NXP Semiconductors
User's Guide

Document Number: MCSDKERPCGSUG
Rev. 3, 09/2016

eRPC Getting Started User’s Guide

Contents

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).

Also see the eRPC documentation located in the `<ksdk_install_dir>/multicore_<version>/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).

   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 2 function parameters (2 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.

Let's take the NXP LPCXpresso54114 board as the target dualcore 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, i.e. 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.

Table 1. File locations

Multicore application source and project files	`<ksdk_install_dir>\boards\lpcxpresso54114\multicore_examples \erpc_matrix_multiply\`

Multiprocessor application source and project files	`<ksdk_install_dir>\boards\<board_name>\multiprocessor_examples \erpc_client_matrix_multiply_<transport_layer>\`
	
	`<ksdk_install_dir>\boards\<board_name>\multiprocessor_examples \erpc_server_matrix_multiply_<transport_layer>\`

eRPC source files	`<ksdk_install_dir>\middleware\multicore_<`*version*`>\erpc\`
RPMsg-Lite source files	`<ksdk_install_dir>\middleware\multicore_<`*version*`>\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:

<*ksdk_install_dir*>\boards\lpcxpresso54114\multicore_examples\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*.
- 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:

- program name at the top of the IDL file,
- followed by new data types and constants declarations,
- 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

for Windows® OS	<*ksdk_install_dir*>\middleware\multicore_<*version*>\tools\erpcgen\Windows
for Linux® OS	<*ksdk_install_dir*>\middleware\multicore_<*version*>\tools\erpcgen\Linux_x64 <*ksdk_install_dir*>\middleware\multicore_<*version*>\tools\erpcgen\Linux_x86

Table continues on the next page...

Table 2. eRPC generator application file locations (continued)

for Mac® OS	`<ksdk_install_dir>\middleware\multicore_<version>\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/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/service
        ../../../../../boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/
service/erpc_matrix_multiply.erpc
```

In both cases, the following 4 files are generated into the `<ksdk_install_dir>/boards/lpcxpresso54114/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 multiprocessor examples, the erpc file and pre-generated files can be found in `<ksdk_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 as a 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 *Getting Started with Kinetis SDK (KSDK) v.2.0* and/or *Getting Started with SDK v.2.0 for XXX Derivatives* documents located in the *<ksdk_install_dir>/docs/* folder.

Multiprocessor setup of the eRPC application is discussed here too. The behavior of this application is the same as in the multicore case, just the eRPC transport layer needs to be setup correctly in the application.

3.4.1 Multicore server application

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

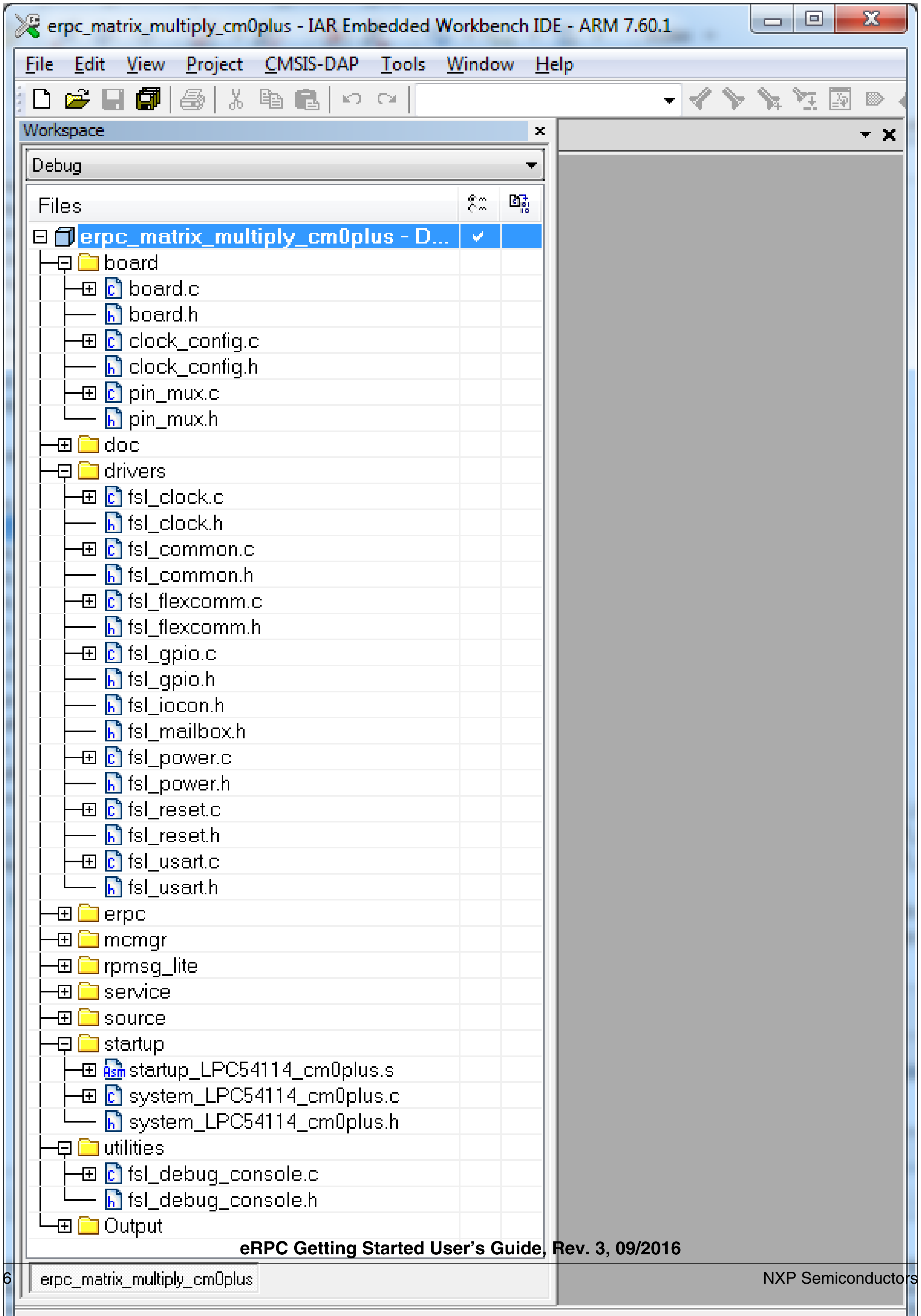
`<ksdk_install_dir>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/iar/`

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 dirvers and utilities belong to the basic project source files and form the skeleton of all KSDK applications. These source files are located in:

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


erpc_matrix_multiply_cm0plus - IAR Embedded Workbench IDE - ARM 7.60.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files

