eRPC Getting Started User's Guide

1 2

3

4

5

1 Before you begin

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

Additionally, see eRPC documentation located in the following folder: <*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 the 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).

Contents

Befo	re you be	egin	1
Creat	te an eRI	PC application	1
eRPO	C exampl	e	2
3.1	Design	ning the eRPC application	2
3.2	Creatir	ng the IDL file	3
3.3	Using	the eRPC generator tool	3
3.4	Creatir	ng eRPC applications	4
	3.4.1	Multicore server application	5
	3.4.2	Multicore client application	13
	3.4.3	Multiprocessor server application	21
	3.4.4	Multiprocessor client application	22
3.5	Runnir	ng the eRPC application	24
Othe	r uses for	r an eRPC implementation	25
Revi	sion histo	orv	



- 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 LPCXpresso54114 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 (CM4) runs the eRPC client.
- The secondary core (CM0+) 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/lpcxpresso54114/multicore_examples/ erpc_matrix_multiply_rpmsg/</mcuxpressosdk_install_dir>
Multiprocessor application source and project files	<mcuxpressosdk_install_dir>/boards/<board_name>/multiprocessor_examples/ erpc_client_matrix_multiply_<transport_layer>/</transport_layer></board_name></mcuxpressosdk_install_dir>
	erpc_server_matrix_multiply_ <transport_layer>/</transport_layer>
eRPC source files	<mcuxpressosdk_install_dir>/middleware/multicore/erpc/</mcuxpressosdk_install_dir>
RPMsg-Lite source files	<mcuxpressosdk_install_dir>/middleware/multicore/rpmsg_lite/</mcuxpressosdk_install_dir>

Table 1. File locations

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/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/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*.
- 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. followed by new data types and constants declarations.
- 3. followed by 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</mcuxpressosdk_install_dir>
Linux [®] OS	<mcuxpressosdk_install_dir>/middleware/multicore/tools/erpcgen/Linux_x64 <mcuxpressosdk_install_dir>/middleware/multicore/tools/erpcgen/Linux_x86</mcuxpressosdk_install_dir></mcuxpressosdk_install_dir>
Mac [®] OS	<mcuxpressosdk_install_dir>/middleware/multicore/tools/erpcgen/Mac</mcuxpressosdk_install_dir>

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/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/service ../../../boards/ lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/service/erpc_matrix_multiply.erpc

In both cases, the following four files are generated into the *ACUXpressoSDK_install_dir>/boards/lpcxpresso54114/ multicore_examples/erpc_matrix_multiply_rpmsg/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 *ACUXpressoSDK_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 Creating eRPC applications

This section does not show how to create a dual-core application from scratch. Instead, it discusses individual source file groups that form the eRPC applications. You can use the dual-core examples provided within the Multicore SDK (MCSDK) package as a starting point (and reference) for cloning these source files to individual user projects.

For more information about building, running, and debugging multicore example applications in different supported toolchains, see the *Getting Started with MCUXpresso SDK* and/or *Getting Started with MCUXpresso SDK for XXX Derivatives* documents located in the *<MCUXpressoSDK_install_dir>/docs/* folder.

Multiprocessor setup of the eRPC application is discussed in this documentation as well. The behavior of this application is the same as in the multicore case, with the exception that the eRPC transport layer needs to be set up correctly in the application.

3.4.1 Multicore server application

The "Matrix multiply" eRPC server project is located in the following folder:

<MCUXpressoSDK_install_dir>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/cm0plus/iar/

The project files for the eRPC server have the _cm0plus 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>/

