
eRPC Getting Started User's Guide

MCSDKERPCGSUG

Rev. 1

03/2016



How to Reach Us:

Home Page:

www.freescale.com

Web Support:

www.freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale 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.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 Freescale 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. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: www.freescale.com/SalesTermsandConditions

Freescale, the Freescale logo, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners. ARM, ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2008-2016 Freescale Semiconductor, Inc.

Document Number: MCSDKERPCGSUG

Rev. 1

03/2016



Non-Disclosure Agreement required

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Before you begin | 4 |
| 1.1 | About this guide | 4 |
| 1.2 | Where to go for more information | 4 |
| 2 | Create an eRPC application | 5 |
| 3 | Step-by-step eRPC example | 6 |
| 3.1 | Designing the eRPC application | 6 |
| 3.2 | Creating the IDL file | 7 |
| 3.3 | Using the eRPC generator tool | 8 |
| 3.4 | Creating eRPC applications | 9 |
| 3.4.1 | Client application | 10 |
| 3.4.2 | Server application | 15 |
| 3.5 | Running the eRPC application | 22 |
| 4 | Generic usage of the eRPC implementation | 22 |
| 5 | Revision history | 24 |

1 Before you begin

1.1 About this guide

This Getting Started Users' Guide is a manual for using Remote Procedure Call (RPC) in embedded (eRPC). This document is written for software developers who are interested in RPC in an embedded world.

1.2 Where to go for more information

For more details, see the *eRPC Reference Manual* (document MCSDKERPCRM) located in the `<ksdk_install_dir>/doc/multicore_<version>/` folder.

2 Create an eRPC application

This chapter describes the generic way how to create a client/server eRPC application. The particular example is discussed in the following chapter.

1. Design the eRPC application – Decide which data types will be sent between applications, and define functions which will 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 written in 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 application.
4. Create the eRPC application:
 - a. Create two projects, where one is for the client side (primary core) and the other 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 projects options.
5. Run the eRPC application – Run both the server and the client applications. Ensure that the server has been run before the client request was sent.

3 Step-by-step eRPC example

This chapter describes the steps required to create the “Matrix multiply” example eRPC application. This example implements just 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.

The example can be run on both the FRDM-KL28T Freedom development platform and the TWR-KL28T72M Tower System module. However, this chapter will only focus on the FRDM-KL28T Freedom development platform. The primary core (core0) runs the eRPC client, and the secondary core (core1) runs the eRPC server. RPMsg (Remote Processor Messaging) is used as the eRPC transport layer.

The application source and project files are located in:

```
<ksdk_install_dir>\boards\frdmkl28t\demo_apps\multicore_examples\erpc_matrix_multiply\
```

The RPMsg related files are located in:

```
<ksdk_install_dir>\middleware\multicore_<version>\open-amp\
```

3.1 Designing the eRPC application

This application has a simple design because it contains only one function. Only three two-dimensional array values are used. Two of them are used as function parameters, while the third one is used as the return value. The IDL file syntax supports arrays with the dimension length set by the number only (current implementation). Because of this, one more variable is declared in the IDL to store information about matrix dimension length, and to allow easy maintenance of the user and server code. For the simple usage of the two-dimensional array, the alias name (new type-definition) for this data type has been declared in the IDL. Declaring this alias name ensures that the same data type can be used across the client and the server applications.

3.2 Creating the IDL file

The created IDL file is located in:

<ksdk_install_dir>\boards\frdmkl28t\multicore_examples\erpc_matrix_multiply\service\erpc_matrix_multiply.erpc

It 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[5][5];

interface MatrixMultiplyService {
    erpcMatrixMultiply(Matrix matrix1, Matrix matrix2) -> Matrix
}
```

The IDL file starts with the program naming. This name is used for naming of all generated outputs. The declaration and definition of the constant variable named *matrix_size* follows. This variable is used for passing information about the length of matrix dimensions to the client/server user code. The alias name for two-dimensional array type named *Matrix* is declared. The interface group named *MatrixMultiplyService* is located at the end of the IDL file. This interface group contains only one function declaration named *erpcMatrixMultiply*. As shown, the function's declaration contains two parameters with data type *Matrix*. Additionally, the returned data type is declared as *Matrix*.

The following steps are best convention to write the IDL file:

- The program name on top of the file.
- New data types and constants declarations.
- Declarations of interfaces and functions at the end.

3.3 Using the eRPC generator tool

The eRPC generator application for Windows® OS is located in:

```
<ksdk_install_dir>\middleware\multicore_<version>\tools\erpcgen\Windows
```

The eRPC generator application for Linux® OS is located in:

```
<ksdk_install_dir>\middleware\multicore_<version>\tools\erpcgen\Linux
```

The files for the “Matrix multiply” example are pregenerated 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 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 this:

```
erpcgen erpc_matrix_multiply.erpc
```

Additionally, another method to create the shim code is:

- The eRPC application can be executed with input commands. The main input commands are:

“-?”/”—help” - Shows supported commands.

“-o <filePath>”/”—output<filePath>” – Sets the output directory.

The command can look like:

```
<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 will look like the following:


```
erpcgen -o
../../../../../../../../boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/service
../../../../../../../../boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/service/erpc_matrix_multiply.erpc
```

In both cases, the following four files are generated into the

<ksdk_install_dir>/boards/frdmkl28t/multicore_examples/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 Linux OS users:

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

3.4 Creating eRPC applications

The eRPC client and server projects for the *IAR Embedded Workbench® for ARM (EWARM)* are covered in this section.

For more information about building, running, and debugging multicore demo applications in EWARM, it is recommended to read *Getting Started with Kinetis MKL28 Dual-Core and IAR Embedded Workbench®* (document KSKDMKL28GSUG) located in the <ksdk_install_dir>/doc/multicore_<version>/ folder.