```
erpc_matrix_multiply_cm0plus - D...
  board
    board.c
    board.h
    clock_config.c
    clock_config.h
    pin_mux.c
    pin_mux.h
  doc
  drivers
    fsl_clock.c
    fsl_clock.h
    fsl_common.c
    fsl_common.h
    fsl_flexcomm.c
    fsl_flexcomm.h
    fsl_gpio.c
    fsl_gpio.h
    fsl_iocon.h
    fsl_mailbox.h
    fsl_power.c
    fsl_power.h
    fsl_reset.c
    fsl_reset.h
    fsl_usart.c
    fsl_usart.h
  erpc
  mcmgr
  rpmsg_lite
  service
  source
  startup
    startup_LPC54114_cm0plus.s
    system_LPC54114_cm0plus.c
    system_LPC54114_cm0plus.h
  utilities
    fsl_debug_console.c
    fsl_debug_console.h
  Output
```

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:

`<ksdk_install_dir>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/service/`

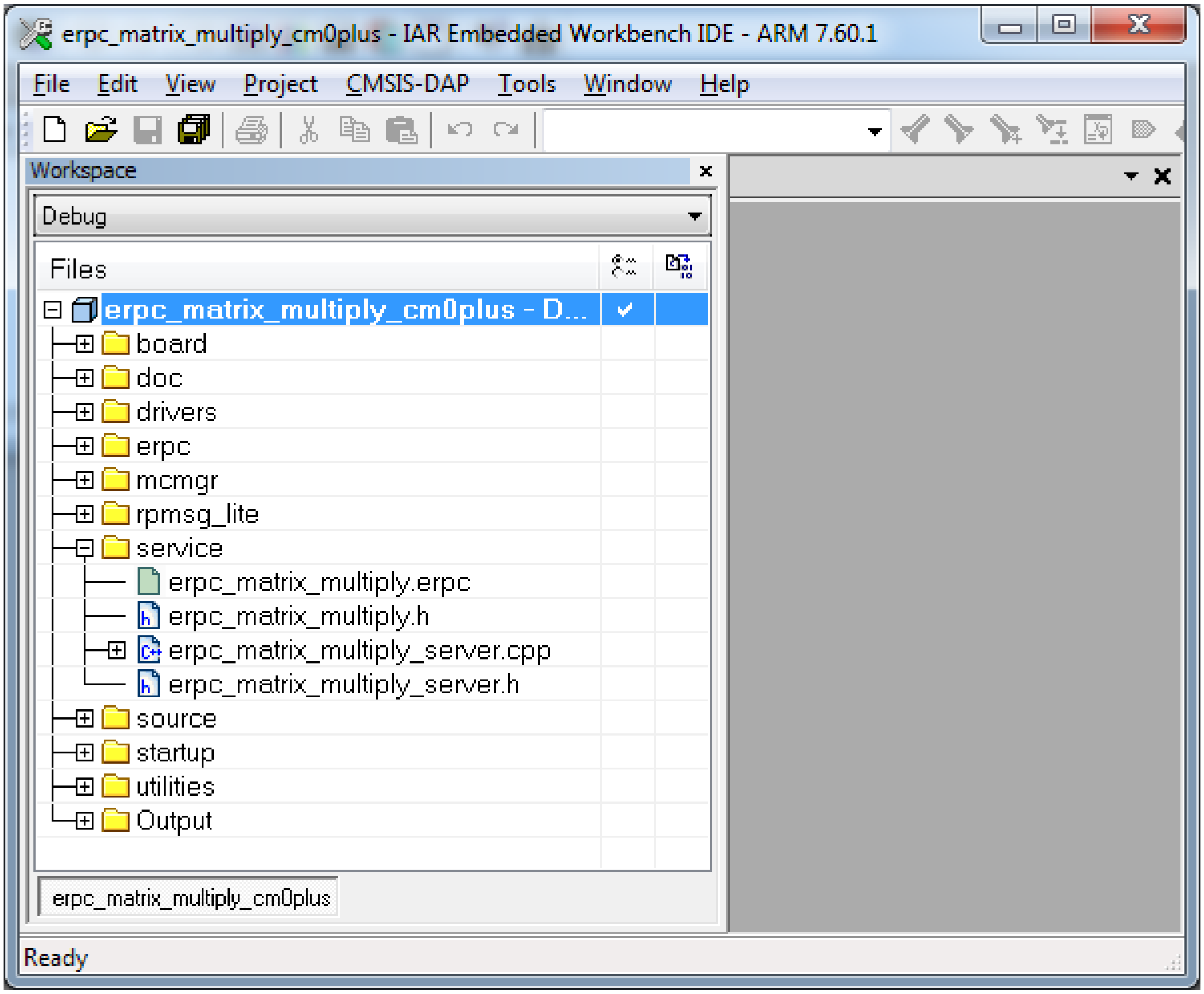

erpc_matrix_multiply_cm0plus - IAR Embedded Workbench IDE - ARM 7.60.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
Debug
Files
erpc_matrix_multiply_cm0plus - D...
board
doc
drivers
erpc
mcmgr
rpmsg_lite
service
erpc_matrix_multiply.erpc
erpc_matrix_multiply.h
erpc_matrix_multiply_server.cpp
erpc_matrix_multiply_server.h
source
startup
utilities
Output
erpc_matrix_multiply_cm0plus
Ready

Figure 2. Server-related generated files

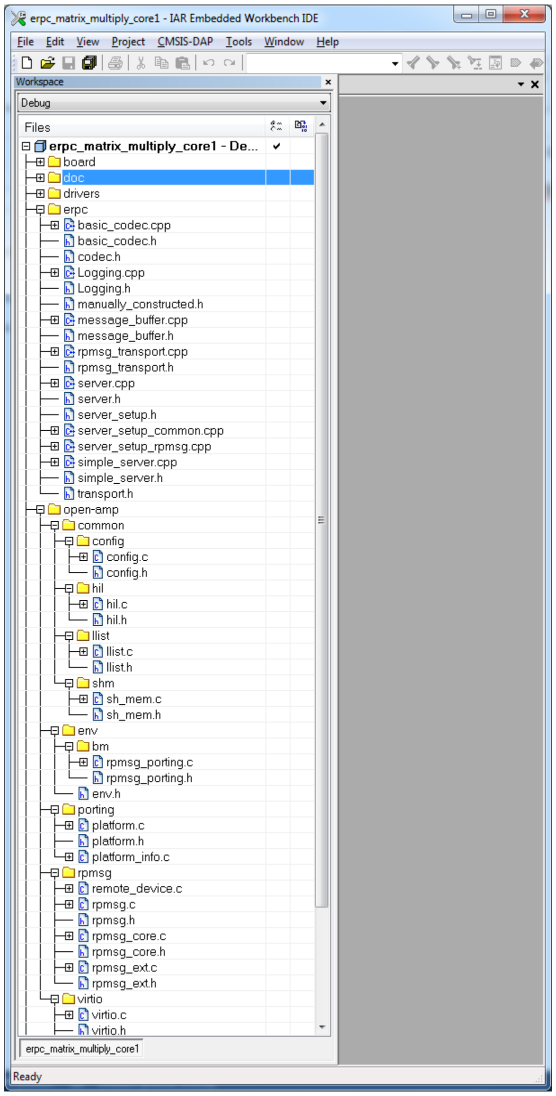
3.4.1.3 Server infrastructure files

The eRPC infrastructure files are located in:

`<ksdk_install_dir>/middleware/multicore_<version>/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 **config** subfolder contains the eRPC configuration.
    - `erpc_config.h` erpc configuration file.
- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
    - Four files, `server.h`, `server.cpp`, `simple_server.h` and `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 (`codec.h`, `basic_codec.h`, and `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 eprc definitions, typedefs and enums.
    - `manually_constructed.h` file is used for allocating static storage for the used objects.
    - Message buffer files are used for storing serialized data: `message_buffer.h` and `message_buffer.cpp`.
    - `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_server_setup.h` and `erpc_server_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_remote.cpp` files needs to be aded into the project in order to allow C-wrapped function for transport layer setup.
- 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, `rpmsg_lite_transport.h` and `rpmsg_lite_transport.cpp` files needs to be added into the server project.


erpc_matrix_multiply_core1 - IAR Embedded Workbench IDE
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files
Ready