Image:	NP <u>T</u> ools <u>W</u> indow <u>H</u> elp
orkspace × Debug	· · · · · · · · · · · · · · · · · · ·
Debug Files Expc_matrix_multiply_cm0plus - D board board.c board.h clock_config.c clock_config.n clock.c fsl_clock.h clock.c fsl_clock.h clock.filex.comm.n clock.filex.com.n fsl_flex.comm.h clock.is fsl_flex.com.h fsl_sopret.c fsl	× ×
Files # ## erpc_matrix_multiply_cm0plus - D / board / board / board / condent / condent / board / condent / con	
erpc_matrix_multiply_cm0plus - D board board.c board.h	<u>₹</u> Bi
Image: Bool of Control o	Iplus - D 🗸
Image: Hermitian Structure Image: Hermitian Structure Image: Hermitian Structure Image: Hermitian	
 board h board h clock_config.c clock_config.h pin_mux.c pin_mux.h doc divers dis_clock.c fsl_clock.h fsl_common.c fsl_flexcomm.c fsl_flexcomm.h fsl_gio.c fsl_gio.h fsl_power.c fsl_power.h fsl_reseth fsl_reseth fsl_sest.h fsl_usart.c fsl_usart.c fsl_sest.h fsl_sest.h fsl_sest.h fsl_sest.h fsl_sest.h fsl_system_LPC54114_cm0plus.s fsl_debug_console.c fsl_debug_console.c fsl_debug_console.h Output 	
Here Clock_config.c Here D pin_mux.c Din_mux.h Here D pin_mux.h Here D fin_mux.h Here D fin_mux.h Here D fin_mux.h Here D fin_mux.h Here D fisl_clock.c Here D fisl_clock.h Here D fisl_clock.h Here D fisl_flexcomm.c Here D fisl_gipio.c Here D fisl_gipio.h Here D fisl_gipio.h Here D fisl_power.c Here D fisl_power.h Here D fisl_reset.h Here D fisl_reset.h Here D fisl_usart.c Here D fisl_usart.c Here D fisl_usart.c Here D repres	
<pre>b clock_config.h b pin_mux.c b pin_mux.c b pin_mux.h b clock.c b fsl_clock.c b fsl_clock.c b fsl_common.c b fsl_common.c b fsl_flexcomm.c b fsl_flexcomm.h b fsl_flexcomm.h b fsl_gpio.c b fsl_gpio.c b fsl_gpio.h b fsl_power.h b fsl_power.h b fsl_power.h b fsl_power.h b fsl_reset.c b fsl_usart.c b fsl_usart.c b fsl_usart.c b fsl_usart.c b fsl_usart.c c b fsl_usart.c c c b fsl_usart.c c c c c c c c c c c c c c c c c c c c</pre>	
Image: Second	
Image: system_LPC54114_cm0plus.s	
Image: Contract of the second seco	
Image: Service Image:	
Here: tsl_clock.c Image: tsl_common.c Image: tsl_common.h Image: tsl_common.h Image: tsl_flexcomm.h Image: tsl_flexcomm.h Image: tsl_spio.h	
Image: Service Image:	
Image: Service Image:	
Image: Signature	
Herein Status Herein Status <td< td=""><td></td></td<>	
Image: Second Secon	
Image: Sill gold.c Image: Sill gold.console.c Image: Sill go	
Isi_gpi0.n Isi_gpi0.n Isi_gpi0.n Isi_gpi0.n Isi_group c.n Isi_gpi0.n Isi_group c.c Isi_power.c Isi_power.h Isi_reset.c Isi_reset.n Isi_susart.c Isi_susart.c Isi_susart.n Ifsi_usart.n Ifsi_secc Isi_secce Isi_secce Isi_secce Isi_secce Isi_secce Isi_startup Ite Isi_startup_LPC54114_cm0plus.s Isi_system_LPC54114_cm0plus.c Isi_system_LPC54114_cm0plus.c Isi_system_LPC54114_cm0plus.h Isi_debug_console.c Isi_debug_console.c Isi_debug_console.h Output	
Isl_boothin Isl_boothin Isl_mailbox.h Isl_power.c Isl_power.h Isl_power.h Isl_reset.c Isl_streset.h Isl_usart.c Isl_usart.h Isl_usart.h Isl_usart.h Isl_service Isl_service Isl_startup_LPC54114_cm0plus.s Isl_system_LPC54114_cm0plus.c Isl_system_LPC54114_cm0plus.c Isl_system_LPC54114_cm0plus.h Isl_debug_console.c Isl_debug_console.h Output	
Image: Status Image: Status Image: Status	
Image: Startup Image	
Image: Statup	
Image: Statup	
Image: Section Image	
Image: Status	
erpc erpc mcmgr erpc orpmsg_lite service source source source system_LPC54114_cm0plus.s System_LPC54114_cm0plus.c System_LPC54114_cm0plus.h System_LPC54114_cm0plus	
Image: Startup Image:	
rpmsg_lite rpmsg_lite service source startup startup_LPC54114_cm0plus.s b system_LPC54114_cm0plus.c b system_LPC54114_cm0plus.h b ottoput	
Image: Service Service Image: Servic	
Image: Source Image: Startup Image: Startup_LPC54114_cm0plus.s Image: System_LPC54114_cm0plus.c Image: System_LPC54114_cm0plus.h	
Image: Startup Image: Startup Image: Startup_LPC54114_cm0plus.s Image: System_LPC54114_cm0plus.c Image: System_LPC54114_cm0plus.h	
Image: Heat Startup_LPC54114_cm0plus.s Image: Heat Startup_LPC54114_cm0plus.c Image: Heat Startup_LPC54114_cm0plus.h	
Image: Head System_LPC54114_cm0plus.c Image: Head System_LPC54114_cm0plus.h Image: Head System0plus.h Image: Hea	olus.s
└── N system_LPC54114_cm0plus.h ├──	plus.c
├──	plus.h
│	
└── ┣] fsl_debug_console.h └─⊞ 🗀 Output	
erpc_matrix_multiply_cm0plus	

