MCSDKERPCGSUG

eRPC Getting Started User Guide

Rev. 12 — 17 June 2024

User guide

Document information

Information	Content
Keywords	eRPC, Getting Started, Remote Procedure Calls, RPC, Embedded, Multicore
Abstract	This Getting Started User Guide document lists the steps to use Remote Procedure Calls (RPC) in embedded multicore microcontrollers (eRPC).

1 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.

2 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:**
   a. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
   b. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
   c. Add infrastructure files.
   d. Add user code for client and server applications.
   e. 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.

3 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.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

2 / 27

Table 1. File locations

Multicore application source and project files	*<MCUXpressoSDK_install_dir>/boards/evkmimxrt1170/multicore_examples/ erpc_matrix_multiply_rpmsg/*
Multiprocessor application source and project files	*<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_ examples/erpc_client_matrix_multiply_<transport_layer>/* *<MCUXpressoSDK_install_dir>/boards/<board_name>/multiprocessor_ examples/erpc_server_matrix_multiply_<transport_layer>/*
eRPC source files	*<MCUXpressoSDK_install_dir>/middleware/multicore/erpc/*
RPMsg-Lite source files	*<MCUXpressoSDK_install_dir>/middleware/multicore/rpmsg_lite/*

3.1 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.

3.2 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/service/erpc_matrix_multiply.erpc

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*.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

3 / 27

- 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.

3.3 Using the eRPC generator tool

Table 2. eRPC generator application file locations

Windows OS	<MCUXpressoSDK_install_dir>/middleware/multicore/tools/erpcgen/Windows
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_multiply.erpc

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`

MCSDKERPCGSUG

User guide

All information provided in this document is subject to legal disclaimers.

Rev. 12 — 17 June 2024

© 2024 NXP B.V. All rights reserved.

4 / 27

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

For Linux OS users:

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

3.4 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:**
   a. Create two projects, where one project is for the client side (primary core) and the other project is for the server side (secondary core).
   b. Add generated files for the client application to the client project, and add generated files for the server application to the server project.
   c. Add infrastructure files.
   d. Add user code for client and server applications.
   e. 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.

3.4.1 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.

3.4.1.1 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>/*

MCSDKERPCGSUG
All information provided in this document is subject to legal disclaimers.
© 2024 NXP B.V. All rights reserved.

User guide
Rev. 12 — 17 June 2024

5 / 27
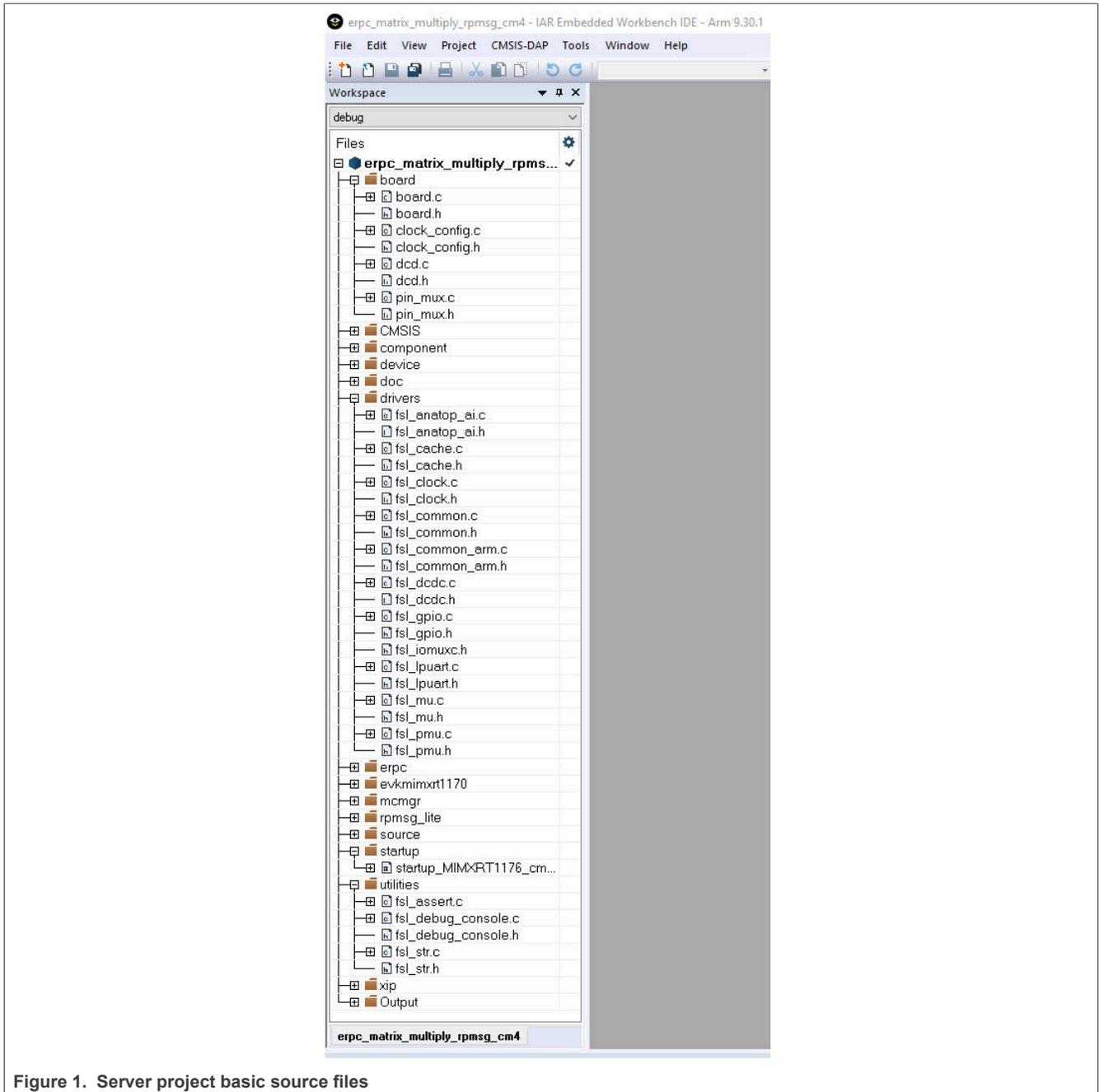
Figure 1. Server project basic source files

