NXP Semiconductors
User's Guide

Document Number: MCUXSDKLPC55XXGSUG
Rev. 0, 12/2019

Getting Started with MCUXpresso SDK for LPC55xx

Contents

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, a USB host and device stack, and various other middleware to support rapid development.

For supported toolchain versions, see the *MCUXpresso SDK Release Notes Supporting LPC55xx* (document MCUXSDKLPC55XXRN).

For more details about MCUXpresso SDK, refer to MCUXpresso-SDK: Software Development Kit for MCUXpresso.

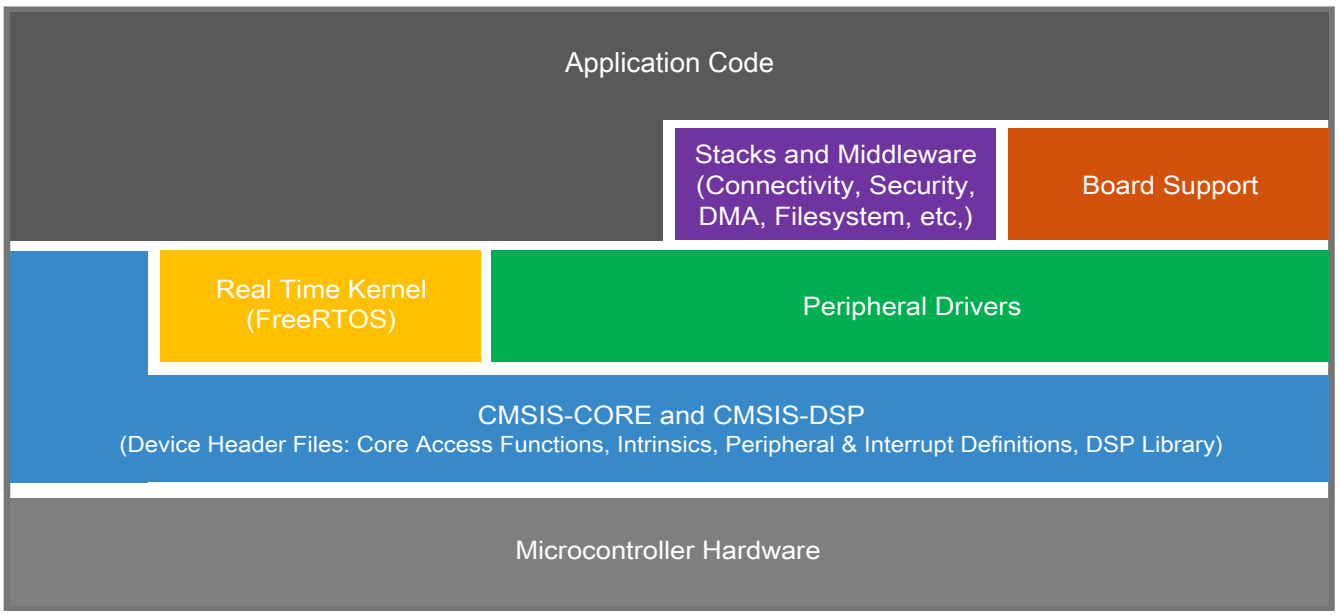

Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support package folders

MCUXpresso SDK board support package 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 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 it contain. These include (but are not limited to):

- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that 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 that show 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).
- `emwin_examples`: Applications that use the emWin GUI widgets.
- `rtos_examples`: Basic FreeRTOS™ OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers
- `usb_examples`: Applications that use the USB host/device/OTG stack.

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 *MCUXpresso SDK API Reference Manual* (document ID: 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:

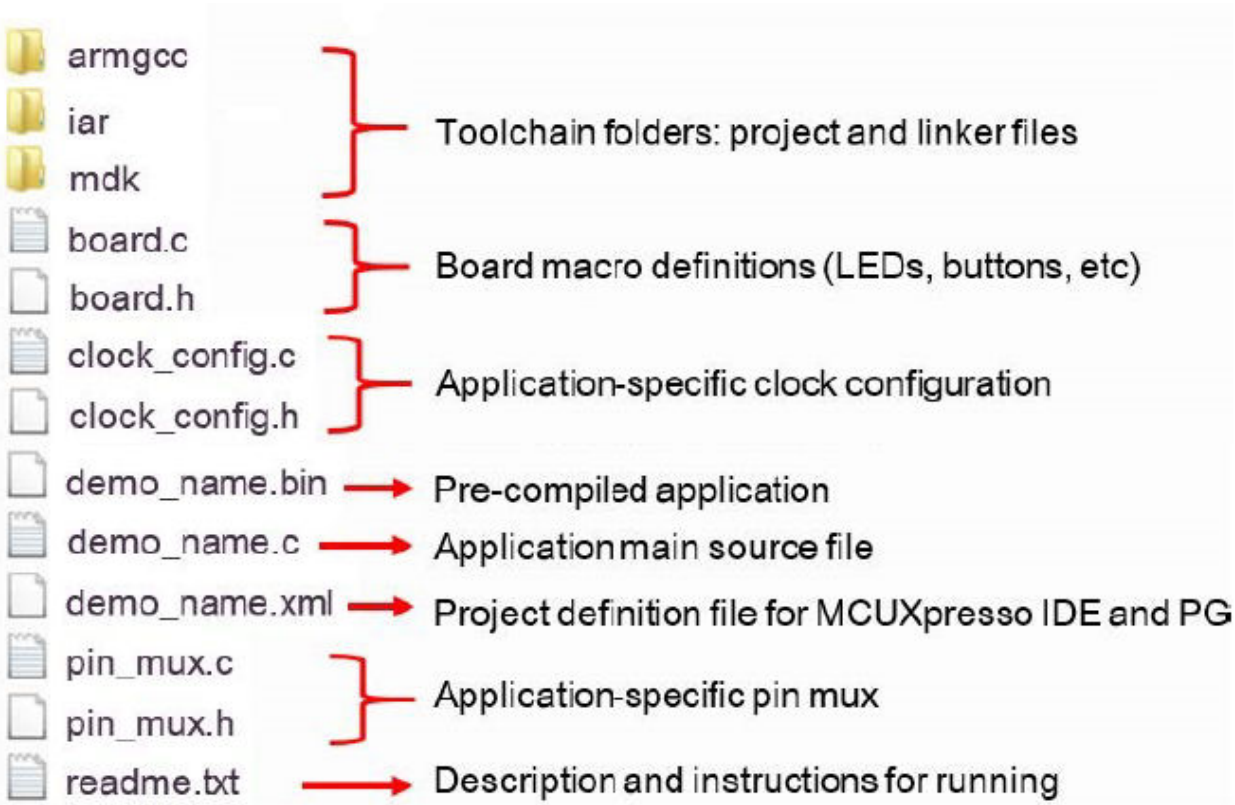

Figure 2. Application folder structure

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 files
- `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, including vector table definitions
- `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. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

There are two boards in LPC55xx SDK: LPCXpresso55S69 and LPCXpresso55S28. This document uses LPCXpresso55S69 as example. Same steps are applied to LPCXpresso55S28 SDK.

Refer document here for LPC55xx 1B silicon IDE update: https://community.nxp.com/docs/DOC-345024

NOTE

The multicore example and trustzone example not support in LPCXpresso55S28 SDK.

3 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 v11.0.1 to build, run, and debug example applications. The hello_world demo application targeted for the LPCXpresso55S69 hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

3.1 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 located outside of the MCUXpresso SDK tree.

3.2 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 **OK** and wait until the import has finished.

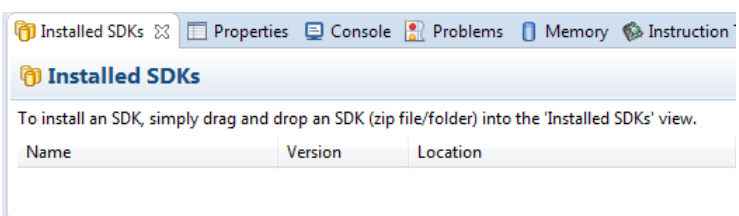

Figure 3. Install an SDK

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

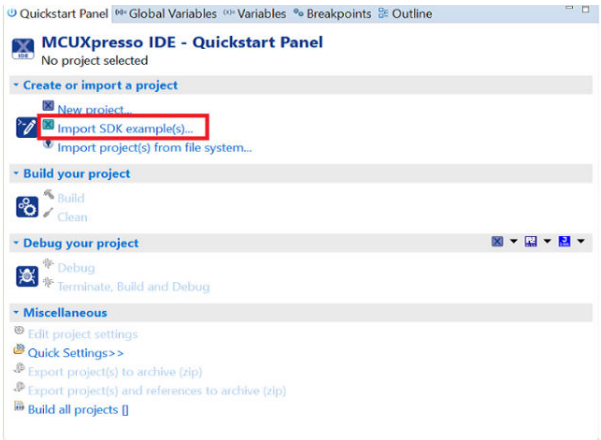

Figure 4. Import an SDK example

3. In the window that appears, expand the **LPC55xx** folder and select **LPC55S69** . Then, select **lpcxpresso55s69** and click **Next**.

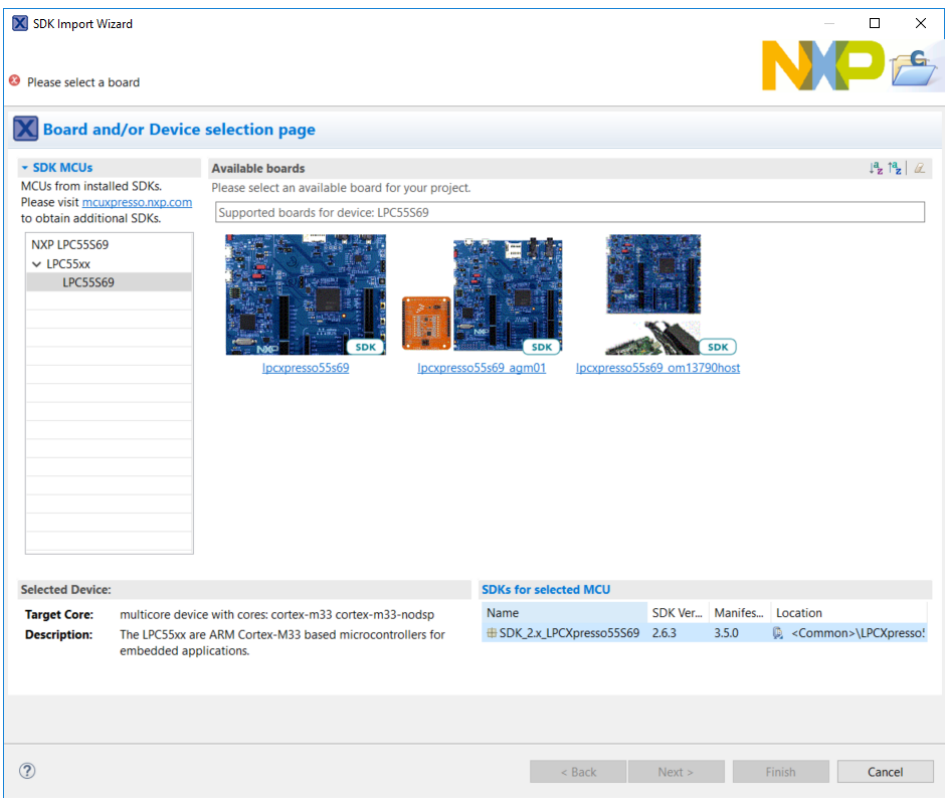

Figure 5. Select LPCXpresso55S69 board

4. Expand the `demo_apps` folder and select `hello_world`. Then, select **UART** as SDK Debug Console and click **Next**.