Figure 1. Server project basic source files eRPC Getting Started User's Guide, Rev. 6, 05/2018

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 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/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/service/

🔀 erpc_matrix_multiply_cm0plus - IAR Embedded W	orkbench IDE - ARM 7.60.1
<u>File Edit View Project CMSIS-DAP Tools V</u>	<u>V</u> indow <u>H</u> elp
D 🛥 🖬 🖪 🎒 X 🖻 💼 🗠 🖂	- 🗸 🏷 🎠 🔁 🔤 🗸
Workspace	× • X
Debug	▼
Files	2: B
□ □ erpc_matrix_multiply_cm0plus - D	✓
📕 🛏 🗀 board	
III ⊨⊞ 🗀 doc	
📗 🛏 🗀 drivers	
📗 🛏 🗀 erpc	
📗 🛏 🗀 mcmgr	
III ⊨⊞ 🗀 rpmsg_lite	
H H I service	
erpc_matrix_multiply.erpc	
I - M erpc_matrix_multiply.h	
III - E C erpc_matrix_multiply_server.cpp	
erpc_matrix_multiply_server.h	
III III startup	
erpc_matrix_multiply_cm0plus	
Ready	.#

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 C/C++. These files are distributed into subfolders.

- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
 - Four files, erpc_server.h, erpc_server.cpp, erpc_simple_server.h 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.h, erpc_basic_codec.h, 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.h file is used for common eRPC definitions, typedefs, and enums.
 - The erpc_manually_constructed.h 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.h, erpc_rpmsg_lite_transport.h and erpc_rpmsg_lite_transport.cpp files needs to be added into the server project.

ile <u>E</u> dit <u>V</u> iew <u>P</u> roject <u>C</u> MSIS-DAP <u>T</u> ools <u></u>	Mindow H	Help			
· · · · · · · · · · · · · · · · · · ·			• 1	14	M 🔄 I
/orkspace	×	¢			• 3
Debug	-				
Files	20 BB	1			
Parpo metrix multiply omfalue - D					
[] erpc_maux_multiply_cmoplus - D [] board					
- 🖯 🗀 erpc					
🗕 🖵 🗀 infra					
🕀 🖻 basic_codec.cpp					
E erpc_common.h					
🛛 🛏 🔂 erpc_version.h					
h 🖂 manually_constructed.h					
🛛 🖃 📴 message_buffer.cpp					
📔 🛏 🔝 message_buffer.h					
🛛 🗠 🖻 server.cpp					
server.h					
□ □ □ = simple_server.cpp					
Simple_server.h					
Static_queue.h					
h transport.h					
erro north					
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □					
Carpo mbf satur h					
enc server setun h					
Herpo_setup mbf romsg zc.cop					
H → ⊕ erpc setup romsa lite zc rem					
erpc transport setup.h					
🗆 🖵 🗀 transports					
Here and the second sec					
🛛 🖵 🔚 rpmsg_lite_zc_transport.h					
—⊞ 🗀 rpmsg_lite					
- 🖽 🗀 service					
Here Source					
He istartup					
utilities					
		1			

