NXP Semiconductors
User's Guide

Document Number: MCUXSDKLPC5410XGSUG
Rev. 0, 08/2019

Getting Started with MCUXpresso SDK for LPC5410x

1 Overview

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for Kinetis and LPC Microcontrollers. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. The MCUXpresso SDK contains FreeRTOS and various other middleware to support rapid development.

For supported toolchain versions, see the MCUXpresso SDK Release Notes Supporting LPCXpresso54102 (document MCUXSDKLPC5410XRN).

For more details about MCUXpresso SDK, see the MCUXpresso SDK homepage MCUXpresso-SDK: Software Development Kit for MCUXpresso.

Contents

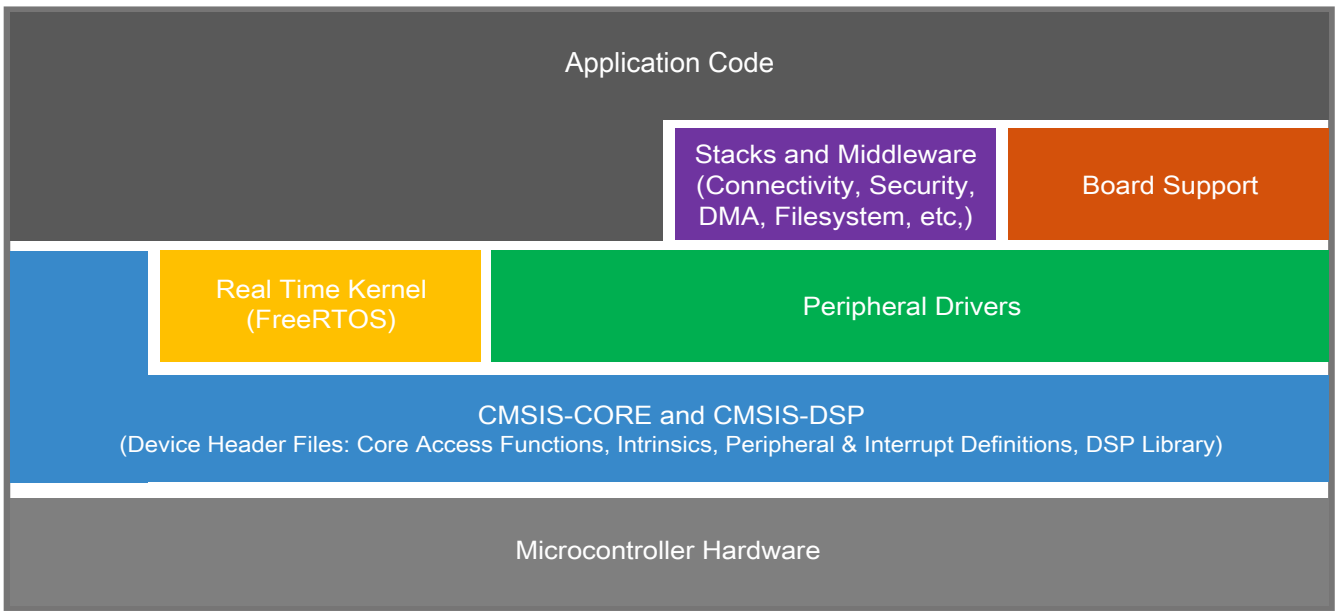

Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support folders

MCUXpresso SDK board support provides example applications for NXP development and evaluation boards for Arm® Cortex®-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <board_name> folder, there are various sub-folders to classify the type of examples they contain. These include (but are not limited to):

- cmsis_driver_examples: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- demo_apps: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications intended to concisely illustrate how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- rtos_examples: Basic FreeRTOS™ OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers

2.1 Example application structure

This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see the *MCUXpresso SDK API Reference Manual* document (MCUXSDKAPIRM).

Each <board_name> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the hello_world example (part of the demo_apps folder), the same general rules apply to any type of example in the <board_name> folder.

In the hello_world application folder you see the following contents:

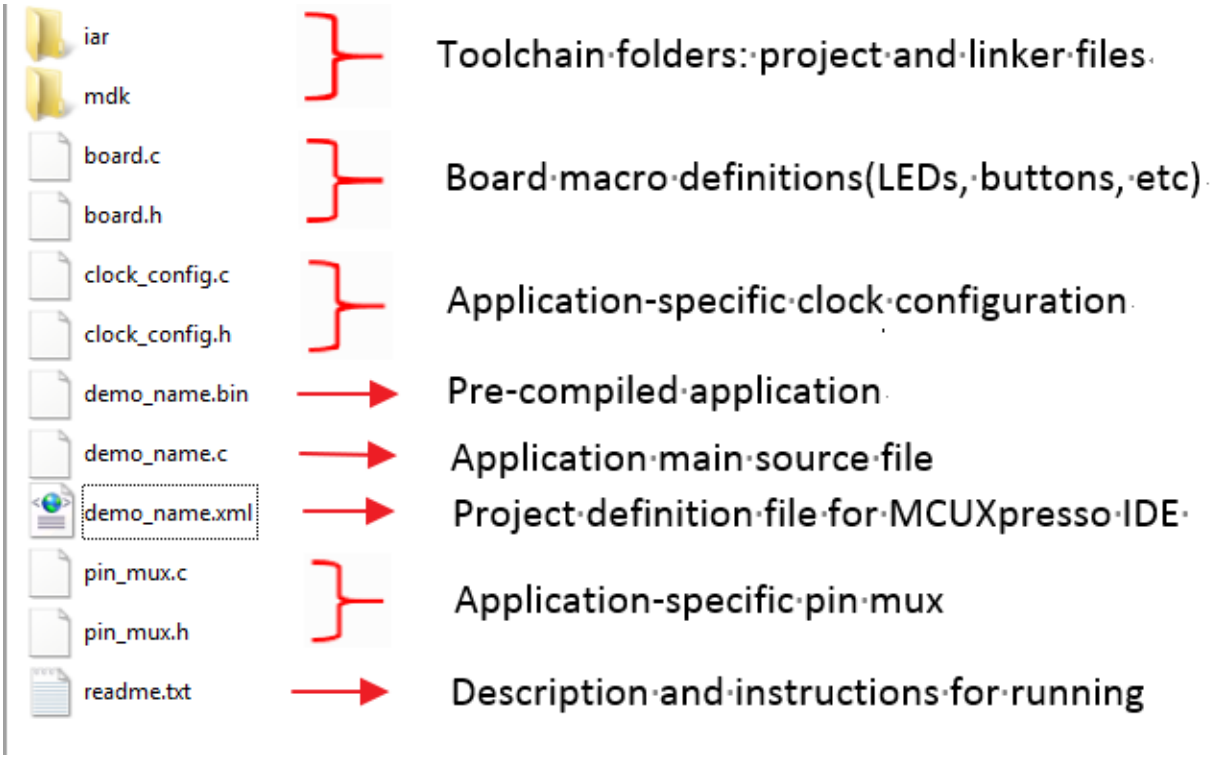

Figure 2. Application folder structure

All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

2.2 Locating example application source files

When opening an example application in any of the supported IDEs, a variety of source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- devices/<device_name>: The device's CMSIS header file, MCUXpresso SDK feature file and a few other things.
- devices/<device_name>/cmsis_drivers: All the CMSIS drivers for your specific MCU.
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU.
- devices/<device_name>/<tool_name>: Toolchain-specific startup code. Vector table definitions are here.
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications.

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the *rtos* folder. Again, the core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

3 Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the LPCXpresso54102 hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

3.1 Build an example application

The following steps helps you build the hello_world example application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/iar

   Using the LPCXpresso54102 hardware platform as an example, the hello_world workspace is located in

   <install_dir>/boards/lpcxpresso54102/demo_apps/hello_world/iar/hello_world.eww

   Other example applications may have additional folders in their path.
2. Select the desired build target from the drop-down menu. For this example, select the “hello_world – debug” target.

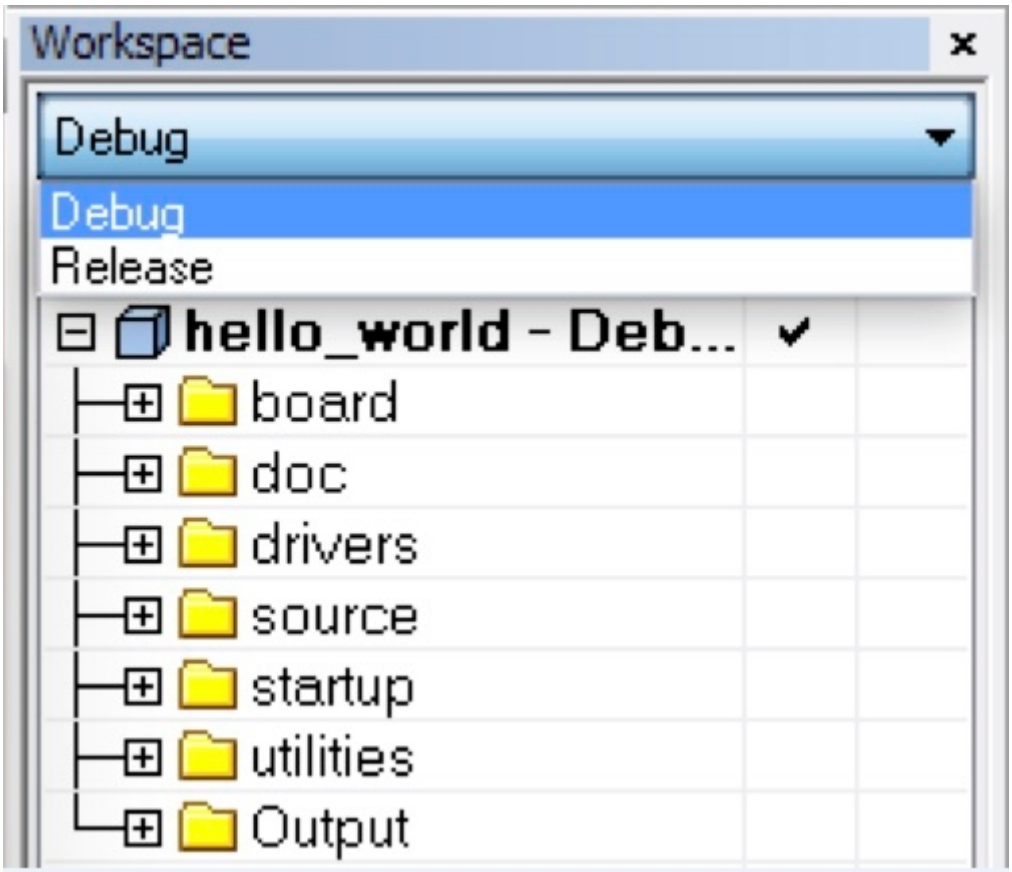

Figure 3. Demo build target selection

3. To build the demo application, click the “Make” button highlighted in red below.

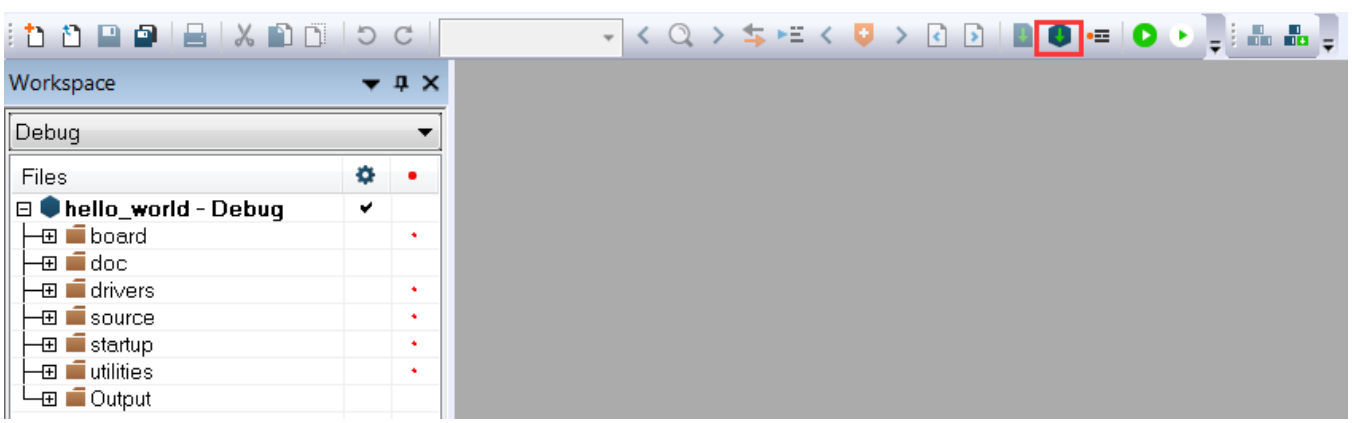

Figure 4. Build the demo application

4. The build completes without errors.

3.2 Run an example application

To download and run the application, perform these steps:

1. Download and install LPCScrypt or the Windows® operating systems driver for LPCXpresso boards from www.nxp.com/lpcutilities. This installs required drivers for the board.
2. Connect the development platform to your PC via USB cable between the Link2 USB connector (named Link for some boards) and the PC USB connector. If connecting for the first time, allow about 30 seconds for the devices to enumerate.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