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

3.4.1 Client application

The “Matrix multiply” eRPC client project is located in:

```
<ksdk_install_dir>/boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/i  
ar/
```

Project files for the eRPC client have the `_core0` suffix.

3.4.1.1 Client project basic source files

The startup files, board-related settings, and utilities belong to the basic project source files.

These files can be found in the following folders:

- `<ksdk_install_dir>/boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/`
- `<ksdk_install_dir>/devices/`

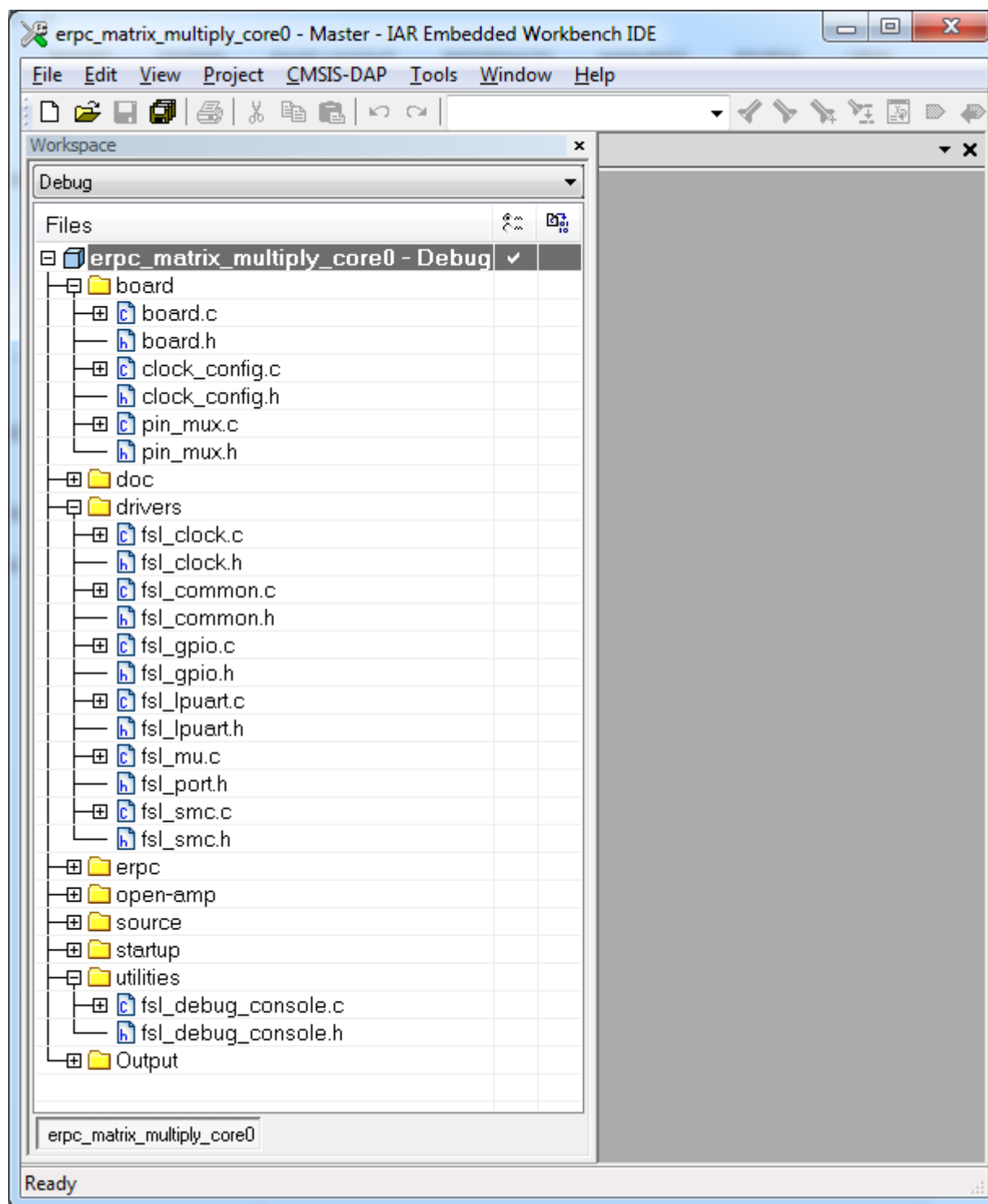


Figure 1: Client application

3.4.1.2 Client-related generated files

The client-related generated files are:

- `erpc_matrix_multiply.h`
- `erpc_matrix_multiply_client.cpp`

These files contain the shim code for functions and data types declared in the IDL file. These functions also call methods for codecs initialization, data serialization, performing eRPC requests, and de-serializing outputs into expected data structures, in case return values are expected. These files are located in the following folder:

```
<ksdk_install_dir>/boards/frdmk128t/multicore_examples/erpc_matrix_multiply/service/
```