```
erpc_matrix_multiply_core1 - De...
  board
  doc
  drivers
  erpc
    basic_codec.cpp
    basic_codec.h
    codec.h
    Logging.cpp
    Logging.h
    manually_constructed.h
    message_buffer.cpp
    message_buffer.h
    rpmsg_transport.cpp
    rpmsg_transport.h
    server.cpp
    server.h
    server_setup.h
    server_setup_common.cpp
    server_setup_rpmsg.cpp
    simple_server.cpp
    simple_server.h
    transport.h
  open-amp
    common
      config
        config.c
        config.h
      hil
        hil.c
        hil.h
      llist
        llist.c
        llist.h
      shm
        sh_mem.c
        sh_mem.h
    env
      bm
        rpmsg_porting.c
        rpmsg_porting.h
      env.h
    porting
      platform.c
      platform.h
      platform_info.c
    rpmsg
      remote_device.c
      rpmsg.c
      rpmsg.h
      rpmsg_core.c
      rpmsg_core.h
      rpmsg_ext.c
      rpmsg_ext.h
    virtio
      virtio.c
      virtio.h
erpc_matrix_multiply_core1
```

Figure 3. Server infrastructure files

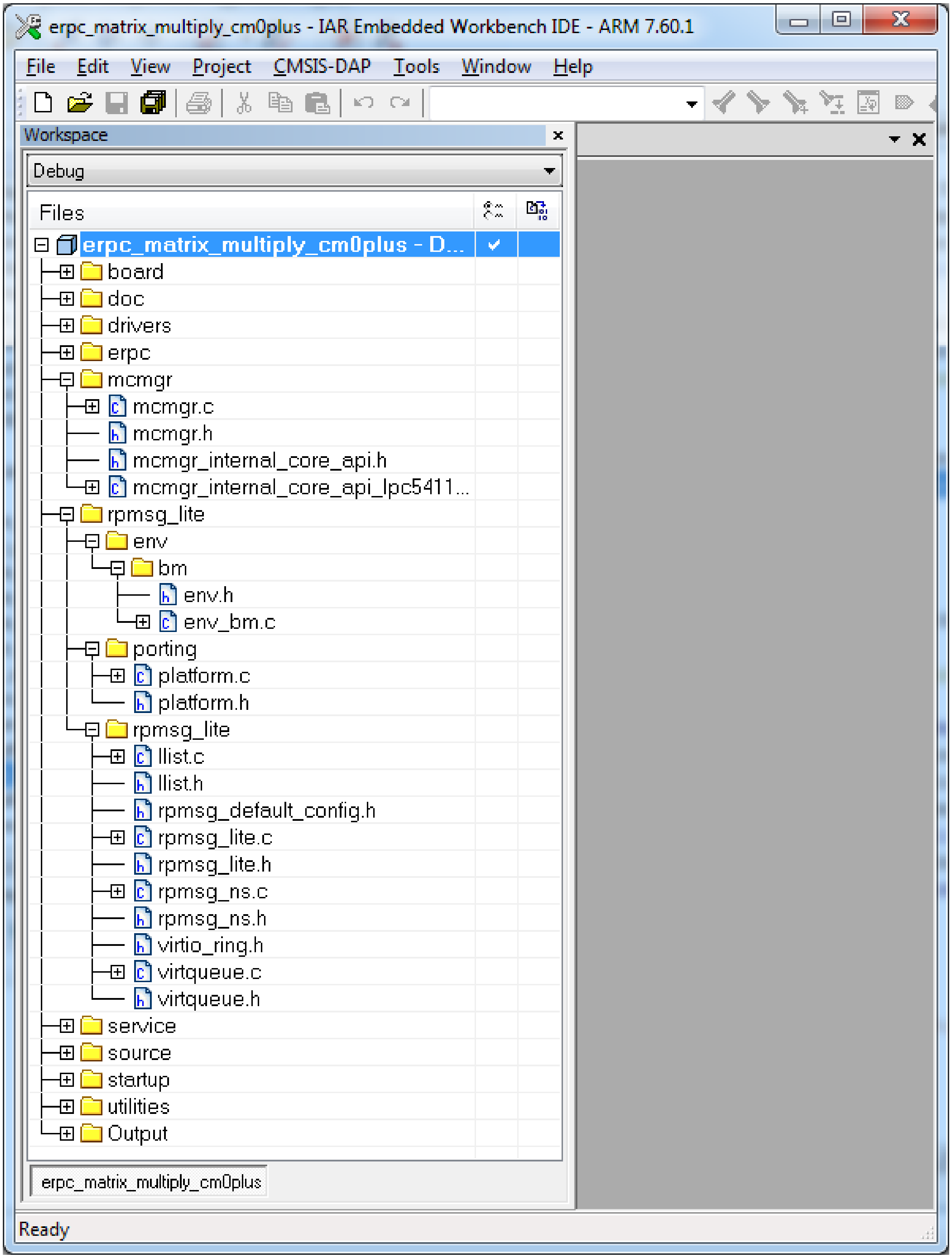
3.4.1.4 Server multicore infrastructure files

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

`<ksdk_install_dir>/middleware/multicore_<version>/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:

`<ksdk_install_dir>/middleware/multicore_<version>/mcmgr/`


erpc_matrix_multiply_cm0plus - IAR Embedded Workbench IDE - ARM 7.60.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files
erpc_matrix_multiply_cm0plus
Ready

```
erpc_matrix_multiply_cm0plus - D...
├─ board
├─ doc
├─ drivers
├─ erpc
├─ mcmgr
│  ├─ mcmgr.c
│  ├─ mcmgr.h
│  ├─ mcmgr_internal_core_api.h
│  └─ mcmgr_internal_core_api_lpc5411...
├─ rpmsg_lite
│  ├─ env
│  │  └─ bm
│  │     ├─ env.h
│  │     └─ env_bm.c
│  ├─ porting
│  │  ├─ platform.c
│  │  └─ platform.h
│  └─ rpmsg_lite
│     ├─ llist.c
│     ├─ llist.h
│     ├─ rpmsg_default_config.h
│     ├─ rpmsg_lite.c
│     ├─ rpmsg_lite.h
│     ├─ rpmsg_ns.c
│     ├─ rpmsg_ns.h
│     ├─ virtio_ring.h
│     ├─ virtqueue.c
│     └─ virtqueue.h
├─ service
├─ source
├─ startup
├─ utilities
└─ Output
```

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 folder:

`<`*`ksdk_install_dir`*`>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/`

The `main_core1.c` file contains two functions:

- The **main()** function contains the code for target board and eRPC server initialization. After the initialization, the matrix multiply service is added and the eRPC server is waiting 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 `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);
    ...
    /* eRPC server side initialization */
    erpc_server_init(transport);
    ...
    /* 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);
           ...
       }
       ...
   }
}
```