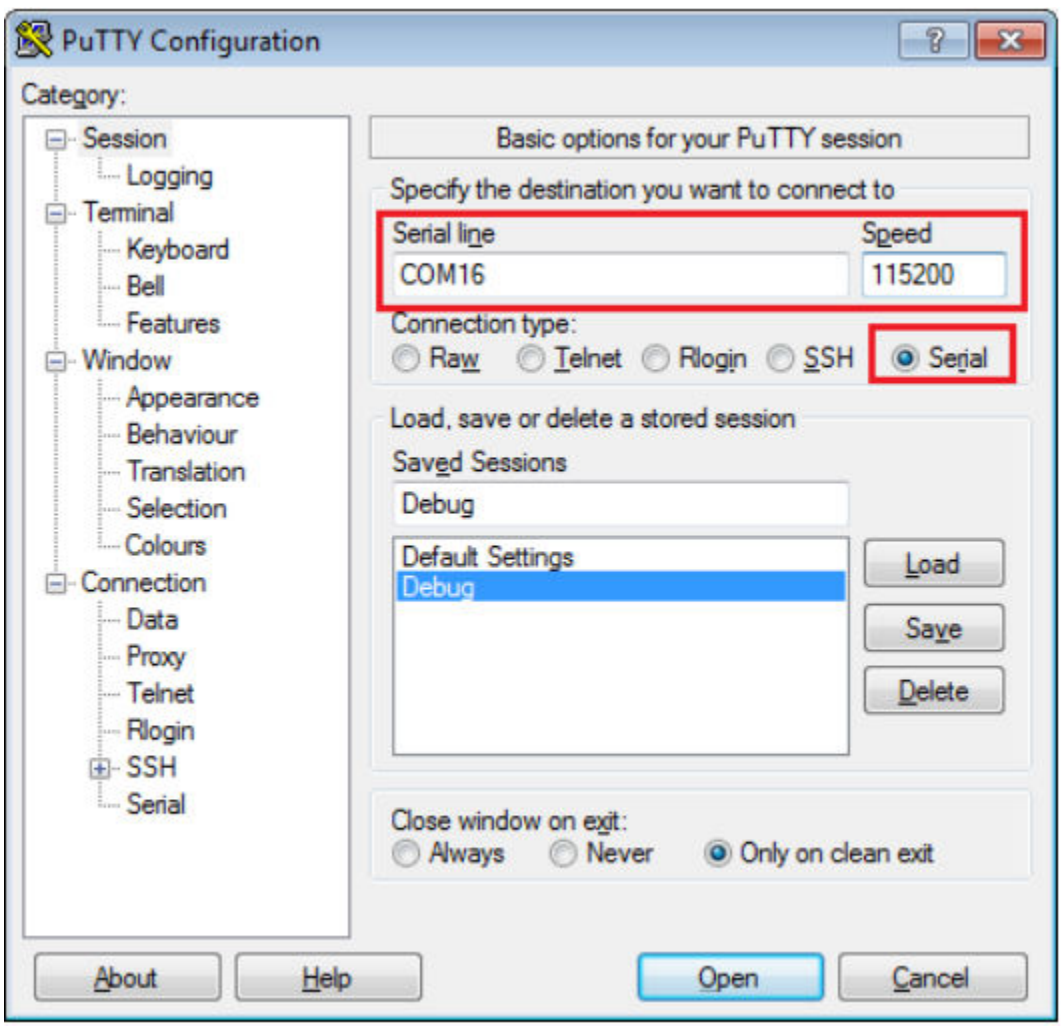

Figure 5. Terminal (PuTTY) configuration

4. In IAR, click the "Download and Debug" button to download the application to the target.

Figure 6. Download and Debug button

5. The application is then downloaded to the target and automatically runs to the main() function.

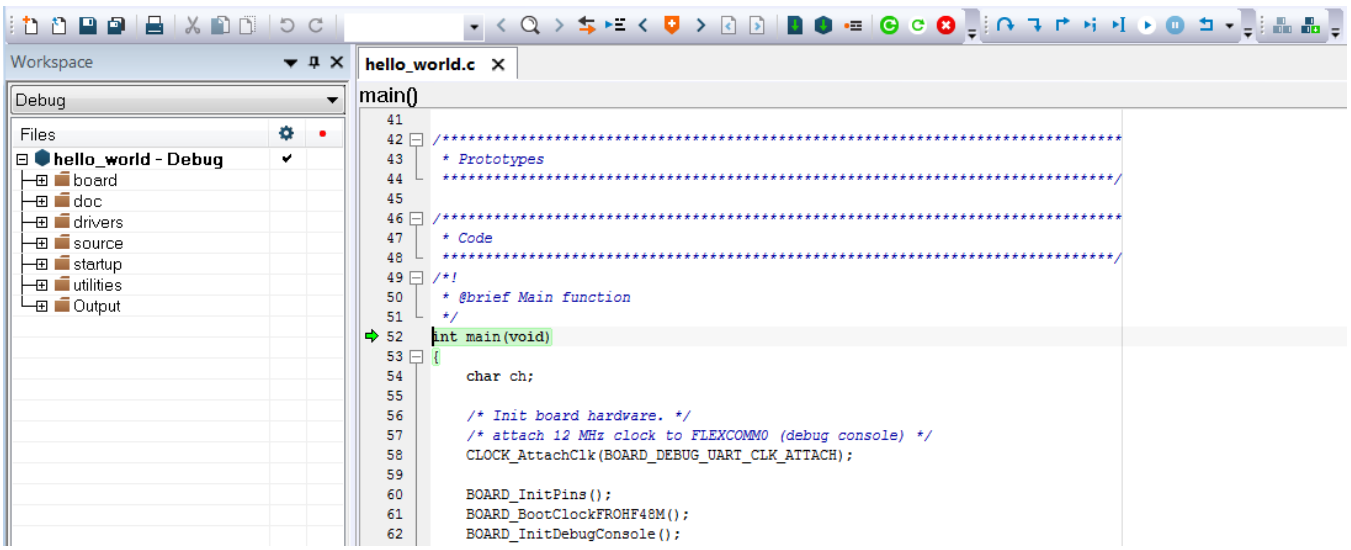

Figure 7. Stop at main() when running debugging

6. Run the code by clicking the "Go" button to start the application.

Figure 8. Go button

7. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.

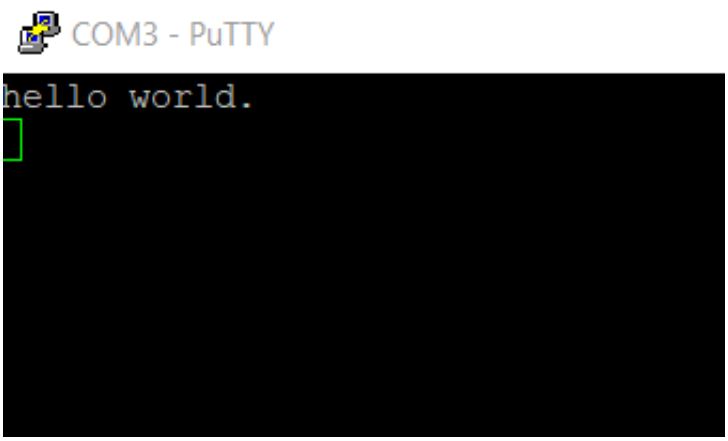

Figure 9. Text display of the hello_world demo

3.3 Build a multicore example application

This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/iar

Begin with a simple dual-core version of the Hello World application. The multicore Hello World IAR workspaces are located in this folder:

<install_dir>/boards/lpcxpresso54102/multicore_examples/hello_world/cm0plus/iar/hello_world_cm0plus.eww

<install_dir>/boards/lpcxpresso54102/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww

Build both applications separately by clicking the “Make” button. It is requested to build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

3.4 Run a multicore example application

The primary core debugger handles flashing both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 4 as described in*Run an example application*. These steps are common for both single core and dual-core applications in IAR.

After clicking the “Download and Debug" button, the auxiliary core project is opened in the separate EWARM instance. Both the primary and auxiliary image are loaded into the device flash memory and the primary core application is executed. It stops at the default C language entry point in the *main()* function.

Run both cores by clicking the "Start all cores" button to start the multicore application.

Figure 10. Start all cores button

During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check the terminal settings and connections.

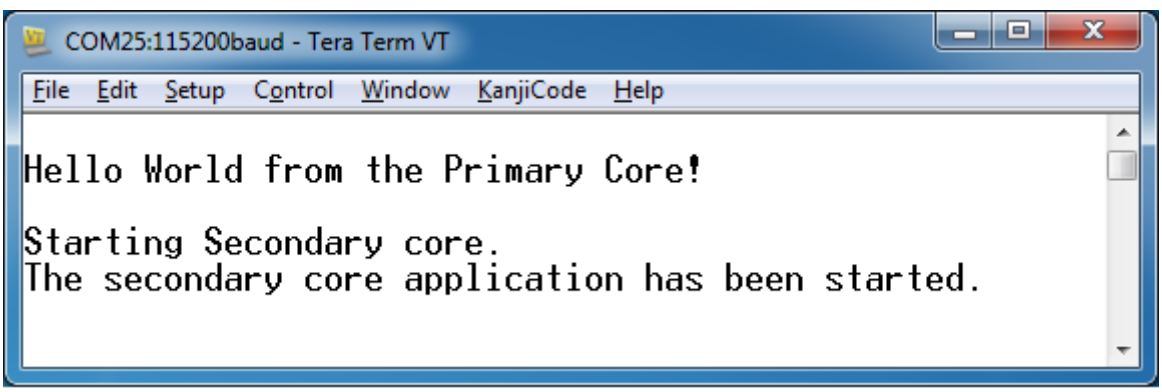

Figure 11. Hello World from primary core message

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and is running correctly. When both cores are running, use the "Stop all cores" and "Start all cores" control buttons to stop or run both cores simultaneously.

Figure 12. "Stop all cores" and "Start all cores" control buttons

4 Run a demo using Keil® MDK/µVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the LPCXpreeso54102 hardware platform is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

4.1 Install CMSIS device pack

After the MDK tools are installed, Cortex® Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions, and flash programming algorithms. Follow these steps to install the appropriate CMSIS pack.

1. Open the MDK IDE, which is called µVision. In the IDE, select the “Pack Installer” icon.

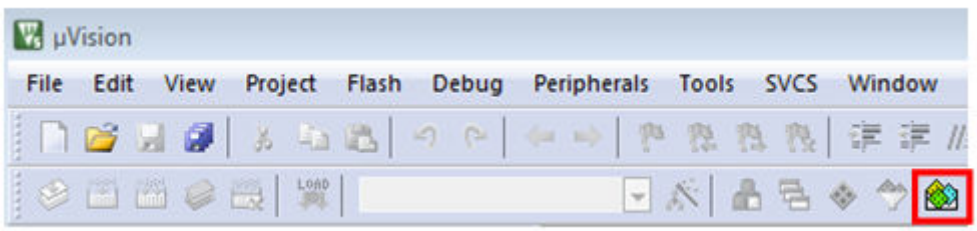

Figure 13. Launch the Pack Installer

2. After the installation finishes, close the Pack Installer window and return to the µVision IDE.

4.2 Build an example application

- Open the desired example application workspace in: <install_dir>/boards/<board_name>/<*example_type*>/<application_name>/mdk

  The workspace file is named <demo_name>.uvmpw, so for this specific example, the actual path is:

  <install_dir>/boards/lpcxpresso54102/demo_apps/hello_world/mdk/hello_world.uvmpw
- To build the demo project, select the "Rebuild" button, highlighted in red.

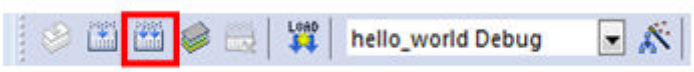

Figure 14. Build the demo

- The build completes without errors.

4.3 Run an example application

To download and run the application, perform these steps:

1. Download and install LPCScrypt or the Windows driver for LPCXpresso boards from www.nxp.com/lpcutilities . This installs required drivers for the board.
2. Connect the development platform to your PC via USB cable between the Link2 USB connector and the PC USB connector. If connecting for the first time, allow about 30 seconds for the devices to enumerate See Section 8.2 to update the debug probe flash using LPCScrypt.

3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

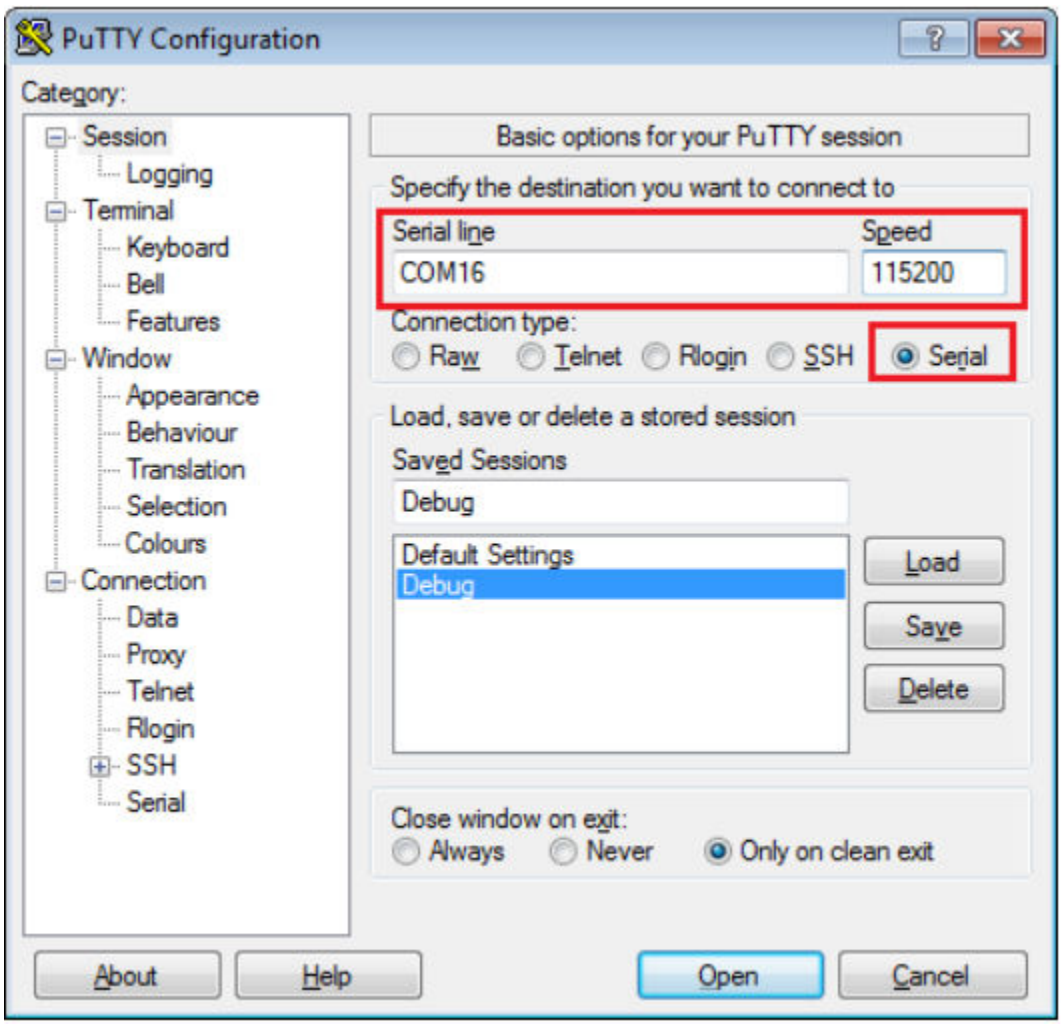

Figure 15. Terminal (PuTTY) configurations

4. In µVision, after the application is properly built, click the "Download" button to download the application to the target.

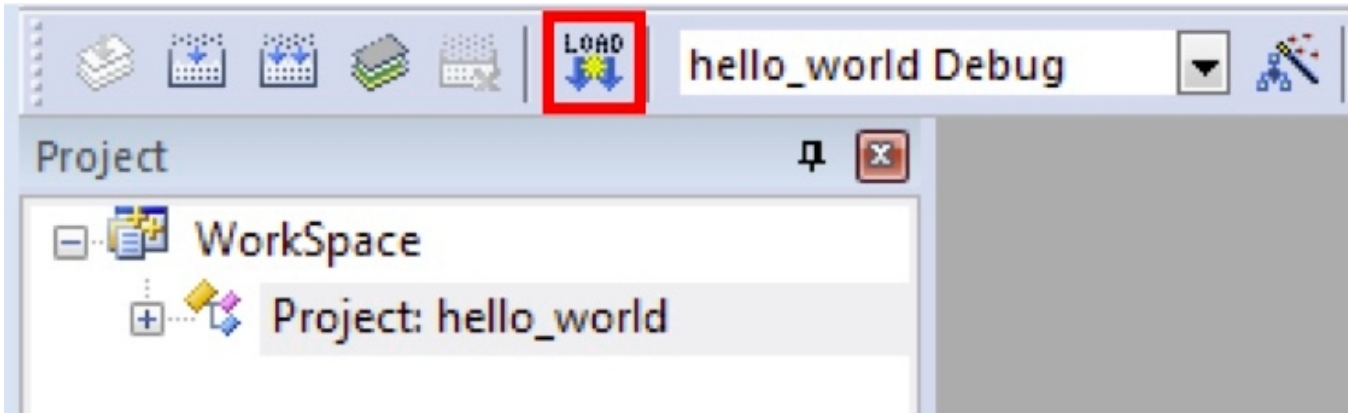

Figure 16. Download button

5. After clicking the “Download” button, the application downloads to the target and should be running. To debug the application, click the “Start/Stop Debug Session” button, highlighted in red.