3.4.1.2 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

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

6 / 27

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/service/
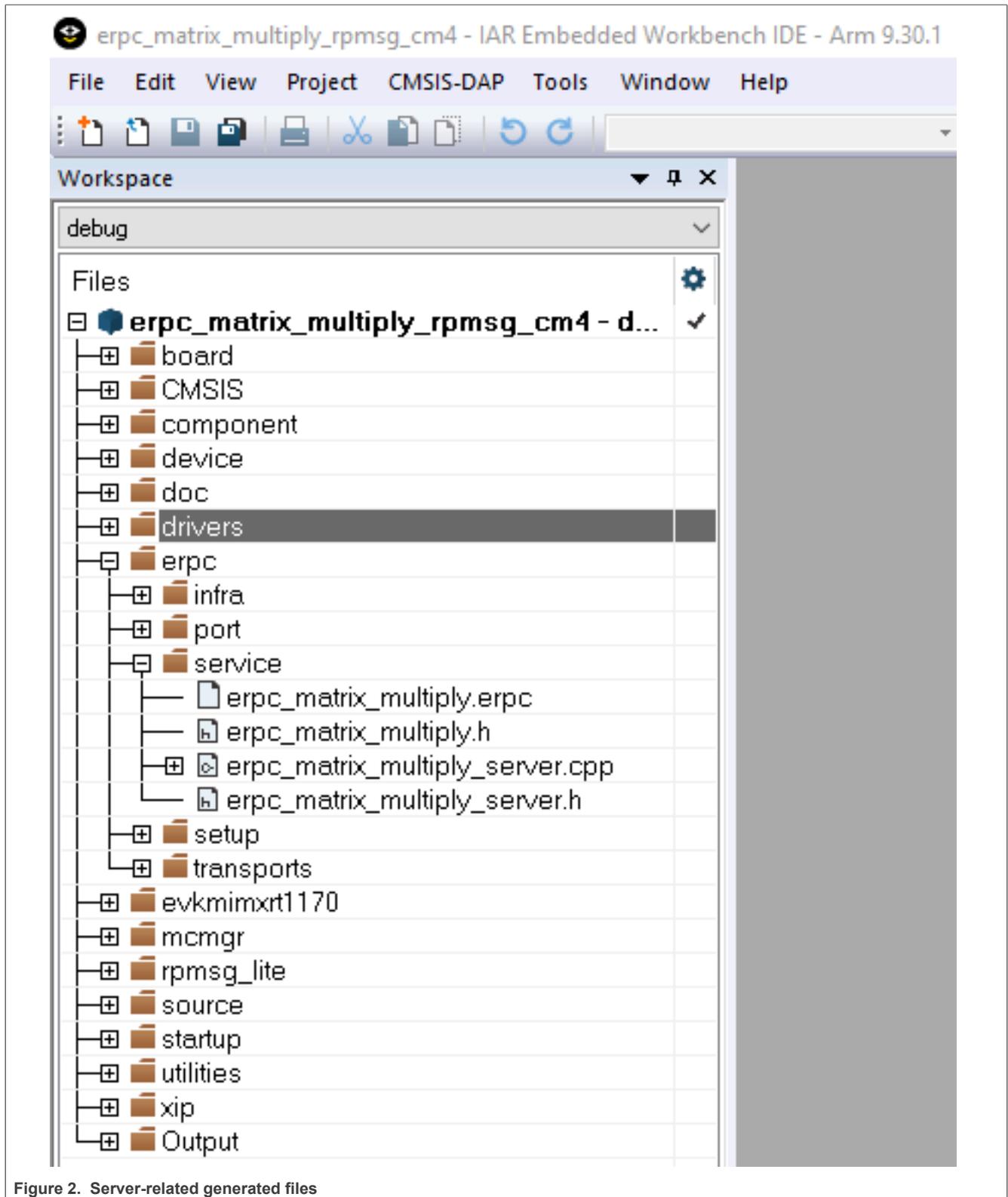
Figure 2. Server-related generated files

3.4.1.3 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.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