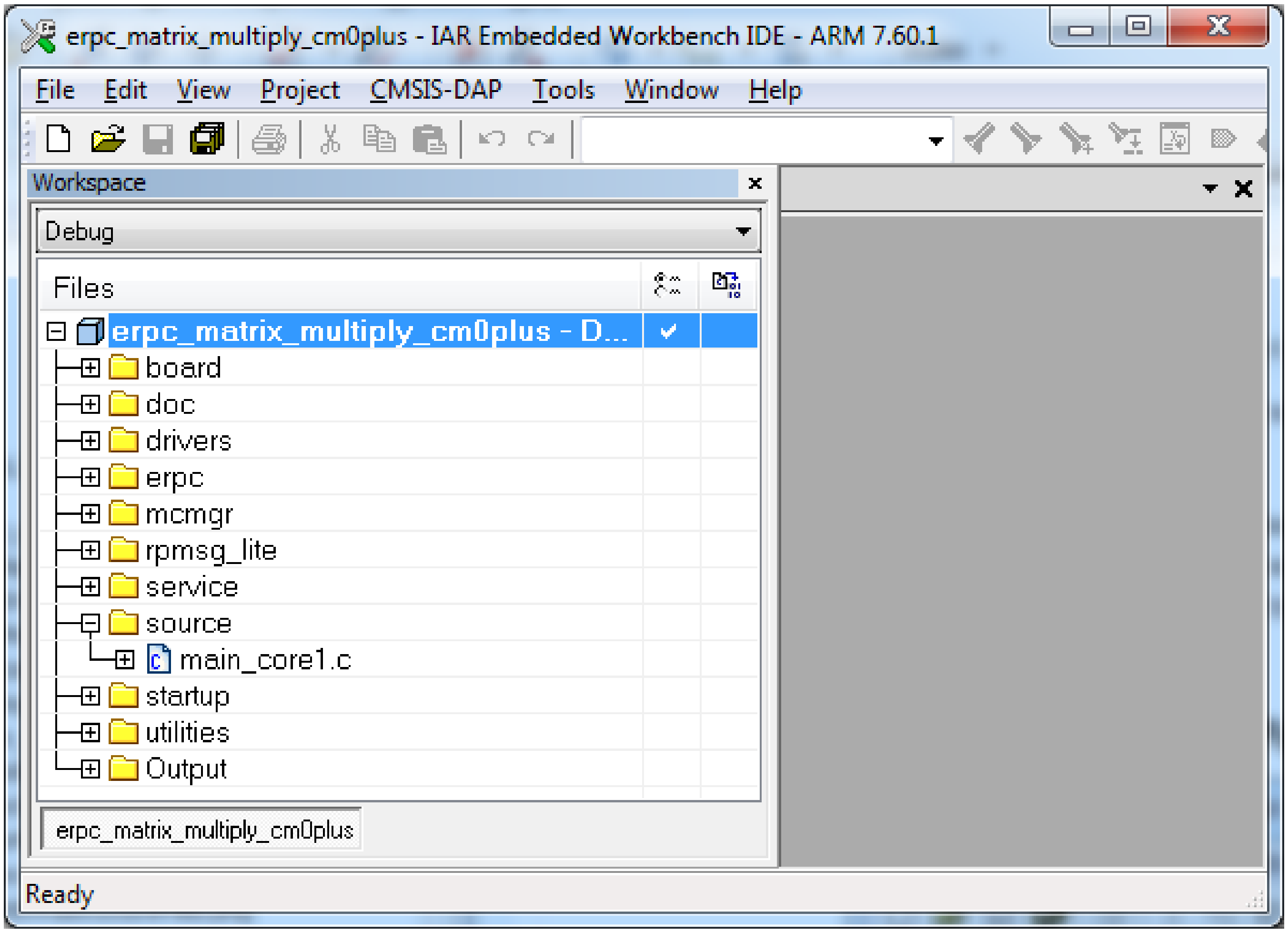


Figure 5. Server user code

3.4.2 Multicore client application

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

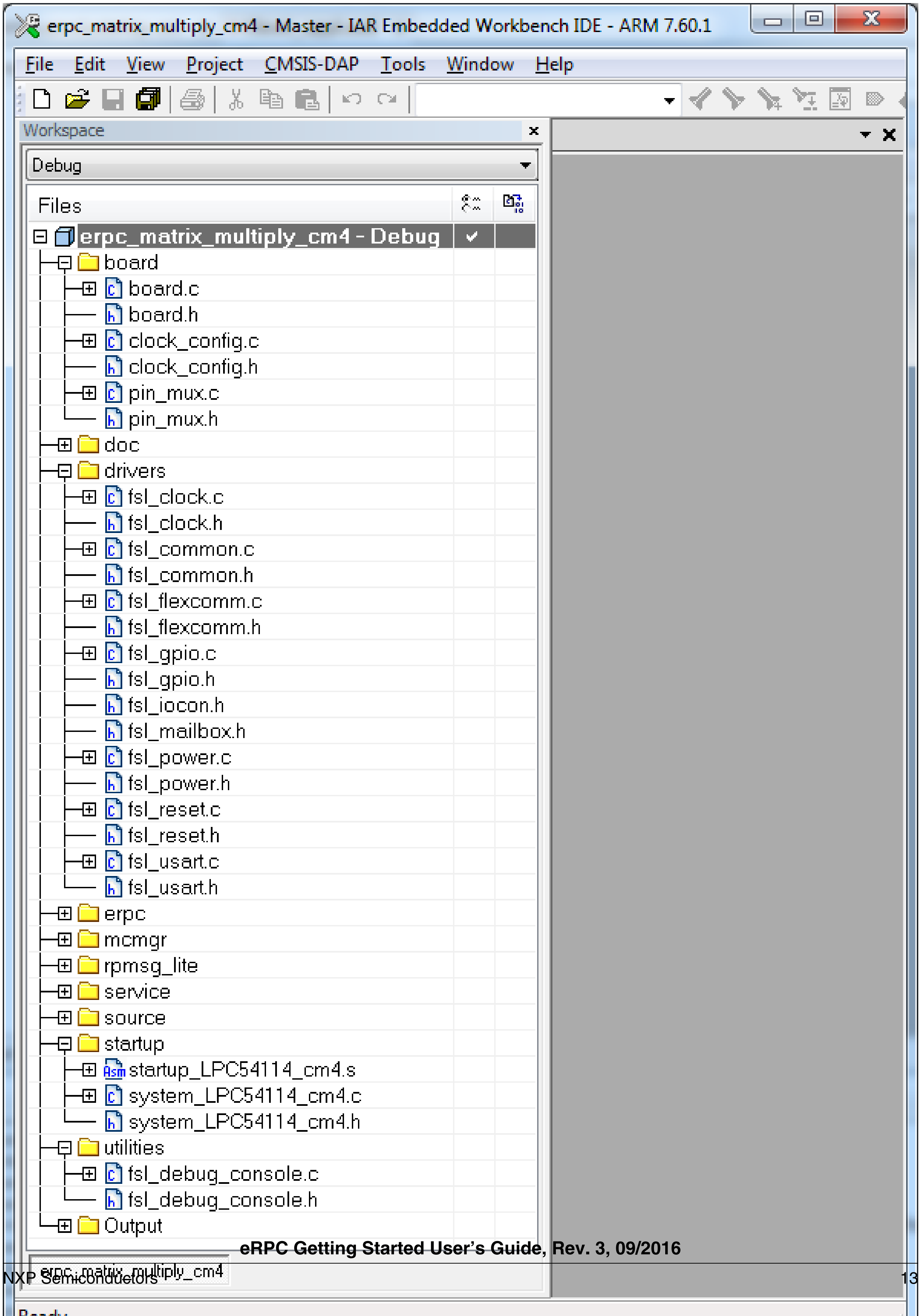
`<ksdk_install_dir>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/iar/`

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 dirvers and utilities belong to the basic project source files and form the skeleton of all KSDK applications. These source files are located in:

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


erpc_matrix_multiply_cm4 - Master - IAR Embedded Workbench IDE - ARM 7.60.1
File
Edit
View
Project
CMSIS-DAP
Tools
Window
Help
Workspace
Debug
Files
erpc_matrix_multiply_cm4 - Debug
board
board.c
board.h
clock_config.c
clock_config.h
pin_mux.c
pin_mux.h
doc
drivers
fsl_clock.c
fsl_clock.h
fsl_common.c
fsl_common.h
fsl_flexcomm.c
fsl_flexcomm.h
fsl_gpio.c
fsl_gpio.h
fsl_iocon.h
fsl_mailbox.h
fsl_power.c
fsl_power.h
fsl_reset.c
fsl_reset.h
fsl_usart.c
fsl_usart.h
erpc
mcmgr
rpmsg_lite
service
source
startup
startup_LPC54114_cm4.s
system_LPC54114_cm4.c
system_LPC54114_cm4.h
utilities
fsl_debug_console.c
fsl_debug_console.h
Output
erpc_matrix_multiply_cm4
Ready

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:

`<`*`ksdk_install_dir`*`>/boards/lpcxpresso54114/multicore_examples/erpc_matrix_multiply/service/`

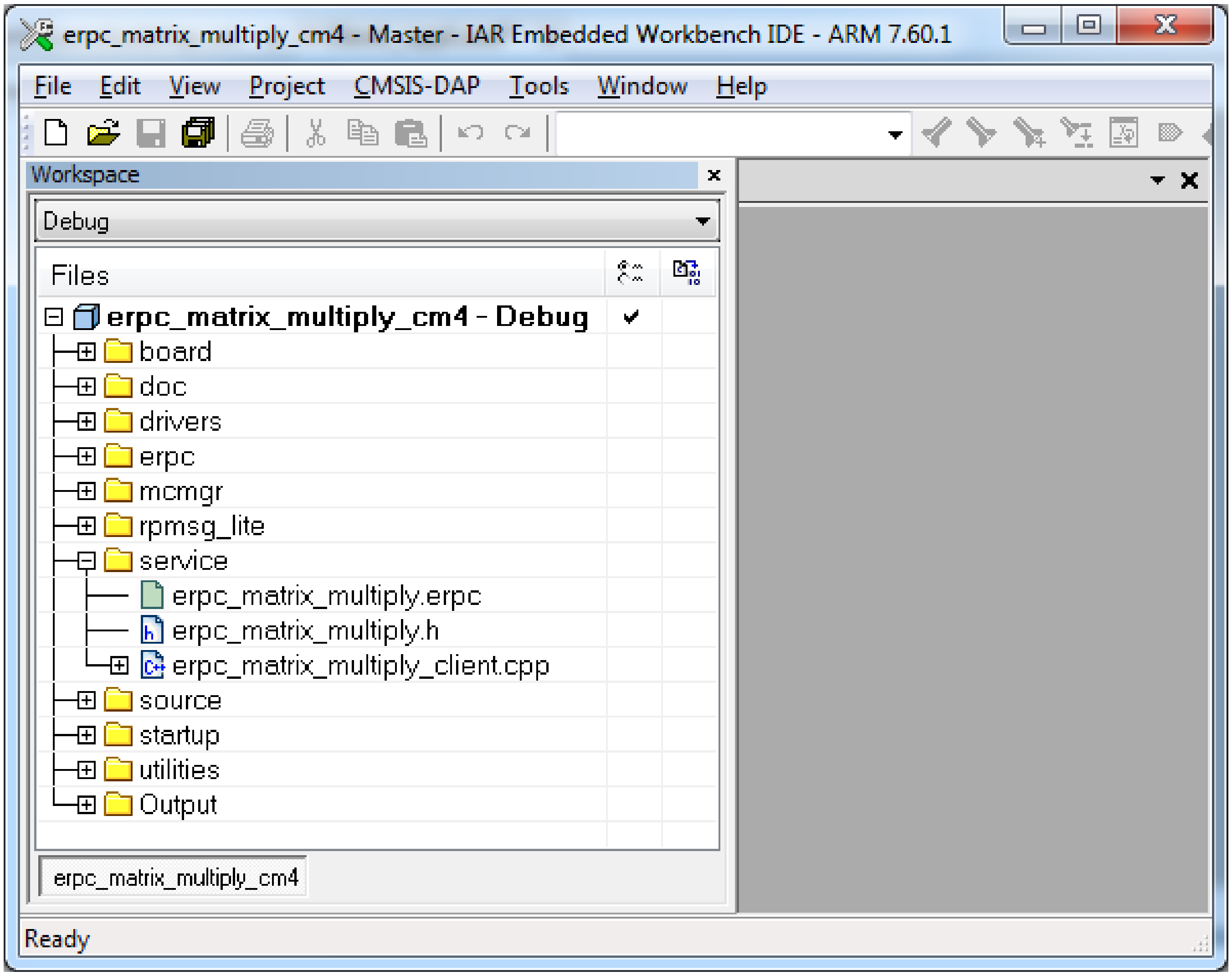


Figure 7. Client-related generated files

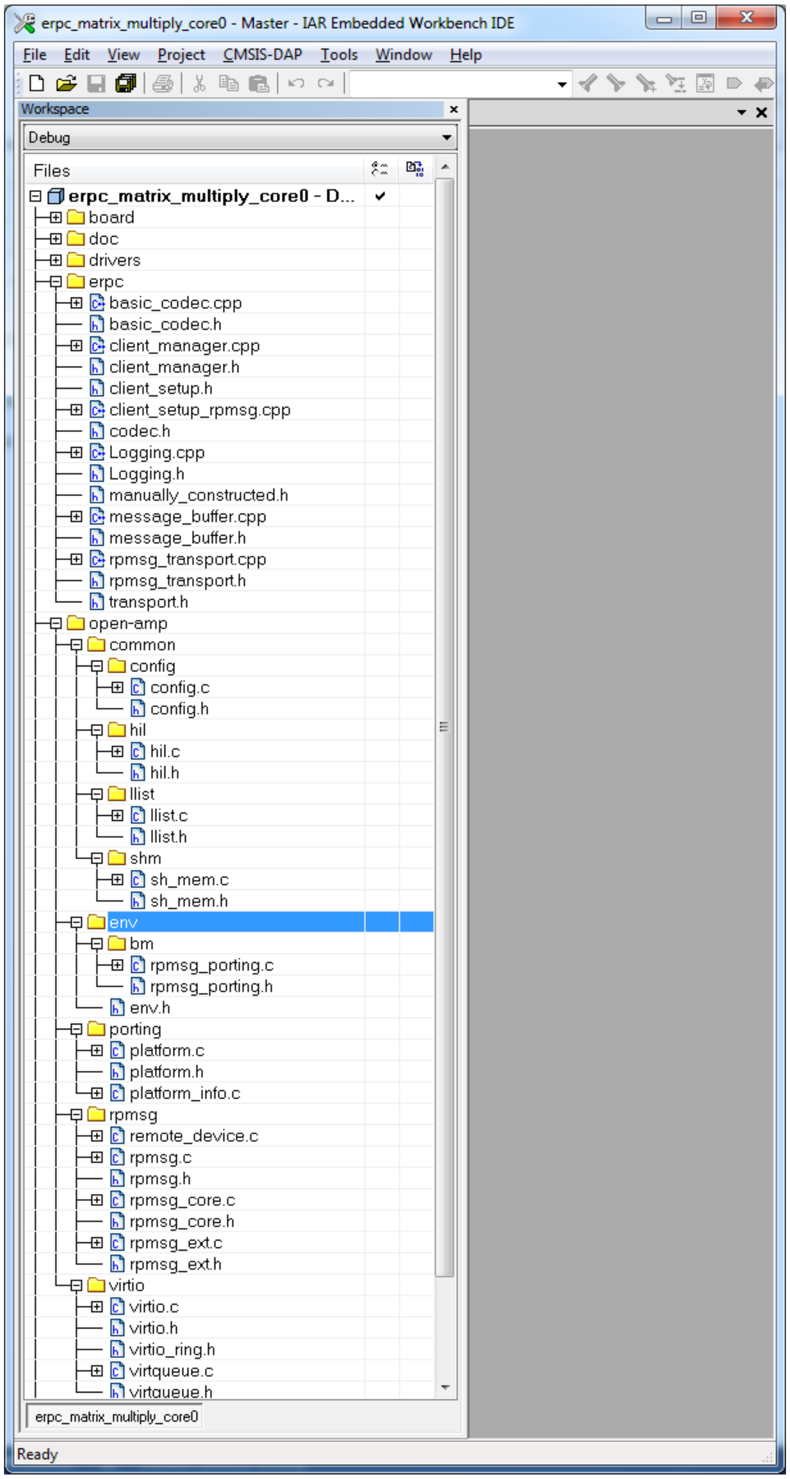
3.4.2.3 Client infrastructure files

The eRPC infrastructure files are located in:

`<`*`ksdk_install_dir`*`>/middleware/multicore_<`*`version`*`>/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 **config** subfolder contains the eRPC configuration.
    - `erpc_config.h` erpc configuration file.
- The **infra** subfolder contains C++ infrastructure code used to build server and client applications.
    - Two files, `client_manager.h` and `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 (`codec.h`, `basic_codec.h`, and `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 eprc definitions, typedefs and enums.
    - `manually_constructed.h` file is used for allocating static storage for the used objects.