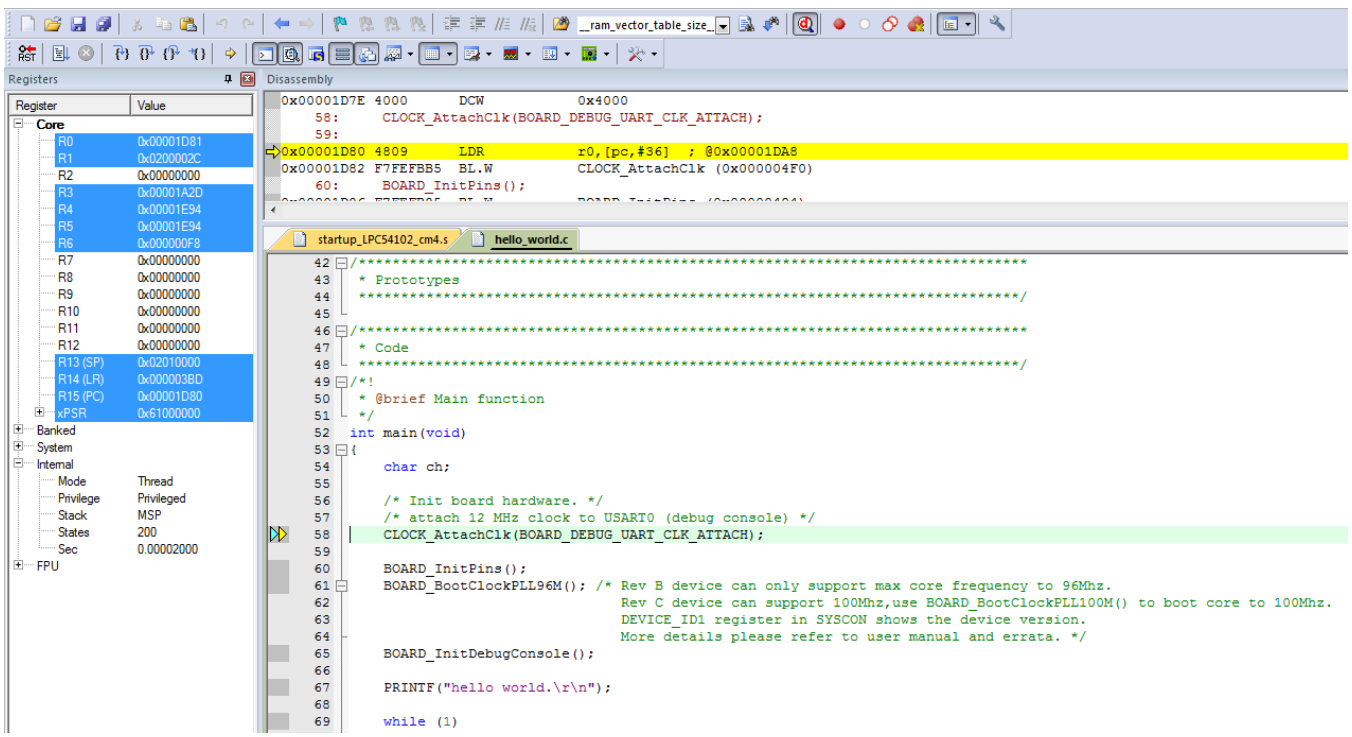

Figure 17. Stop at main() when run debugging

6. Run the code by clicking the “Run” button to start the application.

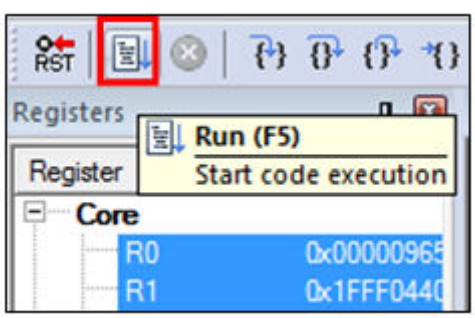

Figure 18. Go button

The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.

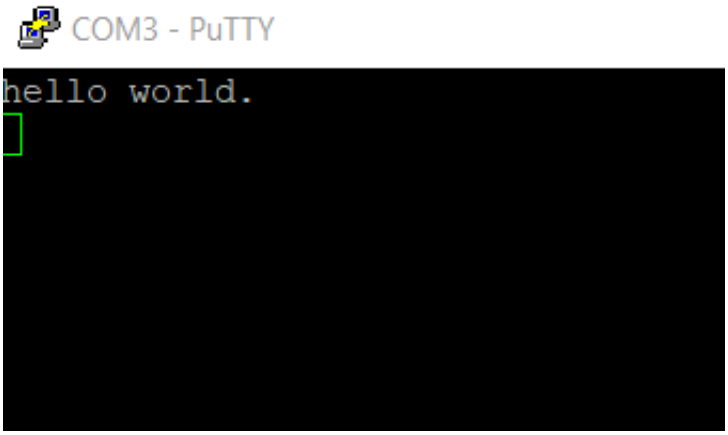

Figure 19. Text display of the hello_world demo

4.4 Build a multicore example application

This section describes the steps to build and run a dual-core application. The demo applications workspace files are located in this folder:

<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/mdk

Begin with a simple dual-core version of the Hello World application. The multicore Hello World Keil MSDK/µVision® workspaces are located in this folder:

<install_dir>/boards/pcxpresso54102/multicore_examples/hello_world/cm0plus/mdk/hello_world_cm0plus.uvmpw

<install_dir>/boards/lpcxpresso54102/multicore_examples/hello_world/cm4/mdk/hello_world_cm4.uvmpw

Build both applications separately by clicking the “Rebuild” button. Build the application for the auxiliary core (cm0plus) first because the primary core application project (cm4) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

4.5 Run a multicore example application

The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in *Run an example application*. These steps are common for both single core and dual-core applications in µVision.

Both the primary and the auxiliary image is loaded into the device flash memory. After clicking the “Run" button, the primary core application is executed. During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this does not appear, check your terminal settings and connections.

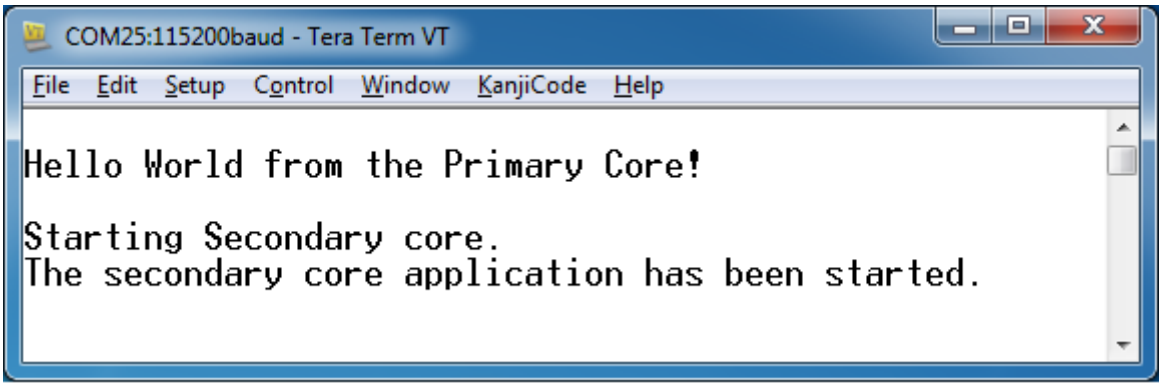

Figure 20. Hello World from primary core message

An LED controlled by the auxiliary core starts flashing indicating that the auxiliary core has been released from the reset and is running correctly.

Attach the running application of the auxiliary core by opening the auxiliary core project in the second µVision instance and clicking the "Start/Stop Debug Session" button. After this, the second debug session is opened and the auxiliary core application can be debugged.

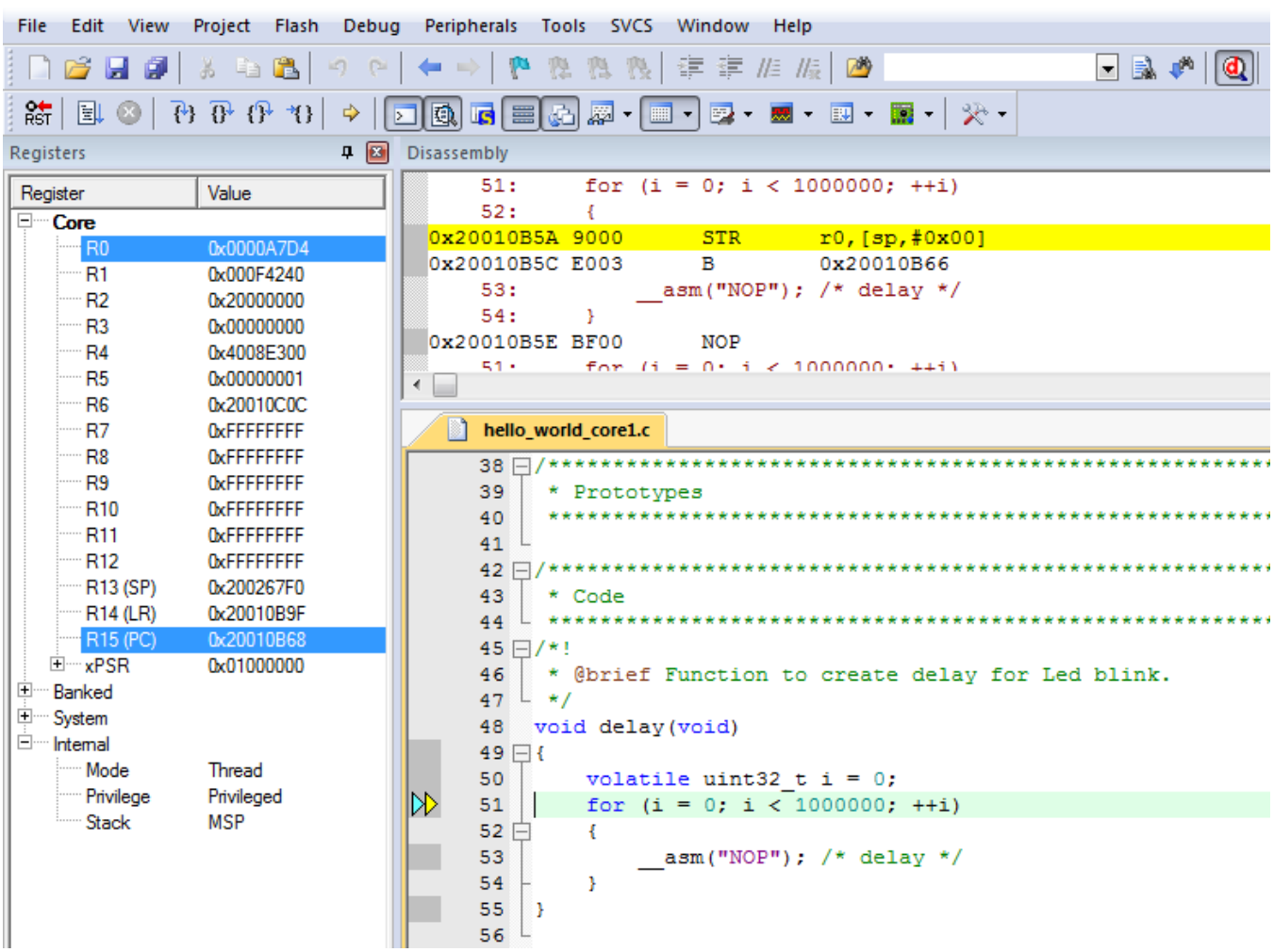

Figure 21. Debugging the auxiliary core application

5 Run a demo using MCUXpresso IDE

NOTE

Ensure that the MCUXpresso IDE toolchain is included when generating the MCUXpresso SDK Package.

This section describes the steps required to configure MCUXpresso IDE to build, run, and debug example applications. The hello_world demo application targeted for the LPCXpresso54102 platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

5.1 Select the workspace location

Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be outside of the MCUXpresso SDK tree.

5.2 Build an example application

To build an example application, follow these steps.

1. Drag and drop the SDK zip file into the “Installed SDKs” view to install an SDK. In the window that appears, click the “OK” button and wait until the import has finished.

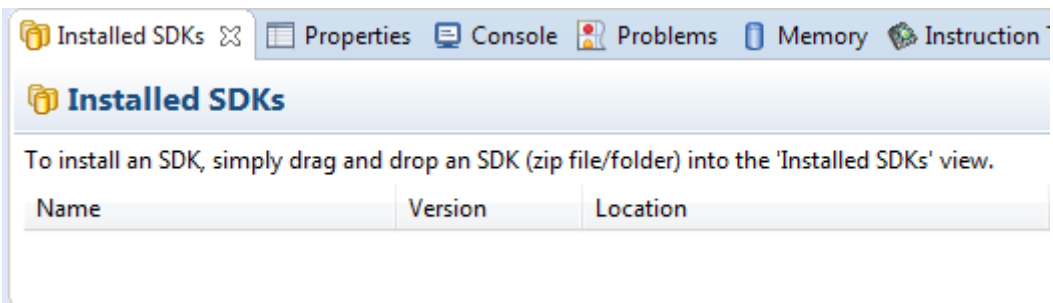

Figure 22. Install an SDK

2. On the *Quickstart Panel*, click “Import SDK example(s)…”.

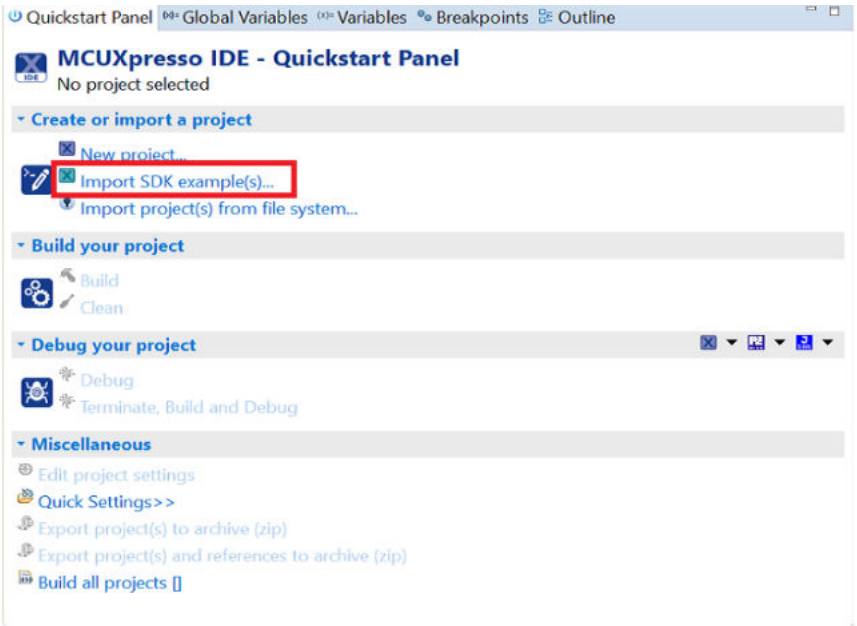

Figure 23. Import an SDK example

3. In the window that appears, expand the “LPC5410x” folder and select “LPC54102J512”. Then, select "lpcxpresso54102” and click the “Next” button.