9 / 27
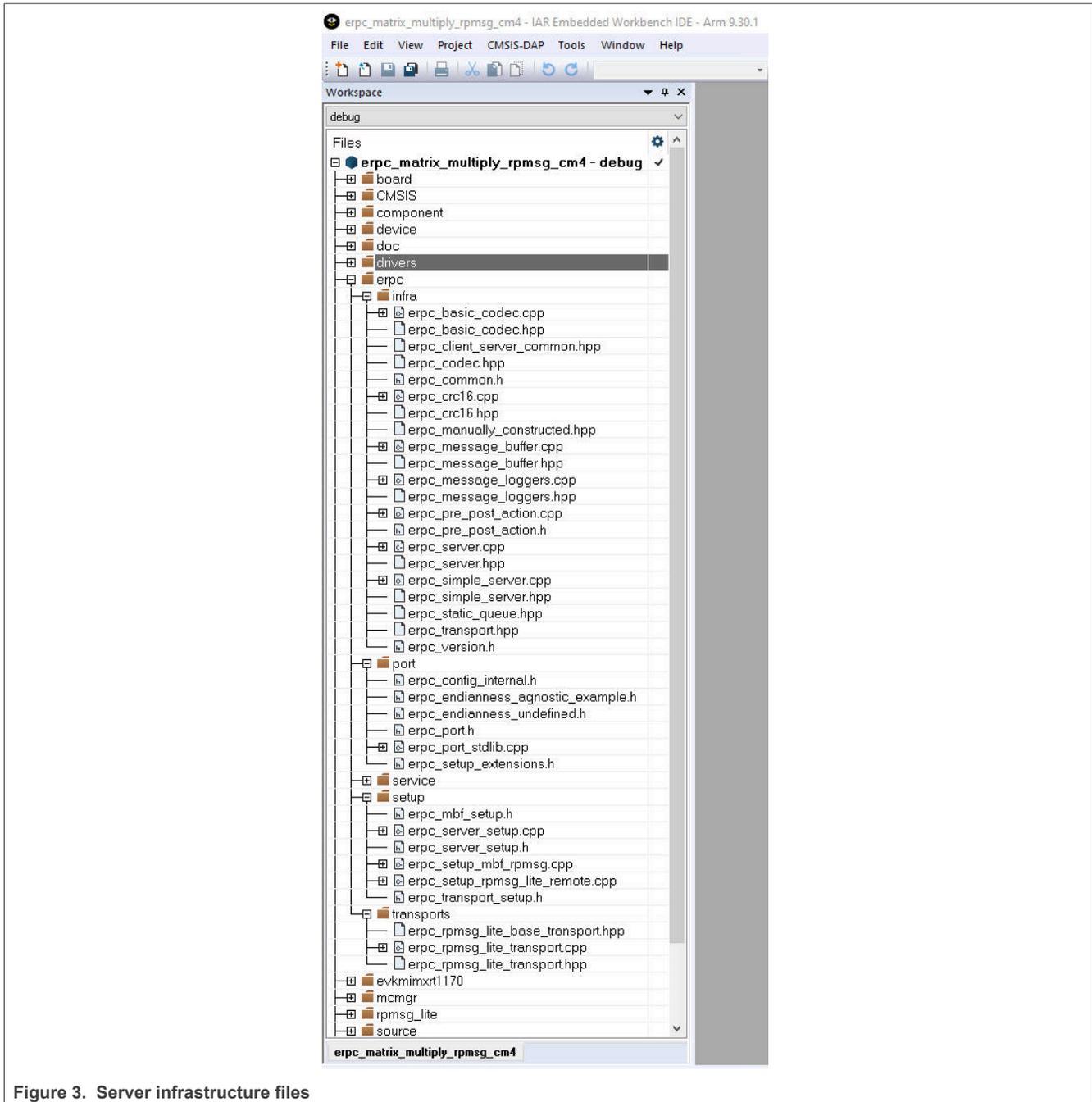
Figure 3. Server infrastructure files

3.4.1.4 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/
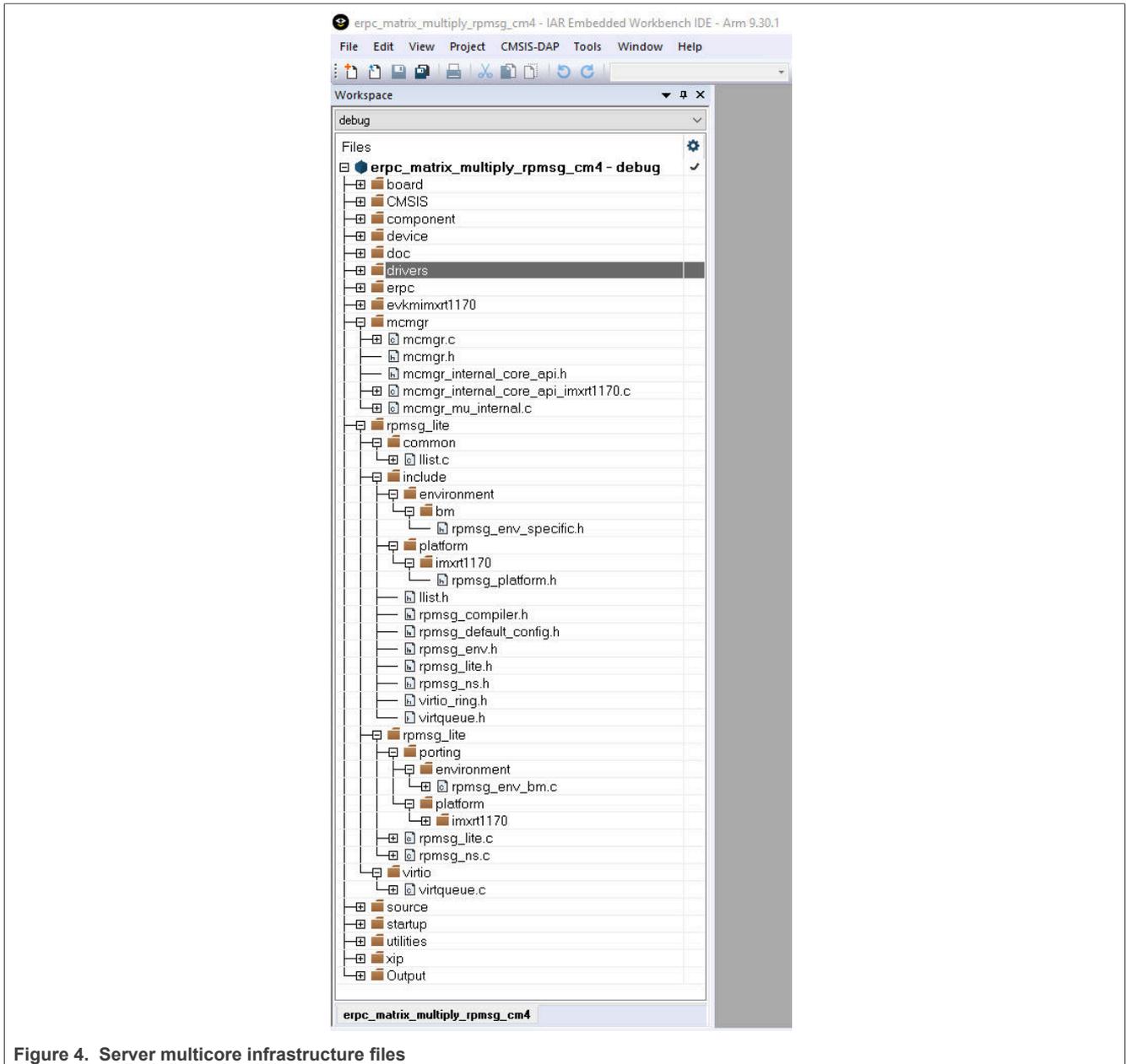
Figure 4. Server multicore infrastructure files