    - Message buffer files are used for storing serialized data: `message_buffer.h` and `message_buffer.cpp`.
    - `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 aded into the project in order to allow C-wrapped function for transport layer setup.
- 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, `rpmsg_lite_transport.h` and `rpmsg_lite_transport.cpp` files needs to be added into the client project.


erpc_matrix_multiply_core0 - Master - IAR Embedded Workbench IDE
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files
Ready

```
erpc_matrix_multiply_core0 - D...  ✔
├⊞ board
├⊞ doc
├⊞ drivers
├⊟ erpc
│ ├⊞ basic_codec.cpp
│ ├── basic_codec.h
│ ├⊞ client_manager.cpp
│ ├── client_manager.h
│ ├── client_setup.h
│ ├⊞ client_setup_rpmsg.cpp
│ ├── codec.h
│ ├⊞ Logging.cpp
│ ├── Logging.h
│ ├── manually_constructed.h
│ ├⊞ message_buffer.cpp
│ ├── message_buffer.h
│ ├⊞ rpmsg_transport.cpp
│ ├── rpmsg_transport.h
│ └── transport.h
├⊟ open-amp
│ ├⊟ common
│ │ ├⊟ config
│ │ │ ├⊞ config.c
│ │ │ └── config.h
│ │ ├⊟ hil
│ │ │ ├⊞ hil.c
│ │ │ └── hil.h
│ │ ├⊟ llist
│ │ │ ├⊞ llist.c
│ │ │ └── llist.h
│ │ └⊟ shm
│ │   ├⊞ sh_mem.c
│ │   └── sh_mem.h
│ ├⊟ env
│ │ ├⊟ bm
│ │ │ ├⊞ rpmsg_porting.c
│ │ │ └── rpmsg_porting.h
│ │ └── env.h
│ ├⊟ porting
│ │ ├⊞ platform.c
│ │ ├── platform.h
│ │ └⊞ platform_info.c
│ ├⊟ rpmsg
│ │ ├⊞ remote_device.c
│ │ ├⊞ rpmsg.c
│ │ ├── rpmsg.h
│ │ ├⊞ rpmsg_core.c
│ │ ├── rpmsg_core.h
│ │ ├⊞ rpmsg_ext.c
│ │ └── rpmsg_ext.h
│ └⊟ virtio
│   ├⊞ virtio.c
│   ├── virtio.h
│   ├── virtio_ring.h
│   ├⊞ virtqueue.c
│   └── virtqueue.h
erpc_matrix_multiply_core0
```

Figure 8. Client infrastructure files

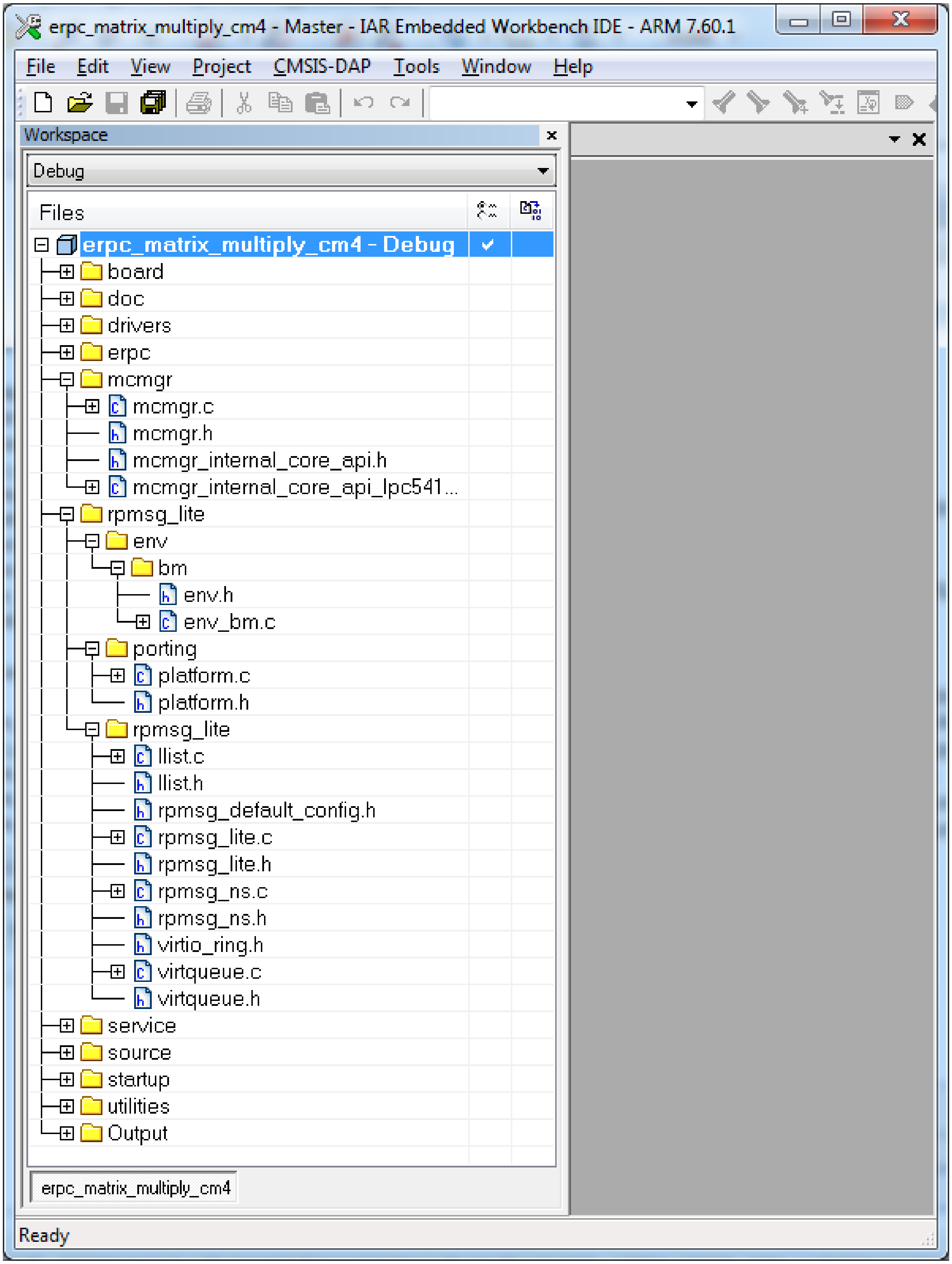
3.4.2.4 Client multicore infrastructure files