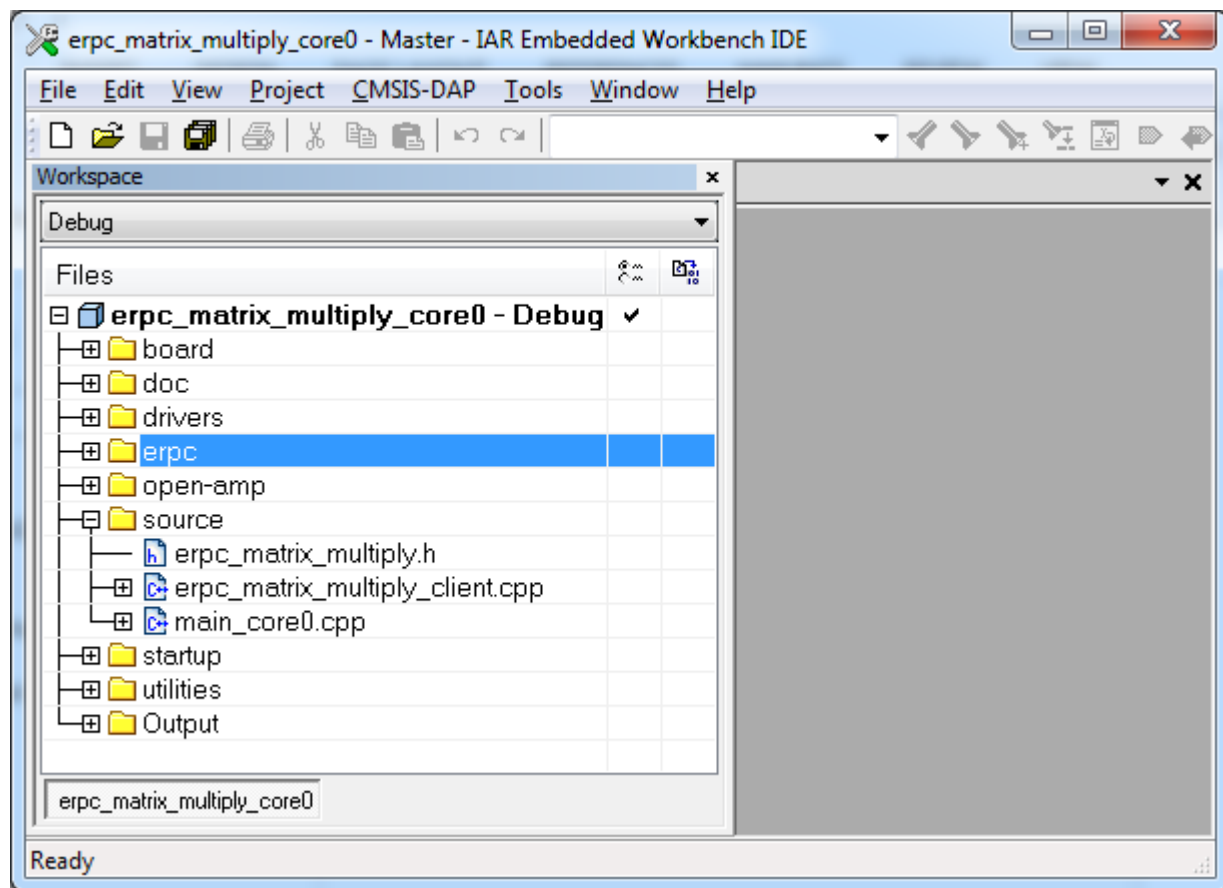


Figure 2: Client-related generated files

3.4.1.3 Client infrastructure files

The eRPC infrastructure files can be found in the following folder:

```
<ksdk_install_dir>/middleware/multicore_<version>/erpc/src/erpc
```

This directory contains files for creating eRPC client and server applications, and links them with the transport layer. These files are split into the following subfolders.

- The *client* directory contains files for creating client side application.
- The *codec* directory contains files for serializing base types.
- The *server* directory contains files for creating server side application.
- The *transport* directory contains files for linking eRPC application with transport layer. Each type of transport layer has its own subfolder.

Two files, `client_manager.h` and `client_manager.cpp`, are added for managing the client side application. The main functionality is to create, perform, and release eRPC requests.

Three files, `codec.h`, `basic_codec.h`, and `basic_codec.cpp`, are added for codecs. Currently, basic codec is the initial and only implementation of codecs.

The following message buffer files are added for storing serialized data: `message_buffer.h` and `message_buffer.cpp`.

RPMsg is used as the transport layer for the communication between cores. The following infrastructure files are added: `rpmsg_transport.h` and `rpmsg_transport.cpp`.

The “Matrix multiply” example also demonstrates the usage of C wrapped functions. It requires adding the following files: `client_setup.h`, `client_setup_rpmsg.cpp`, and `manually_constructed.h`. The initialization and de-initialization function for the client’s application are defined and declared in client setup files. The user does not need to know how to correctly initialize or de-initialize the client side application using C++ functions, and instead just calls these C functions. A manually constructed file is used for allocating static storage for used objects (transport medium, client, codec, and so on).

Because of the RPMsg, the transport layer it is also necessary to include RPMsg-related files. These files can be found in the following folder:

```
<ksdk_install_dir>/middleware/multicore_<version>/open-amp/
```