3.4.1.5 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.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

12 / 27
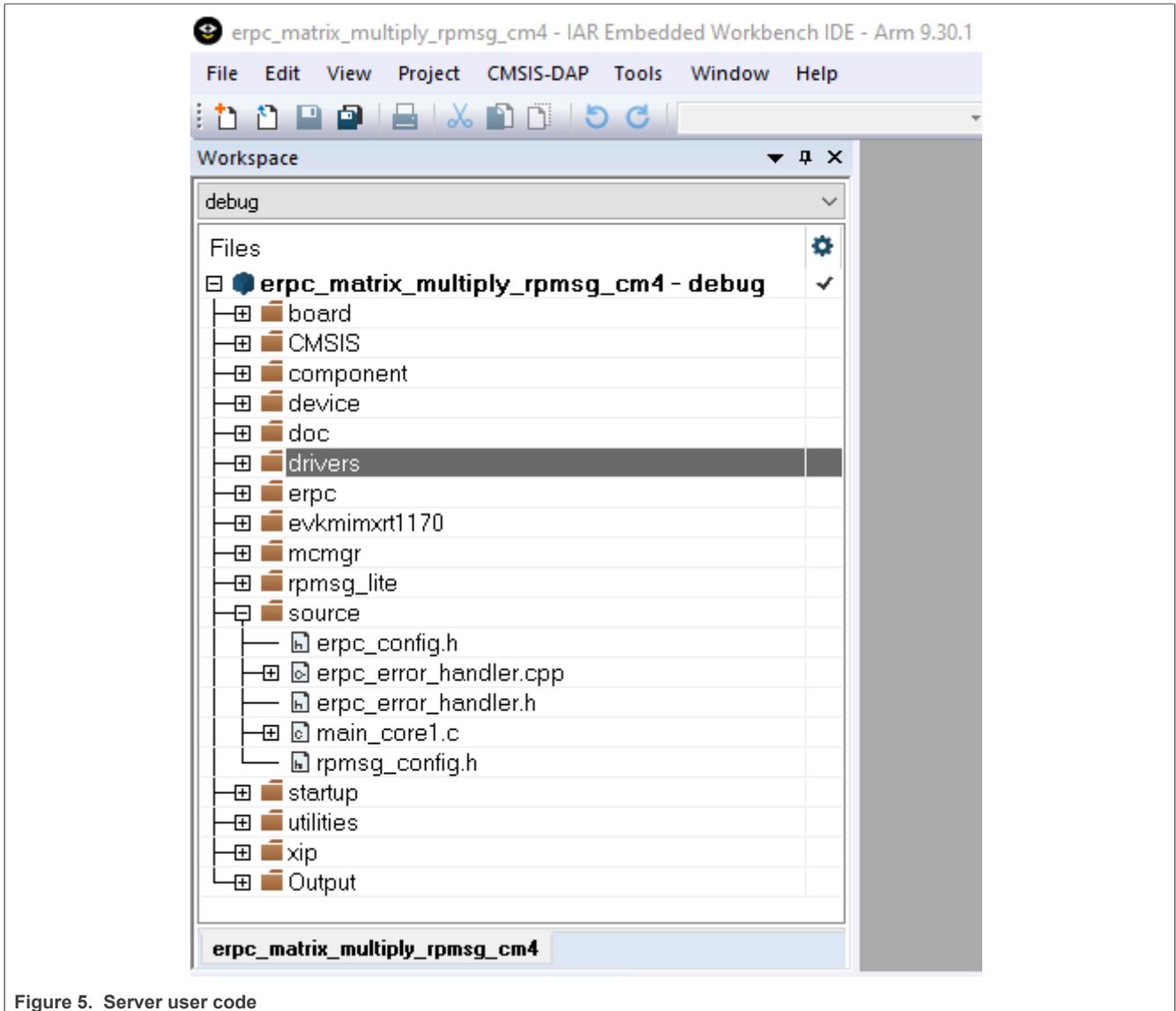
Figure 5. Server user code

3.4.2 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.