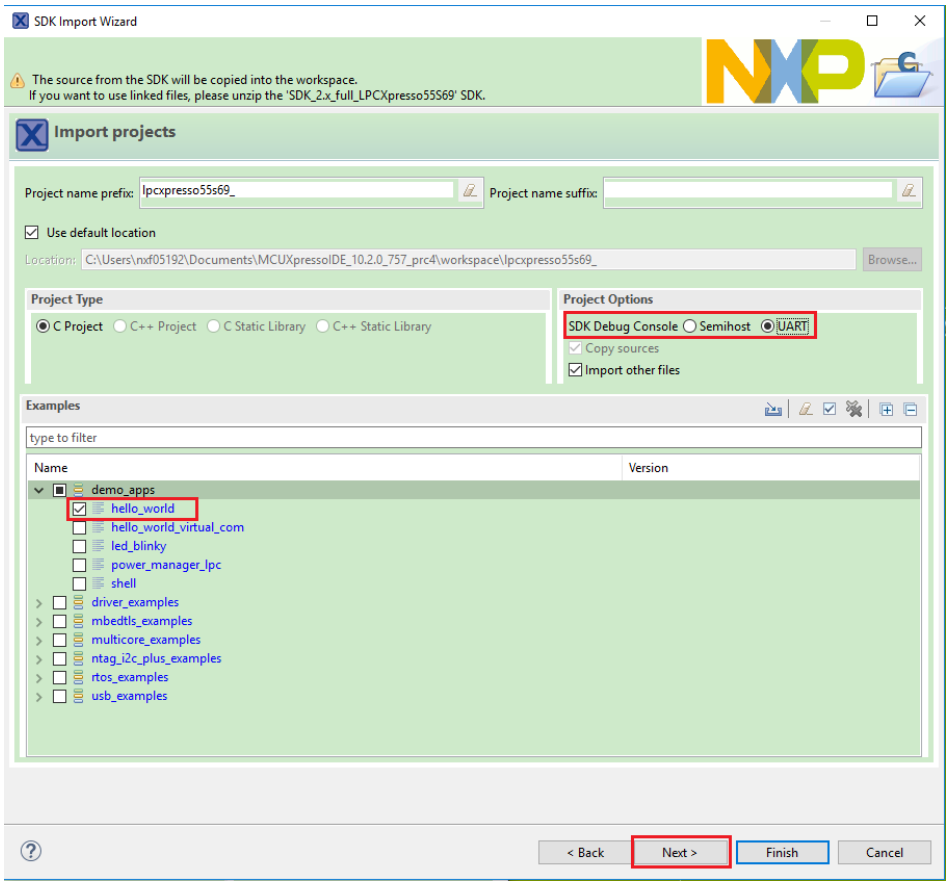

Figure 6. Select `hello_world`

5. Ensure the option **Redlib: Use floating point version of printf** is selected if the cases' print floating point numbers are on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click the **Finish** button.

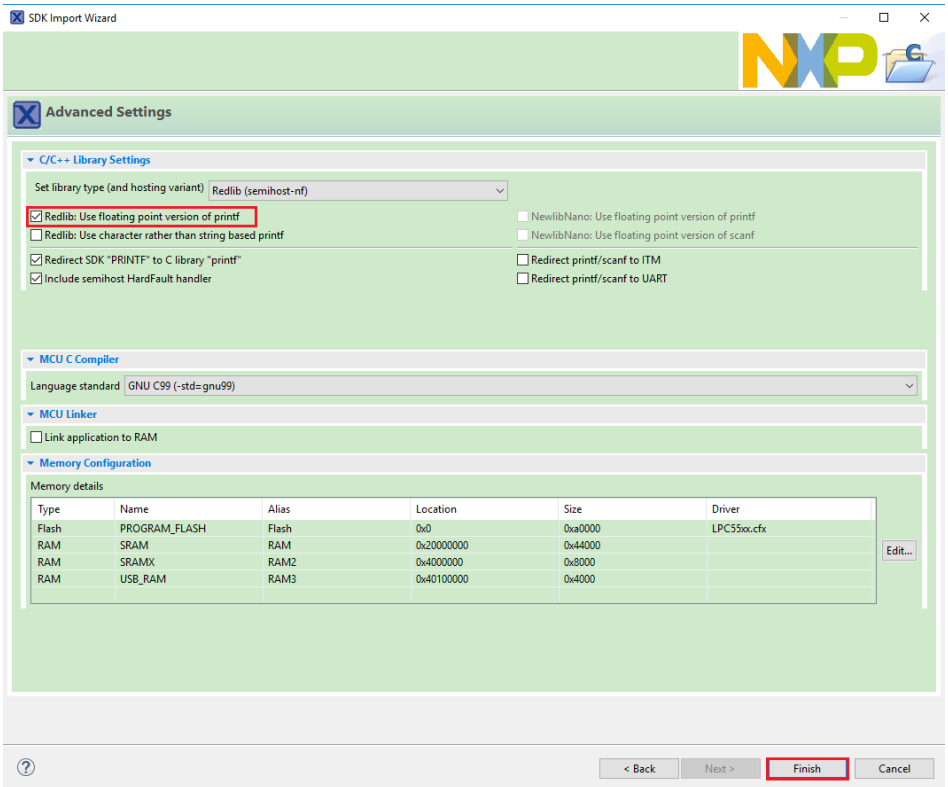

Figure 7. Select "User floating print version of printf"

3.3 Run an example application

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

To download and run the application, perform the following steps:

1. See the table in Default debug interfaces to determine the debug interface that comes loaded on your specific hardware platform. For LPCXpresso boards, install the DFU jumper for the debug probe, then connect the debug probe USB connector.
   - For boards with a P&E Micro interface, see www.pemicro.com/support/downloads_find.cfm to download and install the P&E Micro Hardware Interface Drivers package.
2. Connect the development platform to your PC via a USB cable.
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 How to determine COM port). 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

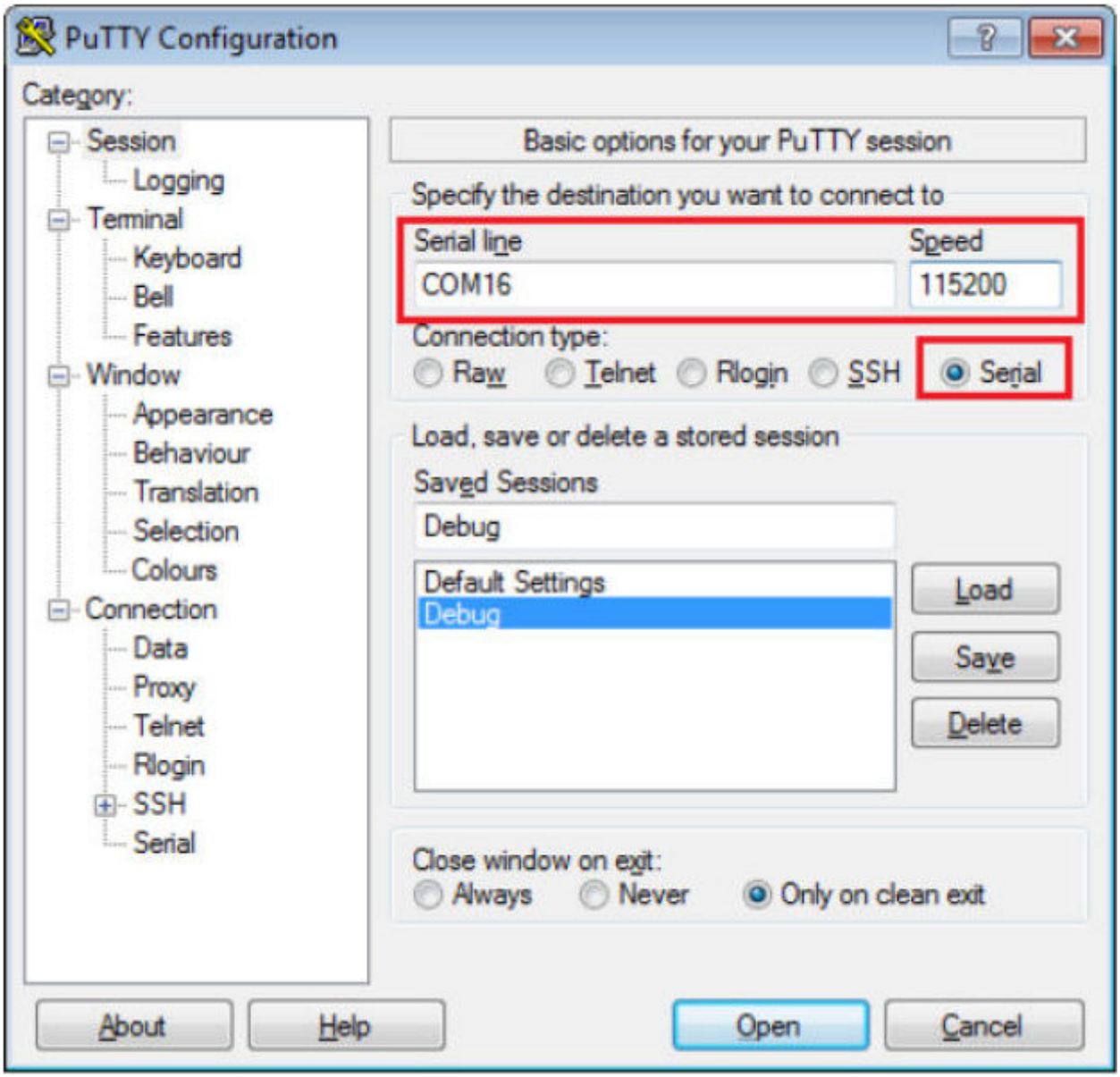

Figure 8. Terminal (PuTTY) configurations

4. On the **Quickstart Panel**, click on **Debug** `lpcxpresso55s69_hello_world [Debug]` to launch the debug session.

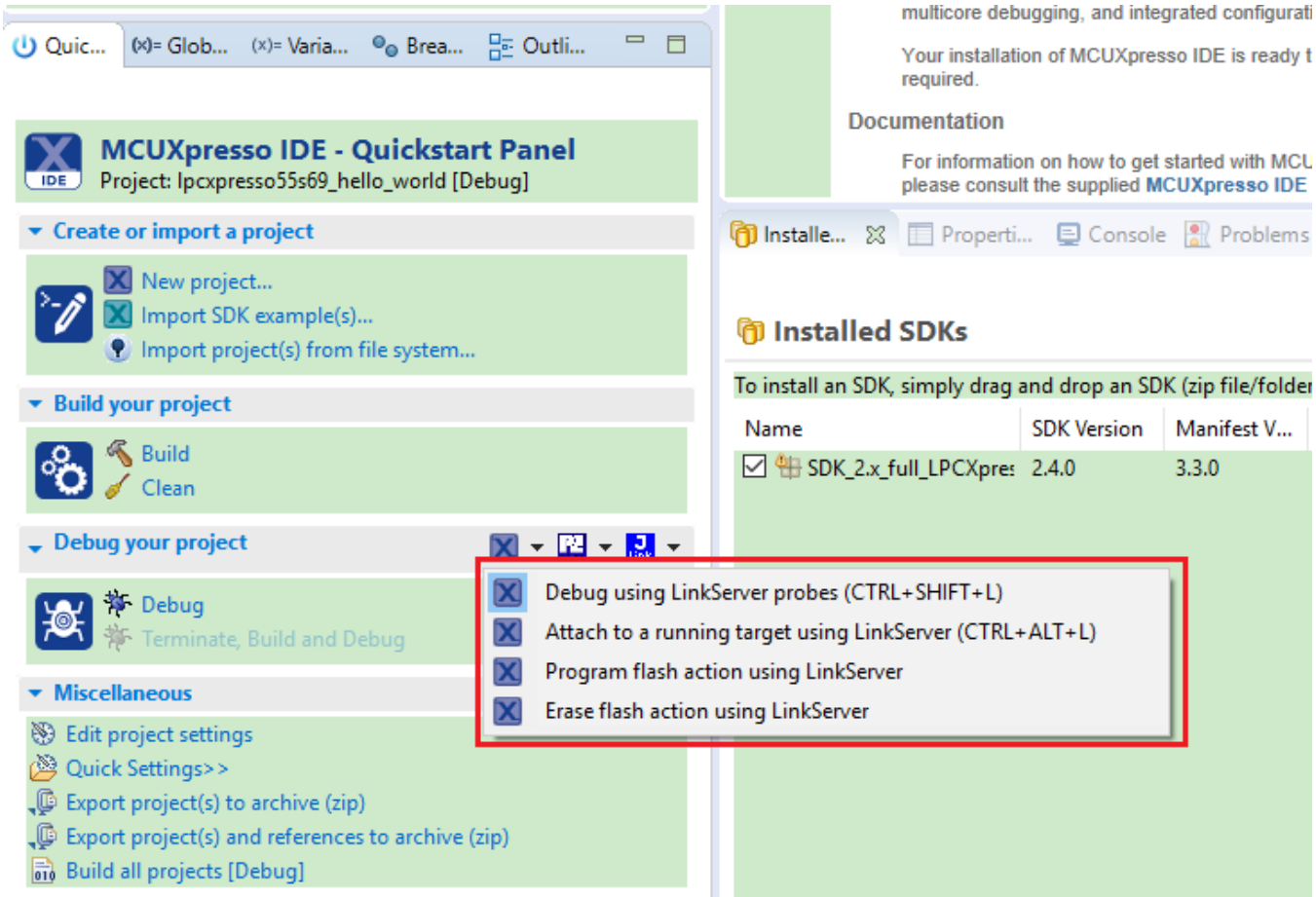

Figure 9. Debug `hello_world` case

5. 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 **OK**. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)