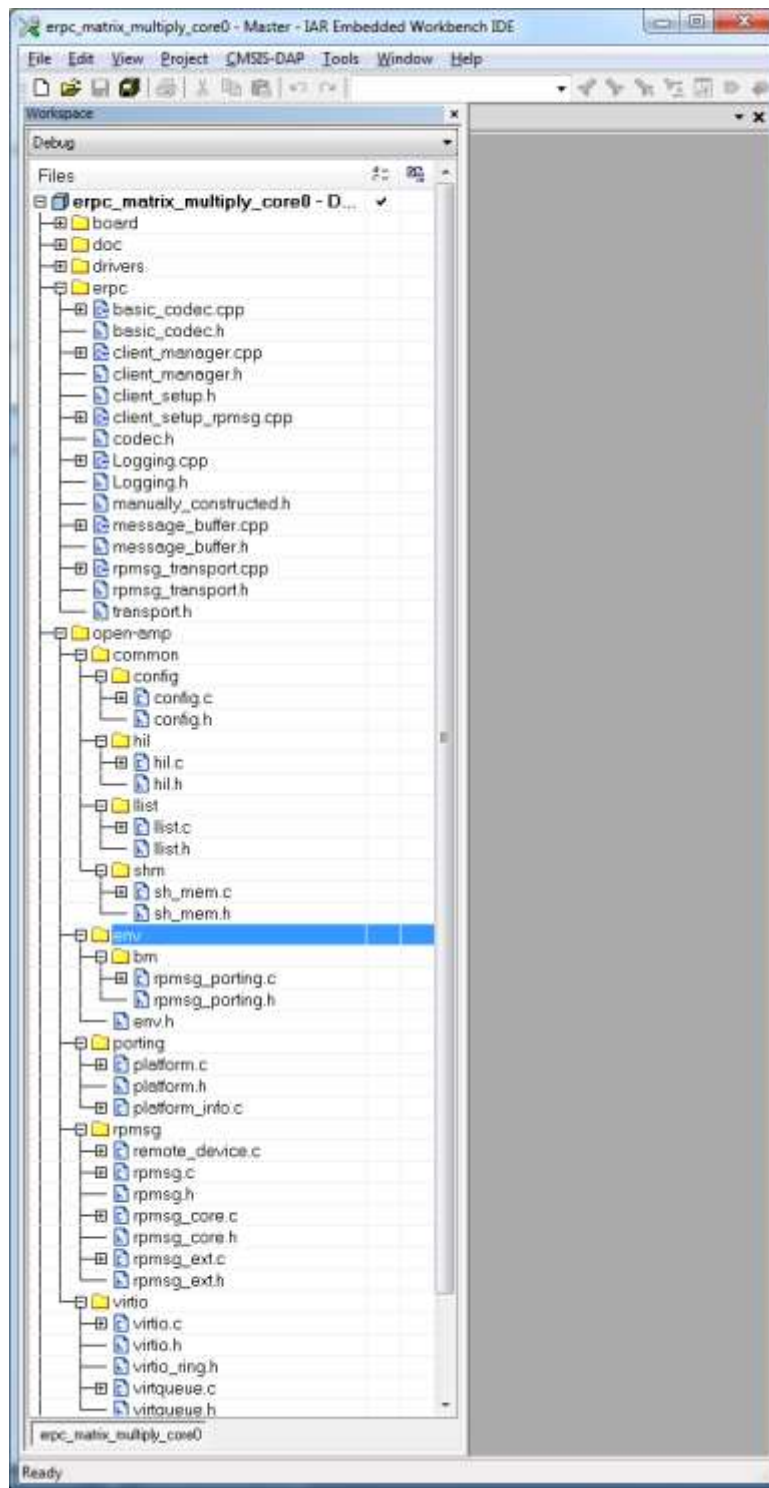


Figure 3: Client infrastructure files

3.4.1.4 Client user code

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

```
<ksdk_install_dir>/boards/frdmkl28t/multicore_example/erpc_matrix_multiply/
```

This file contains the code for hardware, mkl28t7, gpio, 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. The matrix multiplication can be issued repeatedly when pressing a software board button.

