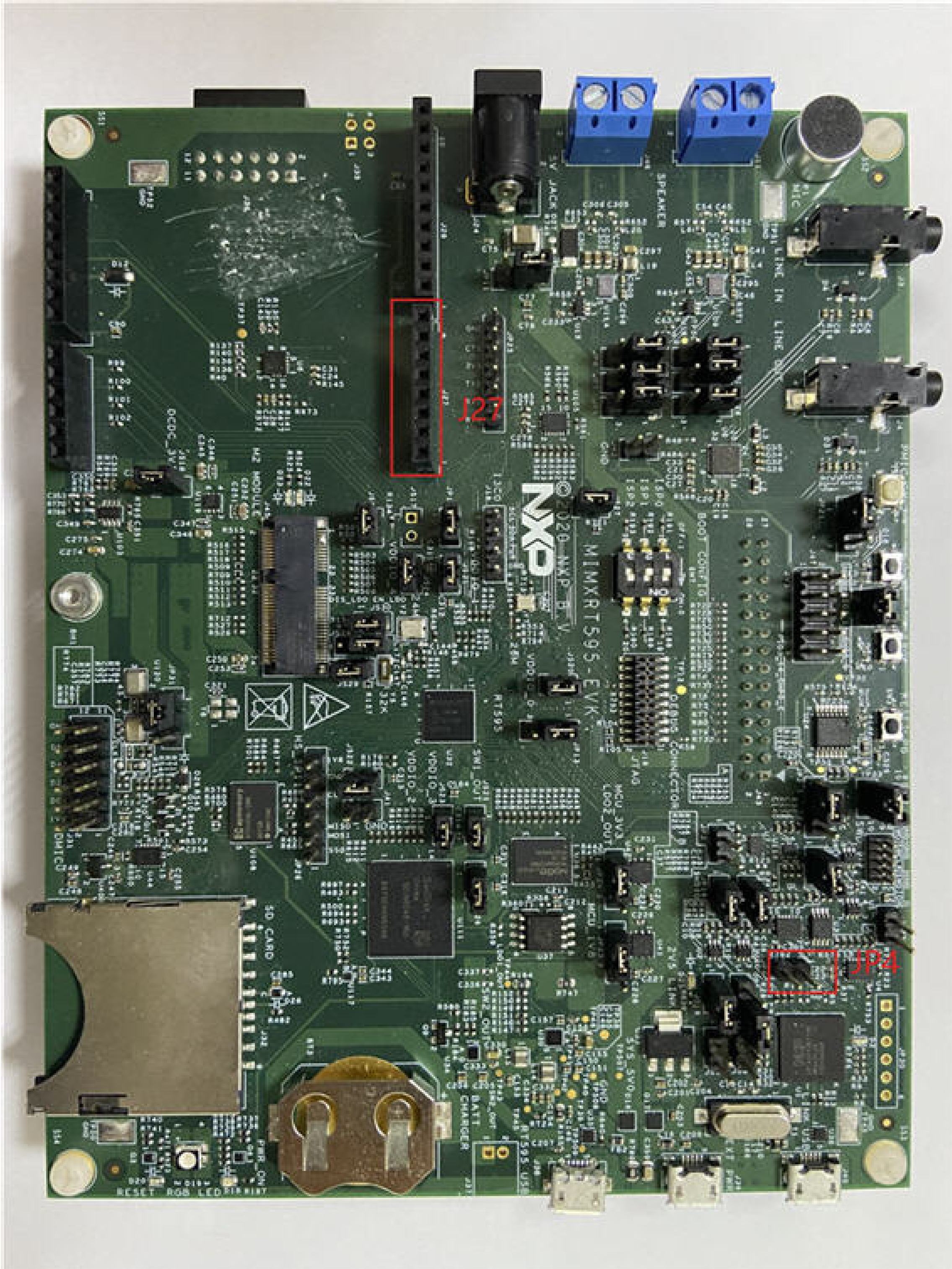
Parent topic:[*Hardware Rework Guide for MIMXRT595-EVK and AW-AM510MA*]

Hardware Rework Guide for MIMXRT595-EVK and AW-CM358MA This section is a brief hardware rework guidance of the Ethermind Bluetooth stack on the NXP i.MX MIMXRT595-EVK board and AW-CM358MA. The AW-CM358MA user guide is available here. The hardware rework has one part:

- Debug console serial rework

Hardware rework **Debug console serial rework:**

- Connect J39 with external power.
- JP4 1-2
- J27 1 - TX of USB to serial converter
- J27 2 - RX of USB to serial converter



Parent topic: *Hardware Rework Guide for MIMXRT595-EVK and AW-CM358MA*

Hardware Rework Guide for MIMXRT1040-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the Edgefast Bluetooth PAL on the NXP i.MX MIMXRT1040-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework
- Wake pin rework

Hardware rework

1. HCI UART rework
   - Solder R93 and R96
2. PCM interface rework
   - Solder R70 and R79; remove R76 and R86; Connect J80.
3. Wake pin rework
   - When using 2LL M.2 module, remove R456 and R457 to avoid the module has an impact on boot configuration.

Note: Make sure to disconnect J80 when debugging. Otherwise, the debugger downloading fails.

Parent topic:*Hardware Rework Guide for MIMXRT1040-EVK and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1060-EVKC and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework has two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

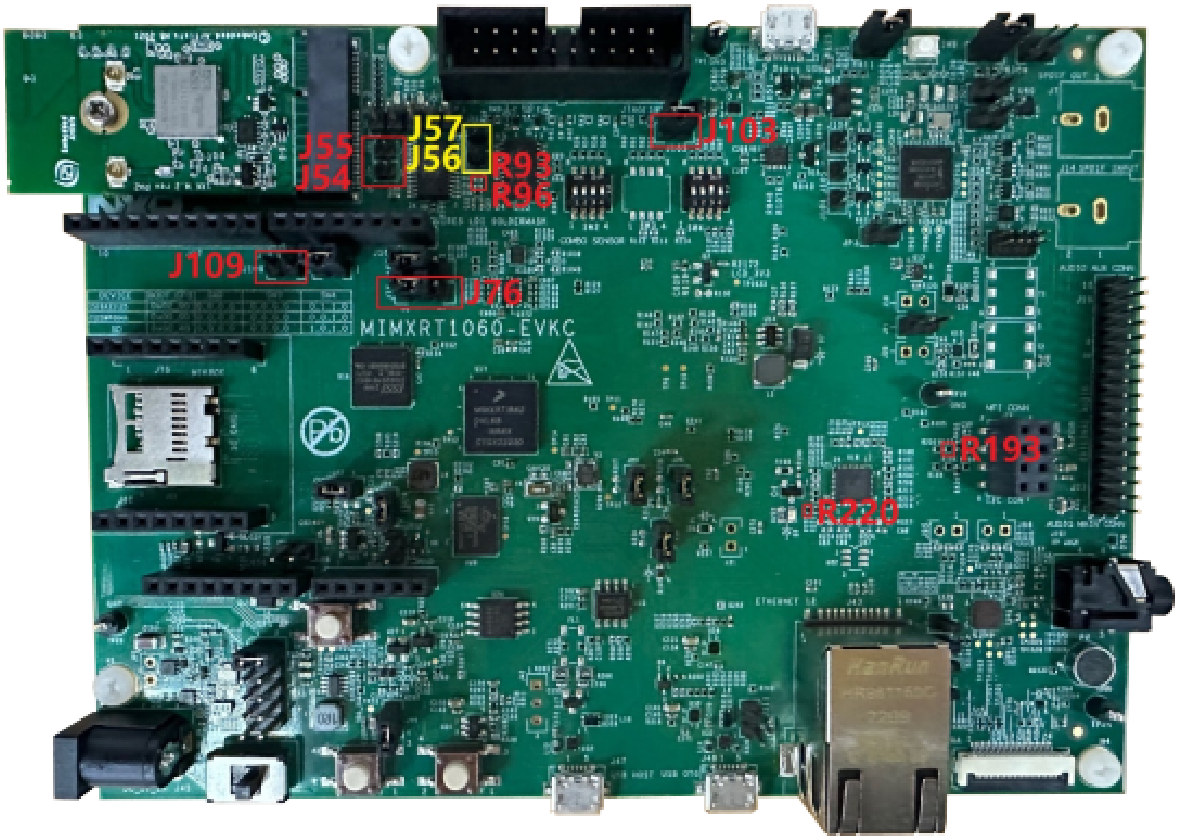
- **HCI UART rework**

1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

- **PCM interface rework**

1. Remove J54 and J55, connect J56 and J57.
2. Remove R220.
3. Connect J103.

Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



Parent topic:*Hardware Rework Guide for MIMXRT1060-EVKC and Murata M.2 Module*

Hardware Rework Guide for MIMXRT1060-EVKC and Murata 2EL M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP i.MX MIMXRT1060-EVKC and the Murata 2EL M.2 solution - direct M.2 connection to Embedded Artists' Rev-A1 (2EL) M.2 modules.

The hardware rework has three parts:

- HCI UART rework
- PCM interface rework
- LE Audio Synchronization interface rework (only used on sink side)

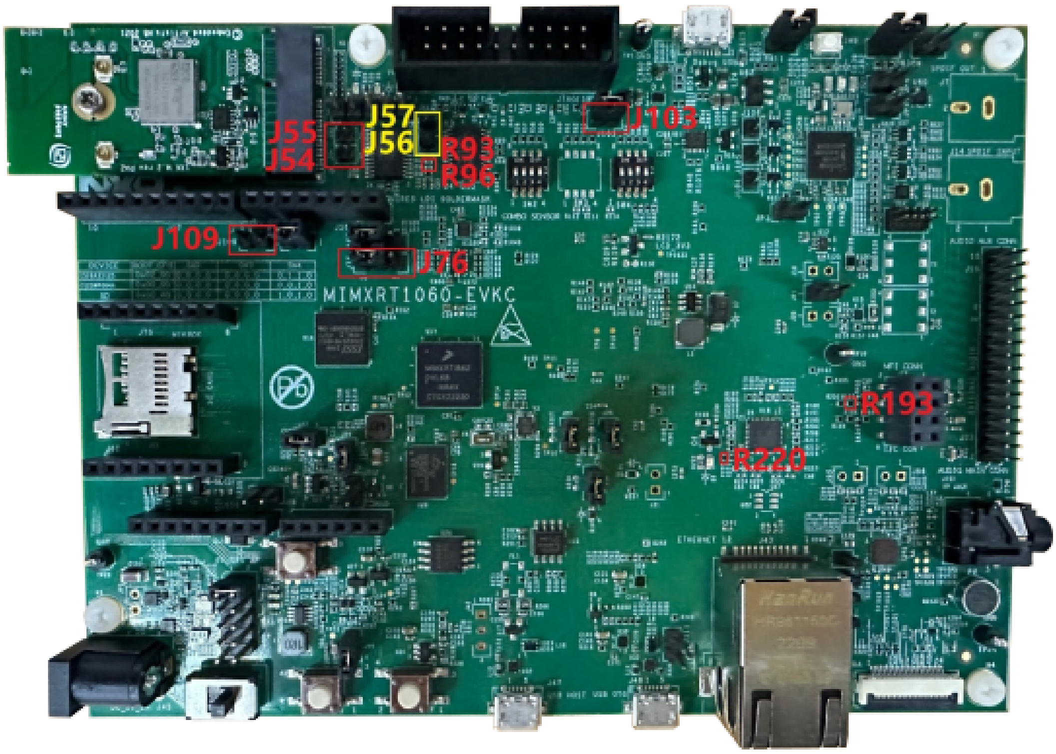
Hardware rework

- HCI UART rework

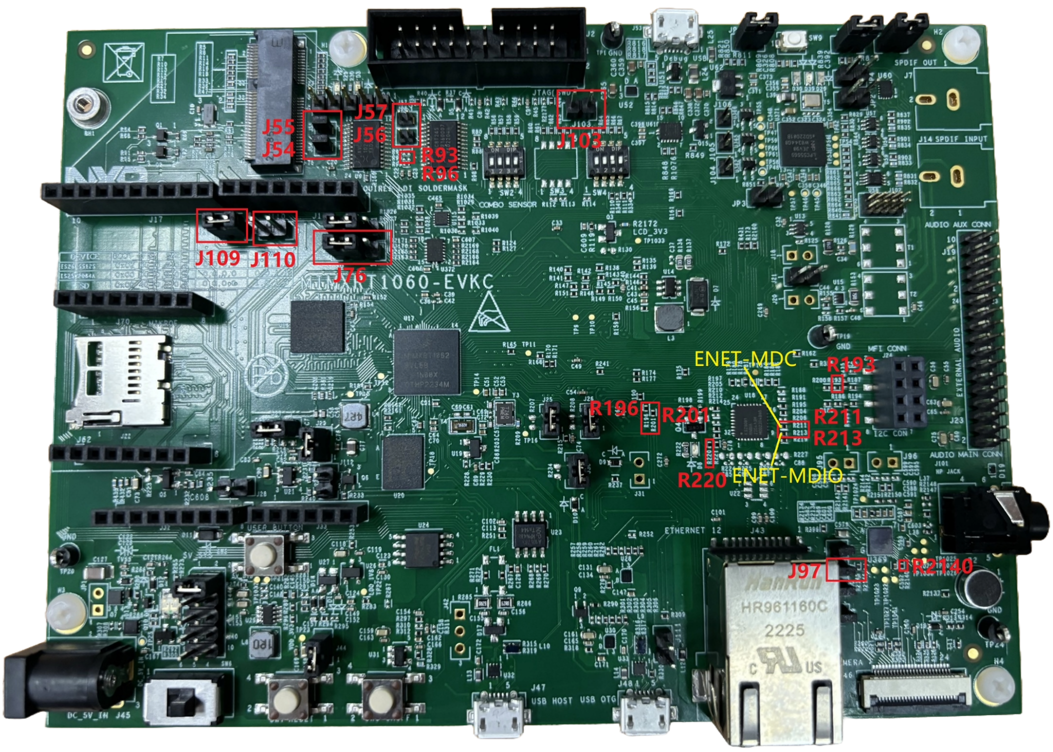
1. Mount R93, R96.
2. Remove R193.
3. Connect J109, connect J76 2-3.