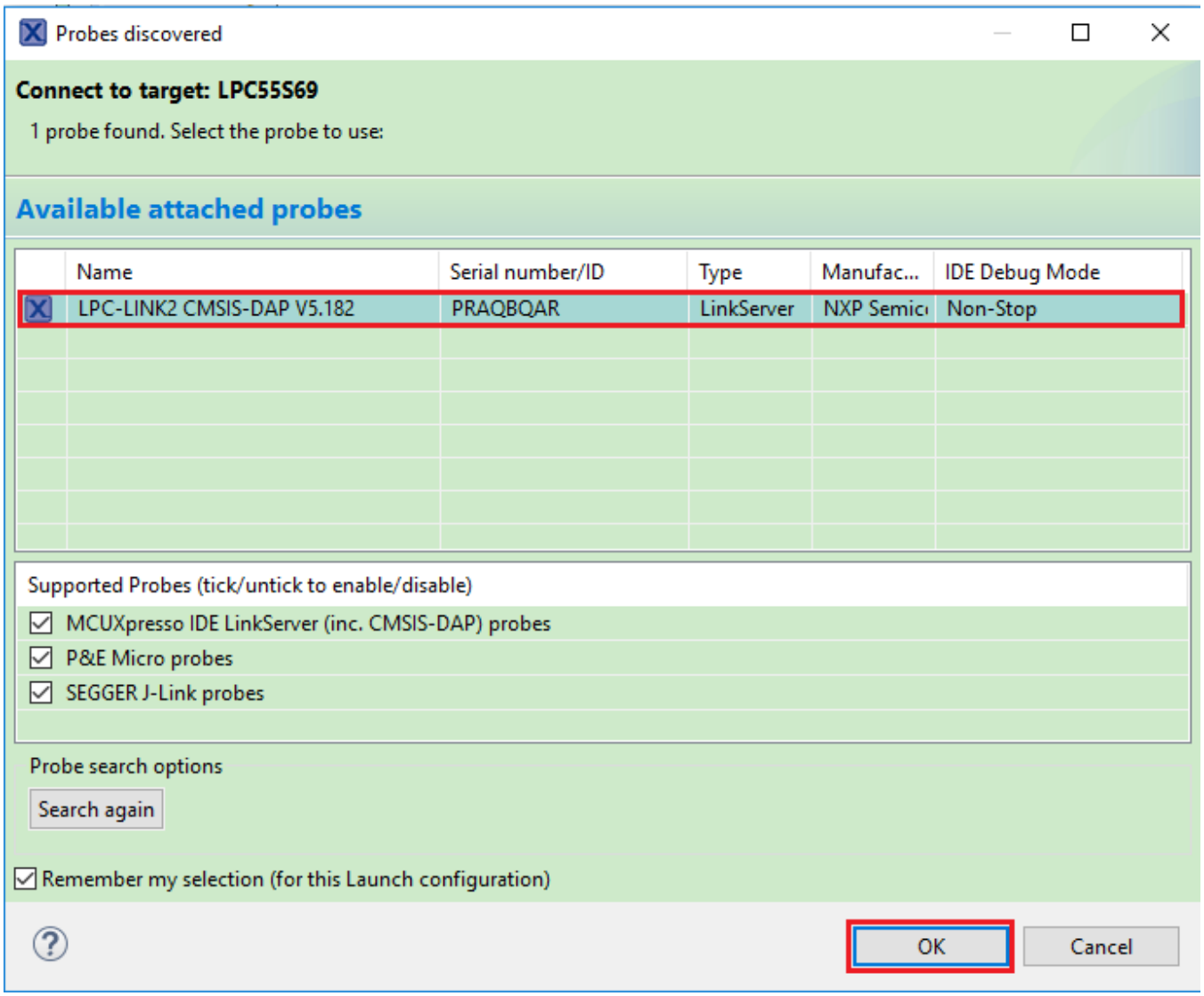

Figure 10. Attached Probes: debug emulator selection

6. The application is downloaded to the target and automatically runs to `main()`.

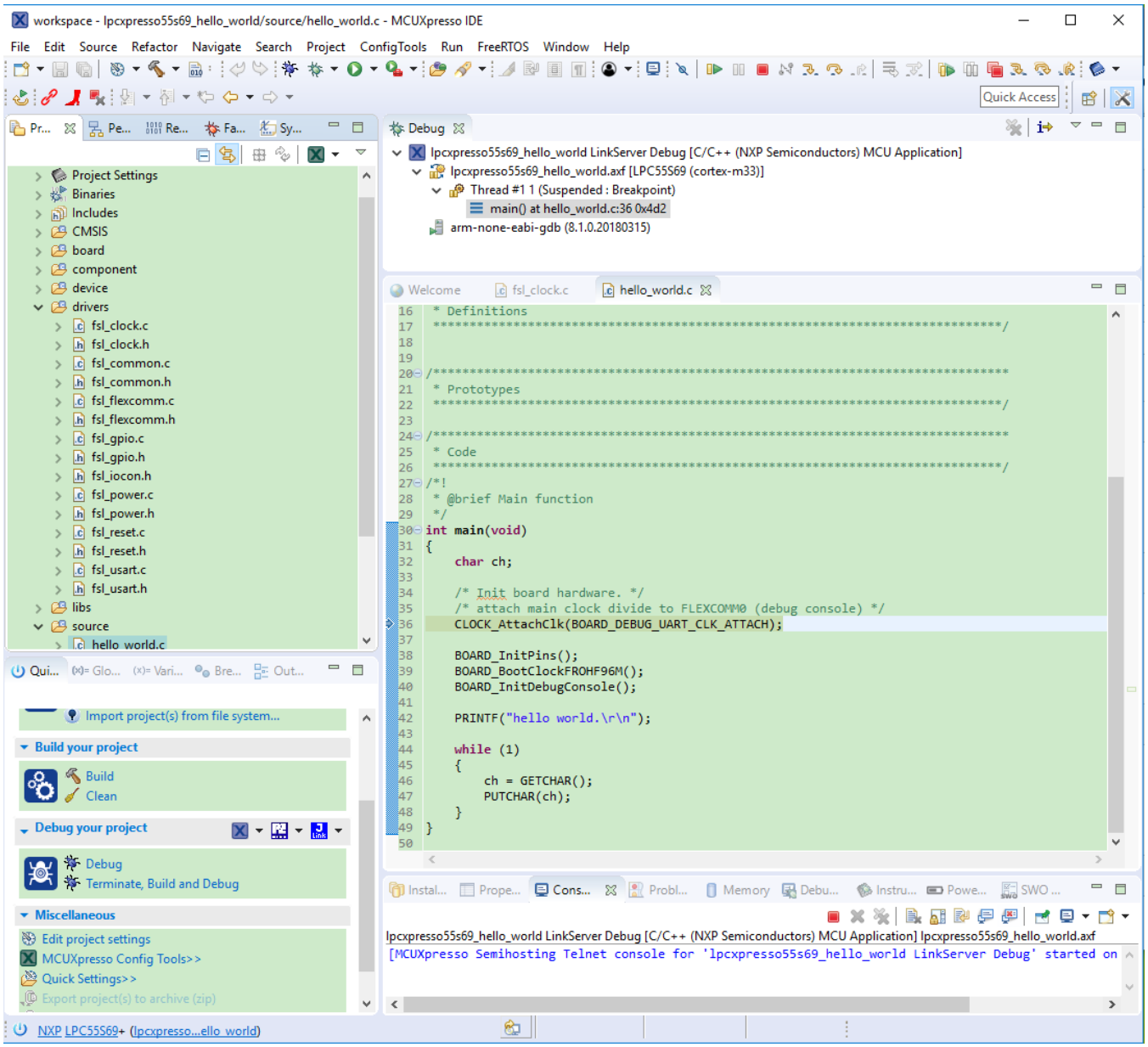

Figure 11. Stop at `main()` when running debugging

7. Start the application by clicking **Resume**.

Figure 12. Resume button

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

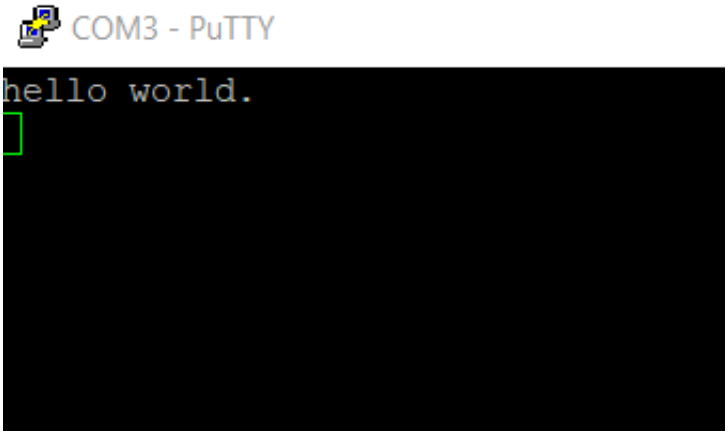

Figure 13. Text display of the hello_world demo

3.4 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 following steps can be applied to any multicore example application in the MCUXpresso SDK. Here, the dual-core version of hello_world example application targeted for the LPCXpresso55S69 hardware platform is used as an example.

1. Multicore examples are imported into the workspace in a similar way as single core applications, explained in Build an example application. When the SDK zip package for LPCXpresso55S69 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 “LPC55xx” folder and select “LPC55S69”. Then, select “lpcxpresso55s69” and click the “Next” button.