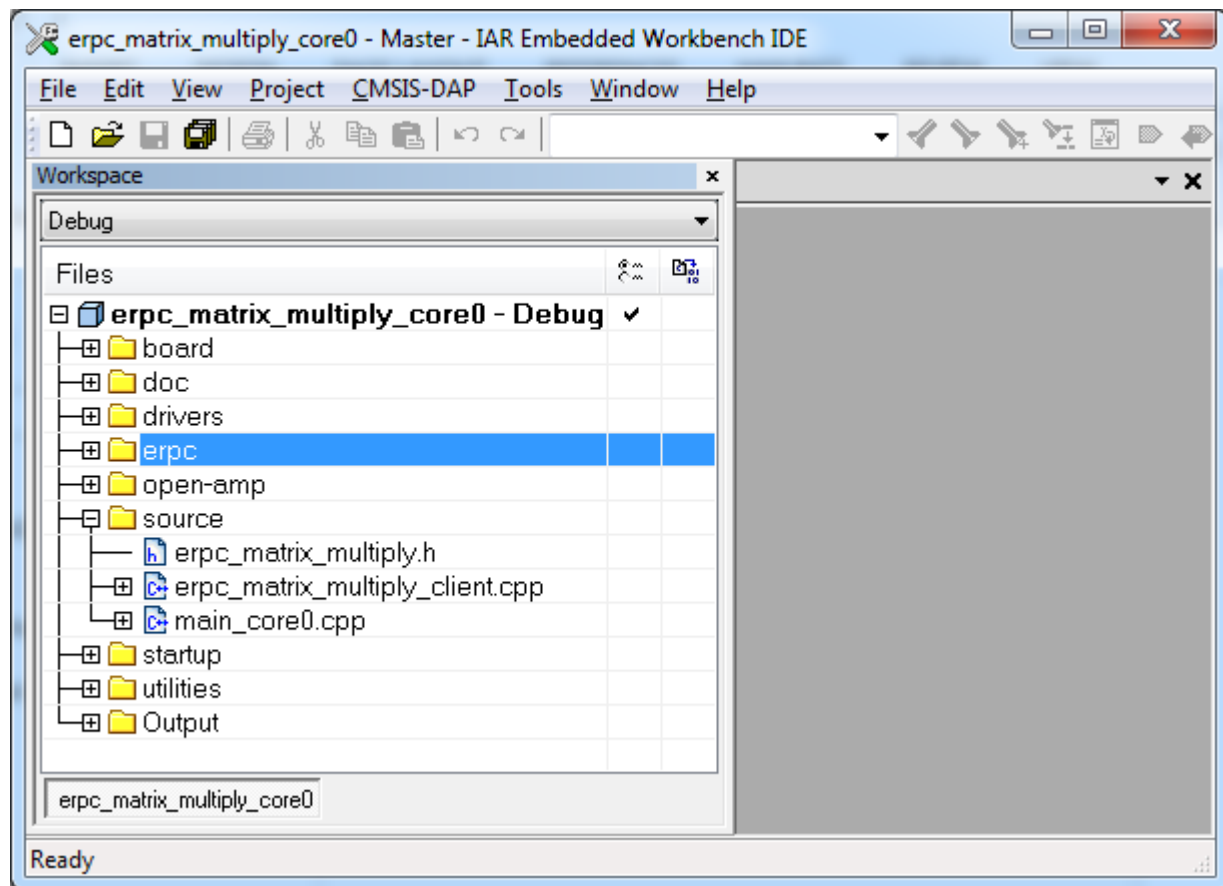


Figure 4: Client user code

3.4.2 Server application

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

```
<ksdk_install_dir>/boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/i  
ar/
```

The project files for the eRPC server have the `_core1` suffix.

3.4.2.1 Server project basic source files

The startup files, board-related settings, and utilities belong to the basic project source files. These files can be found in the following folders:

- `<ksdk_install_dir>/boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/`
- `<ksdk_install_dir>/devices/`

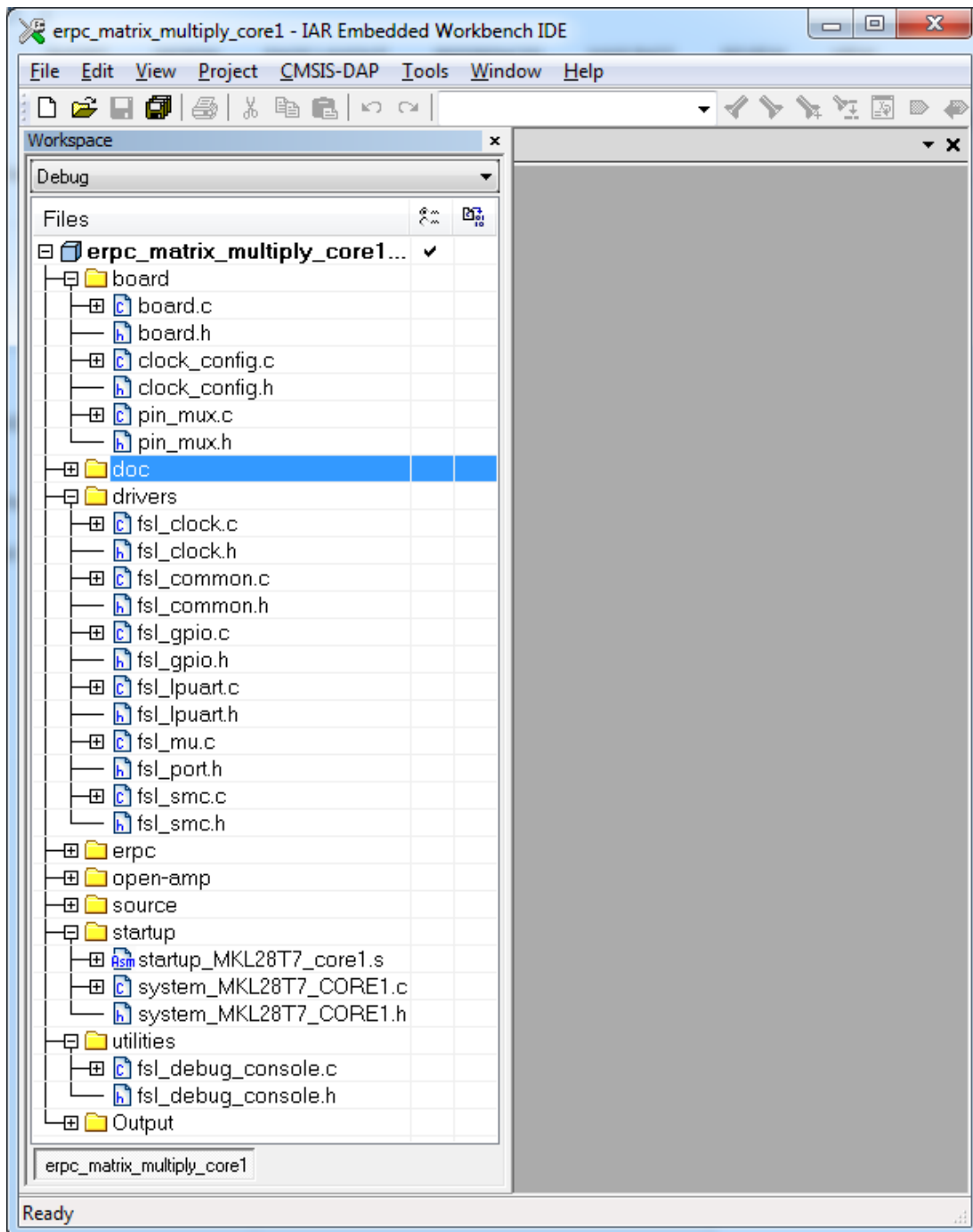


Figure 5: Server project basic source files

3.4.2.2 Server-related generated files

The server-related generated files are:

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

These 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 files are located in the following folder:

<ksdk_install_dir>/boards/frdmk128t/multicore_examples/erpc_matrix_multiply/service/