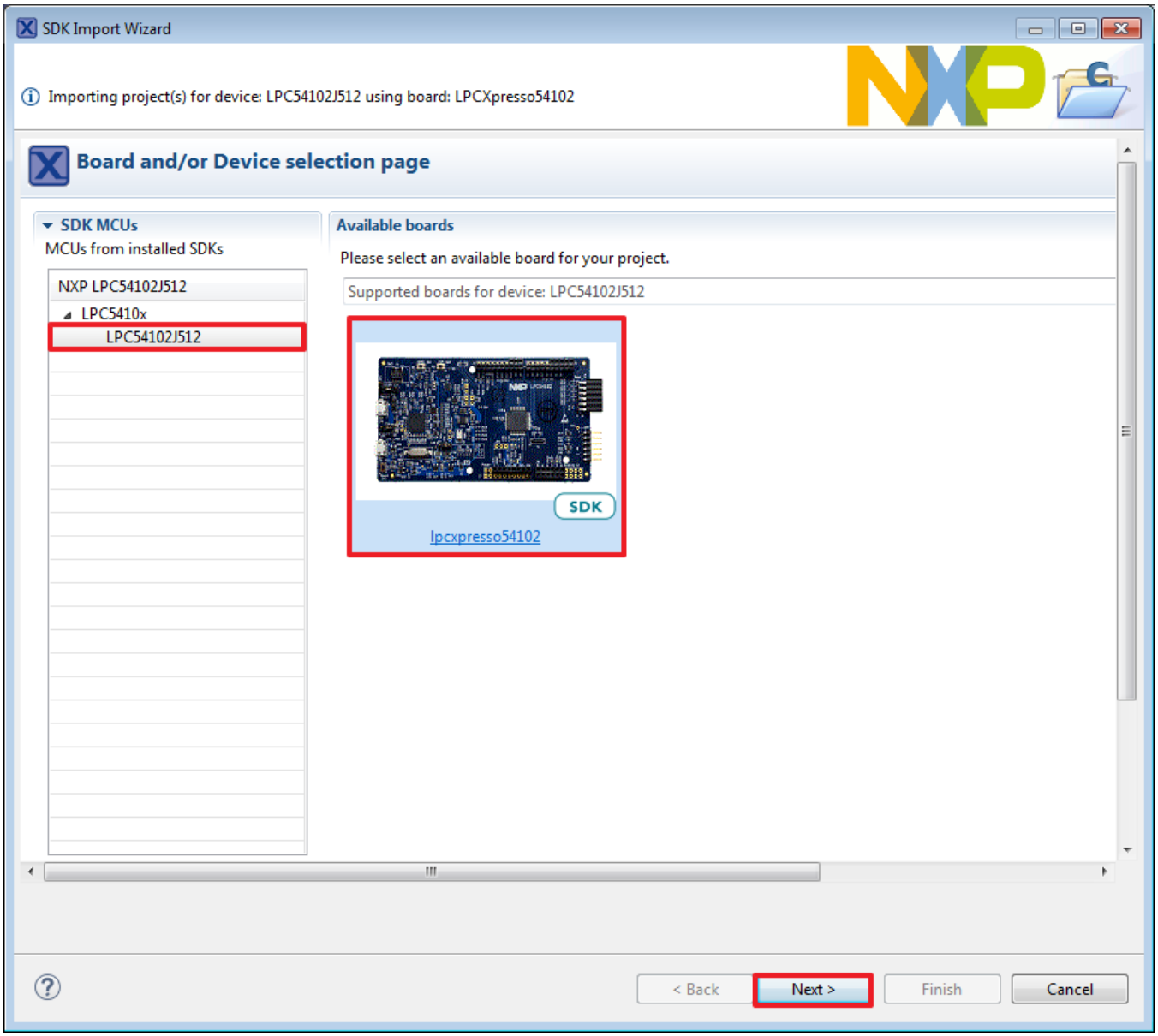

Figure 24. Select LPCXpresso54102 board

4. Expand the “demo_apps” folder and select “hello_world”. Then, click the "Next” button.

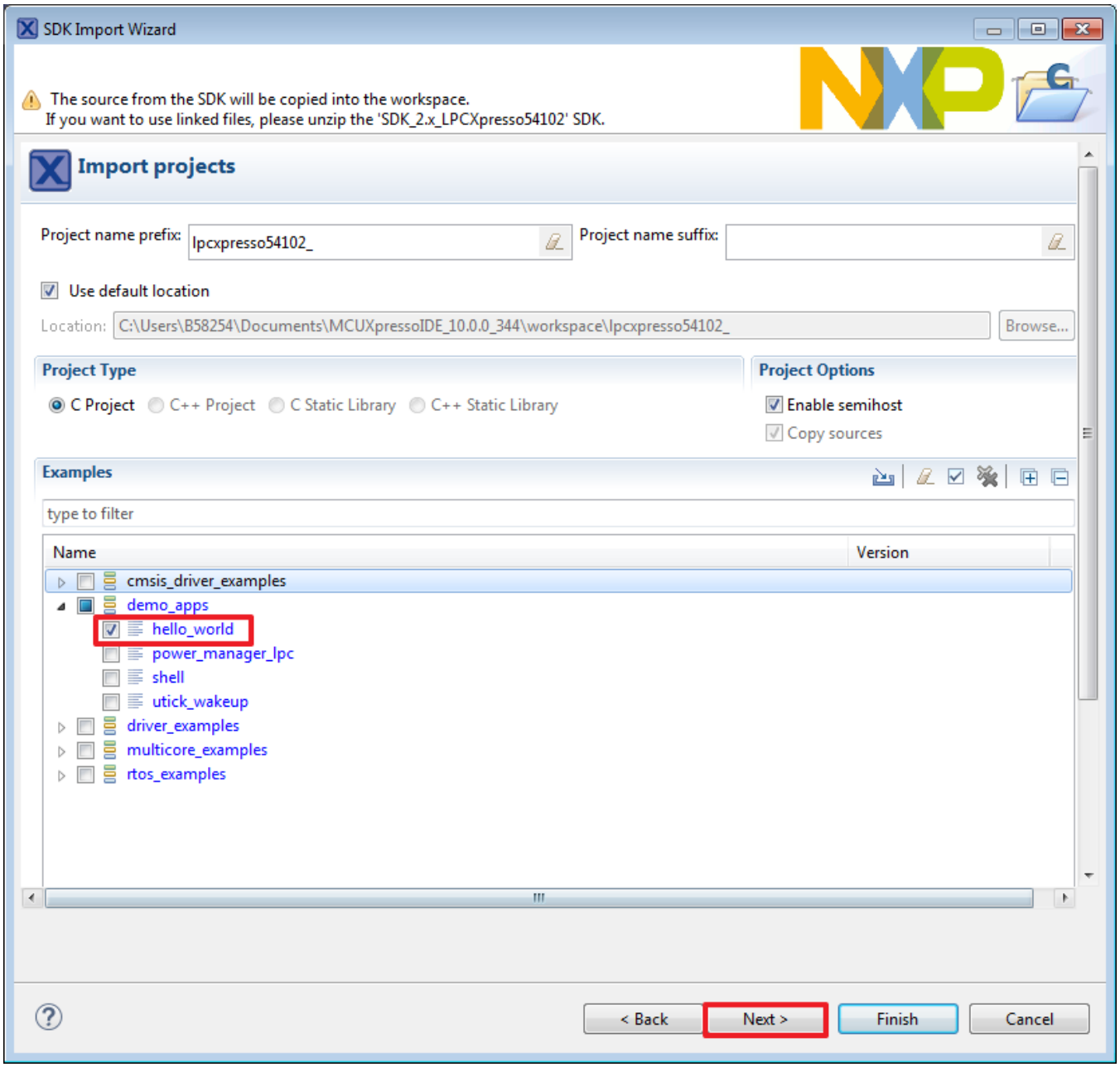

Figure 25. Select "hello_world"

5. Ensure the option “Redlib: Use floating point version of printf” is selected if the cases print floating point numbers on the terminal (for demo applications such as dac32_adc12, dac_adc, dac_cadc, ecompass, sai, coremark, mbedtls_benchmark, wolfssl_benchmark, and for mmcau_examples such as mmcau_api). Otherwise, there is no need to select it. Click the “Finish” button.

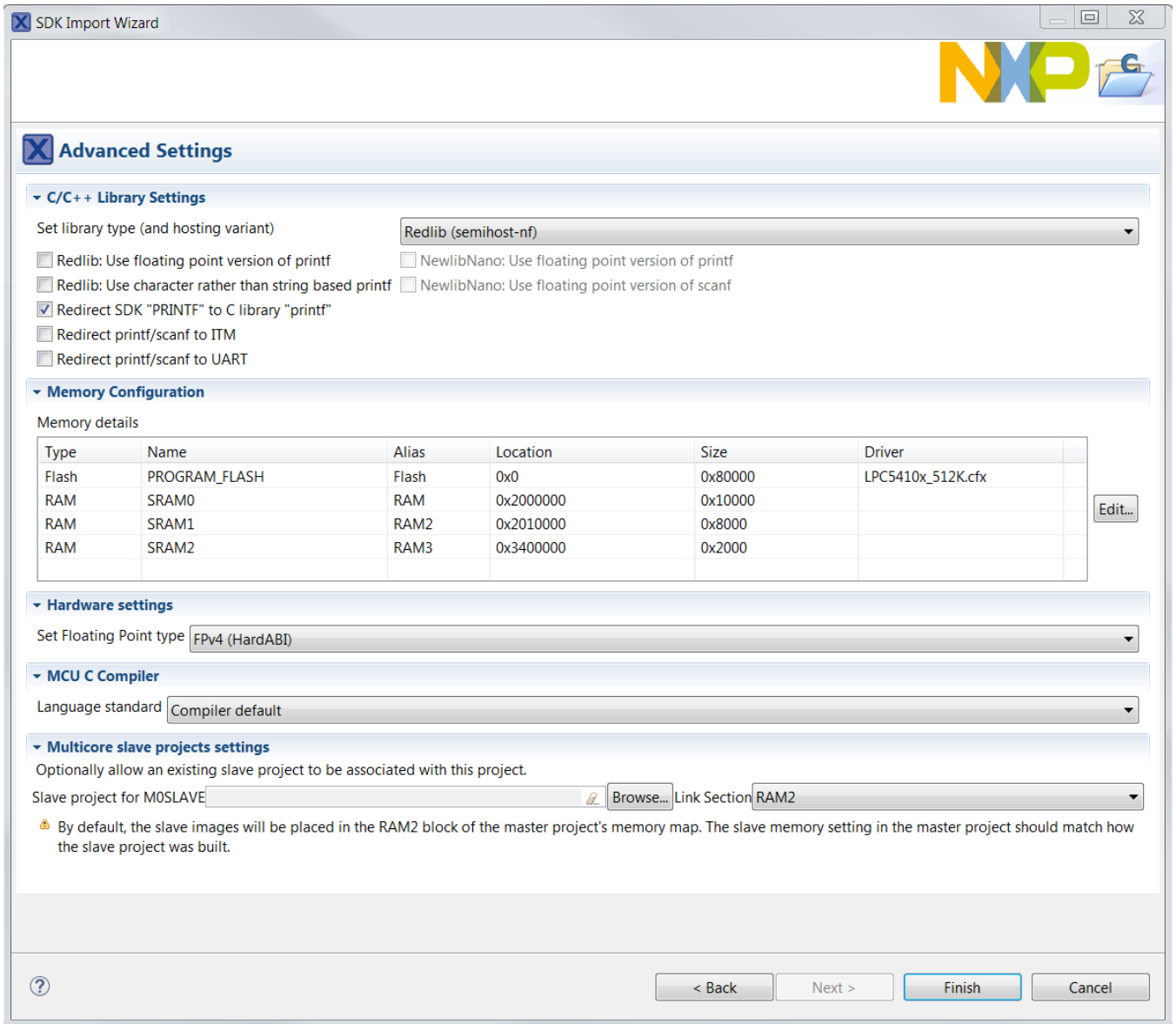

Figure 26. Select "User floating print version of printf"

5.3 Run an example application

For more information on debug probe support in the MCUXpresso IDE v11.0.1, visit community.nxp.com.

To download and run the application, perform these steps:

1. On the *Quickstart Panel*, click on "Debug 'lpcxpresso54102_demo_apps_hello_world' [Debug]".

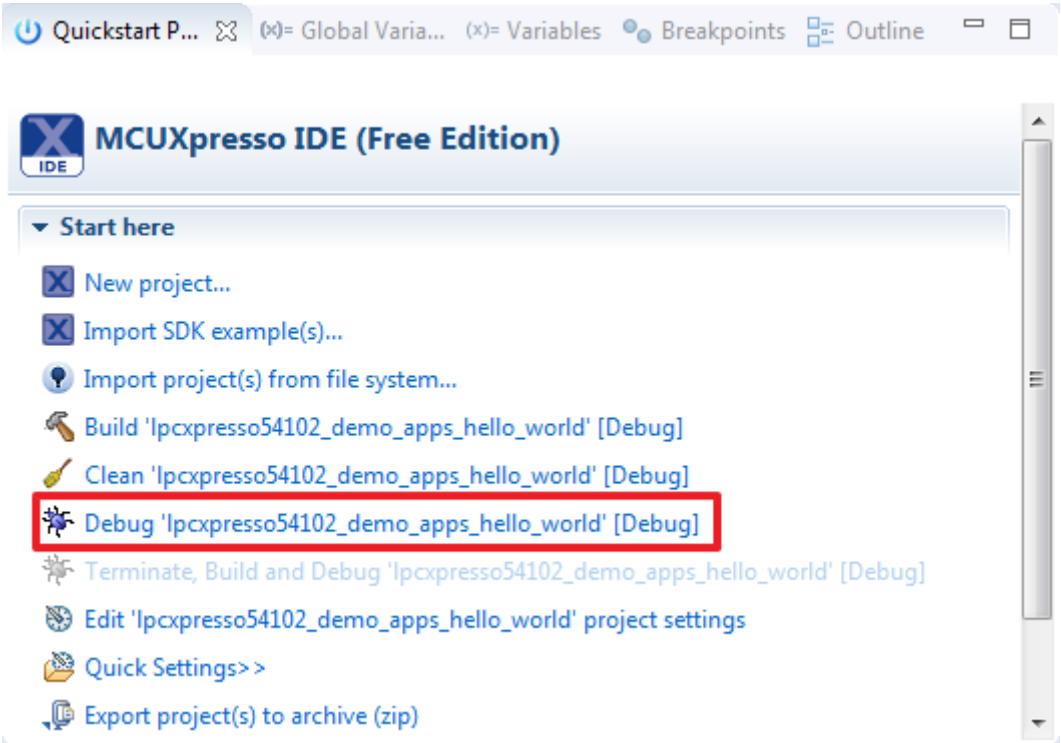

Figure 27. Debug "hello_world" case

2. The first time you debug a project, the Debug Emulator Selection Dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click the “OK” button. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)