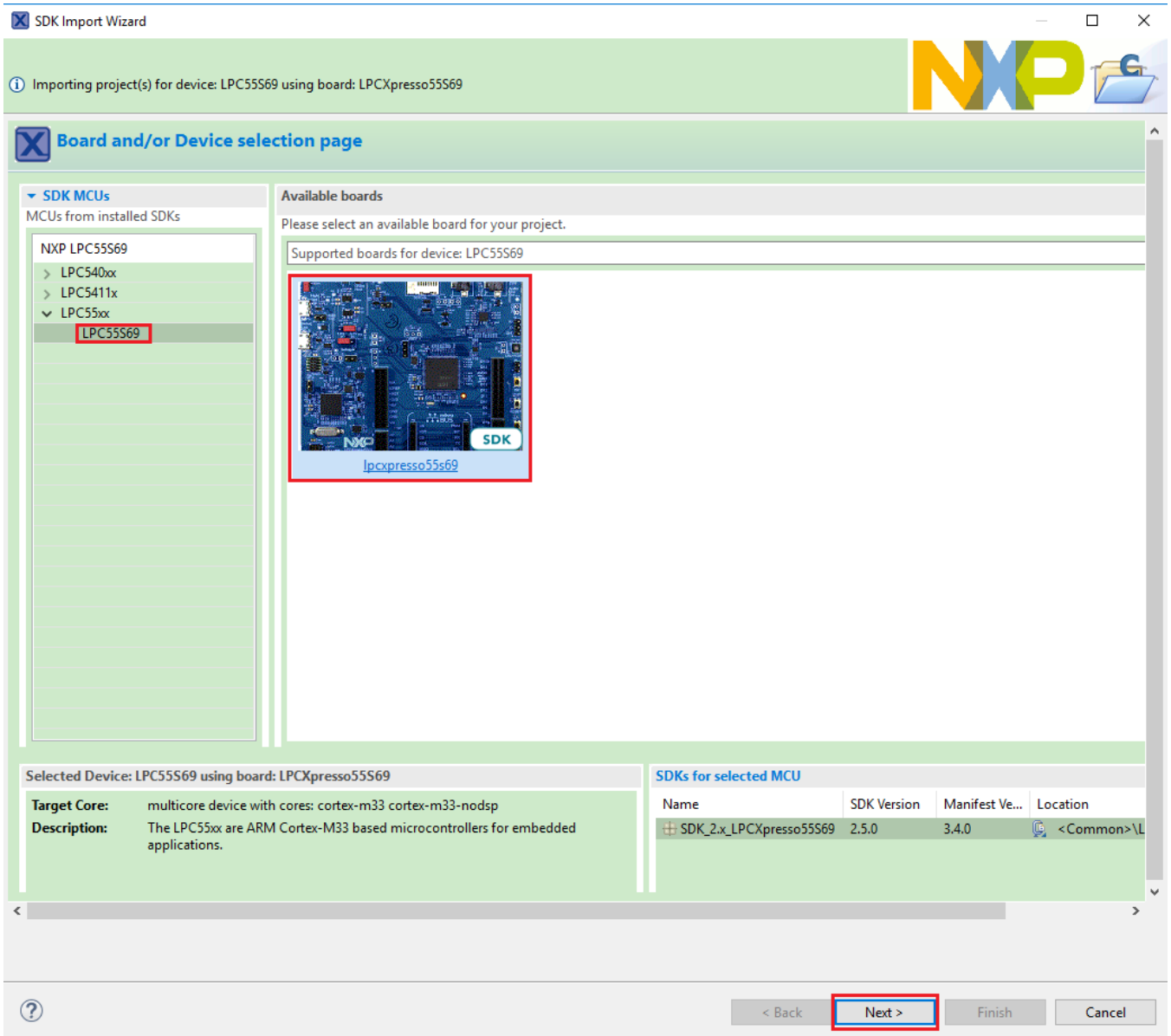

Figure 14. Select the LPCXpresso55S69 board

2. Expand the “multicore_examples” folder and select "hello_world_cm33_core0". The core1 counterpart project is automatically imported with the core0 project, because the multicore examples are linked together and there is no need to select it explicitly. Then select "UART" as SDK Debug Console. Click the “Finish” button.

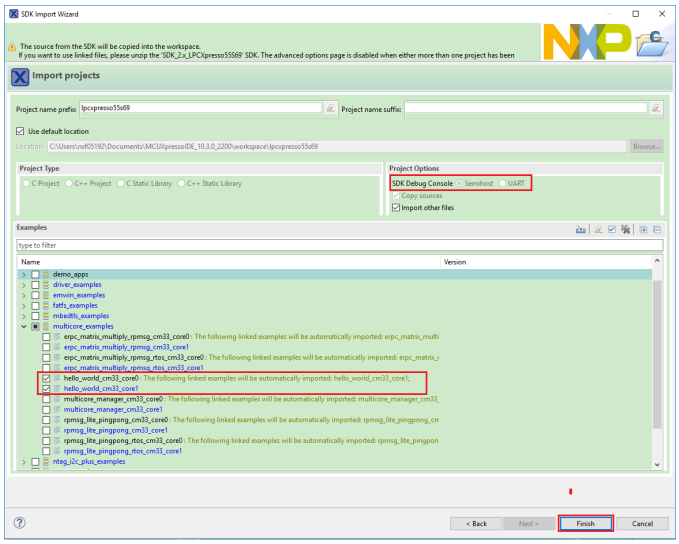

Figure 15. Select the hello_world multicore example

3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the lpcxpresso55s69_hello_world_cm33_core0 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.

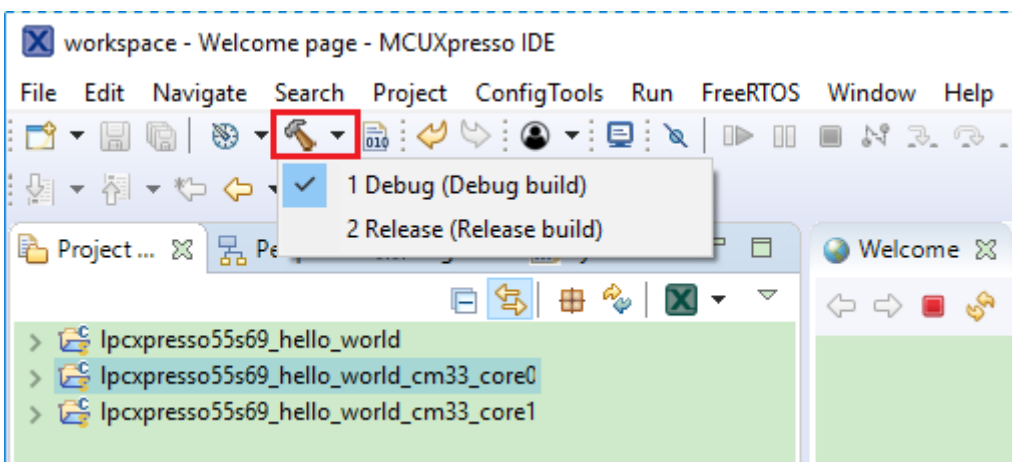

Figure 16. Selection of the build target in MCUXpresso IDE

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 (core0) also causes the referenced auxiliary core application (core1) to build.

NOTE

When the 'Release' build is requested, it is necessary to change the build configuration of both the primary and auxiliary core application projects first. To do this, select both projects in the Project Explorer view and then right click which displays the context-sensitive menu. Select 'Build Configurations->Set Active->Release'. This alternate navigation using the menu item is 'Project->Build Configuration->Set Active->Release'. After switching to the 'Release' build configuration, the build of the multicore example can be started by triggering the primary core application (core0)build.

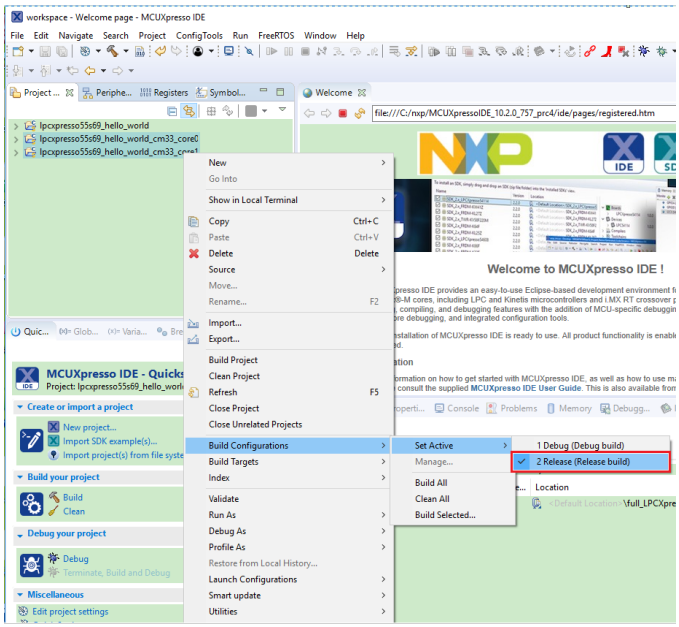

Figure 17. Switching multicore projects into the Release build configuration

3.5 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 *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 which requires selecting the target core. See the following figures as reference.