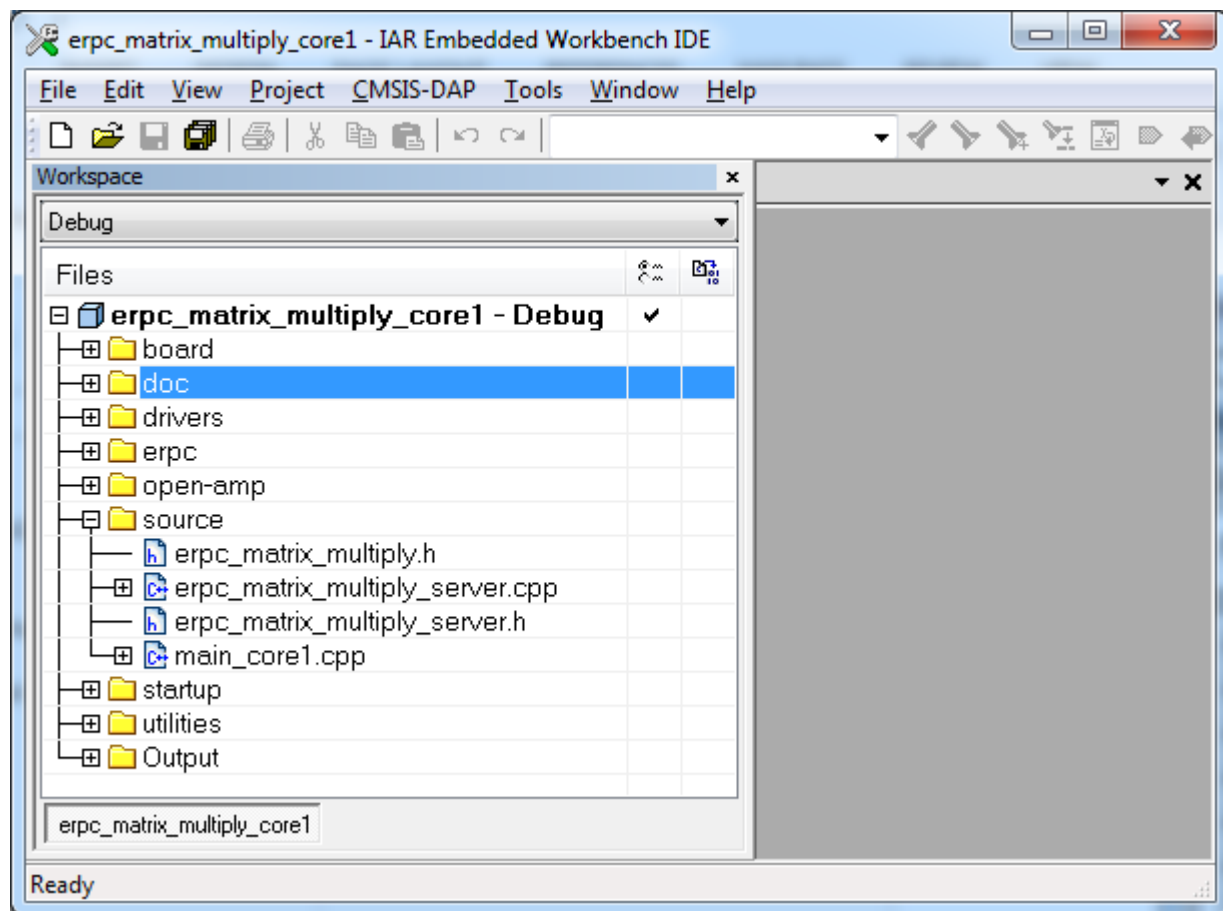


Figure 6: Server-related generated files

3.4.2.3 Server infrastructure files

The eRPC infrastructure files can be located in the following folder:

```
<ksdk_install_dir>/middleware/multicore/erpc/src/erpc
```

This directory contains files for creating the eRPC client and server applications, and links them with the transport layer. These files are split into the following subfolders.

- The *client* directory contains files for creating client side application.
- The *codec* directory contains files for serializing base types.
- The *server* directory contains files for creating server side application.
- The *transport* directory contains files for linking eRPC application with transport layer. Each type of transport layer has its own subfolder.

Three files, `server.h`, `simple_server.h`, and `simple_server.cpp`, are added for running the server on the server side application. The simple server is currently the only implementation of the server. Its role is to catch client requests, and identify and call requested functions implementation and send data back when requested.

Three files, `codec.h`, `basic_codec.h`, and `basic_codec.cpp`, are added for codecs. Currently, the basic codec is the initial and only implementation of codecs.

The following message buffer files are added for storing serialized data: `message_buffer.h` and `message_bufer.cpp`.

RPMsg is used as the transport layer for the communication between cores. The following infrastructure files are added: `rpmsg_transport.h` and `rpmsg_transport.cpp`.

The “Matrix multiply” example demonstrates the usage of C wrapped functions. It requires adding the `server_setup.h`, `server_setup_common.cpp`, `server_setup_rpmgs.cpp`, and `manually_constructed.h` files. The functions for server control are defined and declared in the server setup files. It deals with functions for initialization, de-initialization, run, poll server, and functions for adding service to the server. The user does not need to know how to correctly initialize or de-initialize the server side application using C++ functions, and just calls these C functions. The manually constructed file is used for allocating static storage for used objects (transport medium, client, codec, and so on).