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/

Repc_matrix_multiply_cm0plus - IAR Embedded Wo	orkbench IDE - ARM 7.60.1
File Edit View Project CMSIS-DAP Tools W	/indow Help
Workspace	×
Debug	-
	e
	C~ 11
Pierpc_matrix_multiply_cmUplus - D	
III 🖬 mcmgr.h	
- 🚡 mcmgr_internal_core_api.h	
□ □ □ □ □ mcmgr_internal_core_api_lpc5411	
📗 ————————————————————————————————————	
env.h	
IIII ⊢⊞ CI plattorm.c	
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □	
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □	
lite.h	
rpmsg_ns.c	
📗 📔 🛏 🔂 rpmsg_ns.h	
││ │	
III I III citqueue.c	
Virtqueue.h	
stanup	
erpc_matrix_multiply_cm0plus	
Ready	th.

Figure 4. Server multicore infrastructure files

3.4.1.5 Server user code

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

<MCUXpressoSDK_install_dir>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/cm0plus

The main corel.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()
 {
     /* RPMsq-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_init(transport, message_buffer factory);
     /* Adding the service to the server */
     erpc_add_service_to_server(create_MatrixMultiplyService_service());
     . . .
     while (1)
     {
         /* Process eRPC requests */
         erpc status t status = erpc server poll();
        /* 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/lpcxpresso54114/multicore_examples/* erpc_matrix_multiply_rpmsg folder.

💘 erpc_matrix_multiply_cm0plus - IAR Embedded Workbench II	DE - ARM 7.80.2 🗖 🔍 🕅
<u>File Edit View Project CMSIS-DAP Tools Window H</u>	<u>l</u> elp
- D 😅 🖬 🕼 🕹 👗 🖻 💼 🗠 억	- 🗸 🏷 🎠 🔁 🖻
Workspace ×	- × ×
Debug	
Files 🕅 🕅	
Image: startup Image: startup	
Ready	

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/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/cm4/iar/

Project files for the eRPC client have the _cm4 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>/

🔀 erpc_matrix_multiply_cm4 - Master - IAR Embedded Workbench IDE - ARM 7.60.1							
<u>File Edit View Project CMSIS-DAP Tools</u>	Wind	ow <u>H</u>	<u>l</u> elp				
D 📽 🖬 🗿 🎒 🖁 🔒 🖌 🖬 🕄 🗠 🖓				• •	18	h h 🗄 🗔 🗉	
Workspace		×				¥	
Debug		-					
Files	e~	25					
Tiles	(~~	10					
B perpc_manx_multiply_cm4 - Debug	¥						
board.h							
L ⊢⊕ C clock config.c							
Clock config.h							
⊢⊞ [c] pin mux.c							
i i i jin_mux.h							
🗕 🖵 🗀 drivers							
-⊞ C fsl_clock.c							
H H I I I I I I I I I I I I I I I I I I							
sl_common.h							
I I I I I I I I I I I I I I I I I I I							
III I III III III III III III III IIII IIII							
I I I I I I I I I I I I I I I I I I I							
📗 📙 🔚 fsl_gpio.h							
III - III fsl_iocon.h							
📘 ⊨ 🔂 fsl_mailbox.h							
I I I I I I I I I I I I I I I I I I I							
III ├── ┣ fsl_power.h							
I I I I I I I I I I I I I I I I I I I							
I I I I I I I I I I I I I I I I I I I							
fsl_usart.c							
fsl_usart.h							
Figure Startup							
E Henstanup_LPC54114_cm4.s							
System_LPC34114_Cm4.C							
L HT n fsl debug console c							
fsl debug console h							
erpc_matrix_multiply_cm4							
Ready							

Figure 6. Client application eRPC Getting Started User's Guide, Rev. 6, 05/2018

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 *ACUXpressoSDK_install_dir>/boards/ lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg/service/* folder.