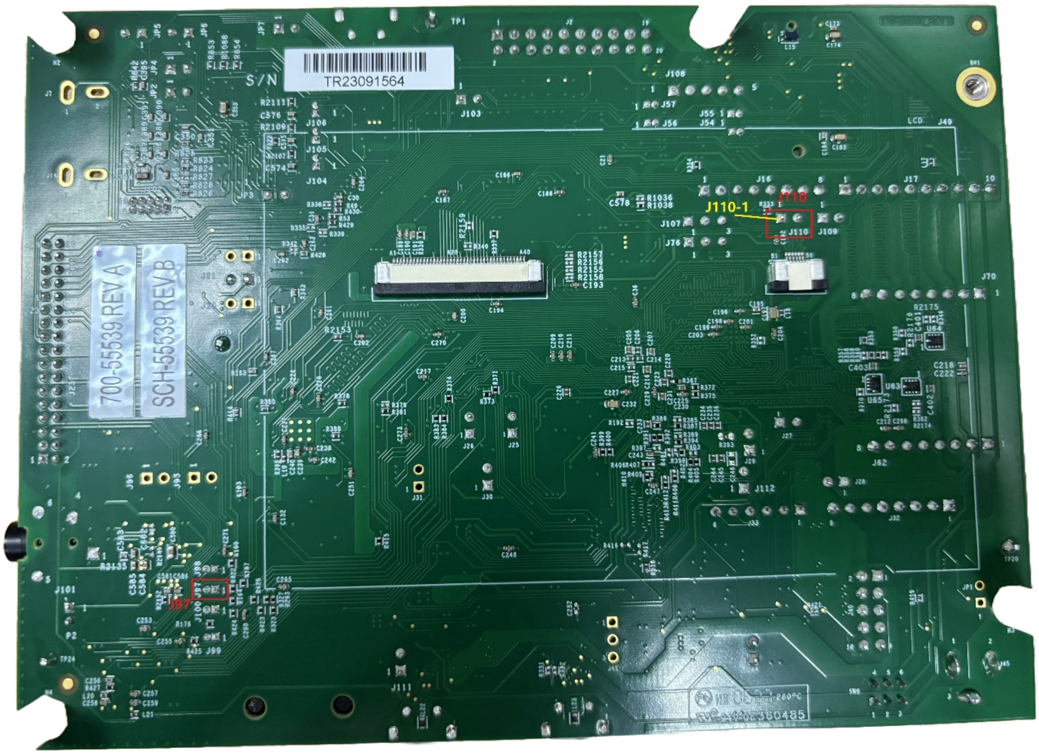
- PCM interface rework
    1. Remove J54 and J55, connect J56, and J57.
    2. Remove R220.
    3. Connect J103.

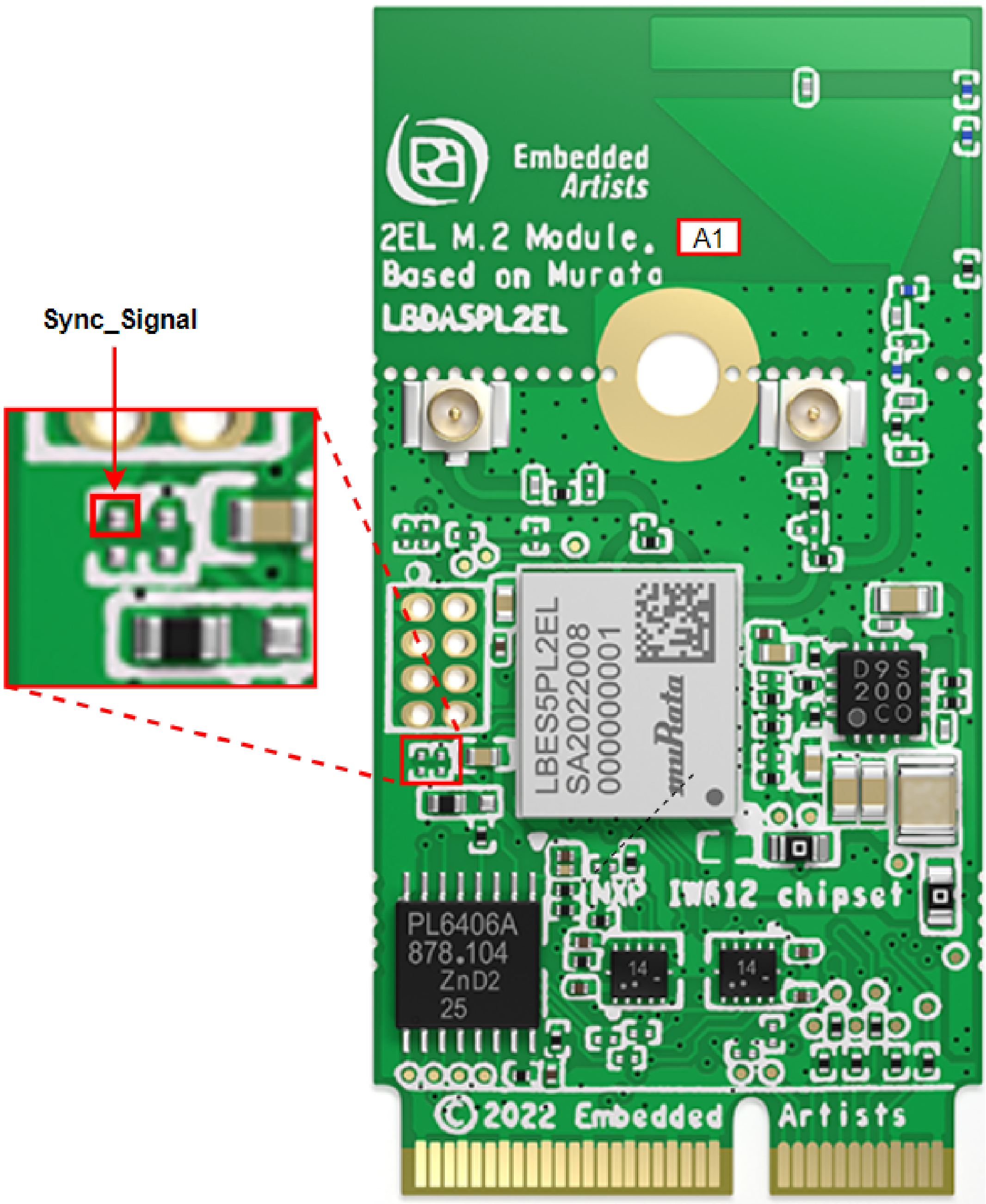
    Note: When J103 is connected, flash cannot be downloaded. So, remove the connection when downloading flash and reconnect it after downloading.



- LE Audio Synchronization interface rework (only used on sink side)
  1. Remove J110 jumper cap.
  2. Remove R196, R201, R213, and R211.
  3. Connect J110-1 (GPT2_CLK) to R2140 (SAI_MCLK).
  4. Connect ENET_MDIO (GPT2_CAP1) with J97 (SAI_SW).
  5. Connect ENET_MDC (GPT2_CAP2) with 2EL's GPIO_27 (Sync Signal).




J110-1
J110



Parent topic:*[Hardware Rework Guide for MIMXRT1060-EVKC and Murata 2EL M.2 Adapter]*

Hardware Rework Guide for MCXN547-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MCXN547-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

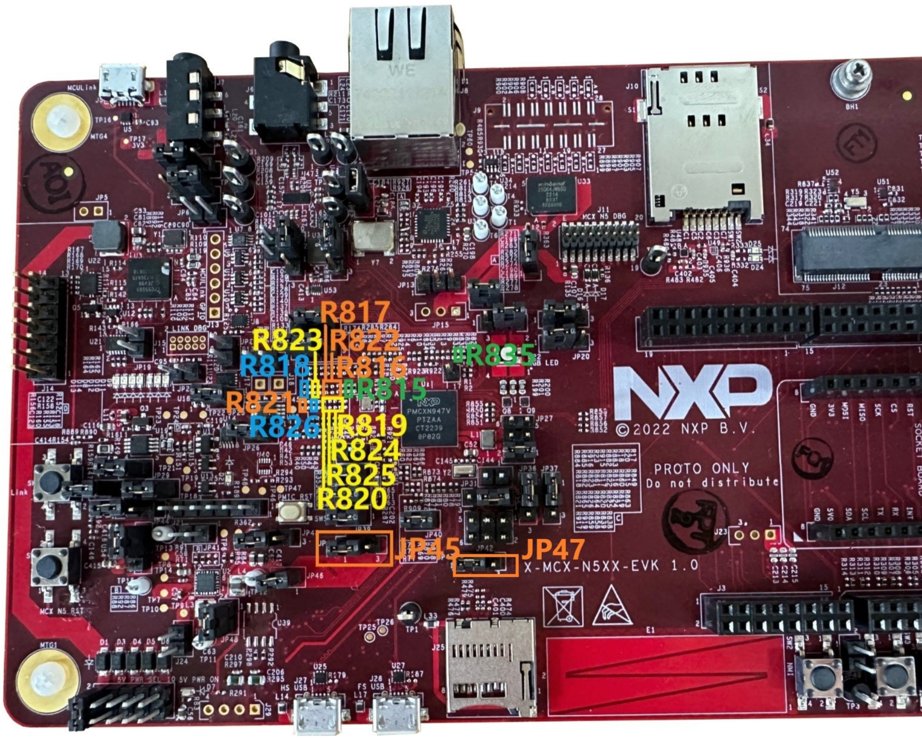
The hardware rework consists of two parts:

- M.2 UART interface
- M.2 SDIO interface

Hardware rework

- M.2 UART interface rework

  - Mount R835
  - Connect JP45 2-3 to supply 1.8V for GPIO4
- M.2 SDIO interface rework
  - Connect JP47 2-3 to supply 1.8V for GPIO2
  - Remove R818, connect R823
  - Remove R819, connect R824
  - Remove R817, connect R822
  - Remove R815, connect R816
  - Remove R820, connect R825



  - Remove R821, connect R826

Hardware Rework Guide for MCXN947-EVK and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MCXN947-EVK board and the Murata's 1XK, 1ZM or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM) or EAR00500 (2LL) M.2 modules.