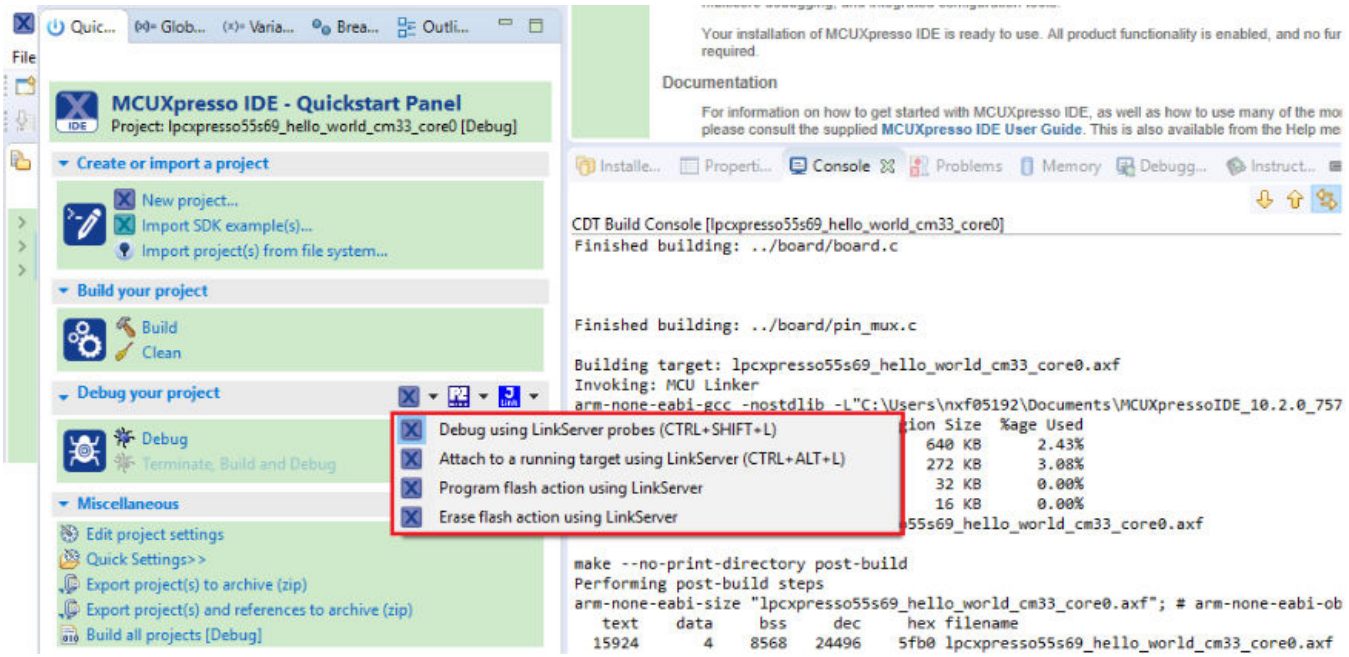

Figure 18. Debug "lpcxpresso55s69_hello_world_cm33_core0" case

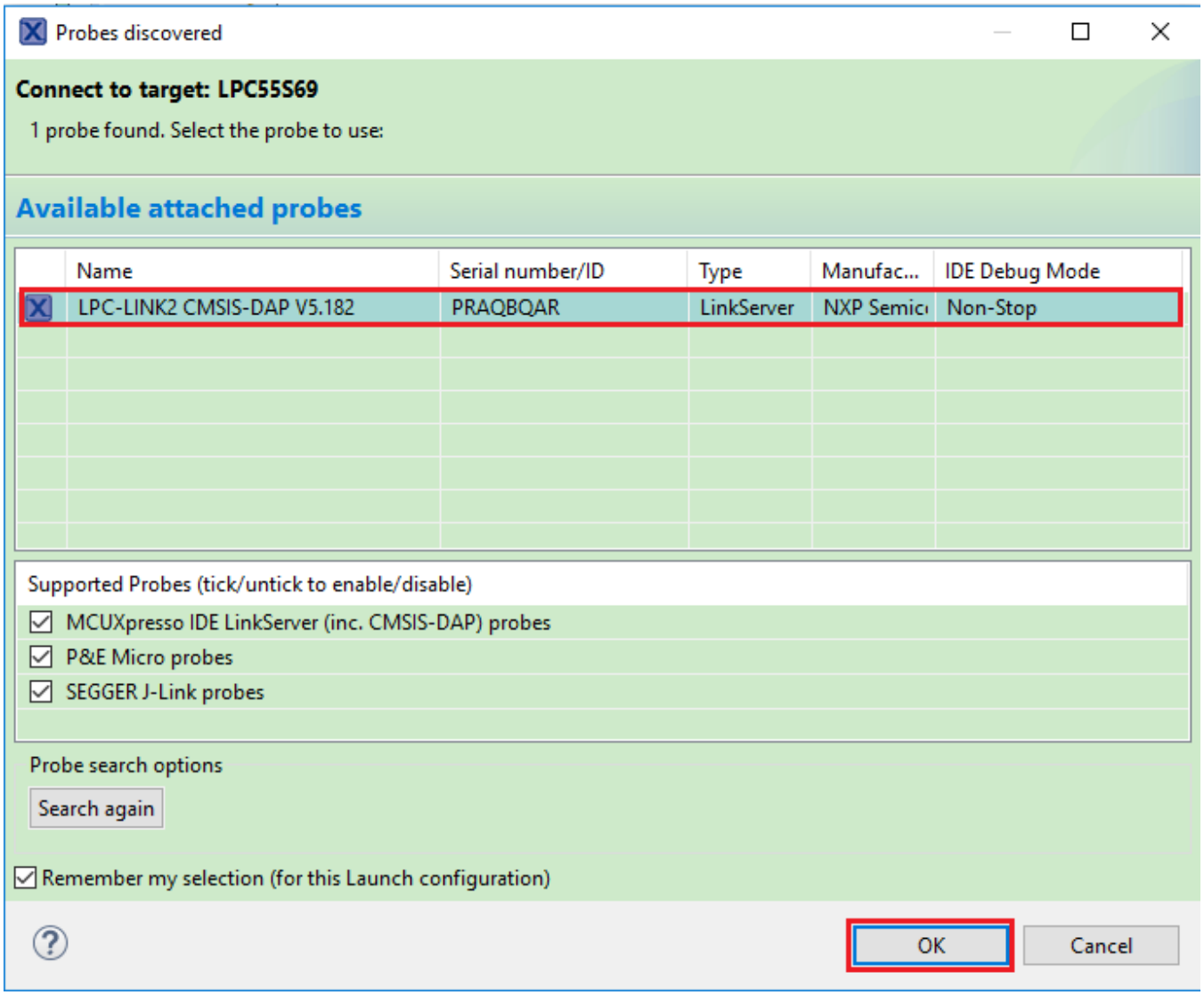

Figure 19. Attached Probes: debug emulator selection

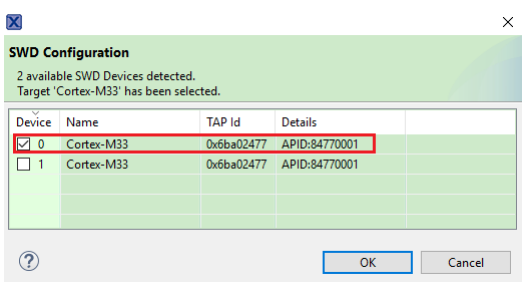

Figure 20. Target core selection dialogue

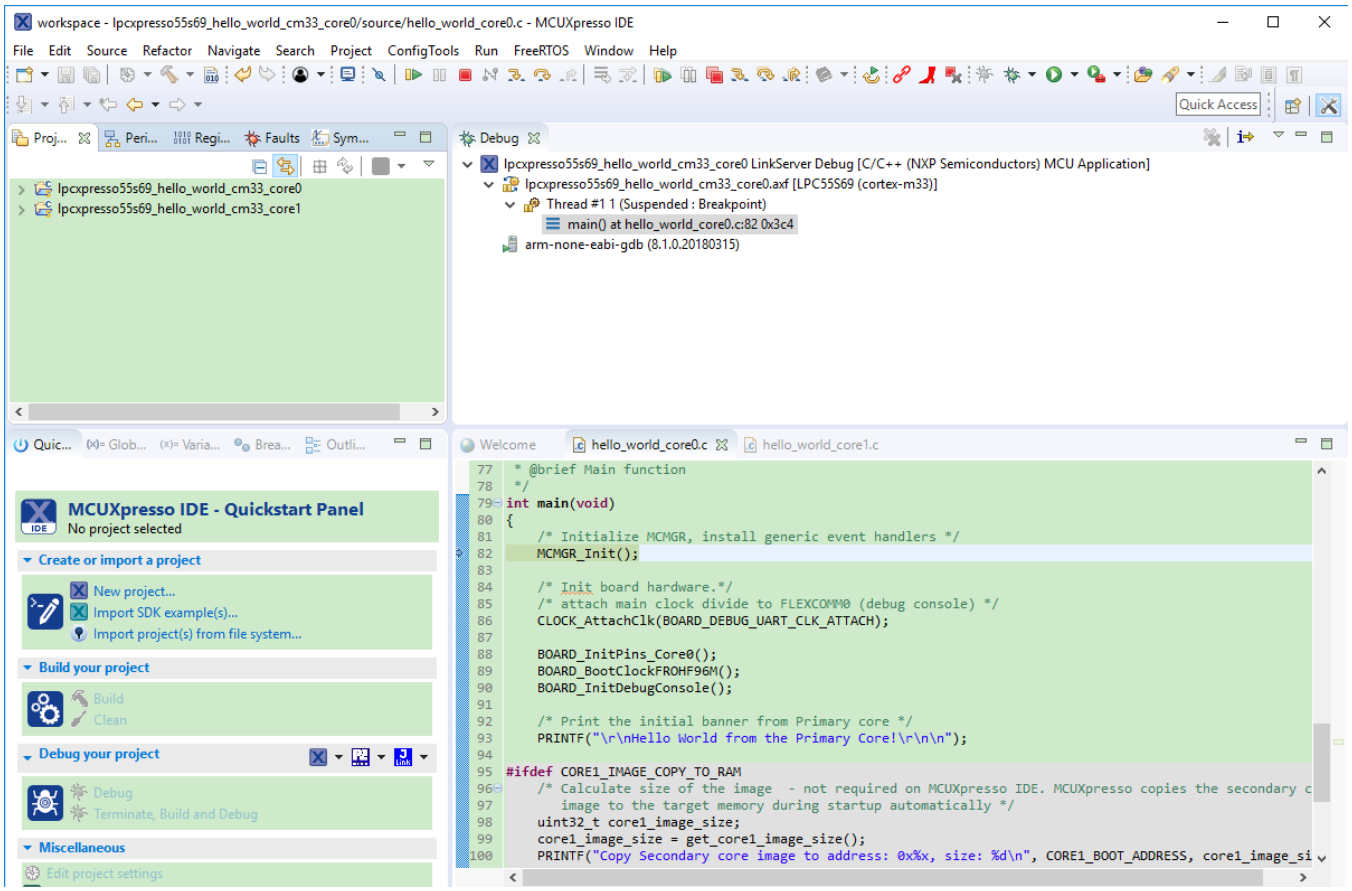

Figure 21. Stop the primary core application at main() when running debugging

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.

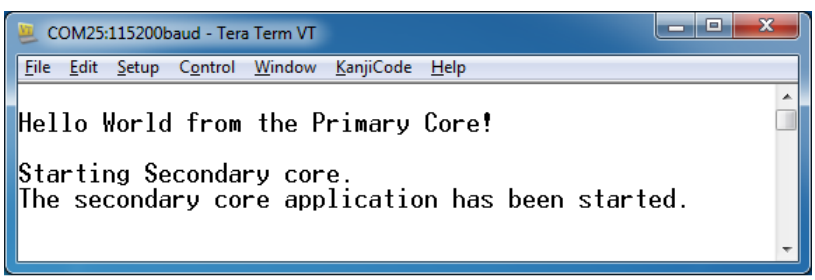

Figure 22. Hello World from the primary core message

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 lpcxpresso55s69_multicore_examples_hello_world_cm33_core1 project (multicore slave project) in the Project Explorer. On the Quickstart Panel, click “Debug ‘lpcxpresso55s69_multicore_examples_hello_world_cm33_core1’ [Debug]” to launch the second debug session.