3.4.2.1 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>/*

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

13 / 27
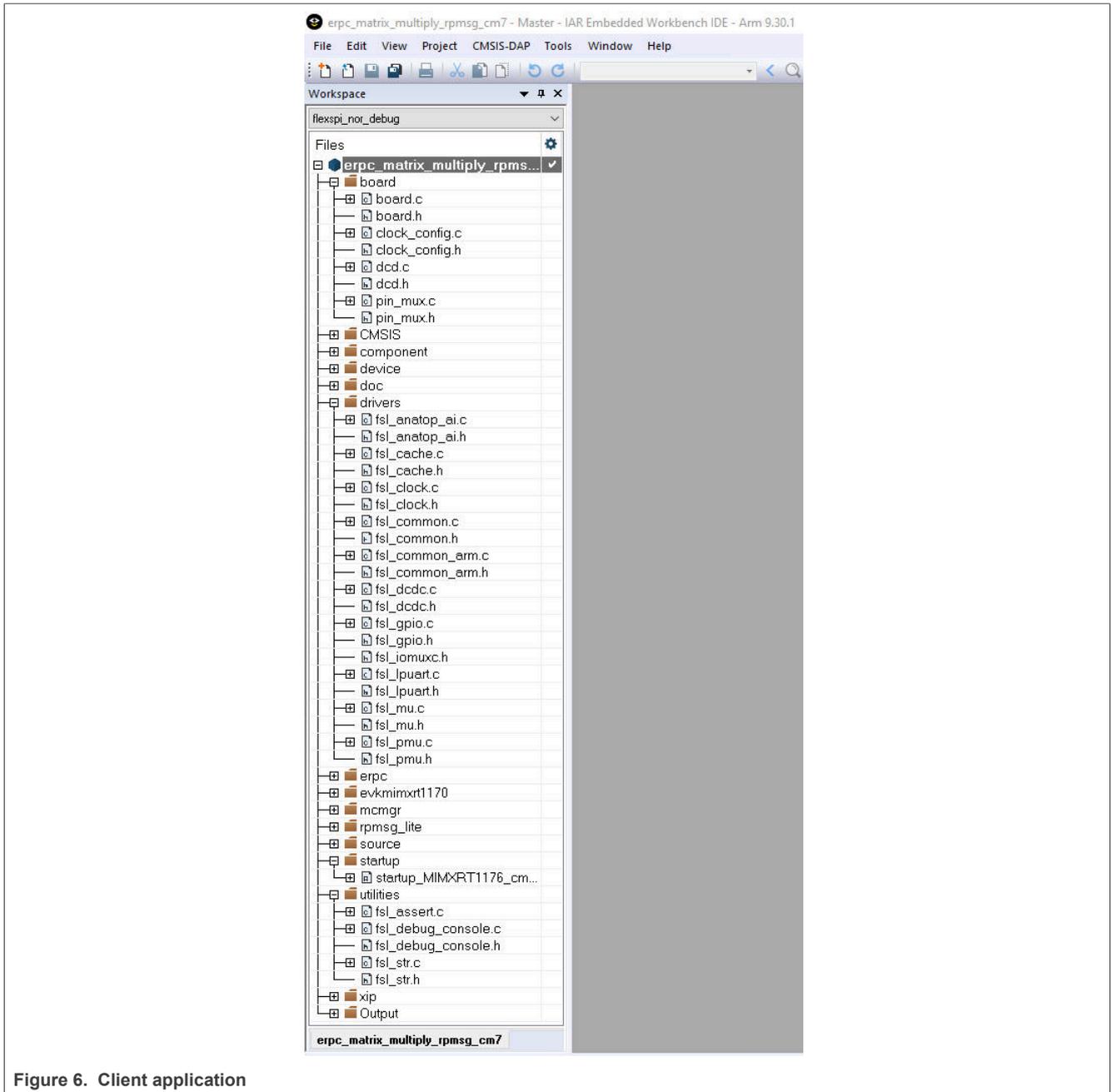
Figure 6. Client application

3.4.2.2 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.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

14 / 27
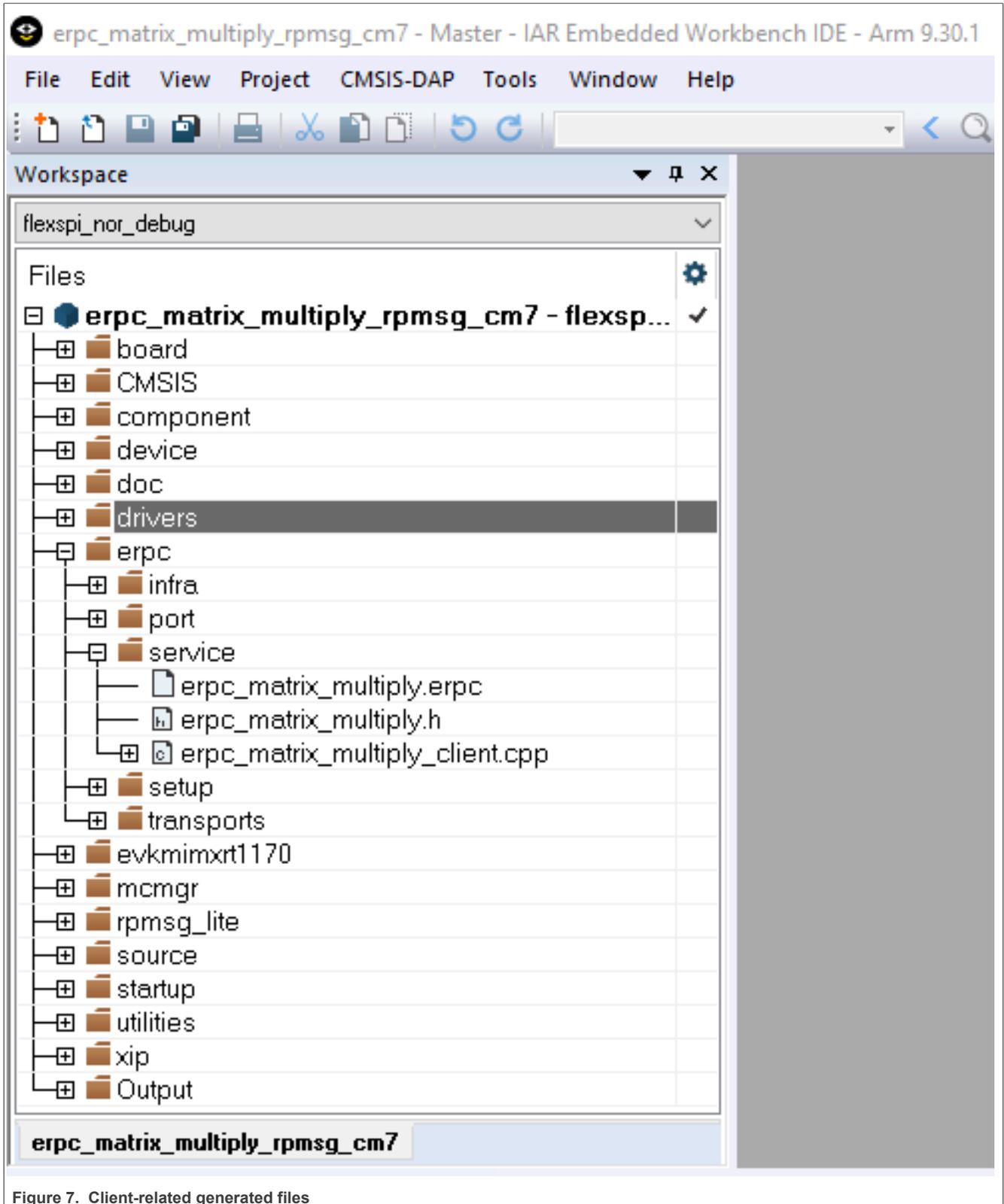
Figure 7. Client-related generated files