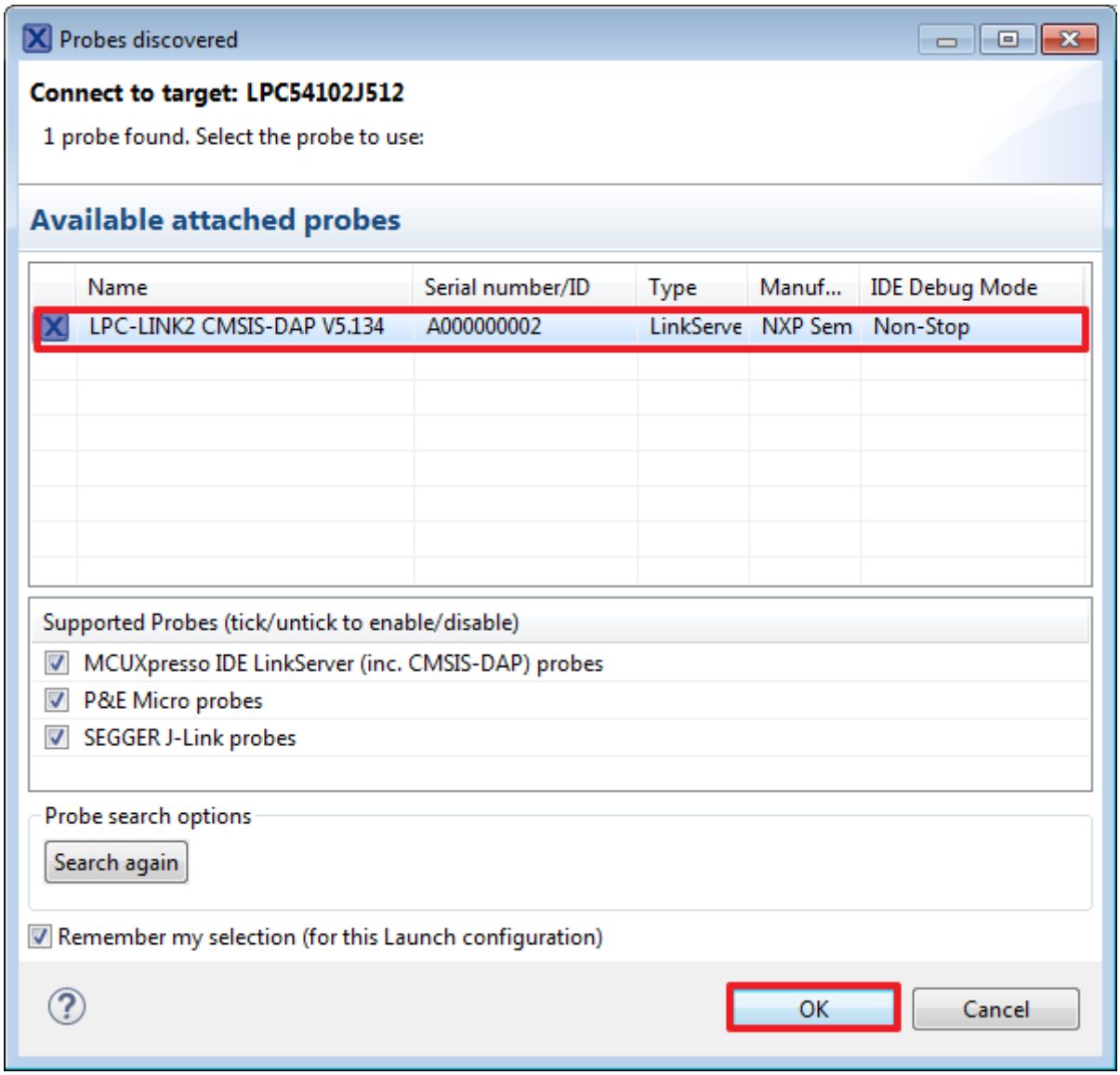

Figure 28. Attached Probes: debug emulator selection

3. The application is downloaded to the target and automatically runs to main():

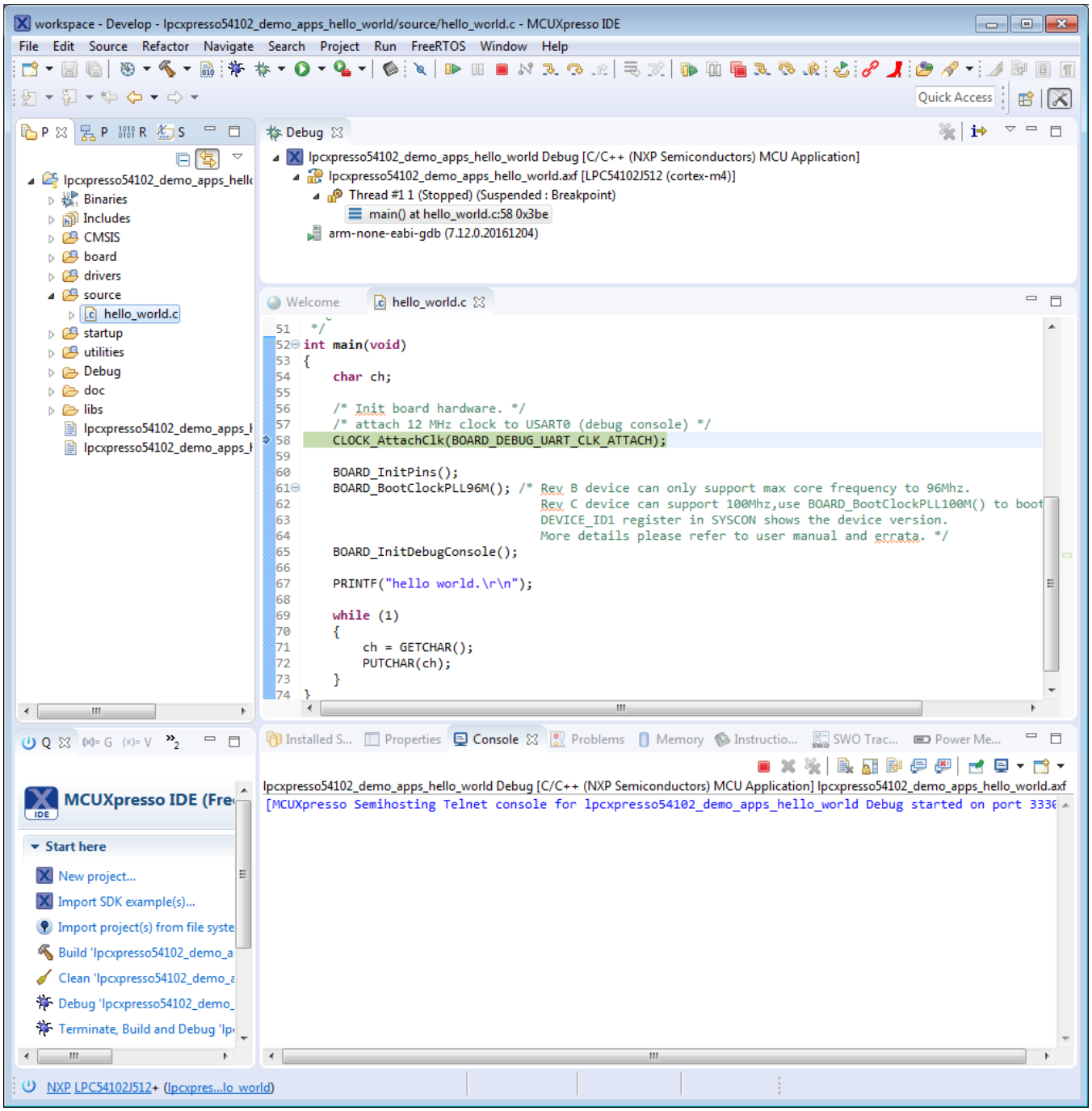

Figure 29. Stop at main() when running debugging

4. Start the application by clicking the "Resume" button.

Figure 30. Resume button

The hello_world application is now running and a banner is displayed on the MCUXpresso IDE console window. If this is not the case, check your terminal settings and connections.

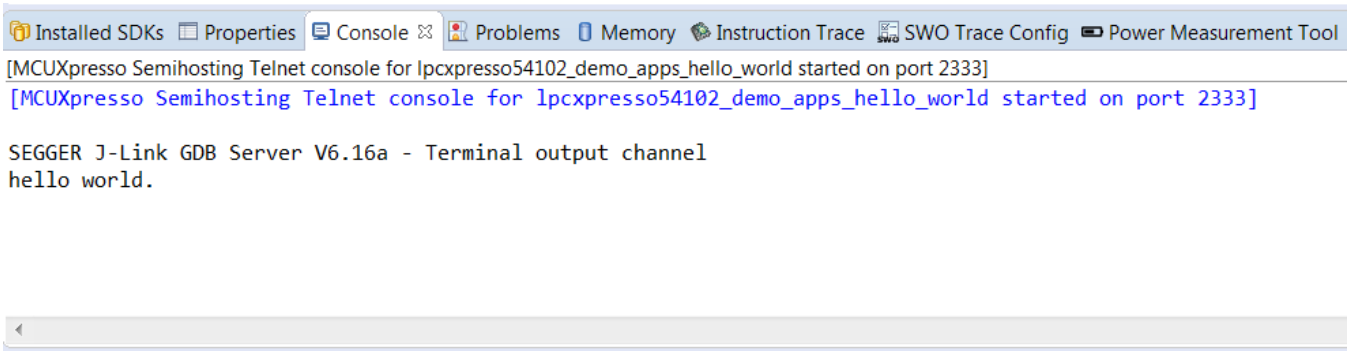

Figure 31. Text display of the hello_world demo

5.4 Build a multicore example application

This section describes the steps required to configure MCUXpresso IDE v11.0.1 to build, run, and debug multicore example applications. The dual-core version of hello_world example application targeted for the LPCXpresso54102 hardware platform is used as an example, though these steps can be applied to any multicore example application in the MCUXpresso SDK

1. Multicore examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for LPCXpresso54102 is installed and available in the “Installed SDKs” view, click “Import SDK example(s) …” on the Quickstart Panel. In the window that appears, expand the “LPCxx” folder and select “LPC54102J512”. Then, select “lpcxpresso54102” and click the “Next” button.

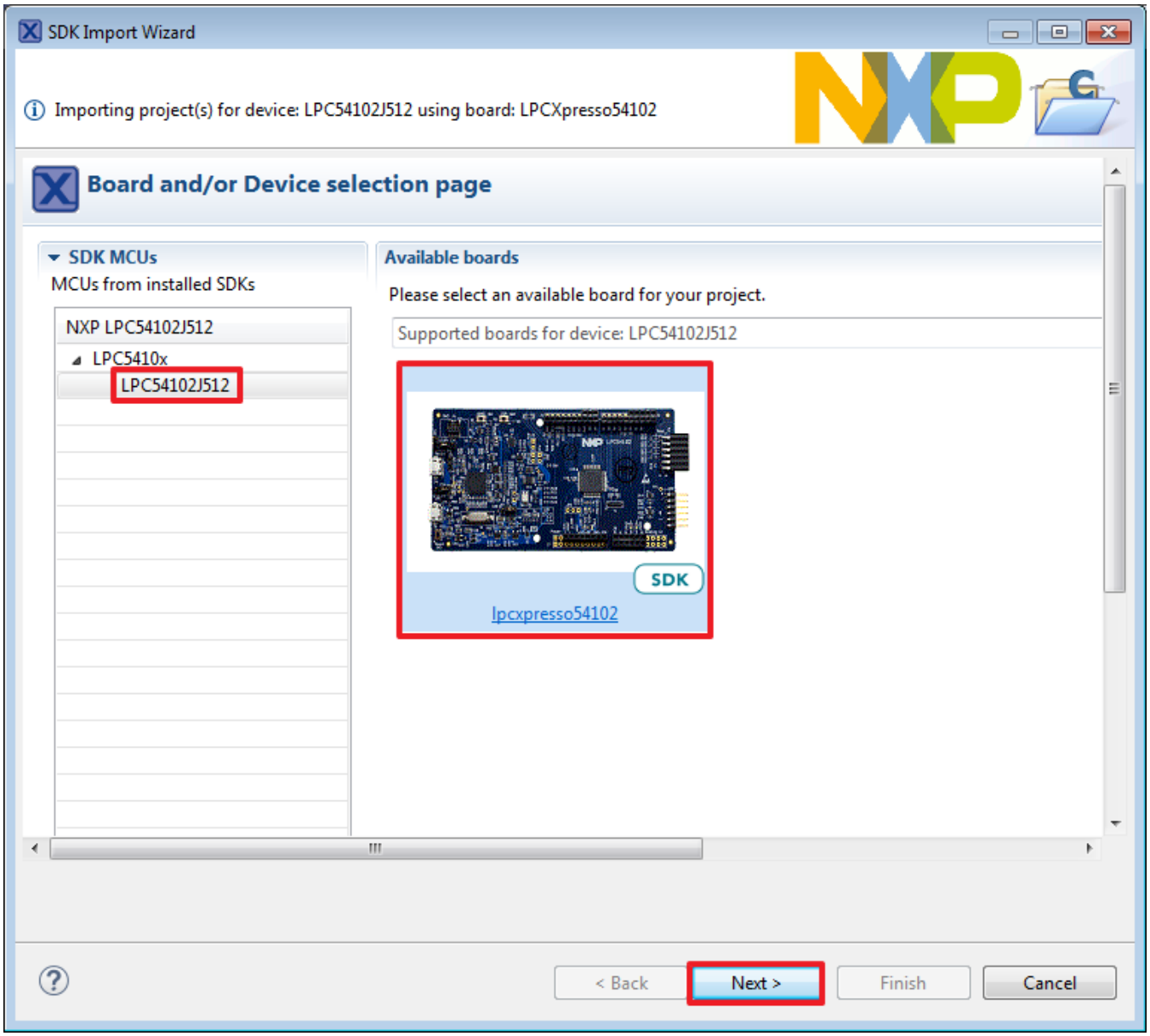

Figure 32. Select the LPCXpresso54102 board

2. Expand the “multicore_examples/hello_world” folder and select “cm4”. Because multicore examples are linked together, the cm0plus counterpart project is automatically imported with the cm4 project, and there is no need to select it explicitly. Click the “Finish” button.