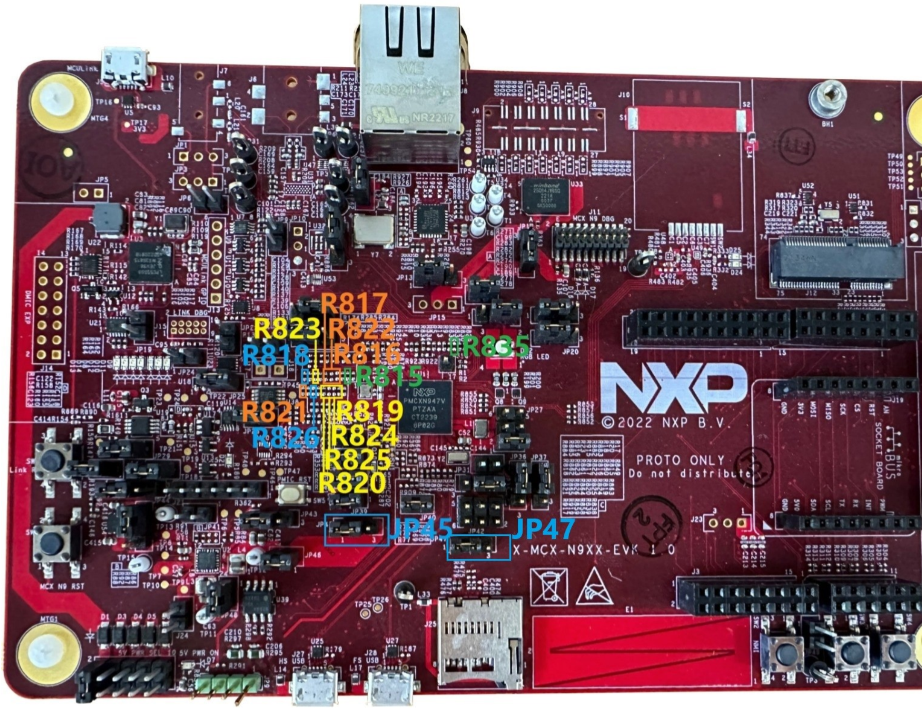
The hardware rework consists of two parts:

- M.2 UART interface
- M.2 SDIO interface

Hardware rework

- M.2 UART interface rework
  - Mount R835
  - Connect JP45 2-3 to supply 1.8V for GPIO4
- M.2 SDIO interface rework
  - Connect JP47 2-3 to supply 1.8V for GPIO2

- Remove R818, connect R823
- Remove R819, connect R824
- Remove R817, connect R822
- Remove R815, connect R816
- Remove R820, connect R825



- Remove R821, connect R826

Hardware Rework Guide for IMXRT1050-EVKB and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP IMXRT1050-EVKB board and the Murata 1XK,1ZM and 2EL solution - direct M.2 connection to Embedded Artists' EAR00385 (1XK) , EAR00364 (1ZM) or EAR00409 (2EL)M.2 modules. The hardware rework consists of three parts:

- Murata uSDM
- HCI UART rework

Hardware rework

- Murata uSD-M.2 jumper settings
  - J12 = 1-2: WLAN-SDIO & BT-PCM = 1.8 V
  - J13 = 1-2: BT-UART & WLAN/BT-CTRL = 3.3 V
  - J1 = 2-3: 3.3 V from uSD connector
- HCI UART interface rework

  Connect the TX/RX/RTS/CTS pins of the two boards as show in Table 1 using the jumper cables included in the Murata's uSD-M.2 Adapter kit as shown in the following table.

Pin name	uSD-M.2 adapter pin	i.MX RT1050-EVKB pin	Pin name of RT1050-EVKB	GPIO name of RT1050-EVKB
BT_UART_TXD_	J9 (pin 1)	J22 (pin 1)	LPUART3_RXD	GPIO_AD_B1_07
BT_UART_RXD_	J9 (pin 2)	J22 (pin 2)	LPUART3_TXD	GPIO_AD_B1_06
BT_UART_RTS_]	J8 (pin 3)	J23 (pin 3)	LPUART3_CTS	GPIO_AD_B1_04
BT_UART_CTS_]	J8 (pin 4)	J23 (pin 4)	LPUART3_RTS	GPIO_AD_B1_05
GND	J7 (pin 7)	J25 (pin 7)	GND	GND

Parent topic:*Hardware Rework Guide for IMXRT1050-EVKB and Murata M.2 Module*

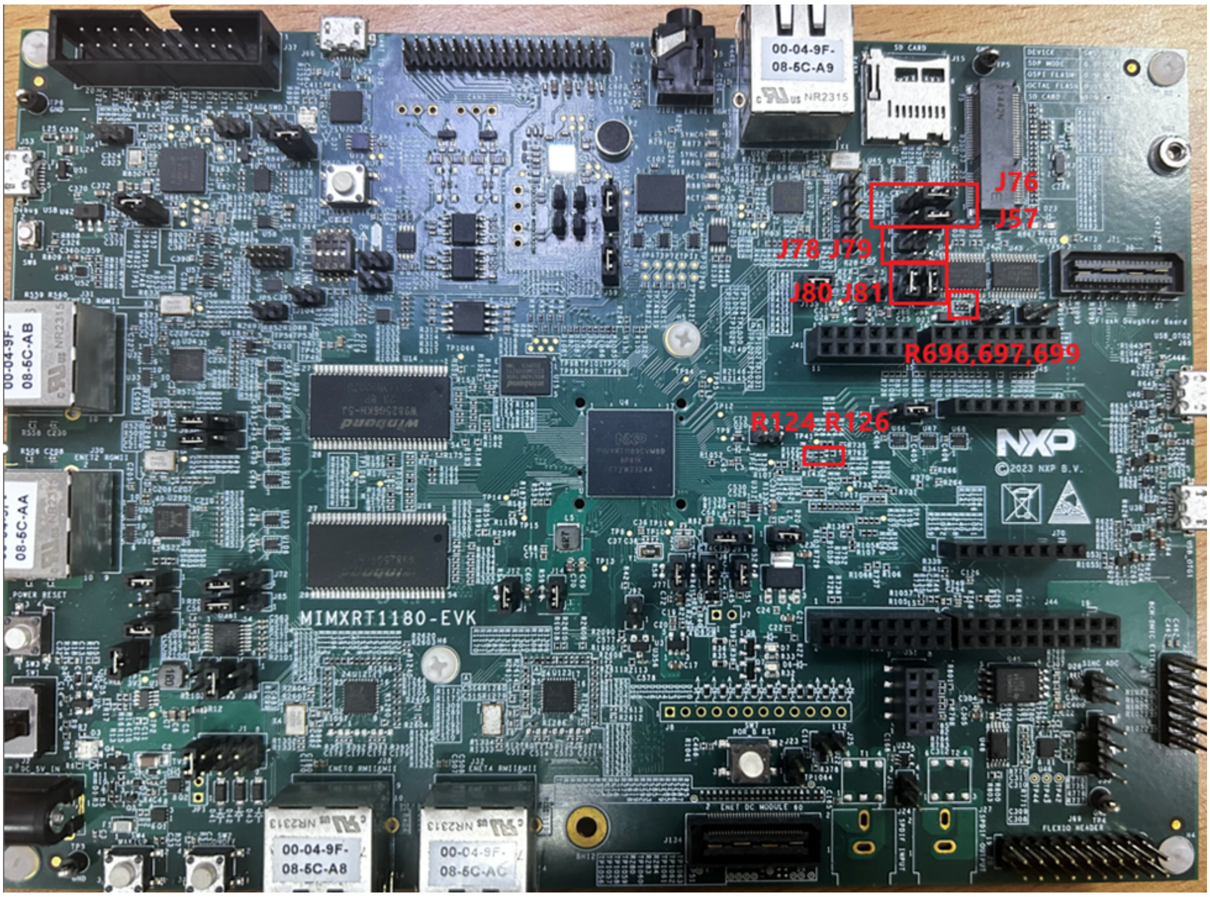
Hardware Rework Guide for MIMXRT1180 and Murata M.2 Module This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP MIMXRT1180 board and the Murata's 1XK, 1ZM, 2EL or 2LL solution - direct M.2 connection to Embedded Artists EAR00385 (1XK), EAR00364 (1ZM), Rev-A1 (2EL) or EAR00500 (2LL) M.2 modules.

The hardware rework consists of two parts:

- HCI UART rework
- PCM interface rework

Hardware rework

- HCI UART rework:
  - Remove: R124,R126
  - Mount R696, R697
  - Connect J57 [2-3], J76 [2-3]
- PCM interface rework
  - Mount R699
  - Disconnect J78 J79
  - Connect J80 J81



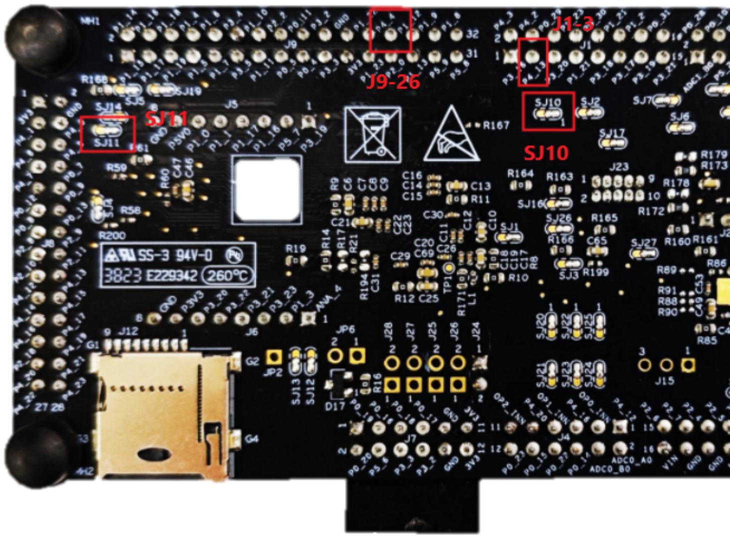
Hardware Rework Guide for FRDM-MCXN947 and X-FRDM-WIFI-M.2 Adapter This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and X-FRDM-WIFI-M.2 or the Murata's 2LL EAR00500 (2LL) M.2 modules solution.

The hardware rework consists of one part:

- UART interface rework

Hardware rework

- UART interface rework
  - Remove SJ11 1-2, connect SJ11 2-3
  - Remove SJ10 1-2, connect J1-3 to J9-26
- X-FRDM-WIFI-M.2 jumper setting
  - Connect J8(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J24(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J19(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J25(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J15(On X-FRDM-WIFI-M.2) for 1.8V
  - Connect J16(On X-FRDM-WIFI-M.2) for 3.3V
  - Connect J17(On X-FRDM-WIFI-M.2) for 1.8V



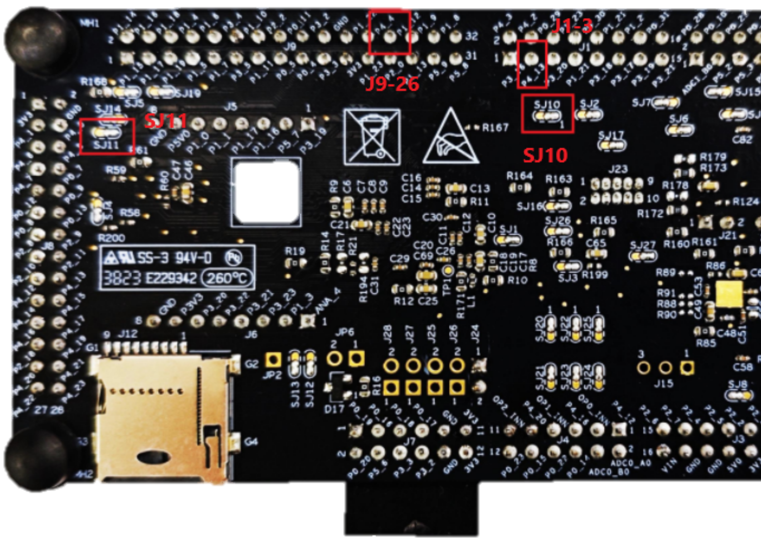
   - Connect J18(On X-FRDM-WIFI-M.2) for 3.3V

Hardware Rework Guide for FRDM-MCXN947 and FRDM-IW416-AW-AM510 This section is a brief hardware rework guidance of the EdgeFast Bluetooth PAL on the NXP FRDM-MCXN947 board and FRDM-IW416-AW-AM510 board. The hardware rework consists of two parts:

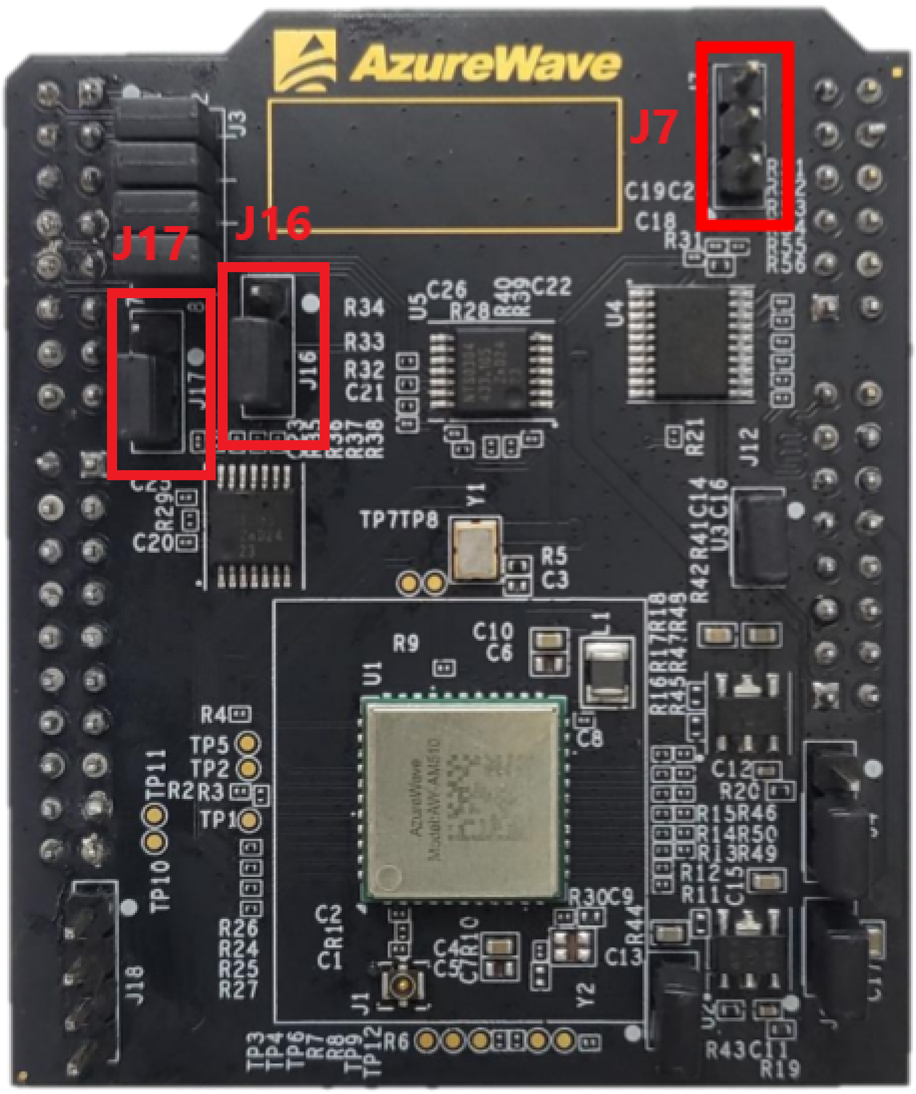
- UART interface rework
- FRDM-IW416-AW-AM510

Hardware rework

- UART interface rework
  - Remove SJ11 1-2, connect SJ11 2-3
  - Remove SJ10 1-2, connect J1-3 to J9-26
- FRDM-IW416-AW-AM510 jumper setting
  - Connect J16 2-3 for 3.3V supply
  - Connect J17 2-3 for 3.3V UART voltage level



  - Connect J7 2-3 for 3.3V SDIO voltage level



Enabling Additional EdgeFast Bluetooth Protocol Abstraction Layer Examples on RT1064

Introduction NXP supports Bluetooth/Bluetooth Low Energy on RT1060EVK and RT1060EVKC. RT1064 has the same MCU die with RT1060EVK and RT1060EVKC and therefore it is possible to migrate the examples.

This document takes *peripheral_ht* as an example and describes the steps to migrate EdgeFast examples from RT1060EVK to RT1064 (based on SDK 2.13.0) and from RT1060EVKC to RT1064 (based on SDK 2.14.0) with different toolchains including IAR, Arm GCC, and MDK.

Migrate examples from RT1060EVK to RT1064 This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDKtoolchains.

Common steps

1. Download *SDK_2.13.0_EVK-MIMXRT1060* and *SDK_2.13.0_EVK-MIMXRT1064*.
2. Copy the following folders from RT1060EVK package to RT1064 package: <install_dir>/components/internal_flash/ <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/.

3. Create a folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evk_install_dir>/boards/evkmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to < rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy *clock_config.[c/h]* and *board.c* from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.
6. Add #define EDGEFAST_BT_LITTLEFS_MFLASH 1 in <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht /app_config.c.
7. Make the following changes in <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth/peripheral_ht/board.h.

```
#define BOARD_FLASH_SIZE (0x800000U)
```

```
#define BOARD_FLASH_SIZE (0x400000U)
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes.

File name	Previous item	New item
peripheral_ht.ewp	1060 1062	1064 1064

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.

```
define symbol m_interrupts_start         = 0x60002000;
define symbol m_interrupts_end           = 0x600023FF;

define symbol m_text_start               = 0x60002400;
define symbol _ROM_END_                  = 0x6057FFFF;
```

```
define symbol m_interrupts_start         = 0x70002000;
define symbol m_interrupts_end           = 0x700023FF;

define symbol m_text_start               = 0x70002400;
define symbol _ROM_END_                  = 0x7017FFFF;
```

```
define exported symbol m_boot_hdr_conf_start = 0x60000000;
define symbol m_boot_hdr_ivt_start           = 0x60001000;
define symbol m_boot_hdr_boot_data_start     = 0x60001020;
define symbol m_boot_hdr_dcd_data_start      = 0x60001030;
```

```
define exported symbol m_boot_hdr_conf_start = 0x70000000;
define symbol m_boot_hdr_ivt_start           = 0x70001000;
define symbol m_boot_hdr_boot_data_start     = 0x70001020;
define symbol m_boot_hdr_dcd_data_start      = 0x70001030;
```

```
BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash secto
BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x60800000 - EDGEFAST_BT_
***/
```

```
BT_LITTLEFS_STORAGE_SECTOR_SIZE = 0x1000;  /* 4k flash sect
BT_LITTLEFS_STORAGE_MAX_SECTORS = (0x70400000 - EDGEFAST_BT
***/
```

Parent topic:*Migrate examples from RT1060EVK to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.
2. Rename the following files.

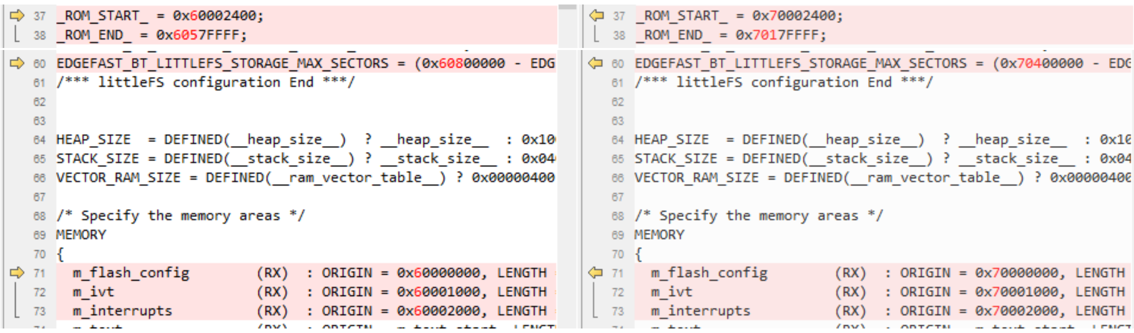
Path	Previous name	**New name**
<rt1064_install_dir>/middleware/wireless/ethermind/	middleware_edgefast_bluetooth_cmake	middleware_edgefast_bluetooth_k32w061_contro cmake

3. Make following changes.

File name	Previous item	New item
config.cmake	1060	1064
	1062	1064
flags.cmake	1062	1064
CMakeLists.txt	1060	1064
	1062	1064

4. *mflash* is used in RT1064 instead of flash_adapter,therefore, comment include(component_flexspi_nor_flash_adapter_rt1064_MIMXRT1064) in *CMakeLists.txt*.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.



Parent topic:[Migrate examples from RT1060EVK to RT1064]()

MDK

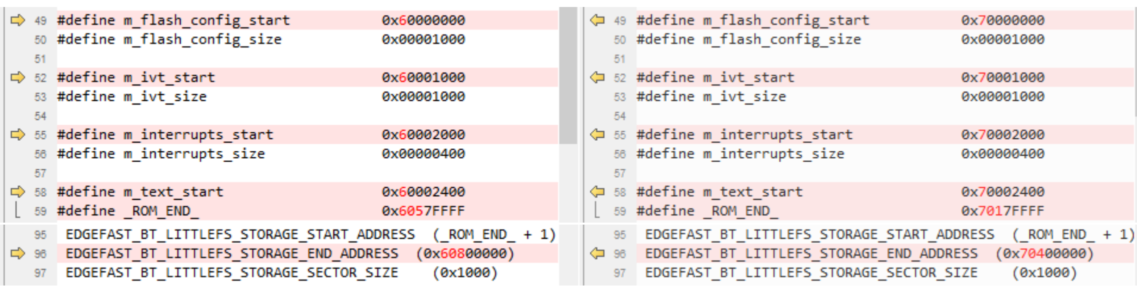
1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

2. Make following changes.

File name	Previous item	New item
peripheral_ht.uvprojx	1060	1064
	1062	1064

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.

4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.



Parent topic: *Migrate examples from RT1060EVK to RT1064*

Migrate examples from RT1060EVKC to RT1064

This topic describes the Common steps and the steps to migrate with the IAR, Arm GCC, and MDK toolchains.

Common steps

1. Download SDK_2.14.0_EVKC-MIMXRT1060 and SDK_2.14.0_EVK-MIMXRT1064.
2. Copy the following folders from the RT1060EVKC package to the RT1064 package: <install_dir>/middleware/edgefast_bluetooth/ <install_dir>/middleware/wireless/ethermind.
3. Create a new folder named edgefast_bluetooth_examples/ under <rt1064_install_dir>/boards/evkmimxrt1064/.
4. Copy the entire folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/peripheral_ht/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/.
5. Copy clock_config.[c/h] and board.c from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/ to <rt1064_installed>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/ to replace the previous files.

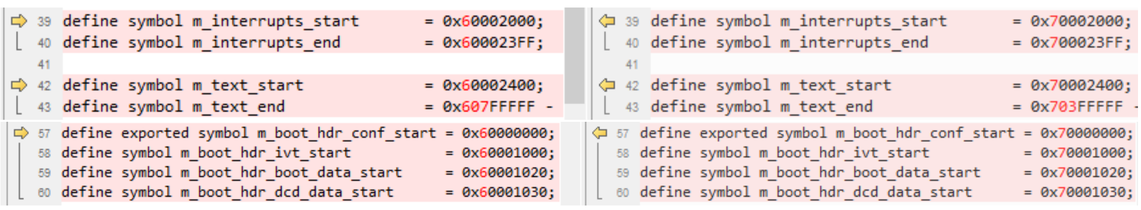
Parent topic: *Migrate examples from RT1060EVKC to RT1064*

IAR

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/iar/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.ewp	1062	1064
	mflash/evkcmimxrt1060	mflash/mimxrt1064
	evkcmimxrt1060	evkmimxrt1064
	6B	6A

3. Rename MIMXRT1062xxxxx_flexspi_nor.icf as MIMXRT1064xxxxx_flexspi_nor.icf and make the following changes.



Parent topic: *Migrate examples from RT1060EVKC to RT1064*

Arm GCC

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/armgcc/.

2. Copy folder from <rt1060evkc_install_dir>/boards/evkcmimxrt1060/edgefast_bluetooth_examples/template/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/ and rename the files.

   |**Path**|**Previous name**|**New name**| |<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/|middleware_edgefast_bluetooth_mcux_linker_template_evkcmimxrt1060.cmake|middleware_edgefast_bluetooth_mcux_linker_template_evkmimxrt1064.cmake| |middleware_edgefast_bluetooth_sdio_template_evkcmimxrt1060.cmake|middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064.cmake|

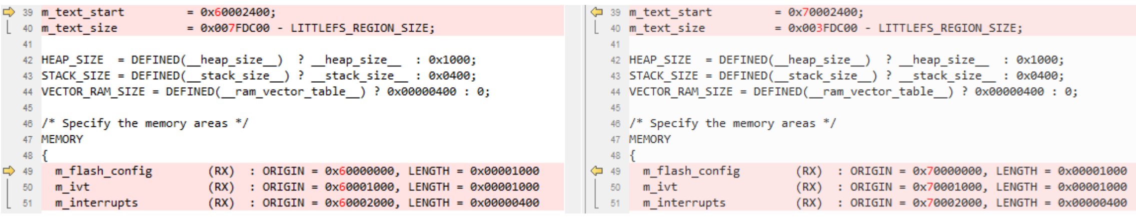
3. Add the following content to<rt1064_install_dir>/devices/MIMXRT1064/all_lib_device.cmake at appropriate location.