😵 erpc_matrix_multiply_cm4 - Master - IAR Embedd	ed W	/orkbe	ench IDI	E - AR	м 7.6	50.1					Σ	
<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>P</u> roject <u>C</u> MSIS-DAP <u>T</u> ools <u>1</u>	<u>W</u> ind	ow <u>H</u>	<u>H</u> elp									
🗅 😅 🖬 🕼 🐇 🝓 🖬 😭 🖬 🕤					-	1	8	1	1	4		-
Workspace		×	-								•	
Debug]										ĺ
Files	20	₿.										
🗆 🗇 erpc_matrix_multiply_cm4 - Debug												
🕀 🗀 board												
Here in the second seco												
🗕 🕀 🧰 drivers												
- 🕀 🗀 erpc												
- 🖽 🚞 mcmgr												
–⊕ 🗀 rpmsg_lite												
📙 🛏 🗋 erpc_matrix_multiply.erpc												
📙 🛏 🔝 erpc_matrix_multiply.h												
📙 🖵 🔂 erpc_matrix_multiply_client.cpp												
- 🕀 🗀 source												
🛏 🗀 startup												
⊨ 🕀 🧰 utilities												
🖵 🖽 🧰 Output												
erpc_matrix_multiply_cm4												
leady												I

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 C/C++. 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.h, erpc_basic_codec.h, 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.h 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.
- 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.
 - erpc_client_setup.h and erpc_client_setup.cpp files needs to be aded 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.h, erpc_rpmsg_lite_transport.h, and erpc_rpmsg_lite_transport.cpp files needs to be added into the client project.

erpc_matrix_multiply_cm4 - Master - IAR Embed	Ided Worl	bench IDE	- ARM 7.80.2		×
ile <u>E</u> dit <u>V</u> iew <u>P</u> roject <u>C</u> MSIS-DAP <u>T</u> ools	Window	<u>H</u> elp			
🗅 🚅 🖬 🕼 🕼 🕼 🖬 🛍 🗠 여 🗌				1 1 12	
/orkspace	3	c			• 3
Debug	-				
Filos	2~ B2	i l			
Res matrix multiply cm4-Dob					
- perpc_mainx_monopry_cm4 - Deb	·				
Here in the second seco					
📔 ⊨ 🔂 basic_codec.h					
- 🖽 📴 client_manager.cpp					
🔝 client_manager.h					
Codec.h					
Figure 1					
I I I I I I I I I I I I I I I I I I I					
🛛 🕂 🕞 erpc_config_internal.h					
🖻 erpc_port.h					
│ │ └─⊞ 🔂 erpc_port_stdlib.cpp					
erpc_client_setup.cpp					
Client_setup.h					
├── IN erpc_mbt_setup.h					
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □					
I Hand the erpc_setup_rpmsg_itte_zc_m					
Express lite zc transport.cpp					
⊢⊕ 🗀 rpmsg_lite					
- 🕀 🗀 source					
He Cartup					
erpc_matrix_multiply_cm4					

Figure 8. Client infrastructure files eRPC Getting Started User's Guide, Rev. 6, 05/2018

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/

Repc_matrix_multiply_cm4 - Master - IAR Embedd	led W	/orkbe	ench IDE - ARM 7.60.1
<u>File Edit View Project CMSIS-DAP Tools</u>	Wind	ow .	<u>H</u> elp
			- < > > > > > > > > > >
Workspace		×	× ×
Debug		-	1
	@	D63	1
	<u>~</u> ~	10	
E erpc_matrix_multiply_cm4 - Debug	×		
H H C mcmgr.c			
III I I I I I I I I I I I I I I I I I			
□			
□ □ □ □ mcmgr_internal_core_api_lpc541			
l			
env_bm.c			
porting			
-⊞ 🖸 platform.c			
III I L B platform.h			
n Instri			
pros lite.h			
🔲 🗖 🗇 rpmsg_ns.c			
📗 ⊨ 🕞 rpmsg_ns.h			
III I Find Virtio_ring.h			
Virtqueue.c			
Hand Carling Contractions			
Utput			
erpc_matrix_multiply_cm4			
Peady			
Incouy			

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/lpcxpresso54114/multicore_example/erpc_matrix_multiply_rpmsg/cm4

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\};
 /* RPMsq-Lite transport layer initialization */
 erpc_transport_t transport;
 transport = erpc transport rpmsq 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 init(transport, message buffer factory);
 /* Set default error handler */
 erpc client set error handler(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/lpcxpresso54114/multicore_examples/erpc_matrix_multiply_rpmsg