3.4.2.3 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.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

16 / 27
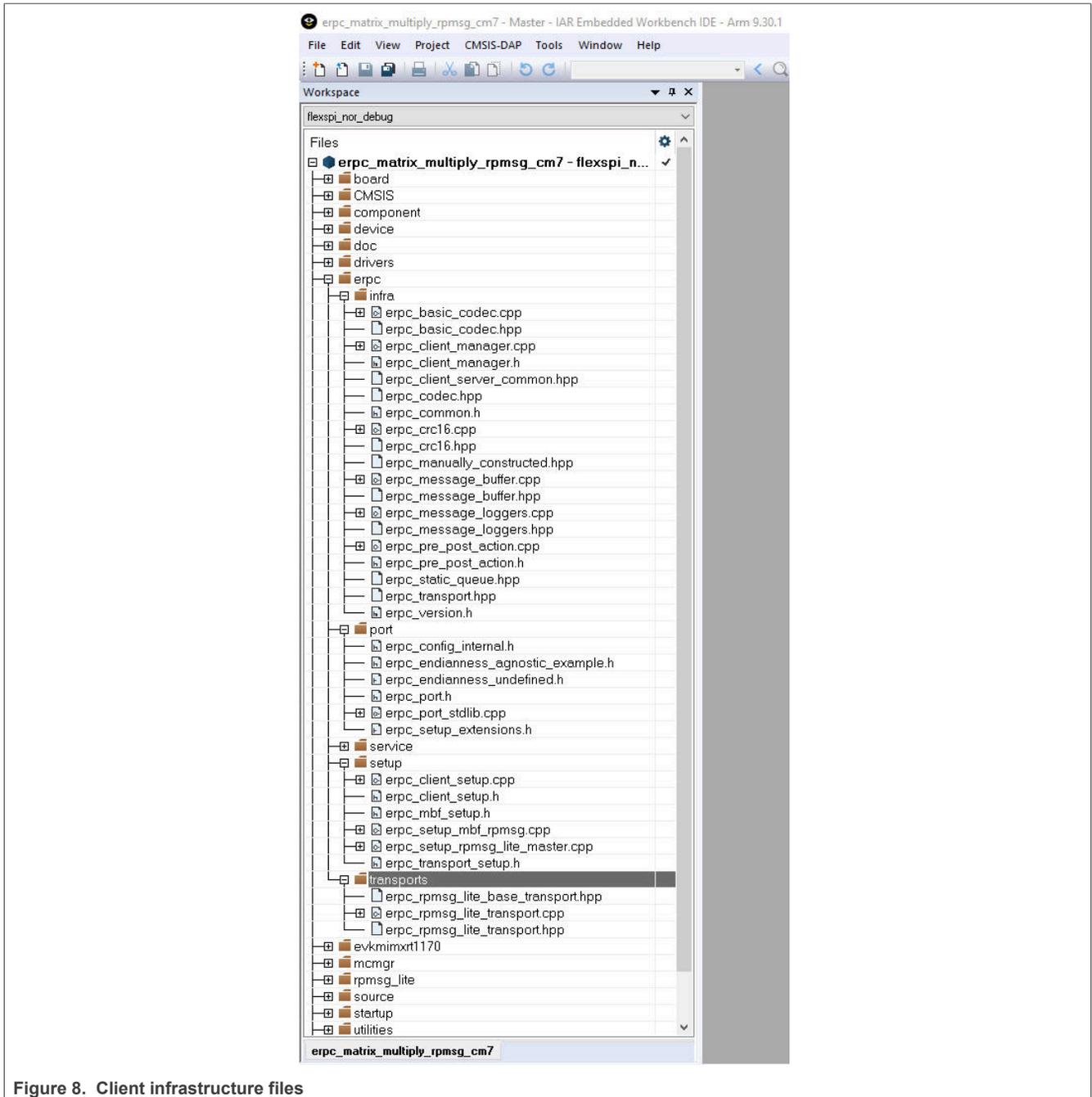
Figure 8. Client infrastructure files