```
...
  ${CMAKE_CURRENT_LIST_DIR}/../../boards
  ${CMAKE_CURRENT_LIST_DIR}/../../boards/evkmimxrt1064/edgefast_bluetooth_examples/
↪template
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/edgefast_bluetooth
  ${CMAKE_CURRENT_LIST_DIR}/../../middleware/wireless/ethermind
...
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_ble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_br_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_cm7f)
include_if_use(middleware_edgefast_bluetooth_btble_ethermind_lib_cm7f)
include_if_use(middleware_edgefast_bluetooth_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci)
include_if_use(middleware_edgefast_bluetooth_common_ethermind_hci_uart)
include_if_use(middleware_edgefast_bluetooth_config_ethermind)
include_if_use(middleware_edgefast_bluetooth_config_template)
include_if_use(middleware_edgefast_bluetooth_extension_common_ethermind)
include_if_use(middleware_edgefast_bluetooth_k32w061_controller)
include_if_use(middleware_edgefast_bluetooth_mcux_linker_template_ evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_pal)
include_if_use(middleware_edgefast_bluetooth_pal_db_gen_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_host_msd_fatfs_ethermind)
include_if_use(middleware_edgefast_bluetooth_pal_platform_ethermind)
include_if_use(middleware_edgefast_bluetooth_porting)
include_if_use(middleware_edgefast_bluetooth_porting_atomic)
include_if_use(middleware_edgefast_bluetooth_porting_list)
include_if_use(middleware_edgefast_bluetooth_porting_net)
include_if_use(middleware_edgefast_bluetooth_porting_toolchain)
include_if_use(middleware_edgefast_bluetooth_porting_work_queue)
include_if_use(middleware_edgefast_bluetooth_profile_bas)
include_if_use(middleware_edgefast_bluetooth_profile_dis)
include_if_use(middleware_edgefast_bluetooth_profile_fmp)
include_if_use(middleware_edgefast_bluetooth_profile_hps)
include_if_use(middleware_edgefast_bluetooth_profile_hrs)
include_if_use(middleware_edgefast_bluetooth_profile_hts)
include_if_use(middleware_edgefast_bluetooth_profile_ipsp)
include_if_use(middleware_edgefast_bluetooth_profile_pxr)
include_if_use(middleware_edgefast_bluetooth_profile_tip)
include_if_use(middleware_edgefast_bluetooth_profile_wu)
include_if_use(middleware_edgefast_bluetooth_sdio_template_evkmimxrt1064)
include_if_use(middleware_edgefast_bluetooth_shell)
include_if_use(middleware_edgefast_bluetooth_shell_ble)
include_if_use(middleware_edgefast_bluetooth_template)
include_if_use(middleware_edgefast_bluetooth_wifi_nxp_controller_base)...
```

4. Make the following changes in the listed order.

File name	Previous item	New item
config.cmake	MIMXRT1	MIMXRT1064xxxxA
mflash_evkcmimxrt1060	mflash_rt1	
1062	1064	
evkcmimxrt1060	evk-mimxrt10	
flags.cmake	1062	1064
6B	6A	
CMakeLists.txt	1062	1064
<rt1064_install_dir>/middleware/edgefast_bluetooth/middleware_edgefast_bluetooth_template.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_common_ethermind_hci_uart.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_k32w061_controller.cmake	1062	1064
<rt1064_install_dir>/middleware/wireless/ethermind/middleware_edgefast_bluetooth_wifi_nxp_controller_base.cmake	evkcmimx	evk-mimxrt1064
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_mcux_ cmake	1062	1064 mxrt1064.
<rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/middleware_edgefast_bluetooth_sdio_t cmake	1062	1064 1.

5. Rename MIMXRT1062xxxxx_flexspi_nor.ld as MIMXRT1064xxxxx_flexspi_nor.ld and make the following changes.



Parent topic:*Migrate examples from RT1060EVKC to RT1064*

MDK

1. Navigate to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.
2. Make the following changes in the listed order.

File name	Previous item	New item
peripheral_ht.uvprojx	1062	1064
	mflash/evkcmimxrt1060	mflash/mimxrt1064
	evkcmimxrt1060	evkcmimxrt1064
	6B	6A

3. Copy evkmimxrt1064_flexspi_nor.ini from <rt1064_install_dir>/boards/evkmimxrt1064/demo_apps/hello_world/mdk/ to <rt1064_install_dir>/boards/evkmimxrt1064/edgefast_bluetooth_examples/peripheral_ht/mdk/.
4. Rename MIMXRT1062xxxxx_flexspi_nor as MIMXRT1064xxxxx_flexspi_nor and make the following changes.

```
43 #define m_flash_config_start      0x60000000
44 #define m_flash_config_size       0x00001000
45
46 #define m_ivt_start               0x60001000
47 #define m_ivt_size                0x00001000
48
49 #define m_interrupts_start        0x60002000
50 #define m_interrupts_size         0x00000400
51
52 #define m_text_start              0x60002400
53 #define m_text_size               0x007FDC00 - LITTLEFS
```