Because of the RMPMsg transport layer, it is also necessary to include RMPMsg-related files. These files can be found in the following folder:

<ksdk_install_dir>/middleware/multicore_<version>/open-amp/

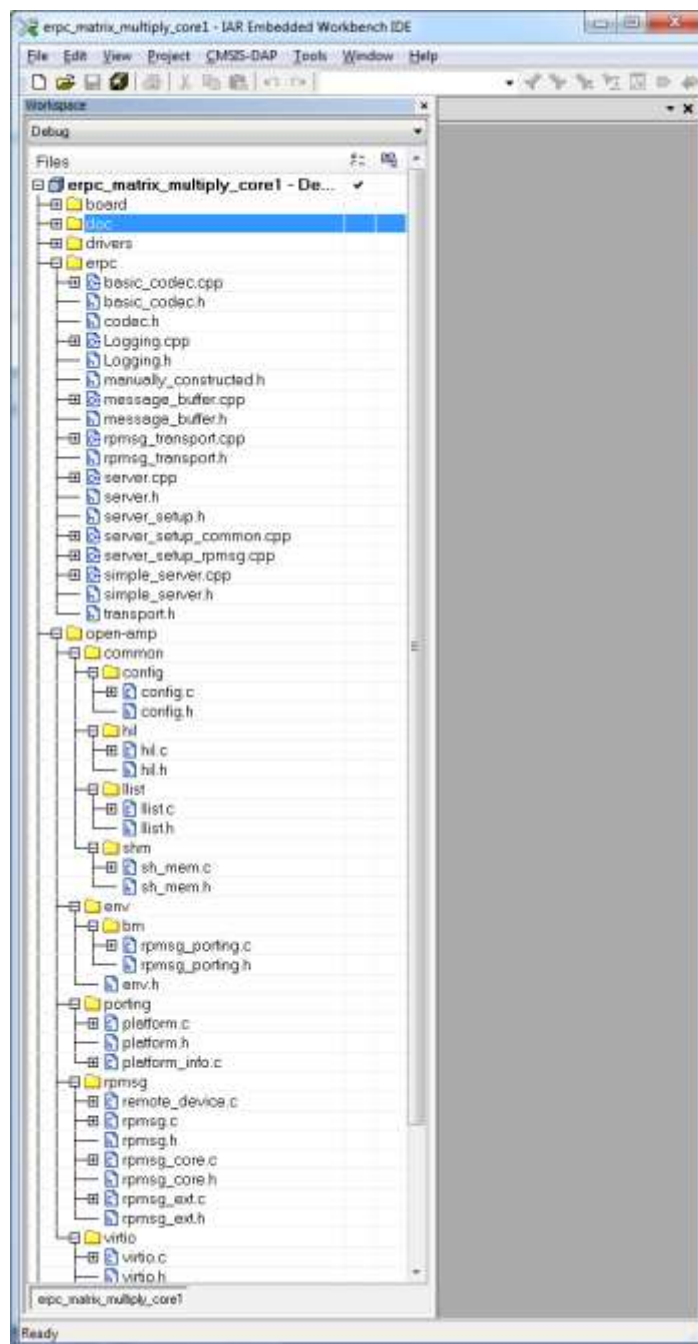


Figure 7: Server infrastructure files

3.4.2.4 Server user code

The server's user code is stored in *main_core1.cpp* file, located in the following folder:

<ksdk_install_dir>/boards/frdmkl28t/multicore_examples/erpc_matrix_multiply/

This file contains two functions:

- The main function contains the code for hardware, mkl28t7, and eRPC server initialization. After initialization, the matrix multiply service is added. When the service is added, the server is waiting for client's requests in the while loop.
- The second function is the implementation of the eRPC function (erpcMatrixMultiply) declared in the IDL file.