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

```
<ksdk_install_dir>/middleware/multicore_<version>/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:

```
<ksdk_install_dir>/middleware/multicore_<version>/mcmgr/
```


erpc_matrix_multiply_cm4 - Master - IAR Embedded Workbench IDE - ARM 7.60.1

```
erpc_matrix_multiply_cm4 - Debug ✓
├ board
├ doc
├ drivers
├ erpc
├ mcmgr
│ ├ mcmgr.c
│ ├ mcmgr.h
│ ├ mcmgr_internal_core_api.h
│ └ mcmgr_internal_core_api_lpc541...
├ rpmsg_lite
│ ├ env
│ │ └ bm
│ │   ├ env.h
│ │   └ env_bm.c
│ ├ porting
│ │ ├ platform.c
│ │ └ platform.h
│ └ rpmsg_lite
│   ├ llist.c
│   ├ llist.h
│   ├ rpmsg_default_config.h
│   ├ rpmsg_lite.c
│   ├ rpmsg_lite.h
│   ├ rpmsg_ns.c
│   ├ rpmsg_ns.h
│   ├ virtio_ring.h
│   ├ virtqueue.c
│   └ virtqueue.h
├ service
├ source
├ startup
├ utilities
└ Output
```

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 folder:

`<ksdk_install_dir>/boards/lpcxpresso54114/multicore_example/erpc_matrix_multiply/`

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 2 matrix variables.
- The erpcMatrixMultiply eRPC function is called to issue the eRPC request and get the result.

There is the possibility 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, matrix2, result_matrix = {0};
...
/* RPMsg-Lite transport layer initialization */
erpc_transport_t transport;
transport = erpc_transport_rpmsg_lite_master_init(src, dst);
...
/* eRPC client side initialization */
erpc_client_init(transport);
...
/* Set default error handler */
erpc_client_set_error_handler(erpc_error_handler);
...
while (1)
{
    /* Invoke the erpcMatrixMultiply function */
    erpcMatrixMultiply((const Matrix *)&matrix1, (const Matrix *)&matrix2, &result_matrix);
    ...
    /* Check if some error occured in eRPC */
    if (g_erpc_error_occurred)
    {
        /* Exit program loop */
        break;
    }
    ...
}
```

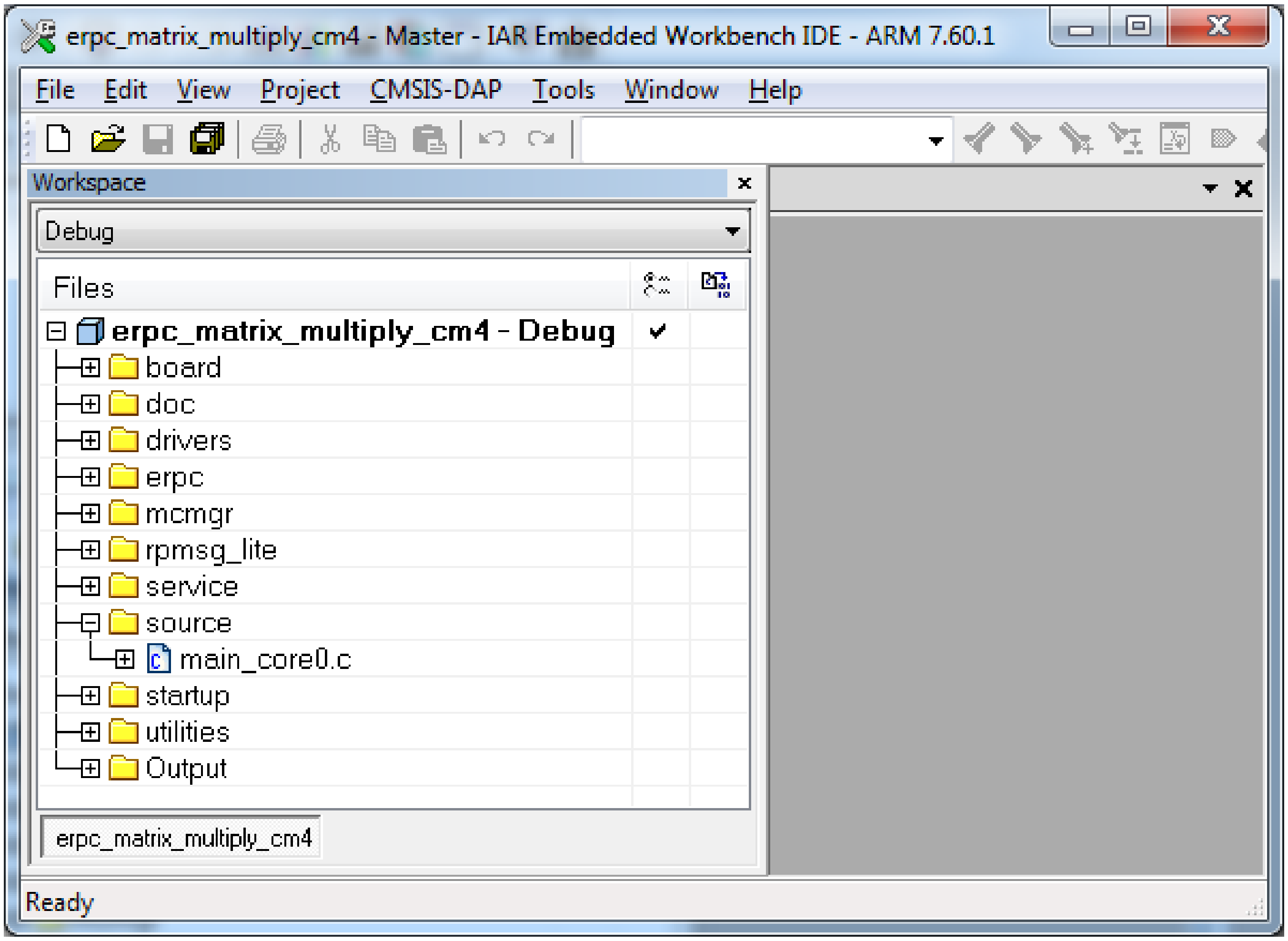

erpc_matrix_multiply_cm4 - Master - IAR Embedded Workbench IDE - ARM 7.60.1
File Edit View Project CMSIS-DAP Tools Window Help
Workspace
Debug
Files
erpc_matrix_multiply_cm4 - Debug
board
doc
drivers
erpc
mcmgr
rpmsg_lite
service
source
main_core0.c
startup
utilities
Output
erpc_matrix_multiply_cm4
Ready

Figure 10. Client user code

3.4.3 Multiprocessor server application

The “Matrix multiply” eRPC server project for multiprocessor applications is located in:

<*ksdk_install_dir*>/boards/<board_name>/multiprocessor_examples/
erpc_server_matrix_multiply_<transport_layer>

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 requiered 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\(d)spi_slave_transport.h`
	`<eRPC base directory>\erpc_c\transports\(d)spi_slave_transport.cpp`

Table continues on the next page...