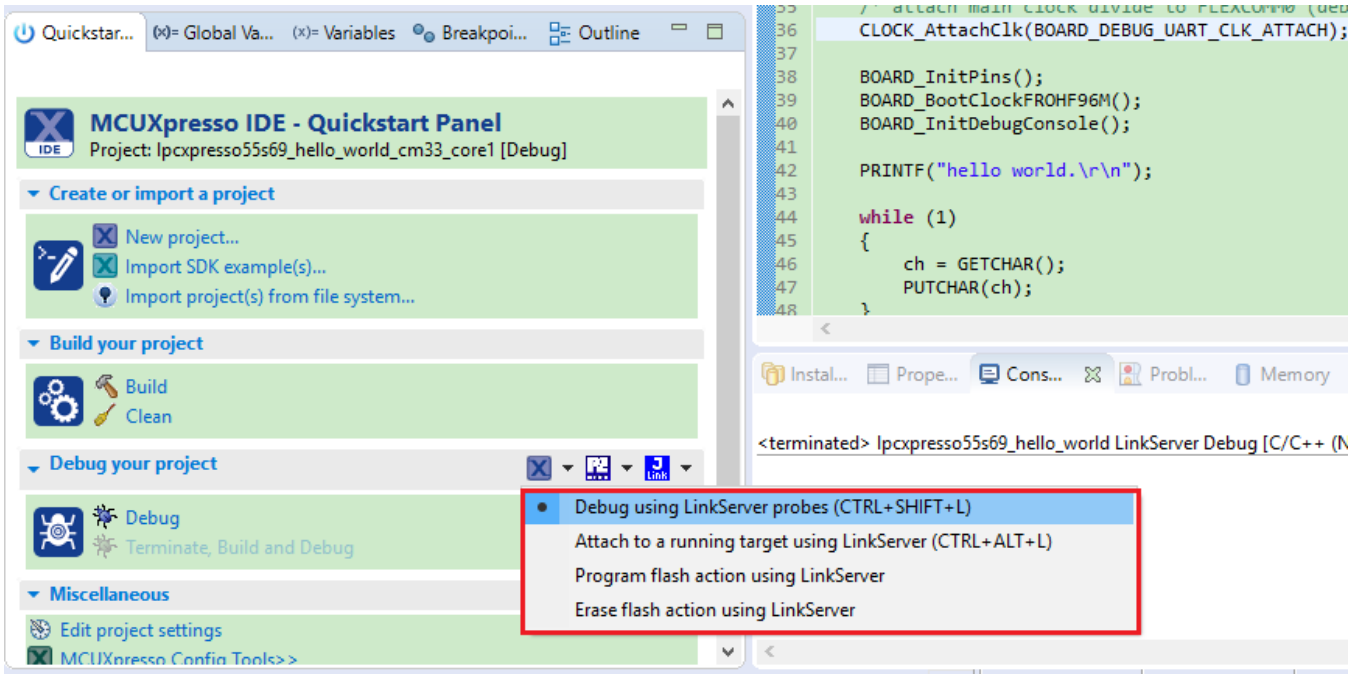

Figure 23. Debug "lpcxpresso55s69_hello_world_cm33_core1" case

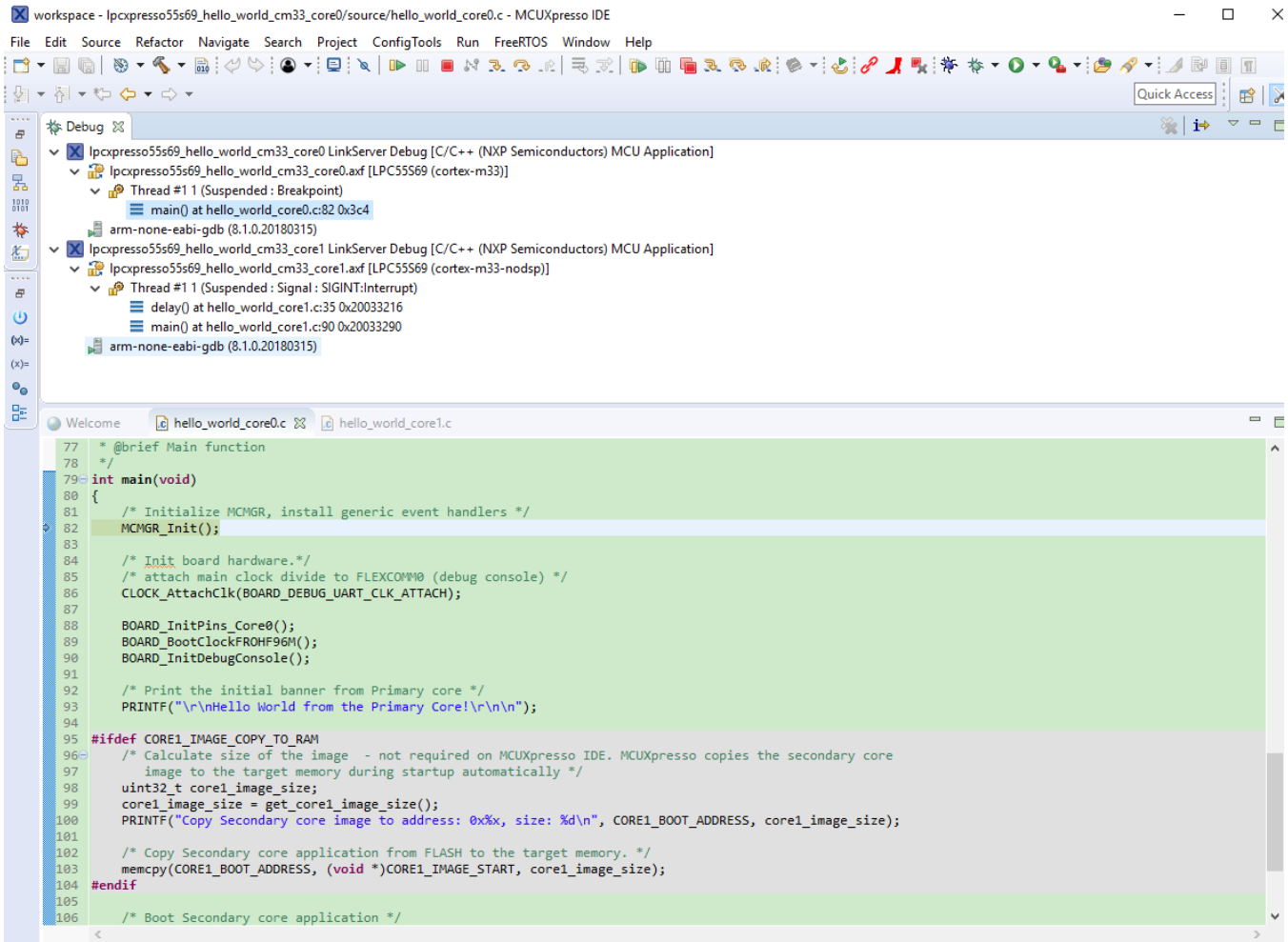

Figure 24. Two opened debug sessions

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 by clicking the "Resume" button. The hello_world multicore application then starts running. The primary core application starts the auxiliary core application during run time, 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.

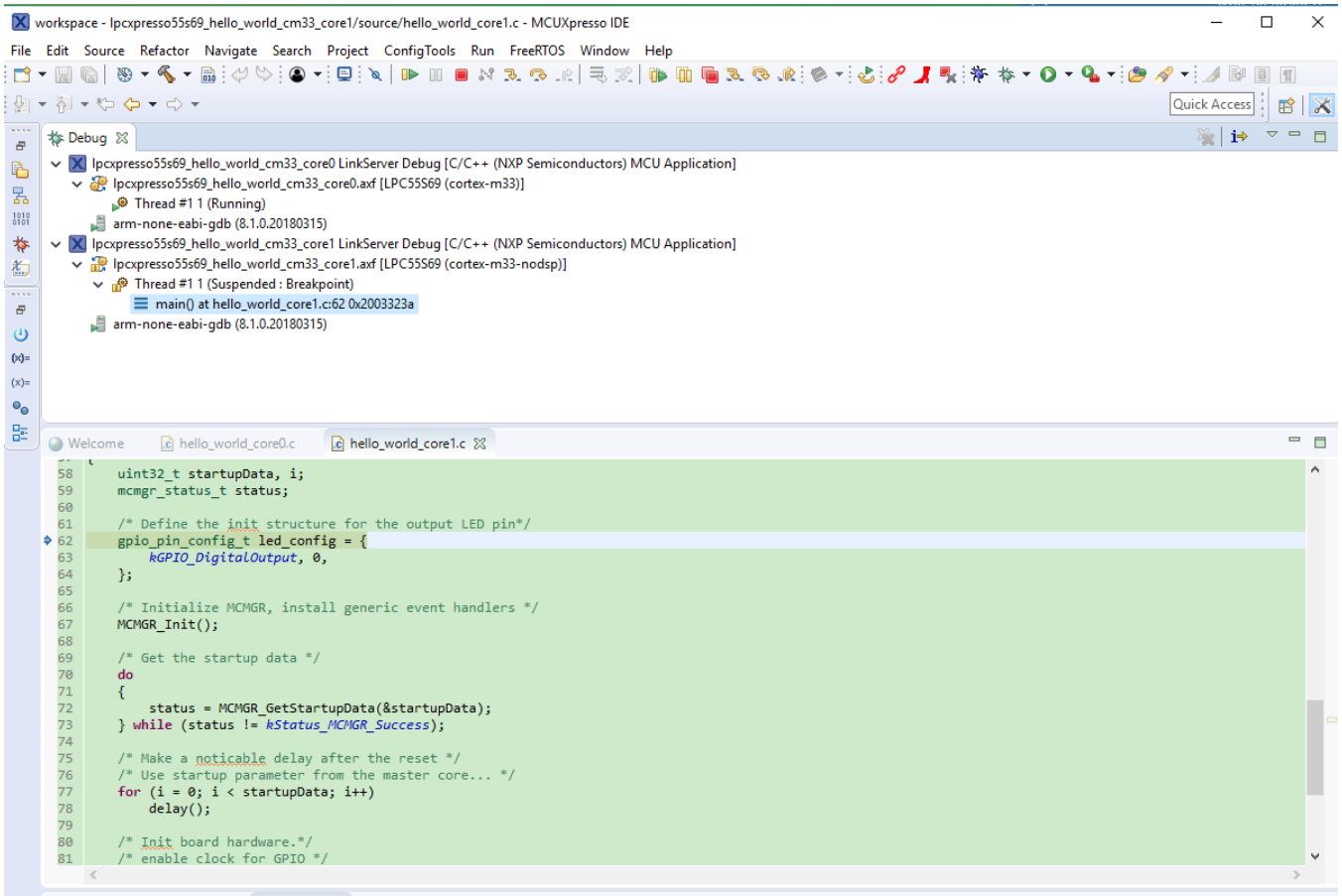

Figure 25. Auxiliary core application stops at the main function

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 the 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.

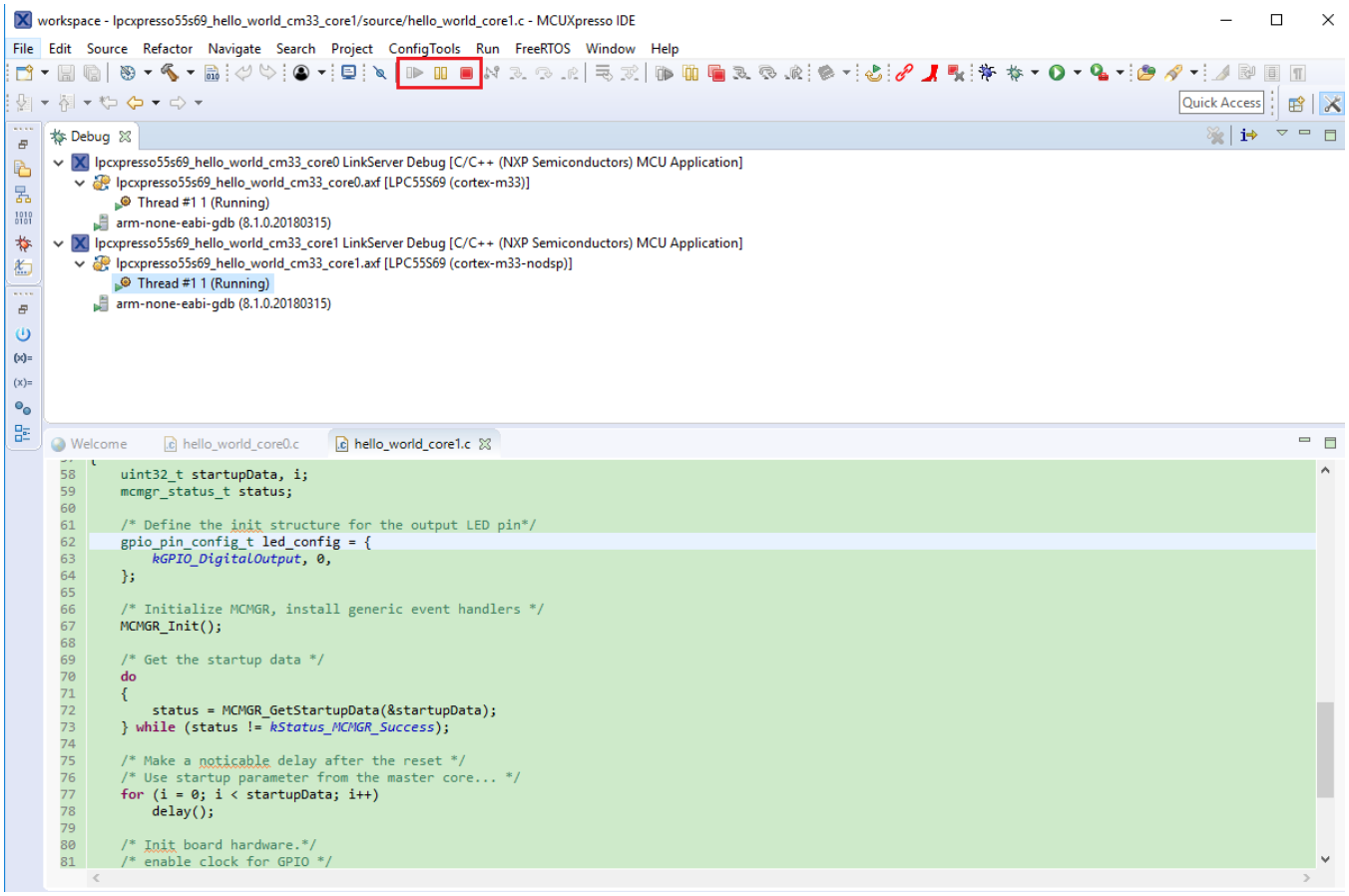

Figure 26. Synchronous suspension/resumption of both cores using the multiple selection of debug sessions and “Suspend”/"Resume” controls

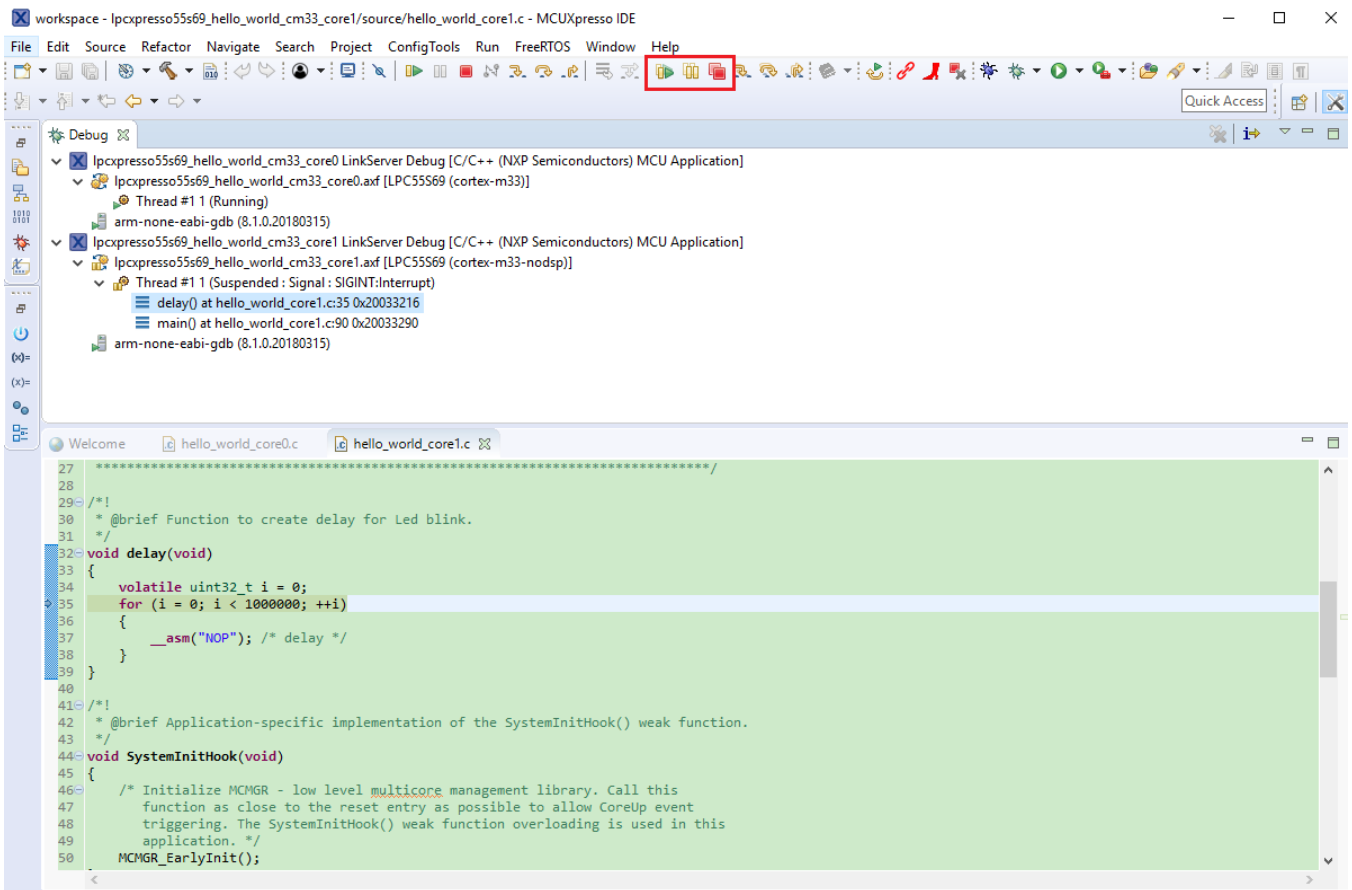

Figure 27. Synchronous suspension/resumption of both cores using the “Suspend All Debug sessions” and the “Resume All Debug sessions” controls