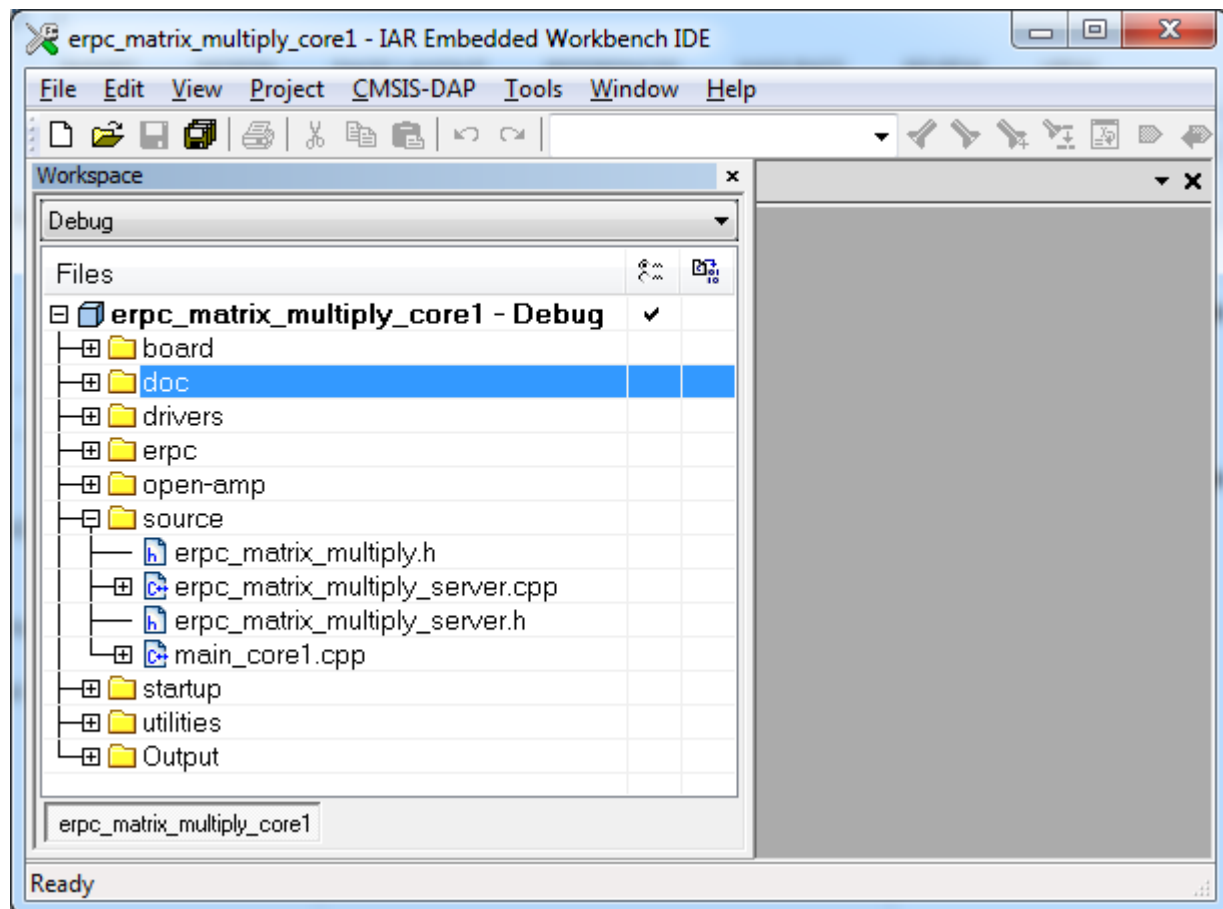
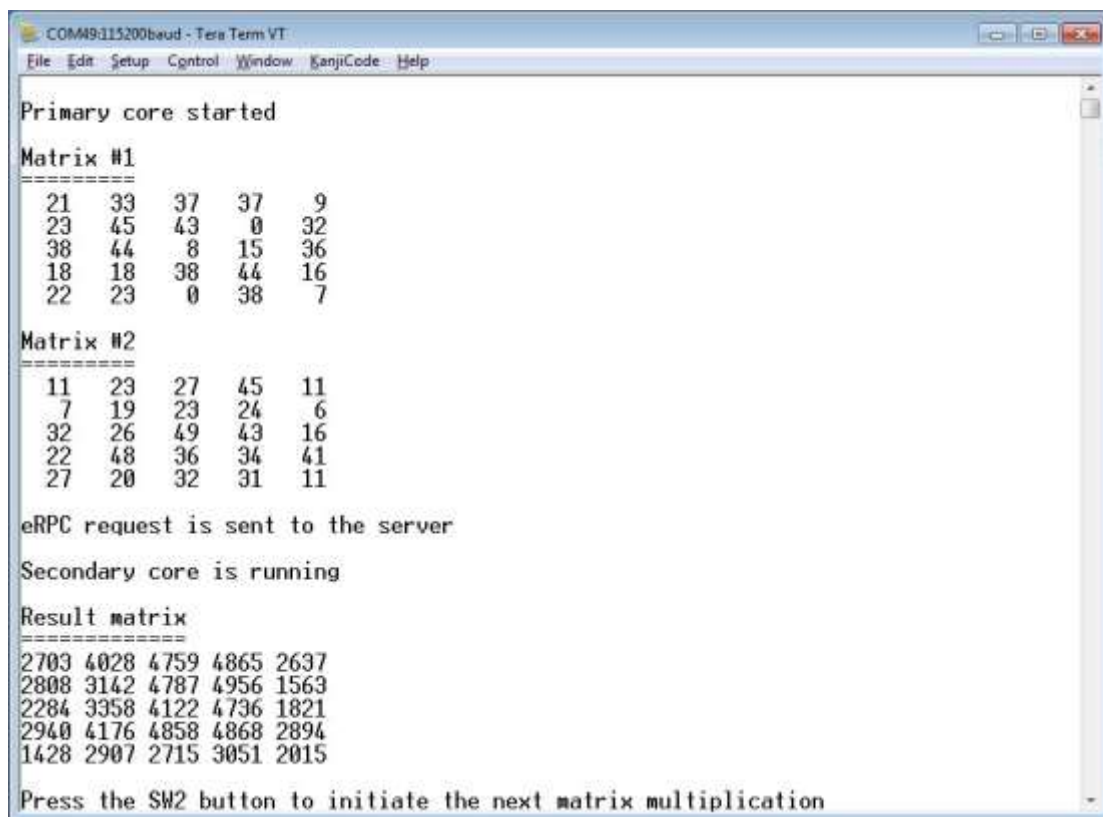


Figure 8: Server user code

3.5 Running the eRPC application

Follow the instructions stated in *Getting Started with Kinetis MKL28 Dual-Core and IAR Embedded Workbench®* (document KSDKMKL28GSUG) document located in the `<ksdk_install_dir>/doc/multicore_<version>/` folder to load both the primary and the secondary core images into the on-chip flash memory and effectively debug the dual-core application. Once the application is running, the following output can be captured on the serial console:



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

Primary core started
Matrix #1
=====
21 33 37 37 9
23 45 43 0 32
38 44 8 15 36
18 18 38 44 16
22 23 0 38 7
Matrix #2
=====
11 23 27 45 11
7 19 23 24 6
32 26 49 43 16
22 48 36 34 41
27 20 32 31 11
eRPC request is sent to the server
Secondary core is running
Result matrix
=====
2703 4028 4759 4865 2637
2808 3142 4787 4956 1563
2284 3358 4122 4736 1821
2940 4176 4858 4868 2894
1428 2907 2715 3051 2015
Press the SW2 button to initiate the next matrix multiplication
```

Figure 9: Running the eRPC application

4 Generic usage of the eRPC implementation

The eRPC implementation is generic, and the usage 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 for creating an eRPC application for the

PC running the Linux operating system. Based on the used type of transport medium, existing transport layers can be used, or a new one can be implemented.

5 Revision history

This revision history table summarizes changes contained in this document.

| Revision history | | |
|------------------|---------|--|
| Revision number | Date | Substantive changes |
| 0 | 09/2015 | Initial release |
| 1 | 03/2016 | Updated to Kinetis SDK v.2.0 and Multicore SDK v.1.1.0 |