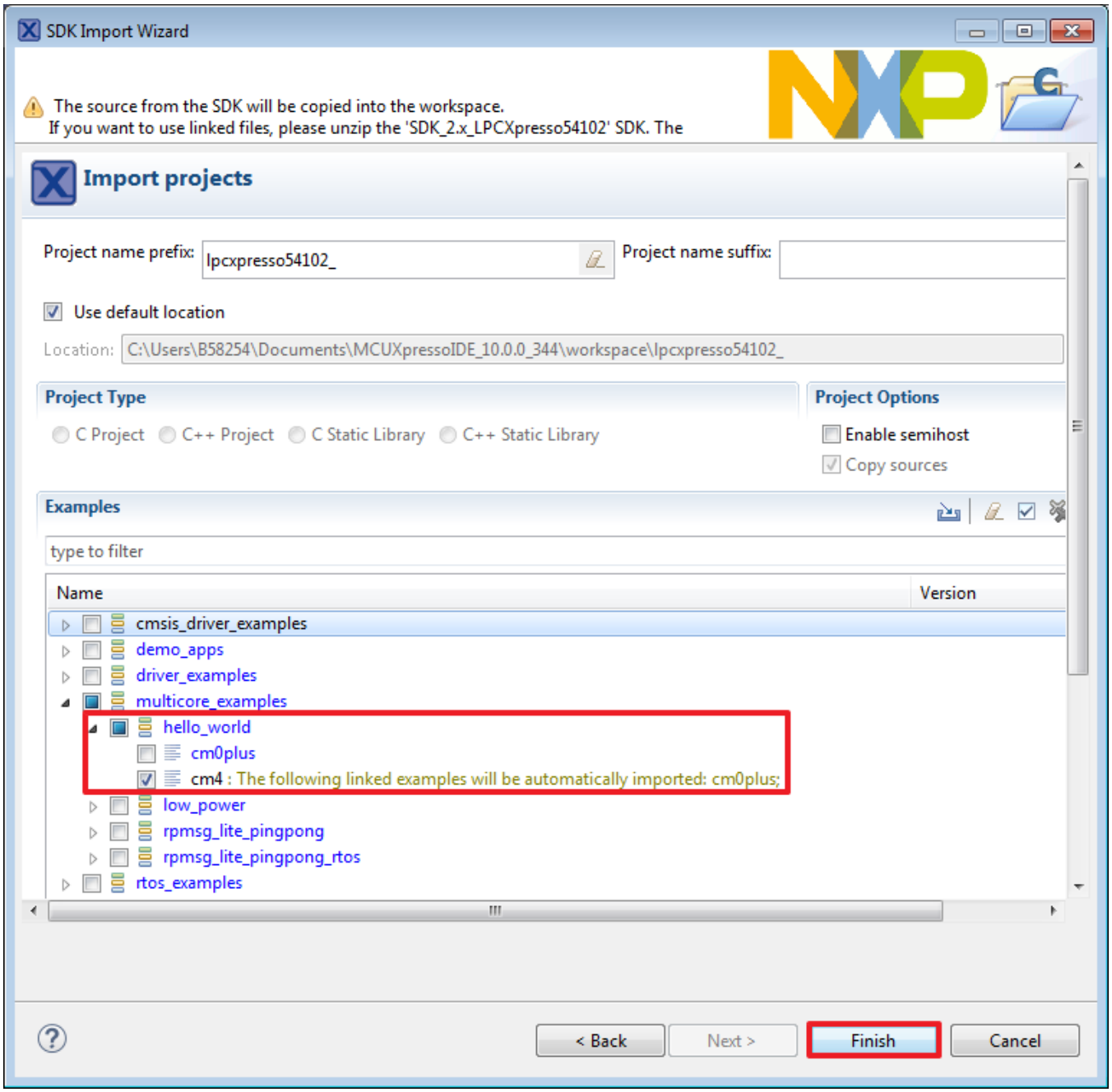

Figure 33. Select the hello_world multicore example

3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the lpcxpresso54102_multicore_examples_hello_world_cm4 project (multicore master project) in the Project Explorer, then choose the appropriate build target, "Debug" or "Release", by clicking the downward facing arrow next to the hammer icon, as shown below. For this example, select the "Debug" target.

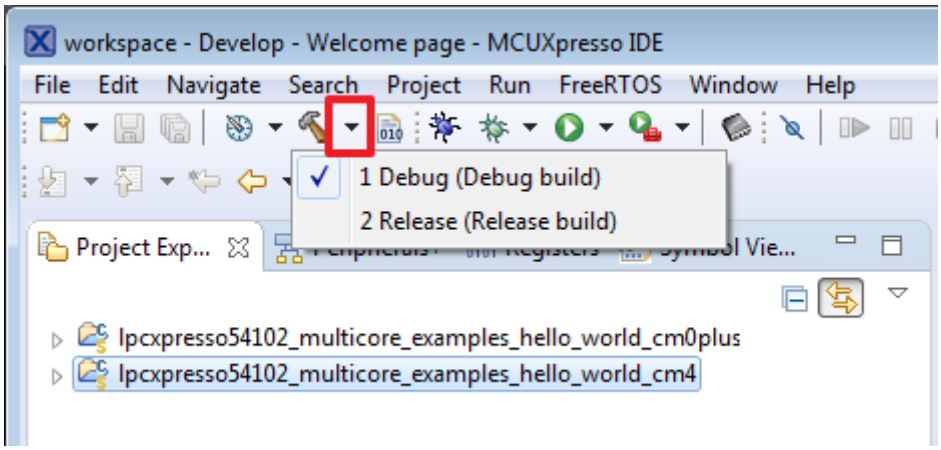

Figure 34. Selection of the build target in MCUXpresso IDE

The project starts building after the build target is selected. Because of the project reference settings in multicore projects, triggering the build of the primary core application (cm4) causes the referenced auxiliary core application (cm0plus) to build as well.

5.5 Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in Section 5.3, "Run an example application". These steps are common for both single core applications and the primary side of dual-core applications, ensuring both sides of the multicore application are properly loaded and started. However, there is one additional dialogue that is specific to multicore examples, and requires selecting the target core. See the following figures as reference.

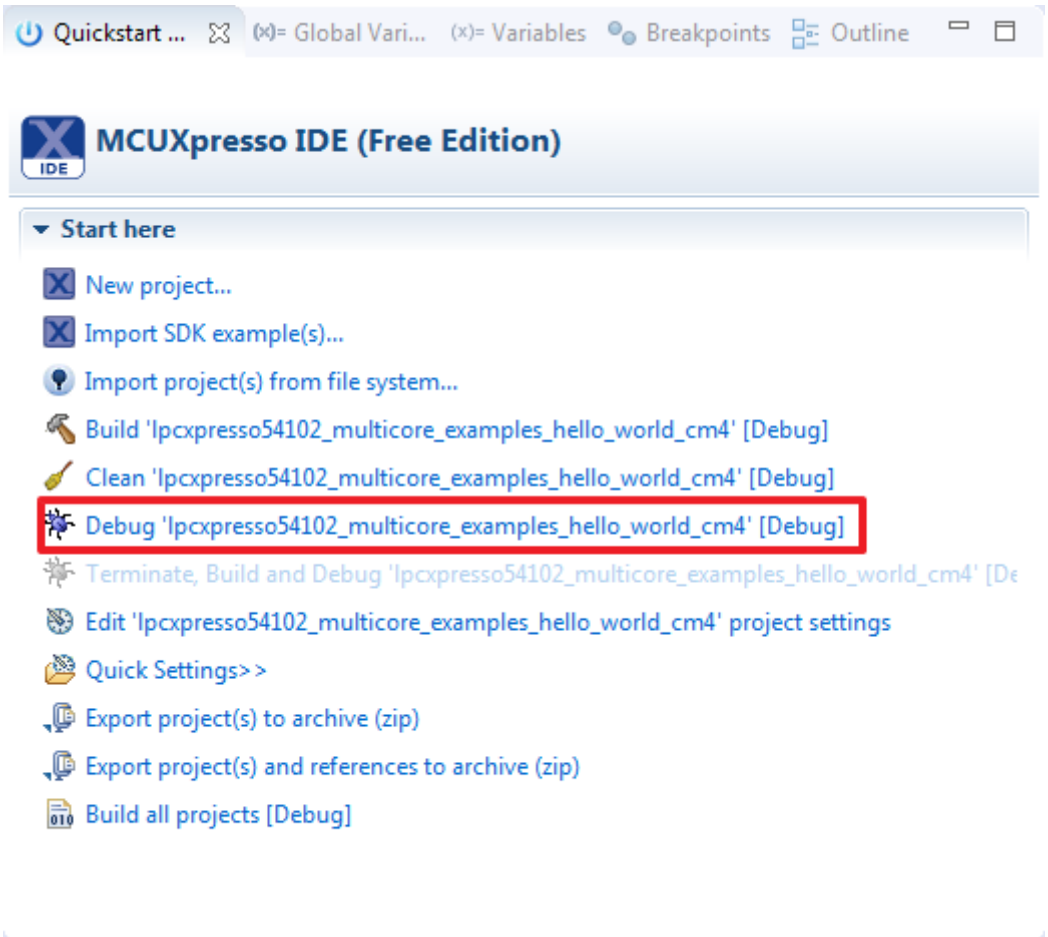

Figure 35. Debug "lpcxpresso54102_multicore_examples_hello_world_cm4" case

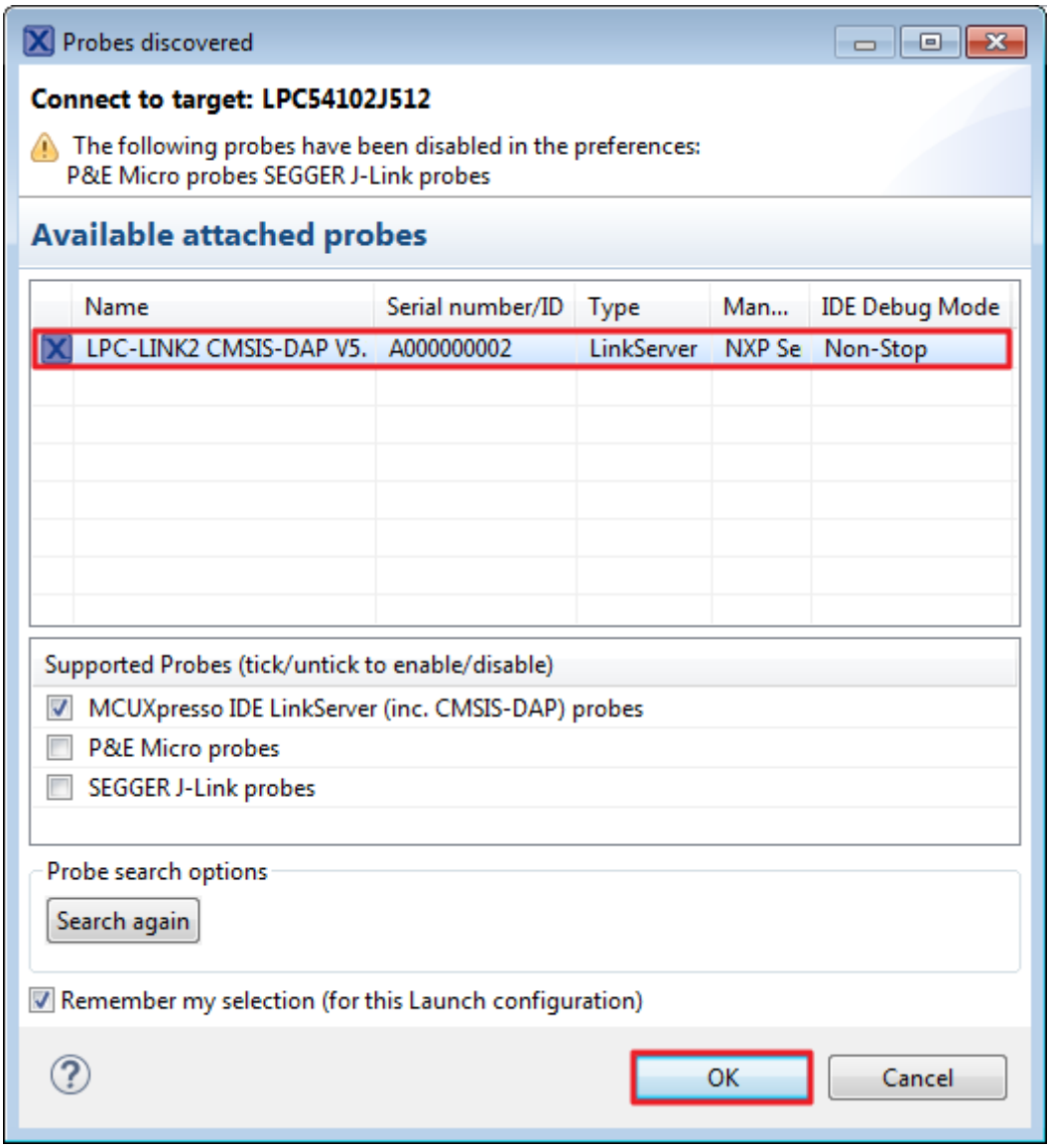

Figure 36. Attached Probes: debug emulator selection

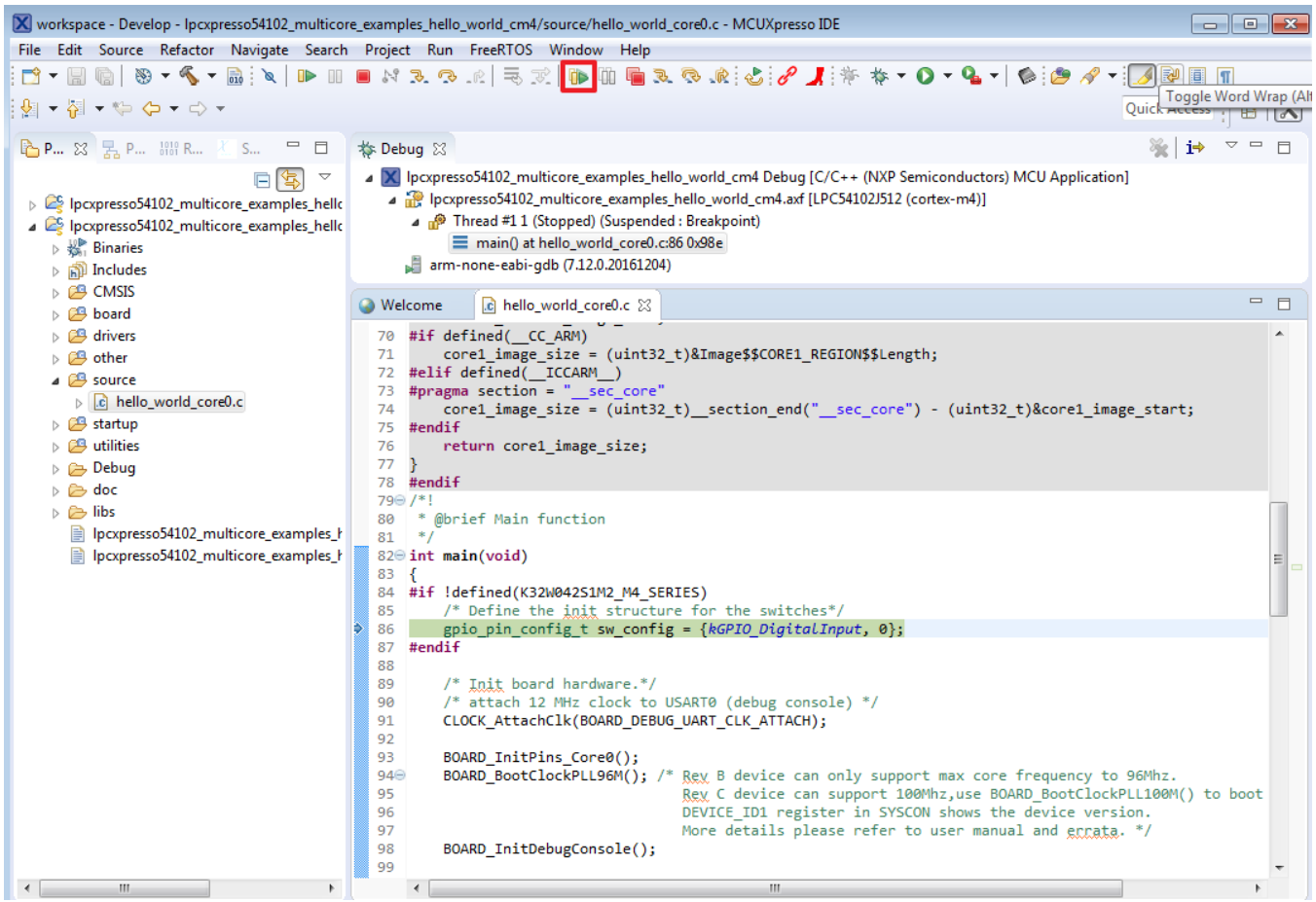

Figure 37. Stop the primary core application at main() when running debugging

After clicking the "Resume All Debug sessions" button, the hello_world multicore application runs and a banner is displayed on the terminal. If this is not the case, check your terminal settings and connections.