3.4.2.4 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/
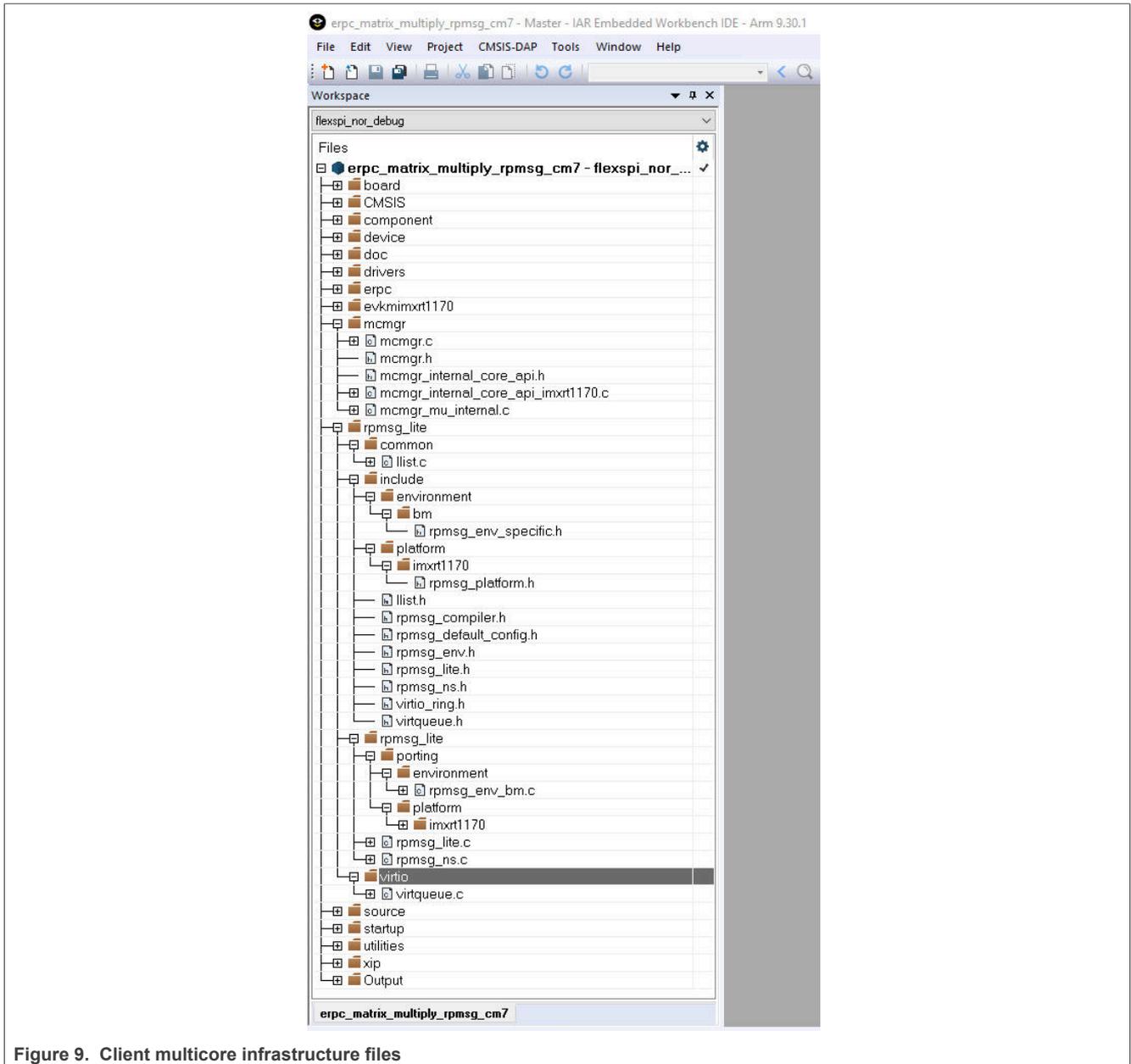
Figure 9. Client multicore infrastructure files

3.4.2.5 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*

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

19 / 27
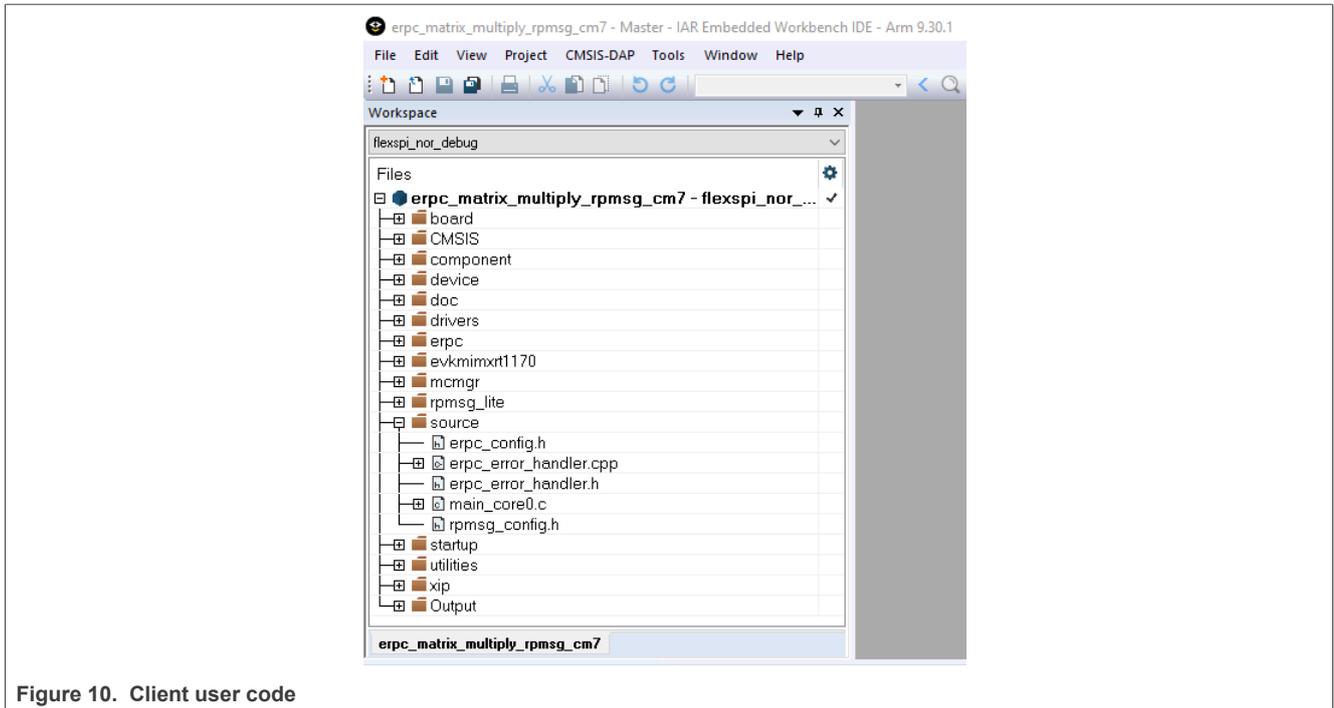
Figure 10. Client user code

3.4.3 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.

Table 3. Transport-related eRPC files for the server side application

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`

3.4.3.1 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.

MCSDKERPCGSUG

User guide

All information provided in this document is subject to legal disclaimers.

Rev. 12 — 17 June 2024

© 2024 NXP B.V. All rights reserved.