🔀 erpc_matrix_multiply_cm4 - Master - IAR Embed	ded Work	bench IDE - ARM 7.80.2 📃 💷 🔀
<u>File Edit View Project CMSIS-DAP Tools</u>	<u>W</u> indow	<u>H</u> elp
- C 🖙 🖬 🕼 😹 🕹 🕹 🖬 💼 🗠 으 🗆		
Workspace	×	- ×
Debug	•	
Files	8:: B	
🛛 🗗 erpc_matrix_multiply_cm4 - Deb	✓	
⊨ ⊕ 🔁 board		
Hand International Internation		
→ = ···································		
🛛 📔 🛏 🖍 erpc_config.h		
-⊞ 🖻 main_core0.c		
II h rpmsg_config.h		
H → ⊕ 🔁 startup		
erpc_matrix_multiply_cm4		
Ready		

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>
	<erpc base="" directory="">/erpc_c/transports/erpc_(d)spi_slave_transport.h</erpc>
	<erpc base="" directory="">/erpc_c/transports/ erpc_(d)spi_slave_transport.cpp</erpc>
UART	<erpc base="" directory="">/erpc_c/setup/erpc_setup_uart_cmsis.cpp</erpc>

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

<erpc base<="" th=""><th>directory>/erpc_c/transports/erpc_</th><th>uart_cmsis_transport.h</th></erpc>	directory>/erpc_c/transports/erpc_	uart_cmsis_transport.h
<erpc base<="" th=""><th>directory>/erpc_c/transports/erpc_</th><th>uart_cmsis_transport.cpp</th></erpc>	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 <<u>MCUXpressoSDK_install_dir</u>>/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 init(transport, message buffer factory);
     /* Adding the service to the server */
     erpc add service to server(create MatrixMultiplyService service());
     . . .
     while (1)
     {
         /* Process eRPC requests */
         erpc status t status = erpc server poll();
        /* handle error status */
        if (status != kErpcStatus Success)
        {
            /* print error description */
            erpc_error_handler(status, 0);
            . . .
        }
        . . .
    }
 }
```

3.4.4 Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in the *ACUXpressoSDK_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>
	<erpc base="" directory="">/erpc_c/transports/ erpc_(d)spi_master_transport.h</erpc>
	<erpc base="" directory="">/erpc_c/transports/ erpc_(d)spi_master_transport.cpp</erpc>
UART	<erpc base="" directory="">/erpc_c/setup/erpc_setup_uart_cmsis.cpp</erpc>
	<pre><erpc base="" directory="">/erpc_c/transports/erpc_uart_cmsis_transport.h</erpc></pre>
	<pre><erpc base="" directory="">/erpc_c/transports/erpc_uart_cmsis_transport.cpp</erpc></pre>

3.4.4.1 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 init(transport, message buffer factory);
 /* Set default error handler */
 erpc client set error handler(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;
     }
```

} ...

3.5 Running the eRPC application

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

😕 COM49:115200baud - Tera Term VT				
<u>Eile E</u> dit <u>S</u> etup C <u>o</u> ntrol <u>W</u> indow <u>K</u> anjiCode <u>H</u> elp				
Primary core started				
Matrix #1				
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				
Matrix #2				
$\begin{array}{cccccccccccccccccccccccccccccccccccc$				
eRPC request is sent to the server				
Secondary core is running				
Result matrix ====================================				
2808 3142 4787 4956 1563 2284 3358 4122 4736 1821 2940 4176 4858 4868 2894 1428 2907 2715 3051 2015				
Press the SW2 button to initiate the next matrix multiplication	-			

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 Revision history

To provide the most up-to-date information, the revision 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.

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 the eRPC v.1.3.0
		Added new sections covering multiprocessor applications.
4	03/2017	Updated to Multicore SDK v.2.2.0 and the eRPC v.1.4.0
5	11/2017	Updated to Multicore SDK v.2.3.0 and the eRPC v.1.5.0 MCUXpresso SDK 2.3.0 release
6	05/2018	Editorial updates for MCUXpresso SDK v2.3.1 and v2.4.0

Table 5. Revision history

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Arm Powered, Cortex, Keil, and μ Vision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2018 NXP B.V.

Document Number MCSDKERPCGSUG Revision 6, 05/2018