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

UART	`<eRPC base directory>\erpc_c\setup\erpc_setup_uart_cmsis.cpp`
	`<eRPC base directory>\erpc_c\transports\uart_cmsis_transport.h`
	`<eRPC base directory>\erpc_c\transports\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 folder:

`<ksdk_install_dir>/boards/<board_name>/multiprocessor_examples/`
`erpc_server_matrix_multiply_<transport_layer>/`

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

```
 /* erpcMatrixMultiply function user implementation */
 void erpcMatrixMultiply(const Matrix *matrix1, const 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);
     ...
     /* eRPC server side initialization */
     erpc_server_init(transport);
     ...
     /* 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);
            ...
        }
        ...
    }
 }
```

3.4.4 Multiprocessor client application

The "Matrix multiply" eRPC client project for multiprocessor applications is located in:

<*ksdk_install_dir*>/boards/<board_name>/multiprocessor_examples/
erpc_client_matrix_multiply_<transport_layer>/iar/

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 requiered 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\(d)spi_master_transport.h`
	
	`<eRPC base directory>\erpc_c\transports\(d)spi_master_transport.cpp`

UART	`<eRPC base directory>\erpc_c\setup\erpc_setup_uart_cmsis.cpp`
	
	`<eRPC base directory>\erpc_c\transports\uart_cmsis_transport.h`
	
	`<eRPC base directory>\erpc_c\transports\uart_cmsis_transport.cpp`

3.4.4.1 Client user code

The client's user code is stored in the `main_client.c` file, located in the folder:

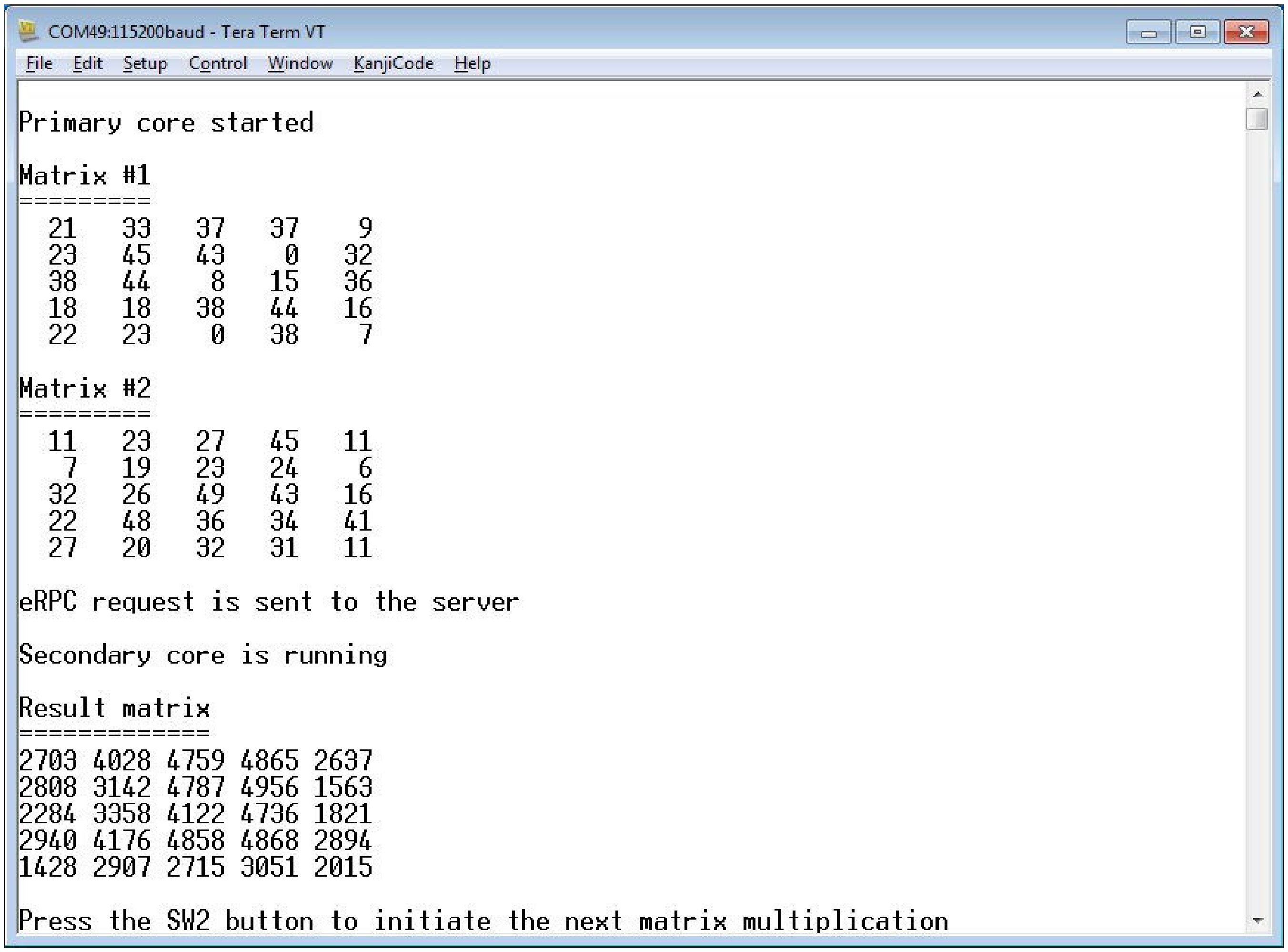
```
<ksdk_install_dir>/boards/<board_name>/multiprocessor_examples/
erpc_client_matrix_multiply_<transport_layer>/
```

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, matrix2, result_matrix = {0};
 ...
 /* RPMsg-Lite 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);
 ...
 /* eRPC client side initialization */
 erpc_client_init(transport);
 ...
 /* Set default error handler */
 erpc_client_set_error_handler(erpc_error_handler);
 ...
 while (1)
 {
     /* Invoke the erpcMatrixMultiply function */
     erpcMatrixMultiply((const Matrix *)&matrix1, (const Matrix *)&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 SDK v.2.0 for LPC5411x Derivatives* (document KSDK20LPC5411XGSUG) (located in the `<ksdk_install_dir>/docs` folder), to load both the primary and the secondary core images into the on-chip memory, and then effectively debug the dual-core application. After the application is running, the serial console should look like:


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

```
Primary core started

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

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

eRPC request is sent to the server

Secondary core is running

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

Press the SW2 button to initiate the next matrix multiplication
```

Figure 11. Running the eRPC application

For multiprocessor applications that are running between PC and the target evaluation board or between two boards, follow instructions in 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 https://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.

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 the eRPC v.1.3.0 Added new sections covering multiprocessor applications.

Document Number MCSDKERPCGSUG
Rev 3
09/2016