```
43 #define m_flash_config_start      0x70000000
44 #define m_flash_config_size       0x00001000
45
46 #define m_ivt_start               0x70001000
47 #define m_ivt_size                0x00001000
48
49 #define m_interrupts_start        0x70002000
50 #define m_interrupts_size         0x00000400
51
52 #define m_text_start              0x70002400
53 #define m_text_size               0x003FDC00 - LITTLEFS
```

Parent topic:*Migrate examples from RT1060EVKC to RT1064*

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

Enabling Additional Edgefast BT PAL Examples on M4 core for RT1170

Introduction RT1170 works with two cores: M7 and M4, on which both all EdgeFast examples can run. However, all the EdgeFast examples in the release package are enabled on M7. Only the A2DP source example is enabled on M4.

EdgeFast projects for both the cores share the demo source files but with different project settings. Therefore, the examples can be migrated.

This document describes the steps to migrate EdgeFast examples from M7 to M4 with different toolchains. There are four main steps required. Additionally, you can also delete the function.

1. Create an M4 project
2. Rearrange source files
3. Rearrange project files

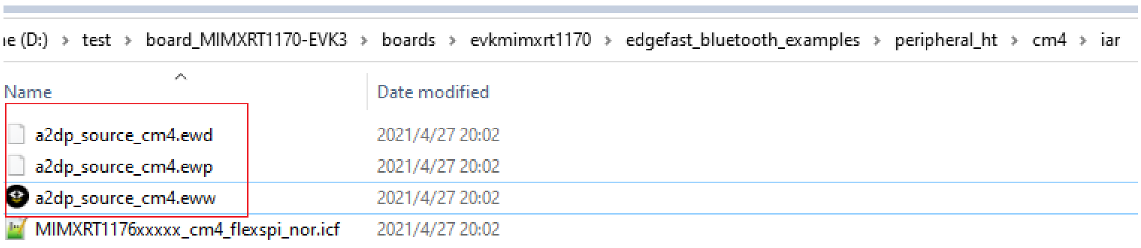
4. Adjust project settings
5. Delete function

In this document, the peripheral_ht example is used to demonstrate how to enable EdgeFast examples on M4 core with IAR and ARMGCC.

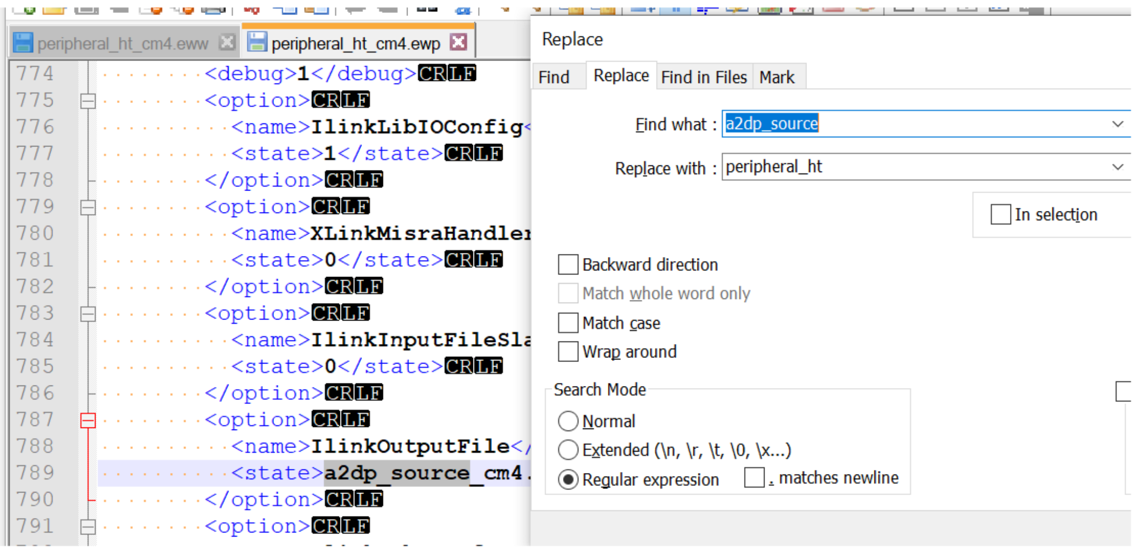
IAR This section describes the steps to create an M4 project with IAR, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project To create an M4 project, perform the following steps:

1. Copy the folder *cm4* in the directory *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_* into the folder in which the example should be enabled. In this case, copy the folder *cm4* into the directory *<install_dir>\boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht*.
2. Open the folder *iar* in the directory *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\periph*
3. Rename the files. Change the file name name *a2dp_source_cm4* to *peripheral_ht_cm4* in all the respective files.



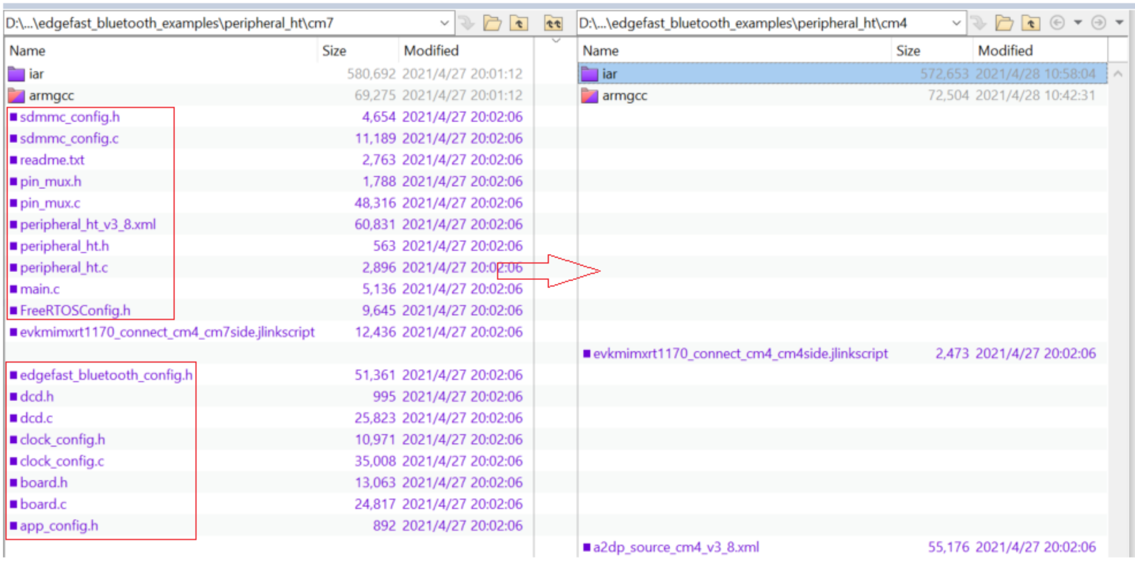
4. Open the files *peripheral_ht_cm4.eww* and *peripheral_ht_cm4.ewp* with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio Code.
5. Search and replace all *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.



Parent topic:*IAR*

Rearrange source files To rearrange source files, perform the following steps:

1. Open the folder *cm4* in the directory *<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\perip* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h from the folder *boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7* to the folder *<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

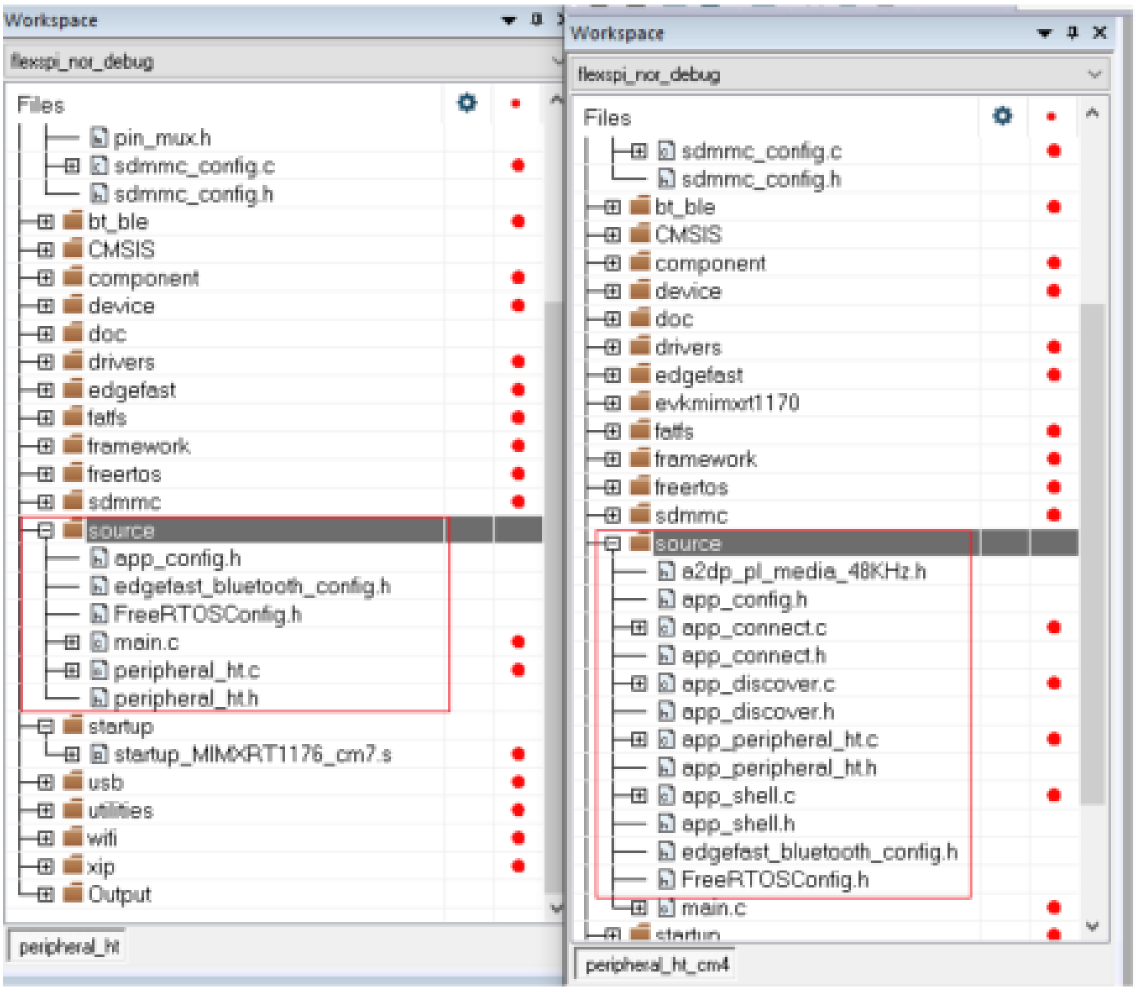


|

Parent topic: *IAR*

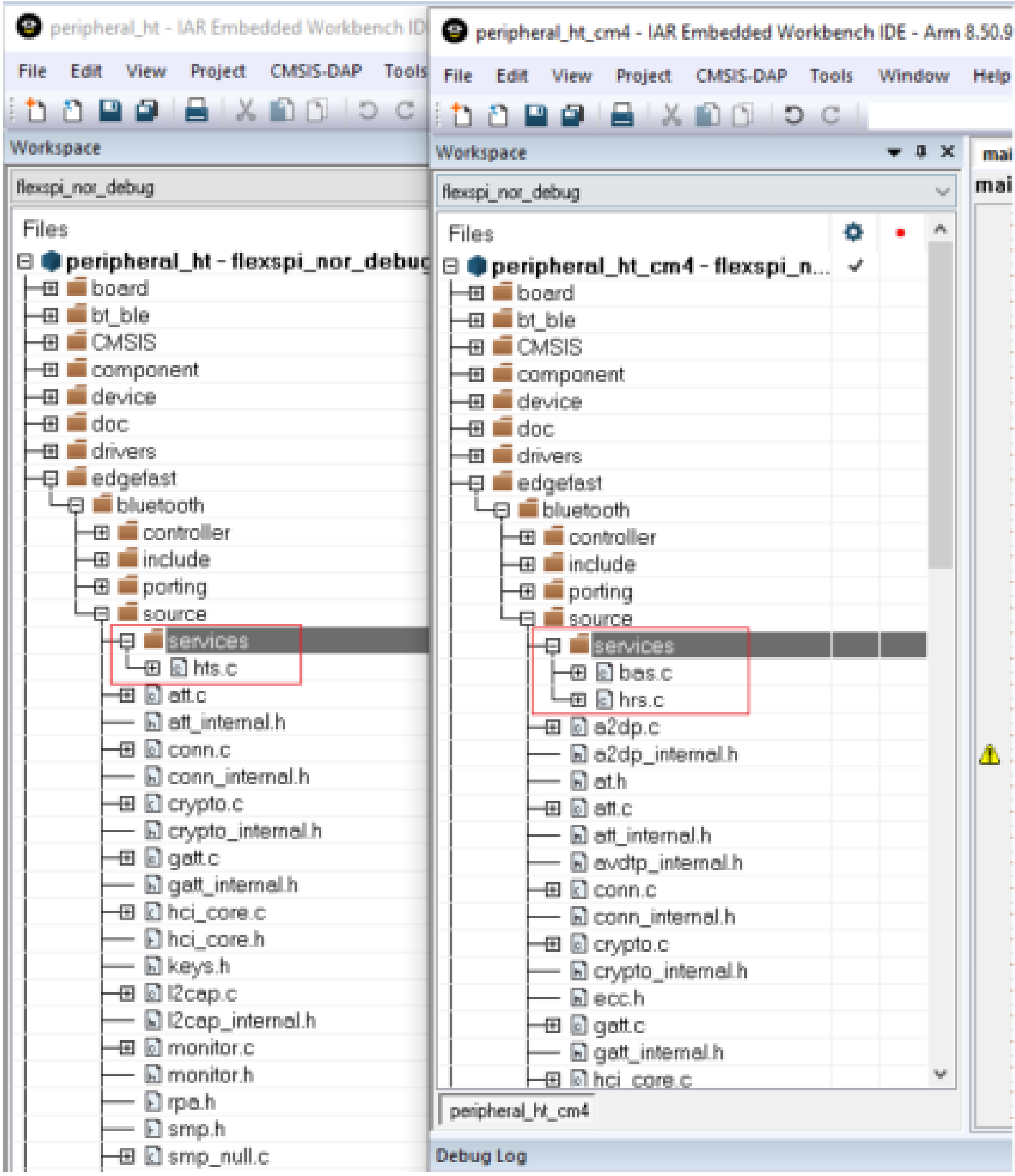
Rearrange project files To rearrange project files, perform the following steps:

1. Open the *peripheral_ht _cm7*and *peripheral_ht _cm4* IAR projects in the directories *<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht \cm7\iar* and *<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht \cm4\iar*.
   1. Compare the whole project directory, find file groups that the cm7 project has but are missing in the cm4 project. Add the missing file groups from the cm7 project into the cm4 project.
   2. Compare the difference between the two groups with the same name. Remove files that do not exist in the cm7 project but exist in the cm4 project. Find files that are available in the cm7 project but are missing in the cm4 project. Add the missing files from the cm7 project into the cm4 project.
2. For example, in the following figure, the files in the source group in the cm4 project must be removed, and the files in the path: *<install_dir>\boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht* with the same name as the files in the cm7 project must be added into the *source* group.



3. Compare the *services* group.

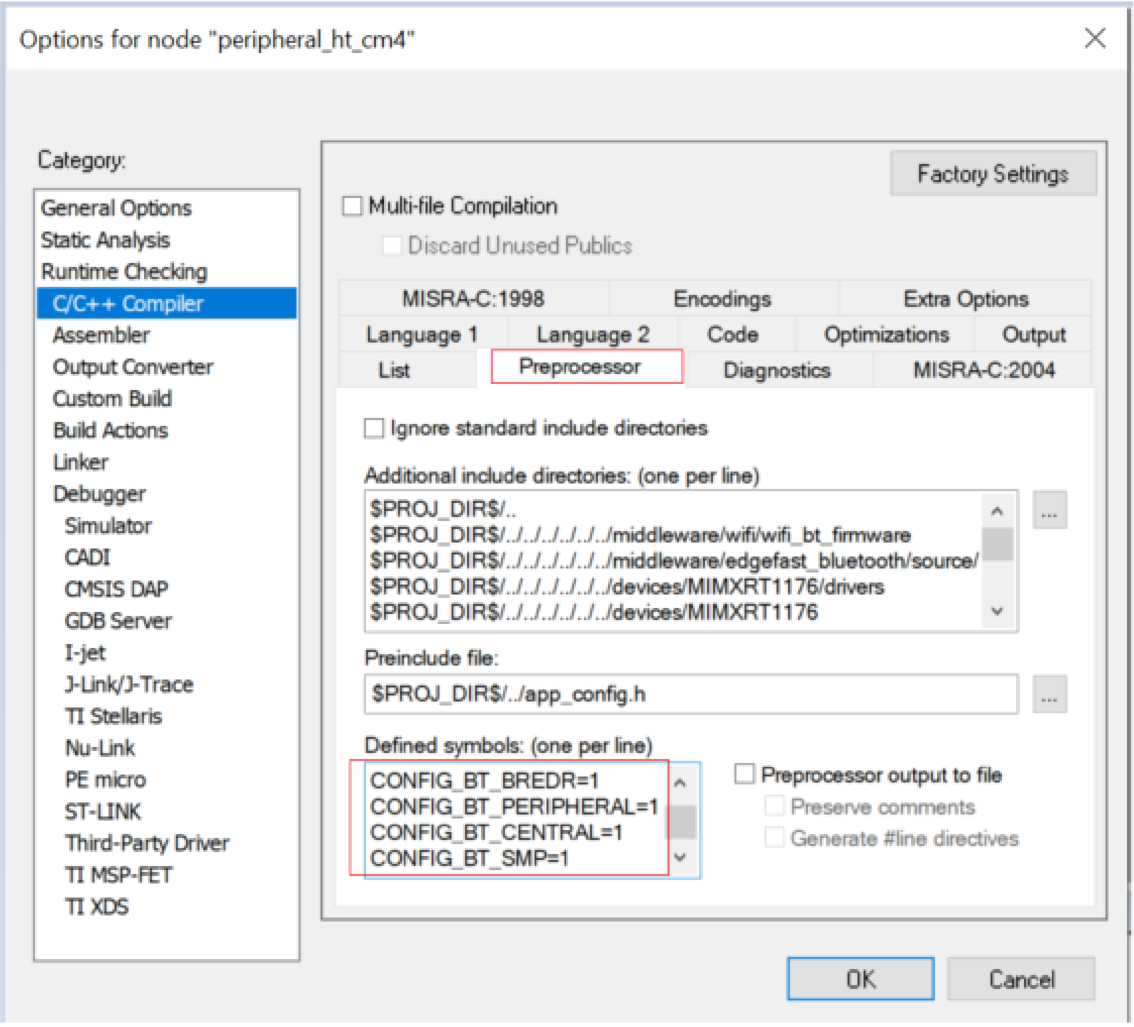
   The peripheral hts profile is in the *services* folder. Add the hts.c file to the *services* group of the *cm4* folder.



Parent topic: *IAR*

Adjust project settings To adjust the project settings, perform the following steps:

1. Compare the macro in the project settings: **Option** > **C/C++ compiler** > **Preprocessor**.
2. Find the macros that do not exist in the **cm4** project but are available in the **cm7** project. Delete these macro. The rule is that **m7** macro setting should be same with **m4**.



|

|

```
The macros are in the**peripheral\_ht\_cm4.ewp** file.