3.6 Build a TrustZone example application

This section describes the steps required to configure MCUXpresso IDE v11.0.1 to build, run, and debug TrustZone example applications. The trustzone version of the `hello_world` example application targeted for the LPCXpresso55S69 hardware platform is used as an example, though these steps can be applied to any TrustZone example application in the MCUXpresso SDK.

1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for LPCXpresso55S69 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 **LPC55xx** folder and select **LPC55S69**. Then, select **lpcxpresso55s69** and click **Next**.

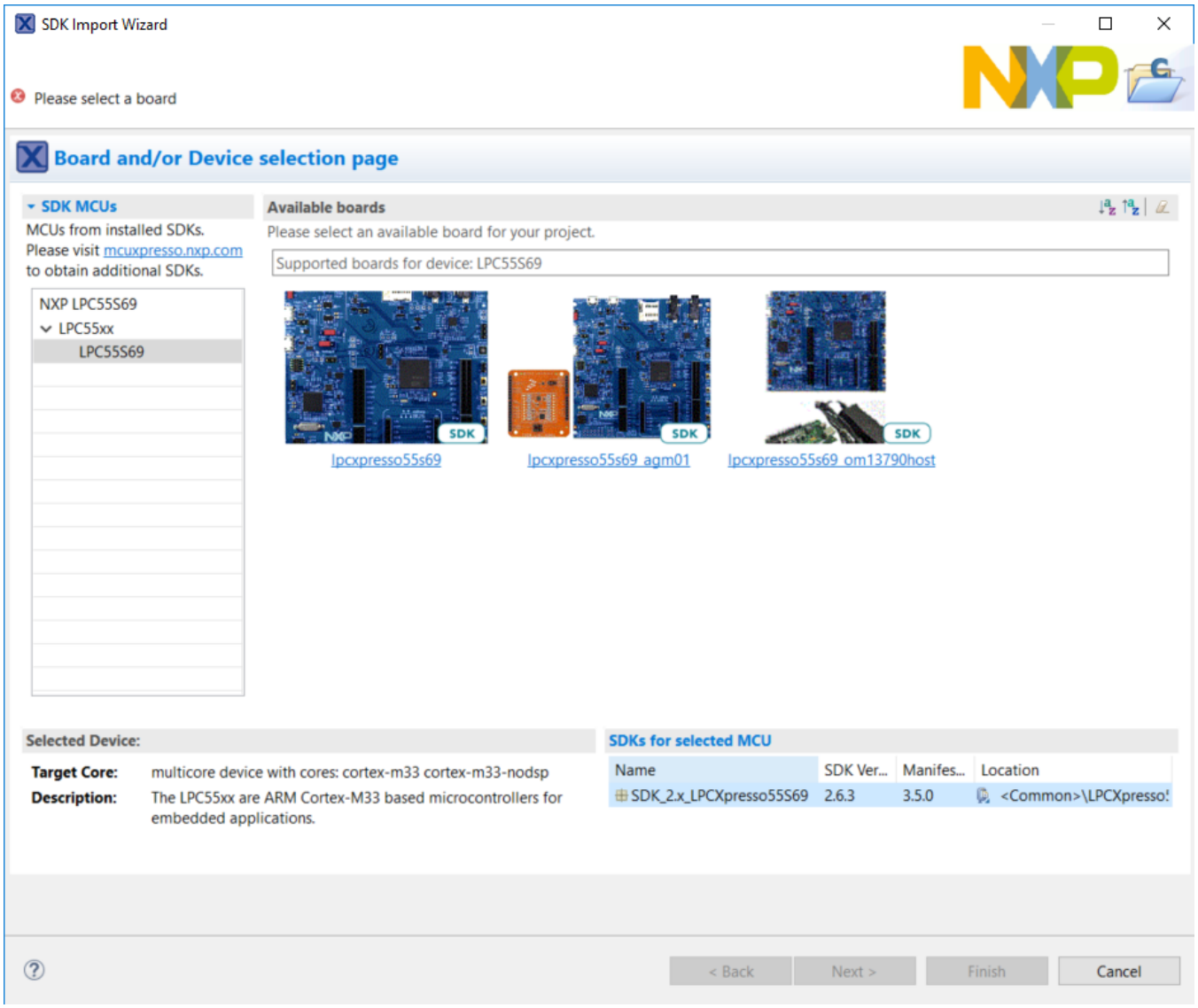

Figure 28. Select the LPCXpresso55S69 board

2. Expand the `trustzone_examples/` folder and select `hello_world_s`. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then select **UART** as SDK Debug Console. Then, click **Finish**.

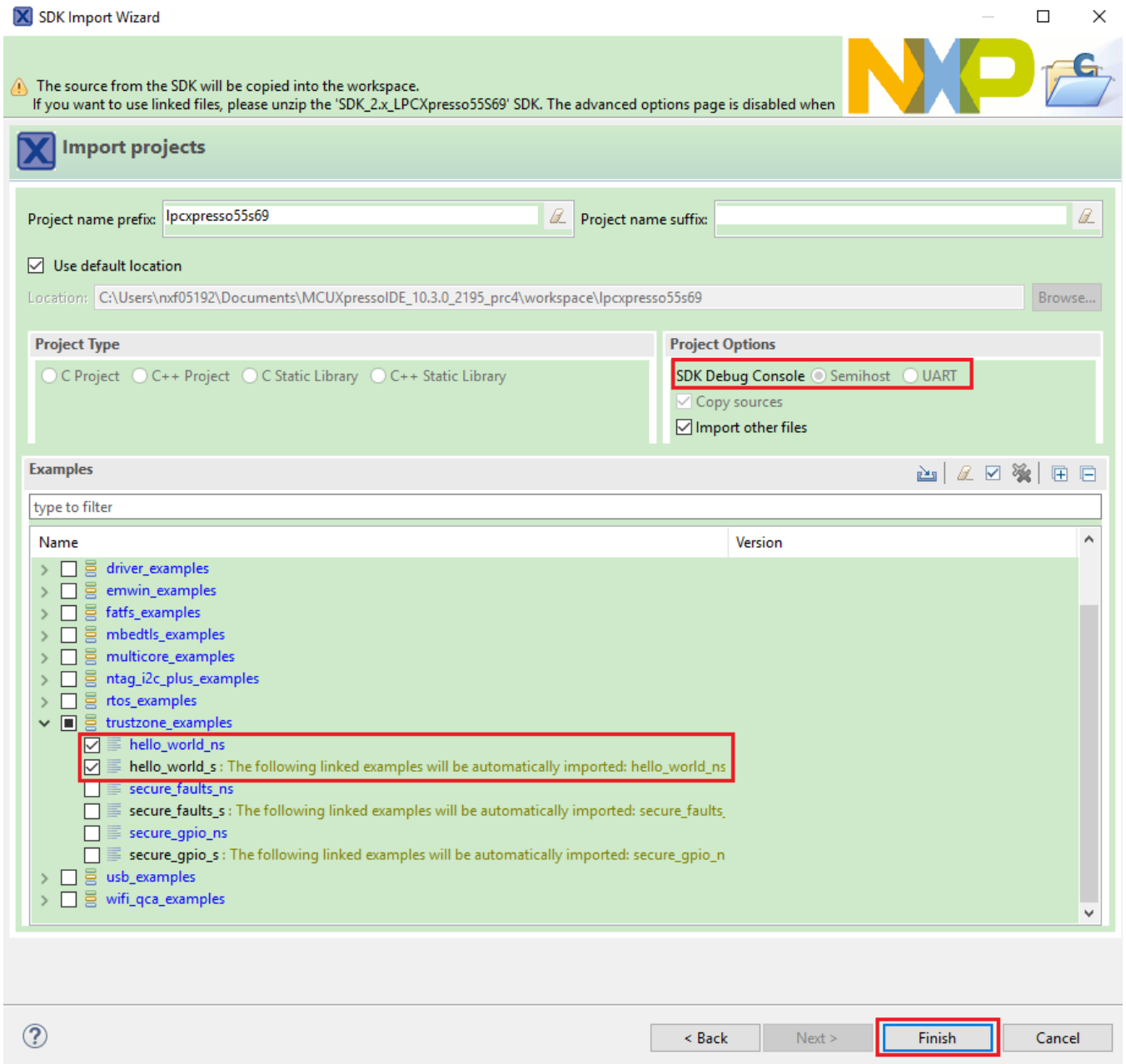

Figure 29. Select the hello_world TrustZone example

3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `lpcxpresso55s69_hello_world_s` project (TrustZone 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 in Figure 30. For this example, select the **Debug** target.

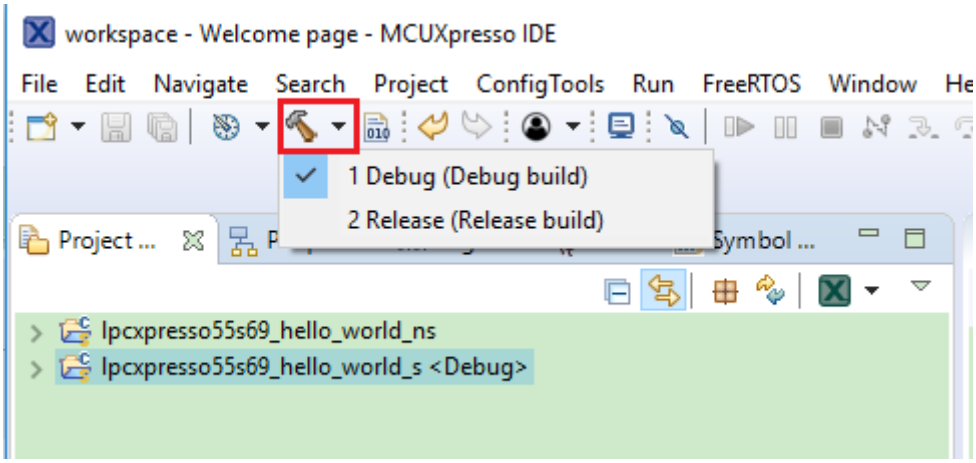

Figure 30. Selection of the build target in MCUXpresso IDE

The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when the secure project since CMSE library is not ready.

NOTE

When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click to select the second project. Right click in the Project Explorer view to display the context-sensitive menu and select **Build Configurations->Set Active- >Release**. This is also possible by using the menu item of **Project->Build Configuration- >Set Active->Release**. After switching to the **Release** build configuration, please build the application for the secure project first.