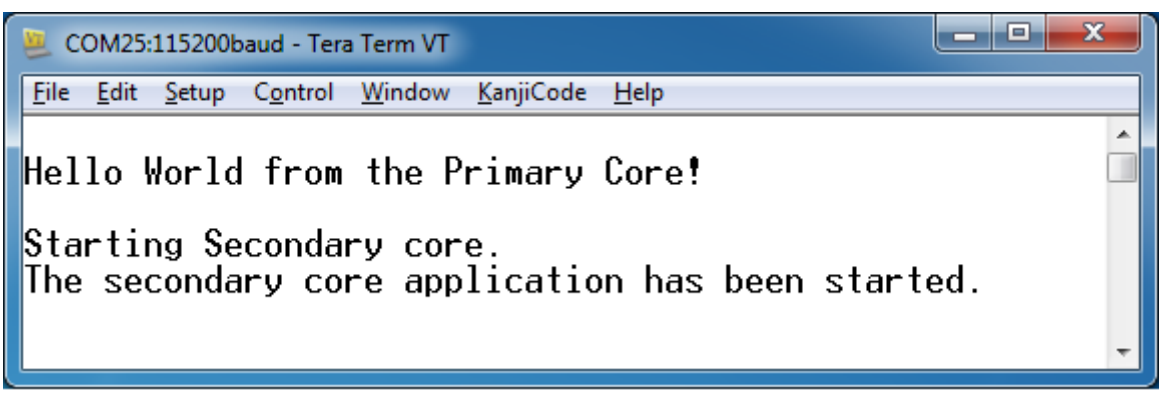

Figure 38. Hello World from the primary core message

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and running correctly. It is also possible to debug both sides of the multicore application in parallel. After creating the debug session for the primary core, perform same steps also for the auxiliary core application. Highlight the lpcxpresso54102_multicore_examples_hello_world_cm0plus project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click “Debug ‘lpcxpresso54102_multicore_examples_hello_world_cm0plus’ [Debug]” to launch the second debug session.

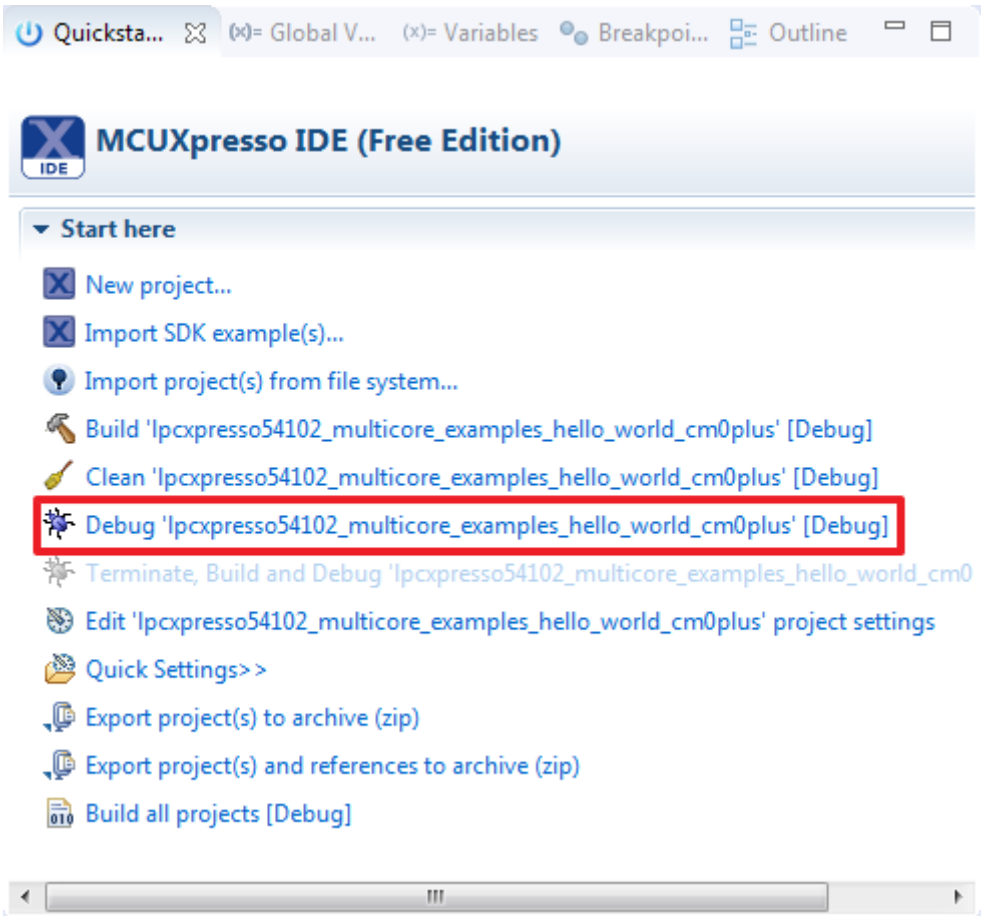

Figure 39. Debug "lpcxpresso54102_multicore_examples_hello_world_cm0plus" case

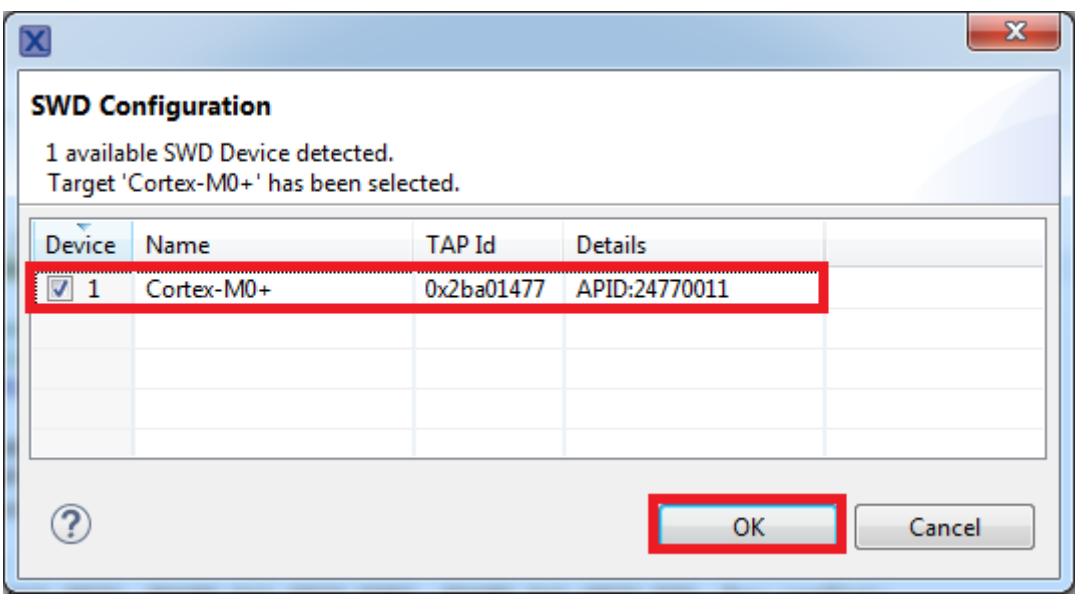

Figure 40. Target core selection dialog

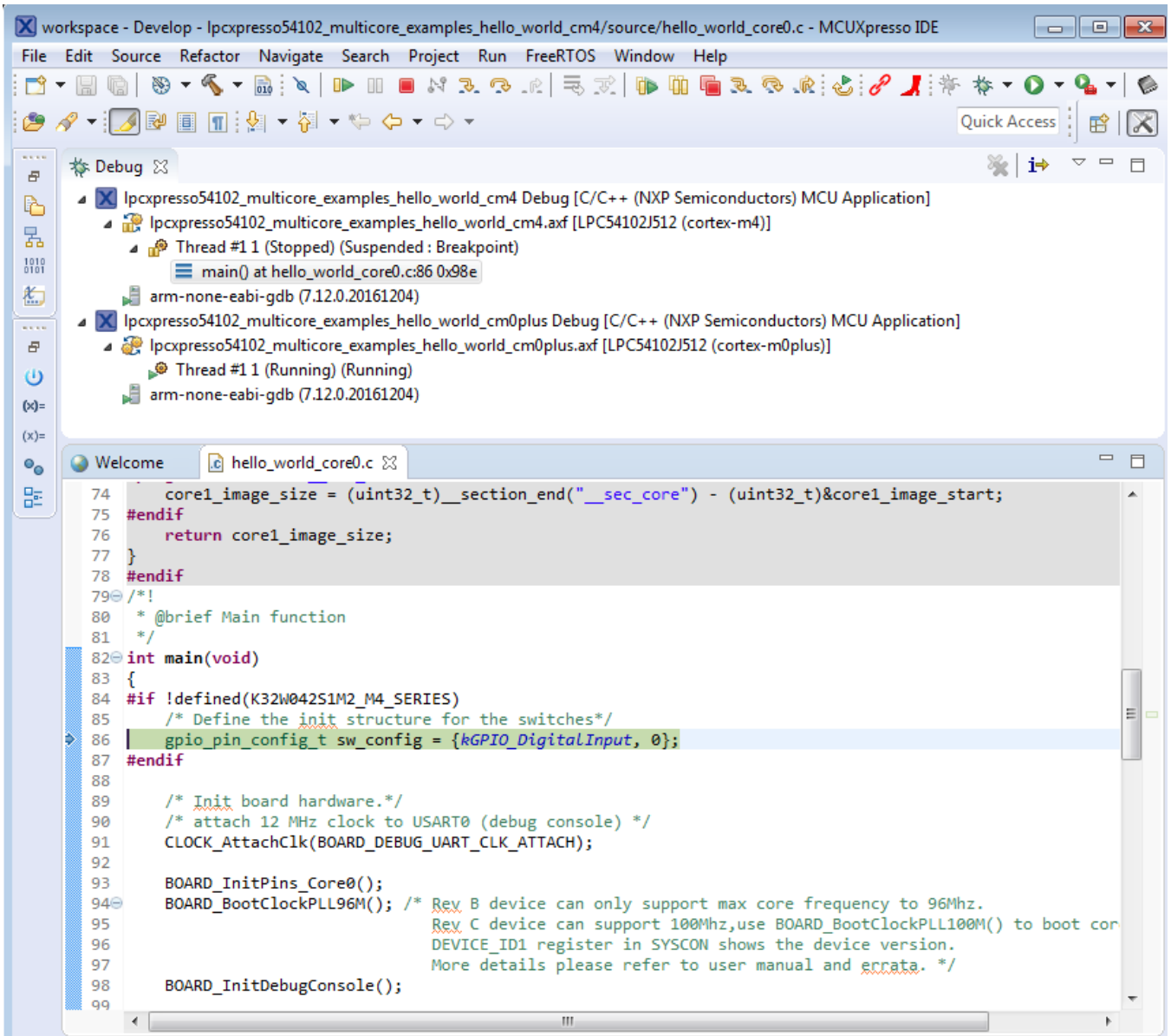

Figure 41. Two opened debug sessions

Now, the two debug sessions should be opened, and the debug controls can be used for both debug sessions depending on the debug session selection. Keep the primary core debug session selected and clicking the "Resume" button. The hello_world multicore application then starts running. The primary core application starts the auxiliary core application during runtime, and the auxiliary core application stops at the beginning of the main() function. The debug session of the auxiliary core application is highlighted. After clicking the “Resume” button, it is applied to the auxiliary core debug session. Therefore, the auxiliary core application continues its execution.

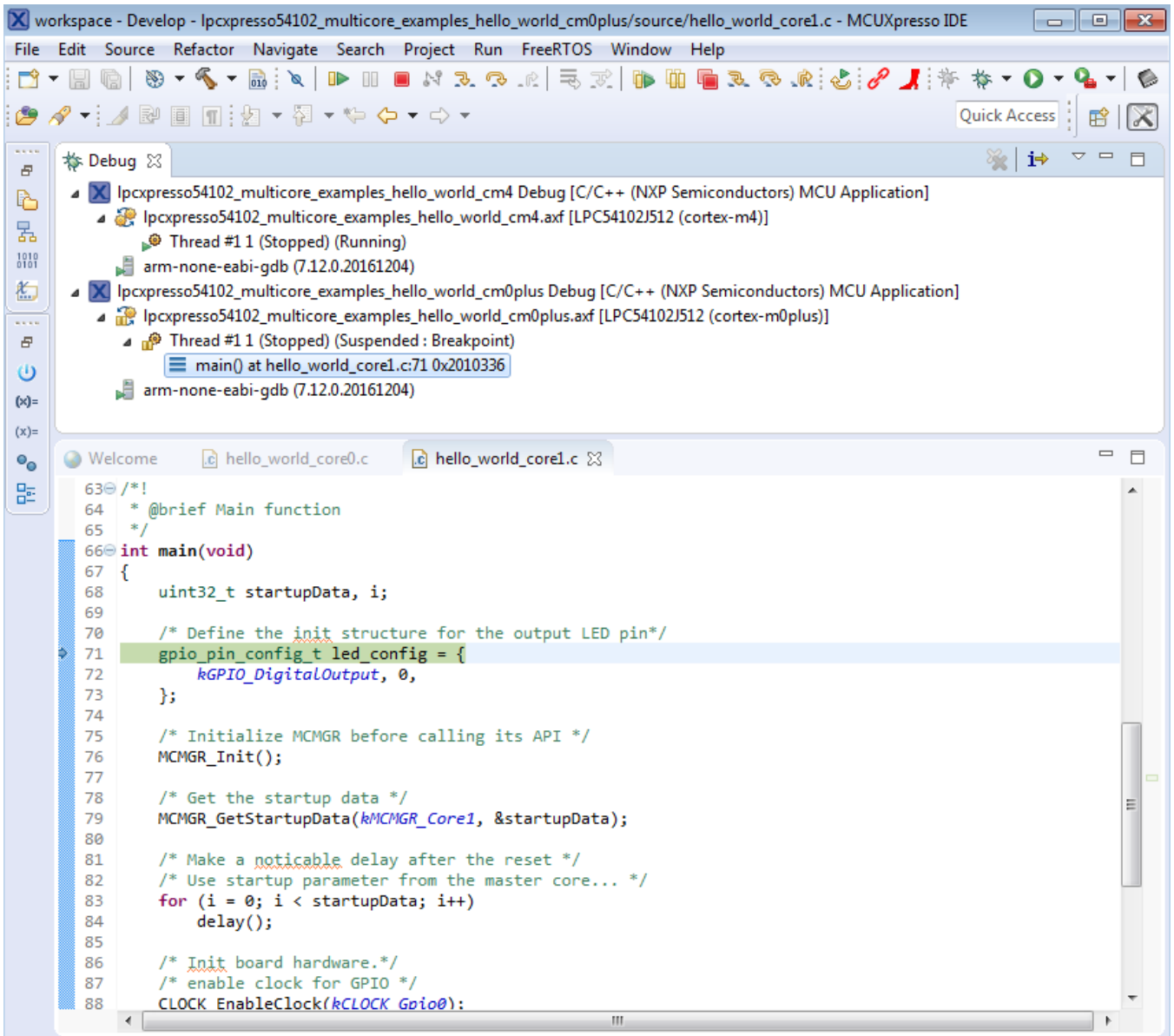

Figure 42. Auxiliary core application stops at the main function

At this point, it is possible to suspend and resume individual cores independently. It is also possible to make synchronous suspension and resumption of both cores. This is done either by selecting both opened debug sessions (multiple selection) and clicking the “Suspend” / ”Resume” control button, or just using the “Suspend All Debug sessions” and the “Resume All Debug sessions” buttons.