|![](../images/image7.png "Compare the peripheral_ht_cm4.ewp file")
```

|

Parent topic:[IAR]

Delete function As a final step, remove the function "*SCB_DisableDCache();* in main.c.

On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:[IAR]

Arm GCC This section describes the steps to create an M4 project with Arm GCC, rearrange source and project files, adjust project settings, and delete function.

Create an M4 project To create an M4 project, perform the following steps:

1. Copy the folder *<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source \cm4* into another folder in which the example should be enabled. In this case, copy the folder *<install_dir>*

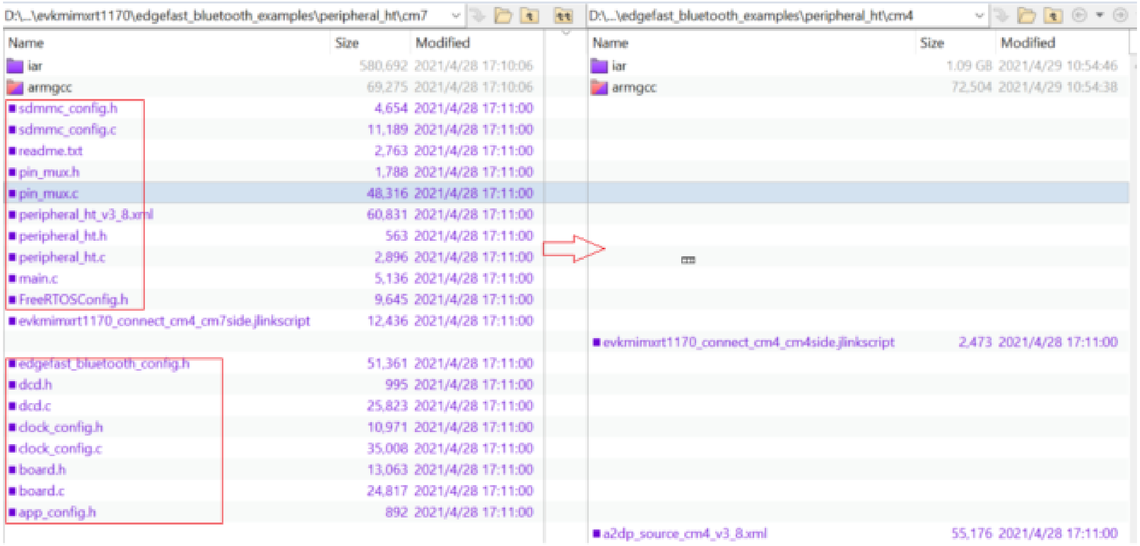
boards\evkmimxrt1170\edgefast_bluetooth_examples\a2dp_source \cm4 into*<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

2. Open the file *CMakeLists.txt* located in the path: *<install_dir> boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\ armgcc*.
3. Search and replace all *a2dp_source_cm4* with *peripheral_ht_cm4*, and then save the files.

Parent topic:*Arm GCC*

Rearrange source files To rearrange source files, perform the following steps:

1. Open the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4* and delete all files with the extensions *.c and *.h.
2. Copy the files with the extensions *.c and *.h in the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7* to the folder *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4*.

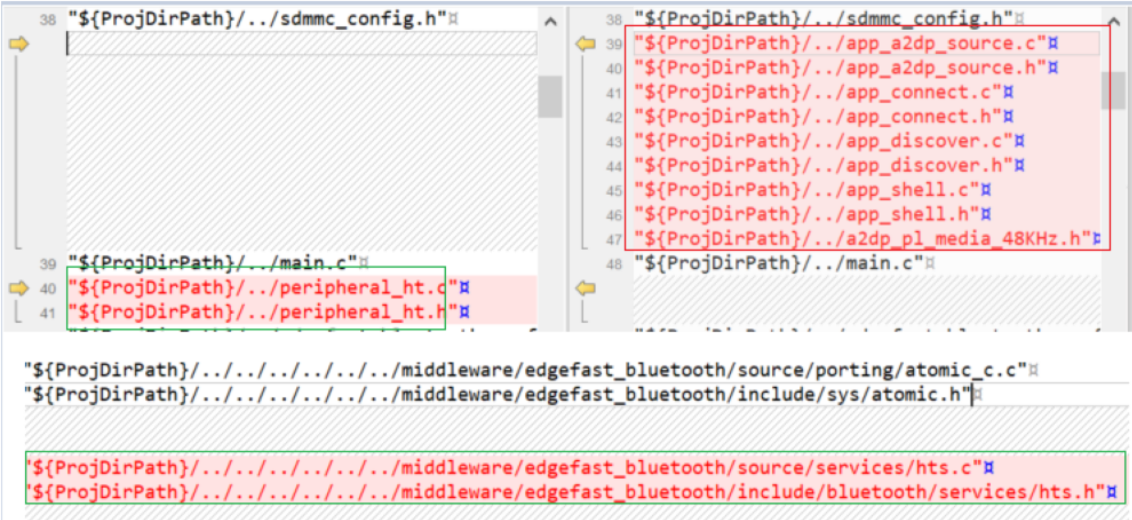


|

Parent topic:*Arm GCC*

Rearrange project files To rearrange project files, perform the following steps:

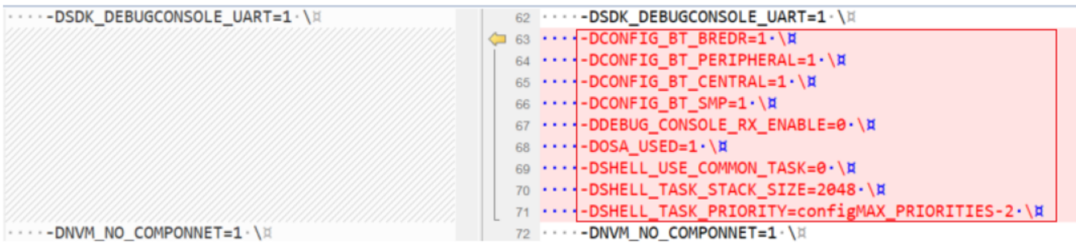
1. Open the *CMakeLists.txt* of the two examples respectively. The two files are in the *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\armgcc* and *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\armgcc* folders respectively.
2. Search the section *add_executable*. Compare the difference between the two sections. Remove files that do not exist in the *cm7* project but are available in the *cm4* project. Add the files that exist in the *cm7* project but are not available in the *cm4* project into the *cm4* project. For example, in the following figure, the files in the red box should be removed and the files in the green box must be added into the *cm4* project.



Parent topic:[Arm GCC]

Adjust project setting To adjust the project settings, perform the following steps:

1. Open the *flags.cmake* of the two examples respectively. The two files are in the *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\armgcc* and *<install_dir>boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\armgcc* folders respectively.
2. Search the ***CMAKE_C_FLAGS_DEBUG*** section.
    1. Compare the macro between the two sections.
    2. Add the macros that do not exist in the **cm4** project but are available in the **cm7** project into the cm4 project. The rule is that macro setting should be same.
    3. Delete the macros highlighted in the red rectangle.



|

Parent topic:[Arm GCC]

Delete function As a final step, remove the function "*SCB_DisableDCache()* in main.c.

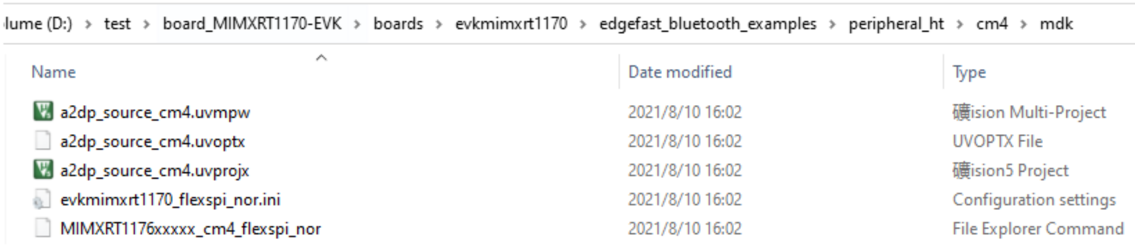
On the completion of the above steps, the M7 project successfully migrates to an M4 project. You can now download and debug the M4 example project.

Parent topic:[Arm GCC]

MDK This section describes the steps to create an M4 project with MDK, rearrange source and project files, adjust project settings, and delete function.

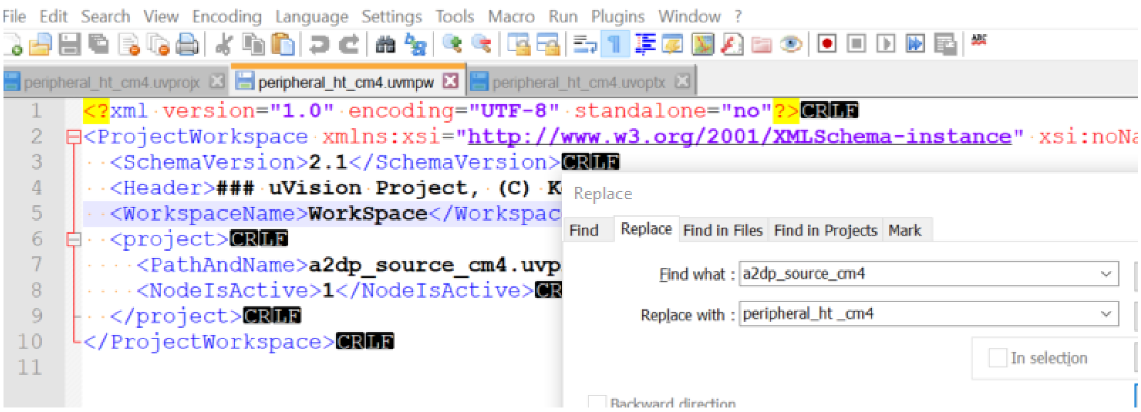
Create an M4 project

1. Copy folder *cm4* from *<install_dir>boards|evkmimxrt1170|edgefast_bluetooth_examples|a2dp_source|cm4* into the folder in where the example must be enabled. In this case, copy folder *cm4* into directory *<install_dir>|boards|evkmimxrt1170|edgefast_bluetooth_examples|peripheral_ht*.