20 / 27

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);
      ...
    }
    ...
  }
}
```

3.4.3.2 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.

Table 4. Transport-related eRPC files for the client side application

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

MCSDKERPCGSUG

User guide

All information provided in this document is subject to legal disclaimers.

Rev. 12 — 17 June 2024

© 2024 NXP B.V. All rights reserved.

21 / 27

Table 4. Transport-related eRPC files for the client side application..._continued_

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`

3.4.3.2.1 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;
  }
  ...
}
```

3.4.4 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:

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

22 / 27
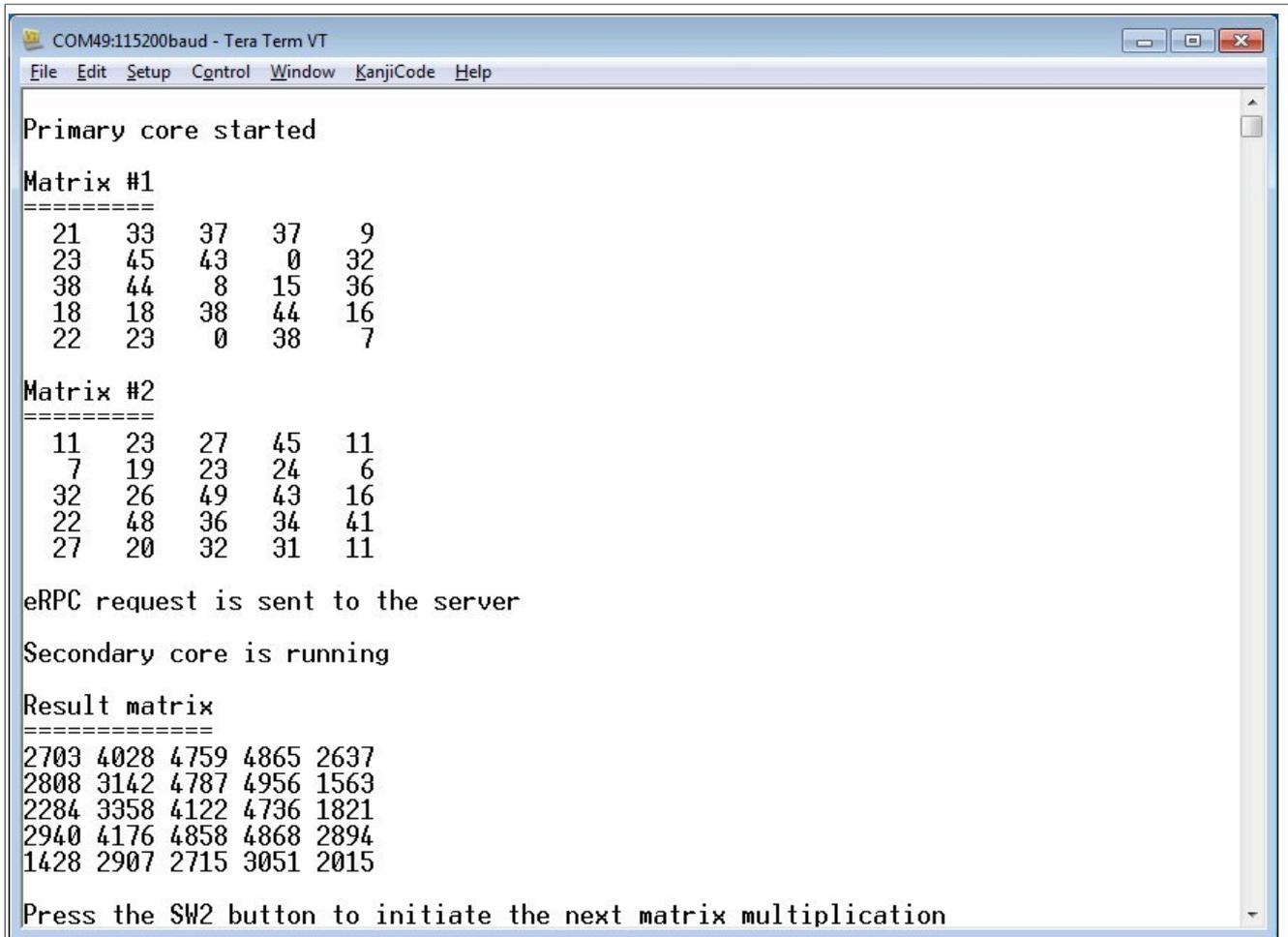
Figure 11. Running the eRPC application

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).

4 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.

5 Note about the source code in the document

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

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

MCSDKERPCGSUG

User guide

All information provided in this document is subject to legal disclaimers.

Rev. 12 — 17 June 2024

© 2024 NXP B.V. All rights reserved.

23 / 27

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

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

6 Revision history

To provide the most up-to-date information, the revisions of our documents on the Internet are the most current. Your printed copy may be an earlier revision.

This revision history table summarizes the changes contained in this document since the last release.

Table 5. Revision history

Revision number	Date	Substantive changes
0	09/2015	Initial release
1	04/2016	Updated to Kinetis SDK v.2.0 and Multicore SDK v.1.1.0
2	09/2016	Updated to Kinetis SDK v.2.0 and Multicore SDK v.2.0.0
3	09/2016	Updated to Multicore SDK v.2.1.0 and eRPC v.1.3.0 Added new sections covering multiprocessor applications
4	03/2017	Updated to Multicore SDK v.2.2.0 and eRPC v.1.4.0
5	11/2017	Updated to Multicore SDK v.2.3.0 and eRPC v.1.5.0 MCUXpresso SDK 2.3.0 release
6	05/2018	Editorial updates for MCUXpresso SDK v2.3.1 and MCUXpresso SDK v.2.4.0
7	12/2019	Editorial and other updates for MCUXpresso SDK v2.7.0
8	01 June 2021	Minor updates for MCUXpresso SDK v2.10.0
9	01 June 2022	Minor updates for MCUXpresso SDK v2.12.0
10	19 December 2022	Editorial and other updates for MCUXpresso SDK v2.13.0
11	27 July 2023	Editorial and other updates for MCUXpresso SDK v2.14.0
12	17 June 2024	Editorial and legal information updates for MCUXpresso SDK v2.16.000

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

MCSDKERPCGSUG

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

User guide

Rev. 12 — 17 June 2024

25 / 27

Contents