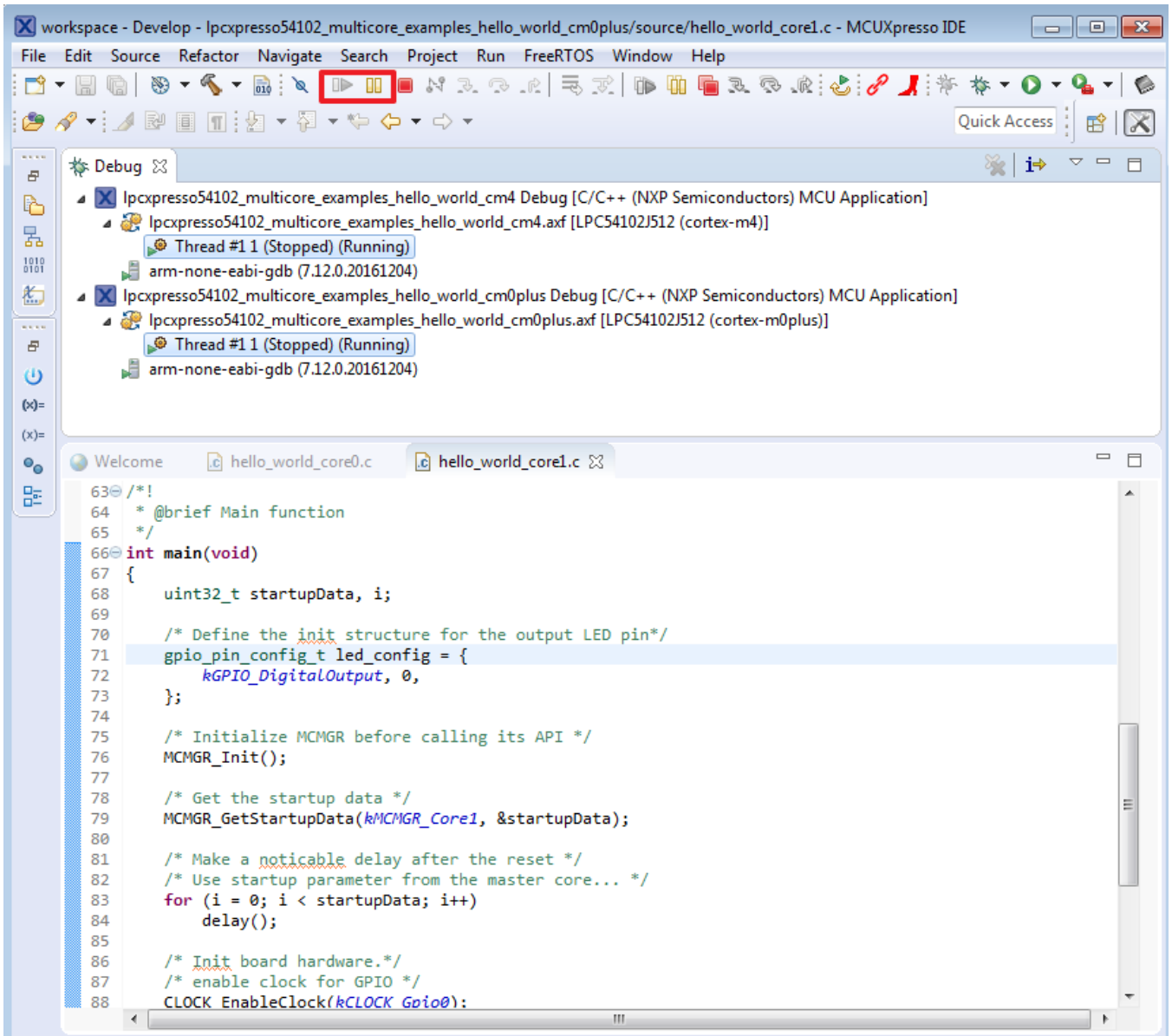

Figure 43. Synchronous suspension/resumption of both cores using the multiple selection of debug sessions and “Suspend”/”Resume” controls

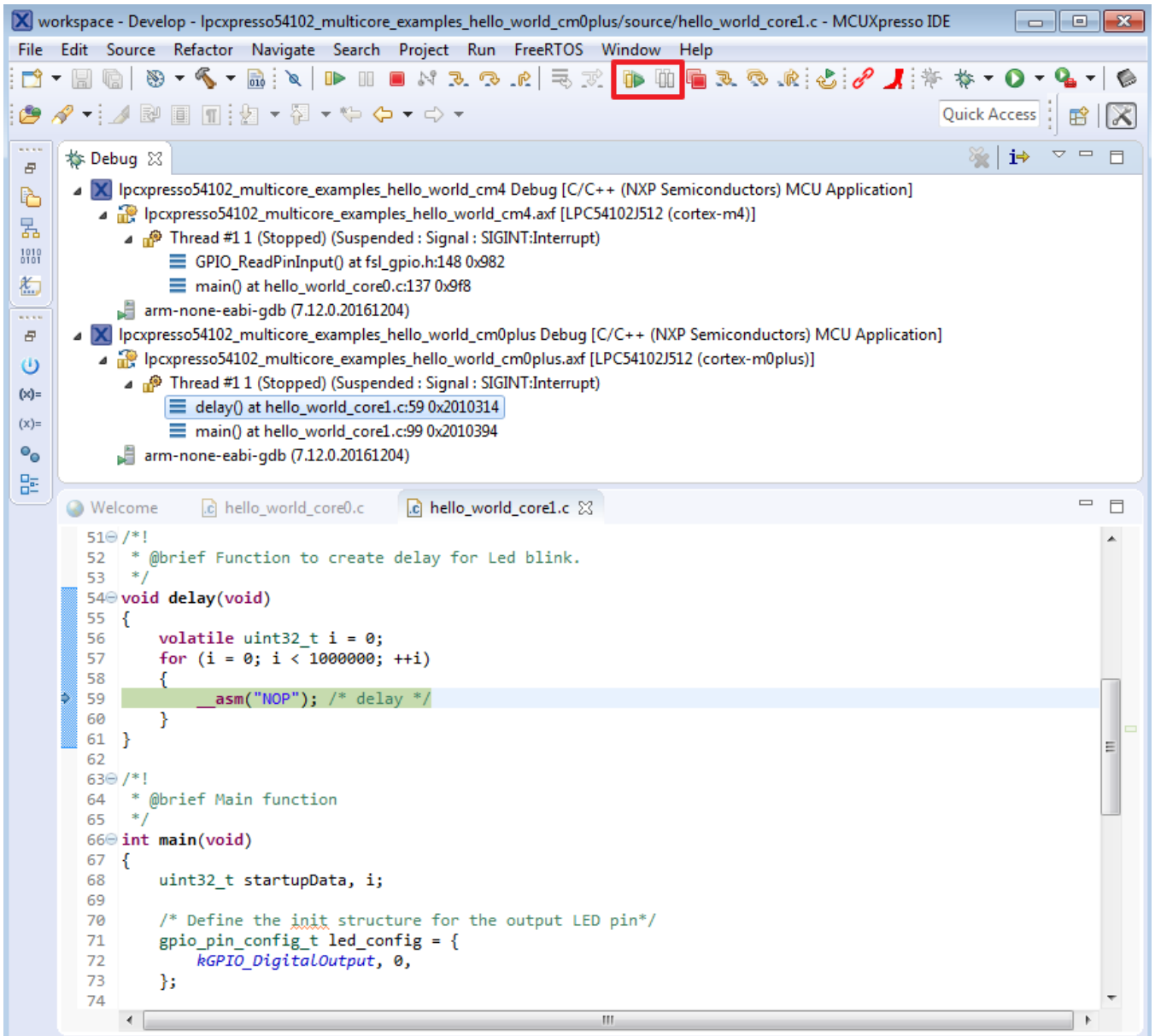

Figure 44. Synchronous suspension/resumption of both cores using the "Suspend All Debug sessions" and the "Resume All Debug sessions" controls

Appendix A How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

1. **Linux:** The serial port can be determined by running the following command after the USB Serial is connected to the host:

```
$ dmesg | grep "ttyUSB"
                    [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
                    [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
```

   There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

2. **Windows:** To determine the COM port open Device Manager in the Windows operating system. Click on the Start menu and type “Device Manager” in the search bar.

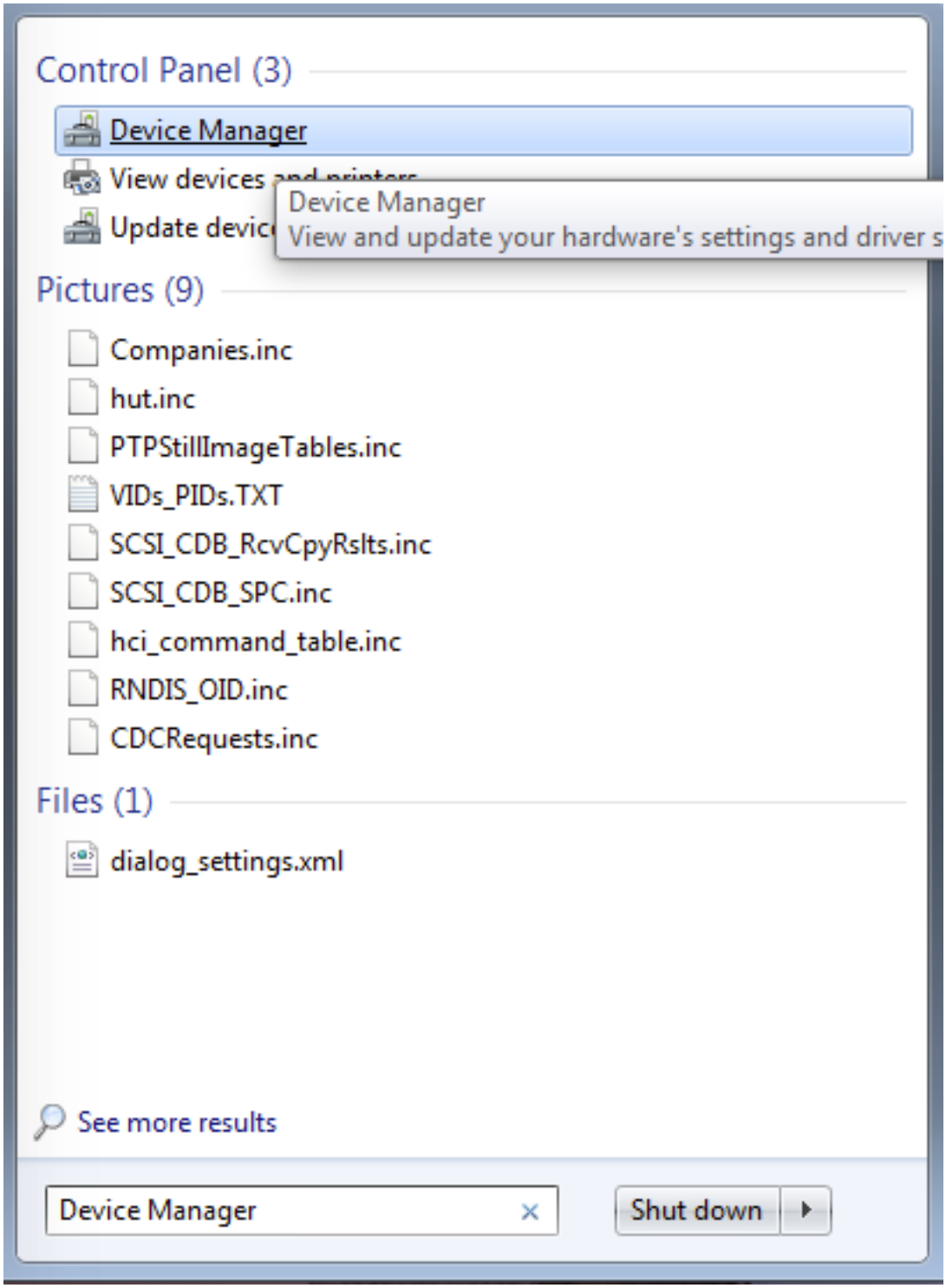

Figure A-1. Device Manager

3. In the Device Manager, expand the “Ports (COM & LPT)” section to view the available ports. The COM port names will be different for all the NXP boards.
   a. LPC-Link2

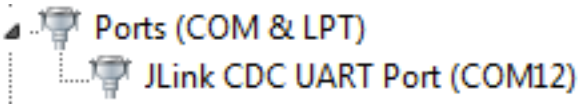

Figure A-2. LPC-Link2

Appendix B Updating debugger firmware

B.1 Updating LPCXpresso board firmware

The LPCXpresso hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called LPCScrypt. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

NOTE

If MCUXpresso IDE is used and the jumper making DFUlink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), LPC-Link2 debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the CMSIS-DAP firmware to the probe before flash memory programming (after clicking the "Debug" button). Using DFU mode ensures most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the LPCScrypt utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from www.nxp.com/lpcutilities.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScrypt user guide (www.nxp.com/lpcutilities, select LPCScrypt, then select documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labelled DFUlink).
4. Connect the probe to the host via USB (use Link USB connector).
5. Open a command shell and call the appropriate script located in the LPCScrypt installation directory (<LPCScrypt install dir>).
   a. To program CMSIS-DAP debug firmware: `<LPCScrypt install dir>/scripts/program_CMSIS`
   b. To program J-Link debug firmware: `<LPCScrypt install dir>/scripts/program_JLINK`
6. Remove DFU link (remove the jumper installed in step 3).
7. Re-power the board by removing the USB cable and plugging it again.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MCUXSDKLPC5410XGSUG
Revision 0, 08/2019