2. Open folder mdk from *<install_dir>boards|evkmimxrt1170|edgefast_bluetooth_examples|peripheral_ht|cm4*



|

3. Change the filename *a2dp_source_cm4* to *peripheral_ht_cm4* respectively.
4. Open the files *peripheral_ht_cm4.*uvmpw* and *peripheral_ht_cm4. uvoptx, peripheral_ht_cm4.uvprojx*with a text editor, such as Notepad, Notepad++, Sublime, or Visual Studio code.
5. Search and replace *a2dp_source_cm4* with *peripheral_ht _cm4*, and then save the files.

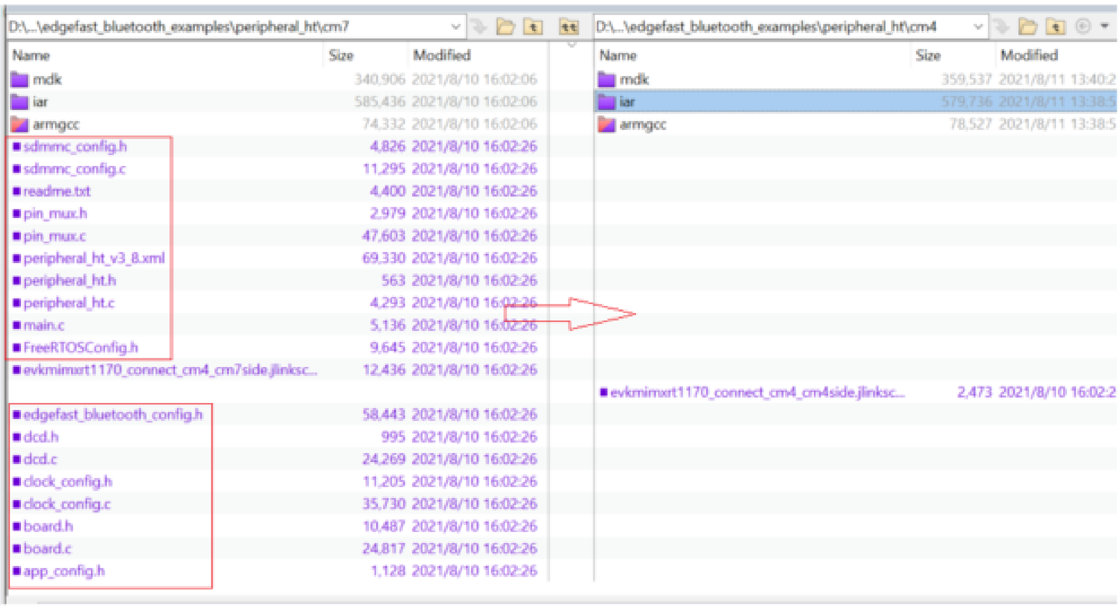


|

Parent topic:*MDK*

Rearrange source files

1. Open folder *cm4* in *<install_dir>*boards|evkmimxrt1170|edgefast_bluetooth_examples|peripheral_ht|cm4*, and delete all files with the .c and .h file name extension.
2. Copy files with the .c and .h filename extension in folder *cm7* with directory *<install_dir> boards|evkmimxrt1170|edgefast_bluetooth_examples|peripheral_ht|cm7*to folder *cm4* with directory *<install_dir> boards|evkmimxrt1170|edgefast_bluetooth_examples|peripheral_ht|cm4*.



|

.

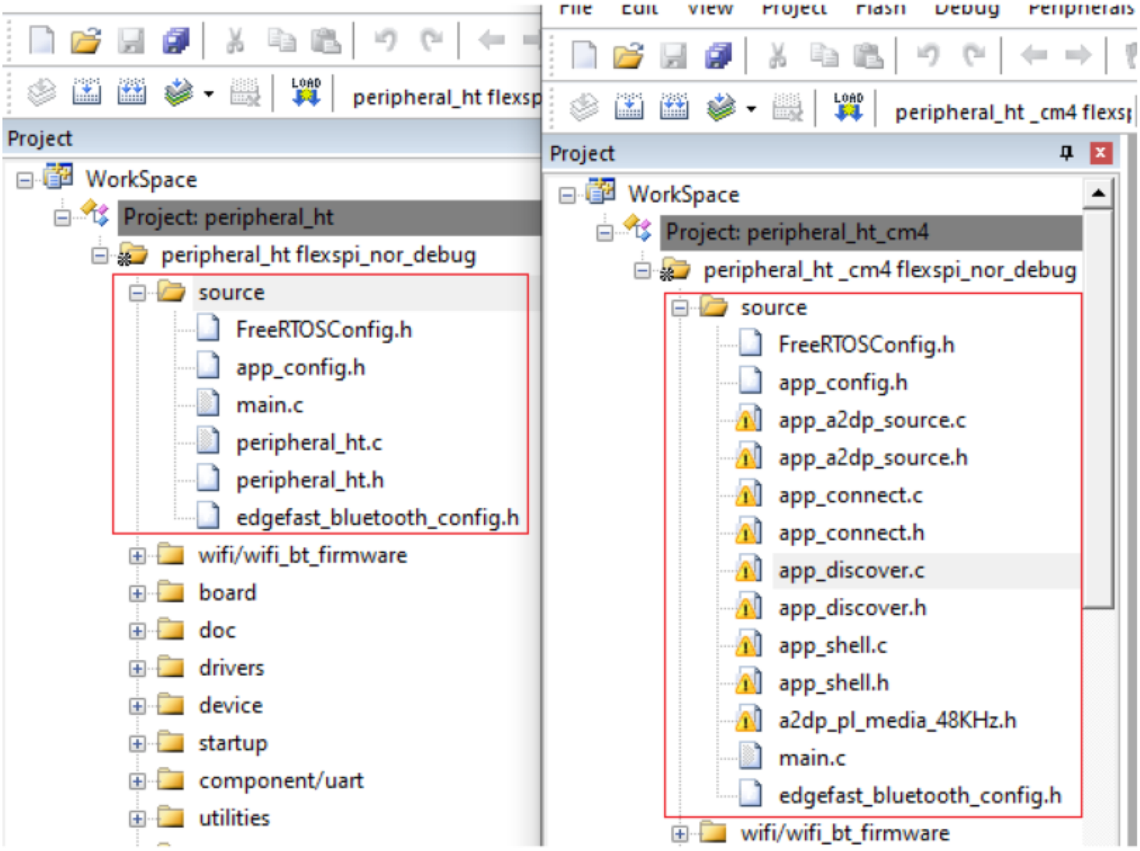
Parent topic:*MDK*

Rearrange project files

1. Open the *peripheral_ht_cm7* and *peripheral_ht_cm4* IAR projects. The two workspaces are located in *<install_dir>**boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm7\mdk* and *<install_dir>**boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4\mdk* respectively.

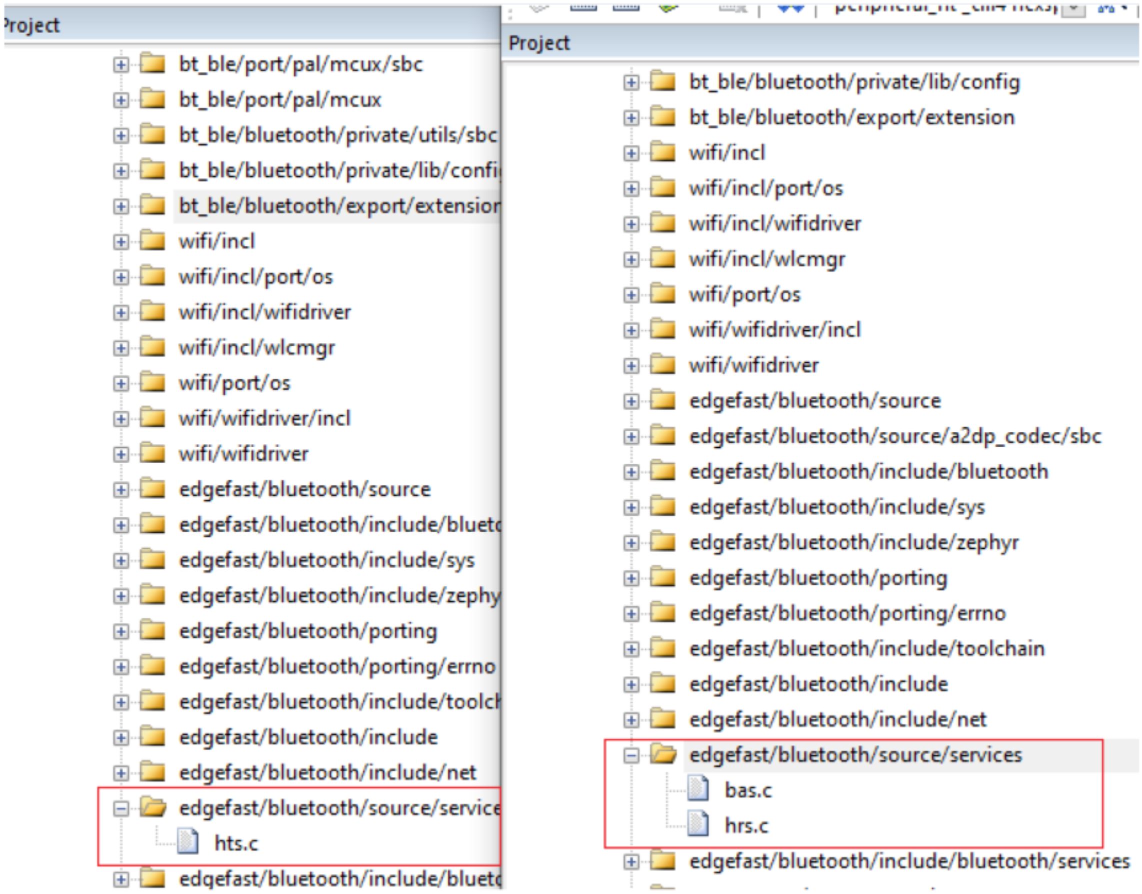
   - Compare the whole project directory, find file groups that the cm7 project has but the cm4 project not and then add these groups into the cm4 project.
   - Compare the difference between the two groups with the same name, remove files that do not exist in the cm7 project but exist in the cm4 project; find files that the cm7 project has but the cm4 project not and then add these files into the cm4 project.

2. For the *source* group, in this case, the files in the source group in the cm4 project must be removed, and the files in the path *<install_dir>\boards\evkmimxrt1170\boards\evkmimxrt1170\edgefast_bluetooth_examples\peripheral_ht\cm4* with the same name as the files in the cm7 project must be added into the *source* group.



3. Compare the **service: group**.

   Peripheral hts profile is located in "**service**" folder. Add the hts.c file to the services group of the cm4 folder.



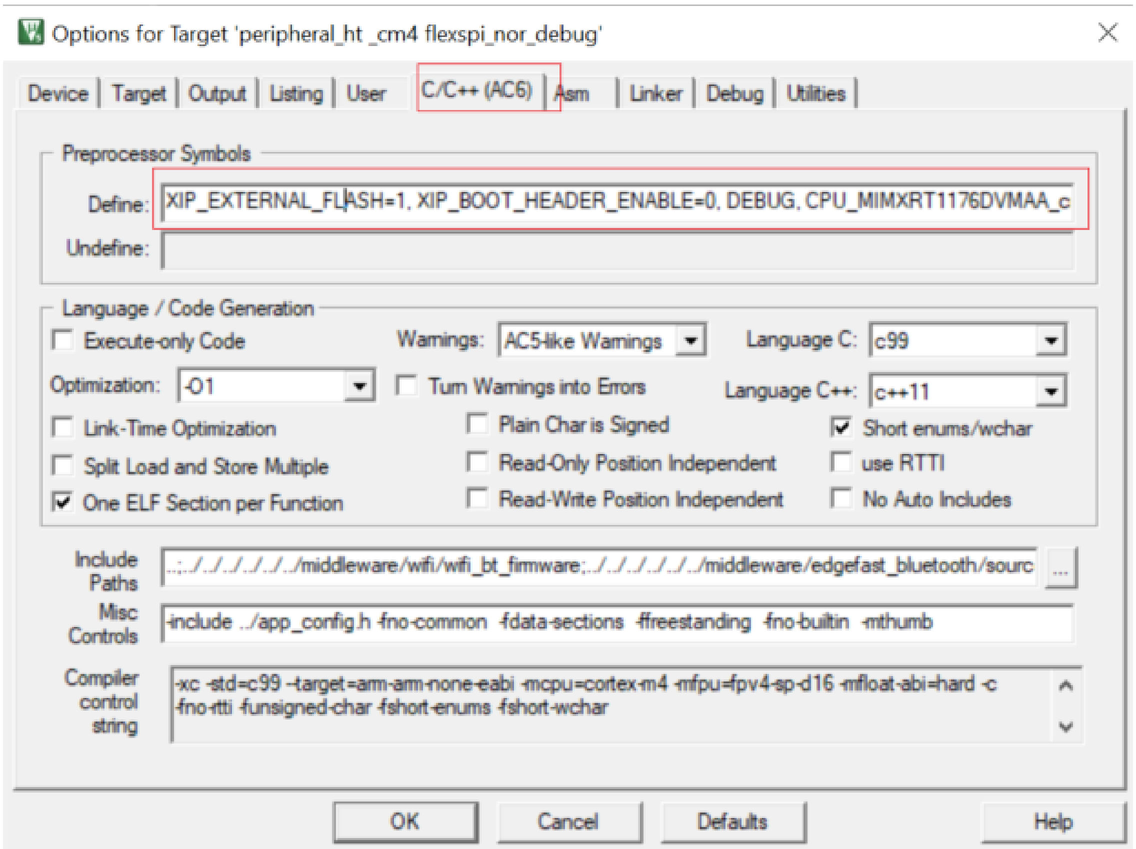
.

Parent topic:[*MDK*]

Adjust project settings

1. Compare the macro in the project settings: *preprocessor symbols*.
2. Compare the macro that does exist in the cm4 project but exists in the cm7 project.
3. Delete the following macro. The rule is that m7 macro setting should be same as m4 .

   The macro could also be found in be eripheral_ht_cm4.uvprojx.



|

```
|![](../images/adjust_mdk_settings2.png "Compare the macro")
```

|

Parent topic:*MDK*

Delete function Remove function *SCB_DisableDCache();* in *main.c*.

On successful completion of the above steps, the M7 project is changed to the M4 project. You can now download and debug the M4 example project.

Parent topic:*MDK*

Note The above steps are based on the **a2dp_source** example and help enable the **peripheral_ht** example on the **m4** core. You can use the same steps for other examples and migrate them from an **m7** project to an **m4** project.

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 ChangeLog

FreeRTOS kernel Readme

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

2.1.4 corehttp

C language HTTP client library designed for embedded platforms.

2.1.5 corejson

JSON parser.

Readme

2.1.6 coremqtt

MQTT publish/subscribe messaging library.

2.1.7 corepkcs11

PKCS #11 key management library.

Readme

2.1.8 freertos-plus-tcp

Open source RTOS FreeRTOS Plus TCP.

Readme