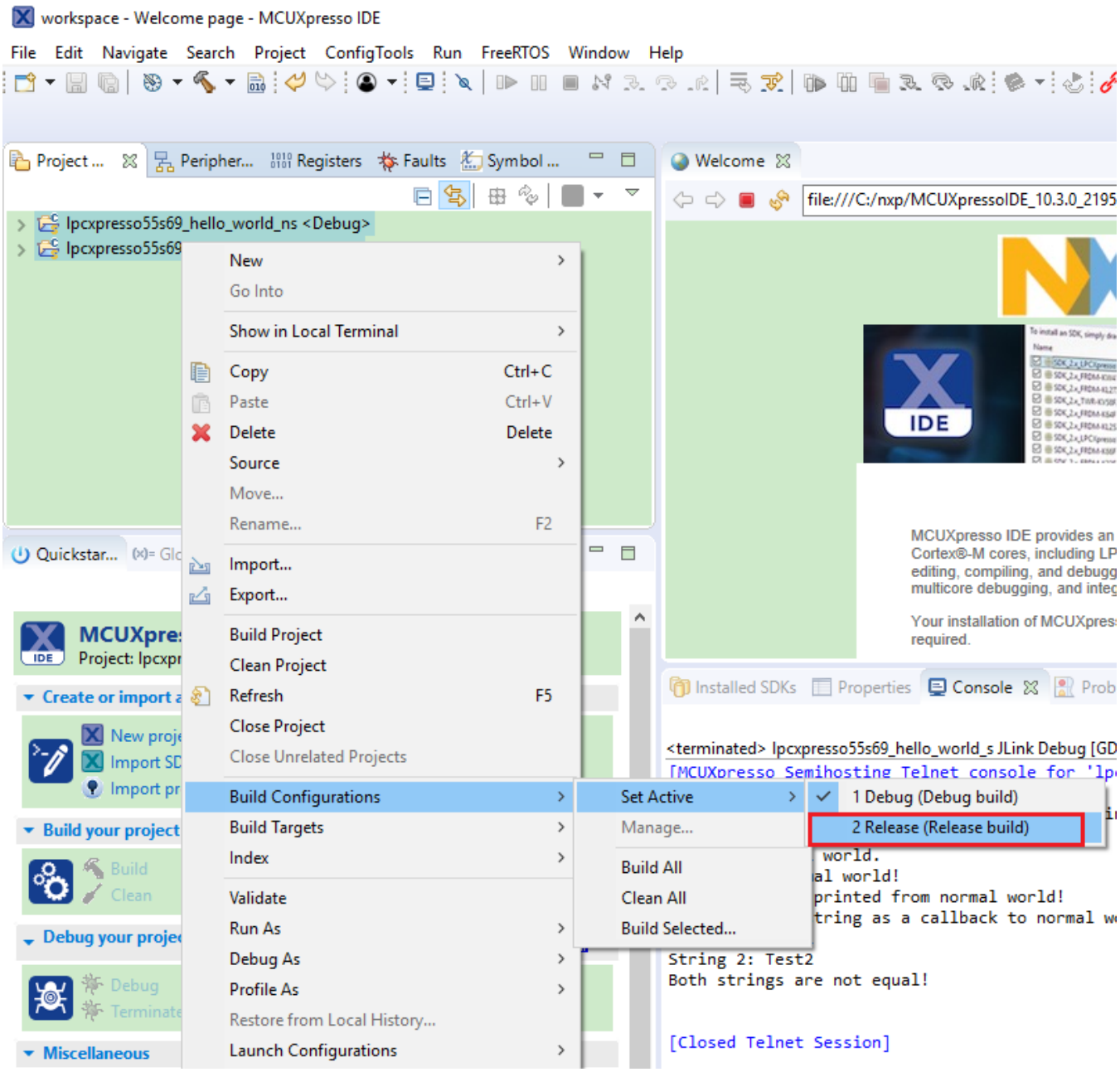

Figure 31. Switching TrustZone projects into the Release build configuration

3.7 Run a TrustZone example application

To download and run the application perform all steps as described in *Section 3.3, "Run an example application"*. These steps are common for single core, dual-core, and TrustZone applications, ensuring both sides of the TrustZone application are properly loaded and started secure application. However, there is one additional dialogue that is specific to TrustZone examples. See the following figures as reference.

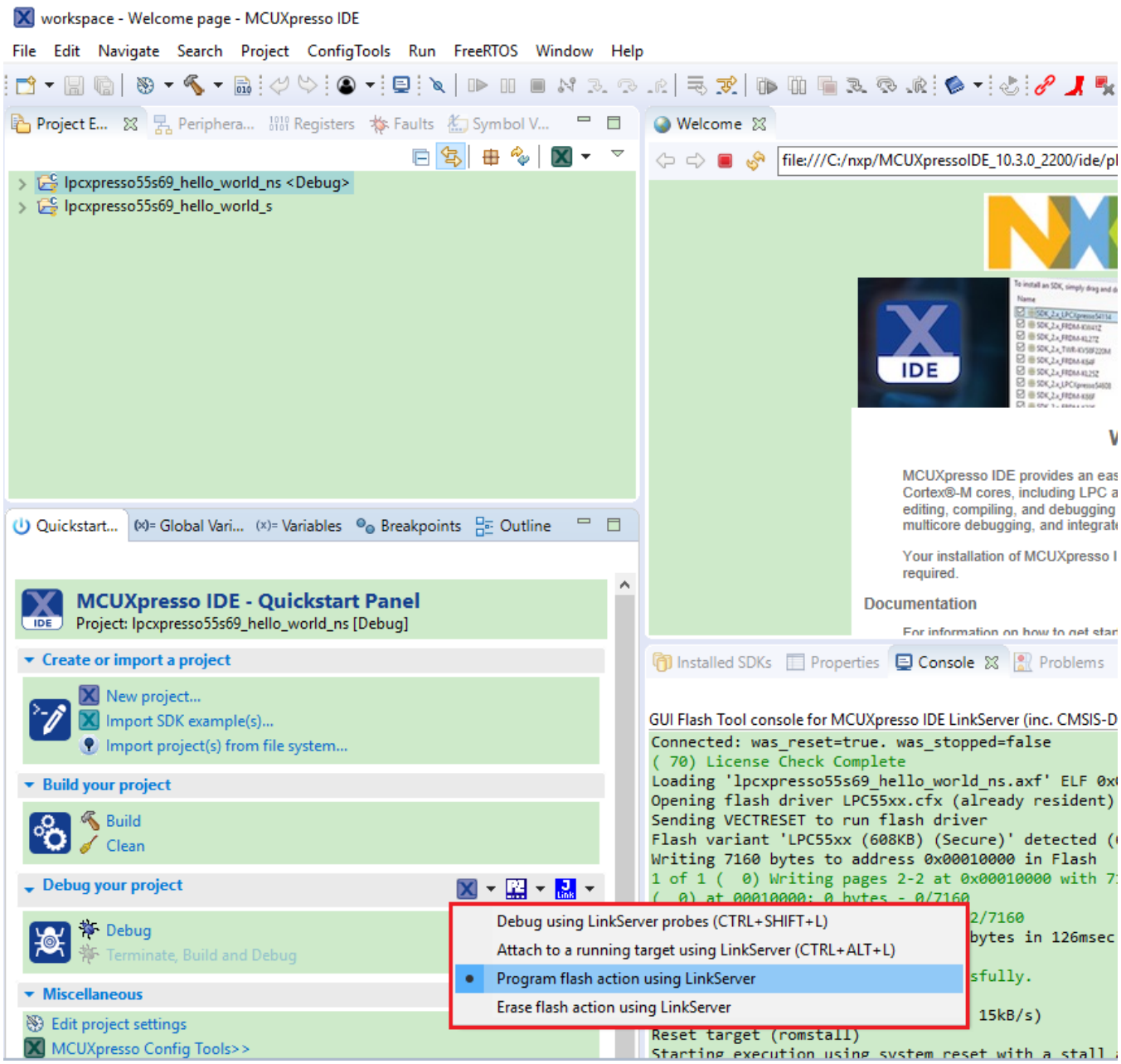

Figure 32. Load `lpcxpresso55s69_hello_world_ns` case

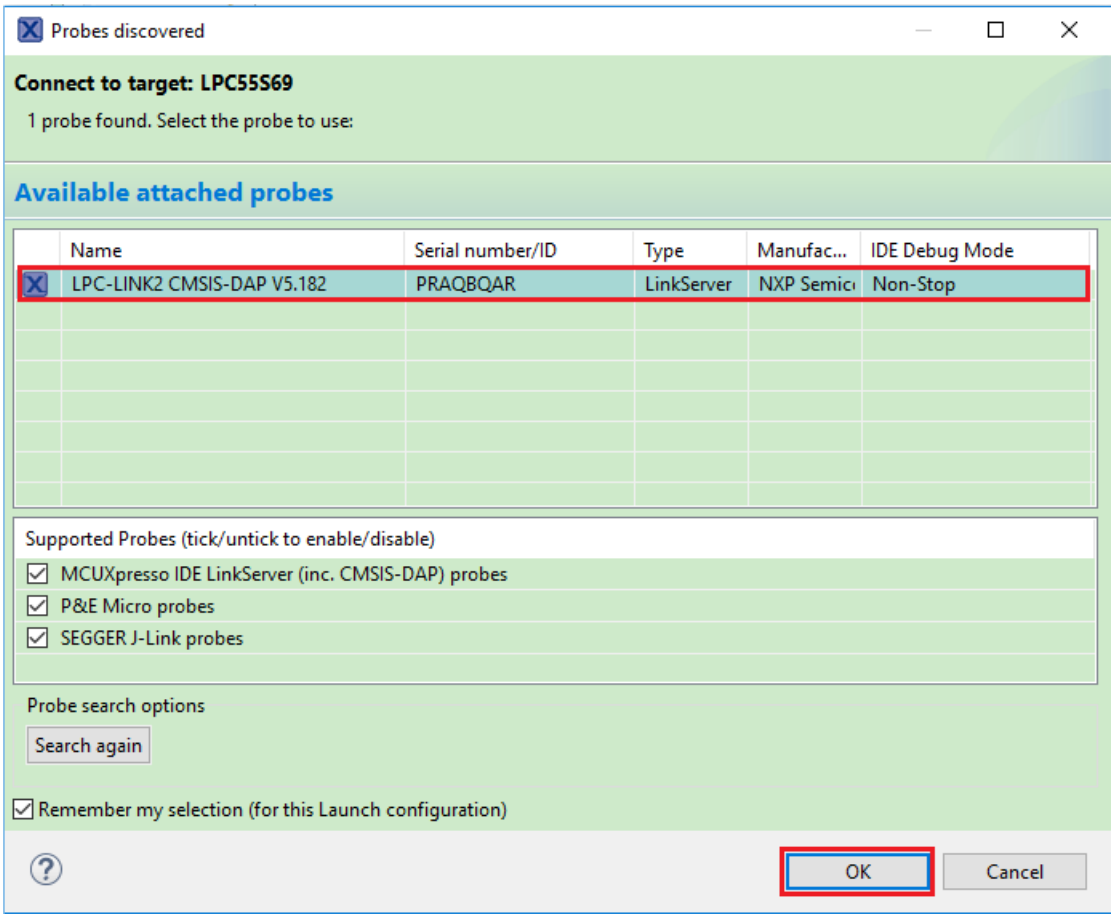

Figure 33. Attached Probes: debug emulator selection

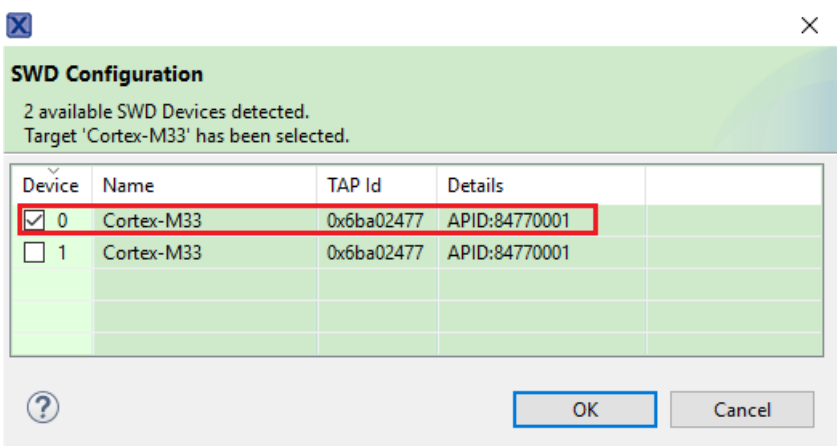

Figure 34. Target core selection dialog

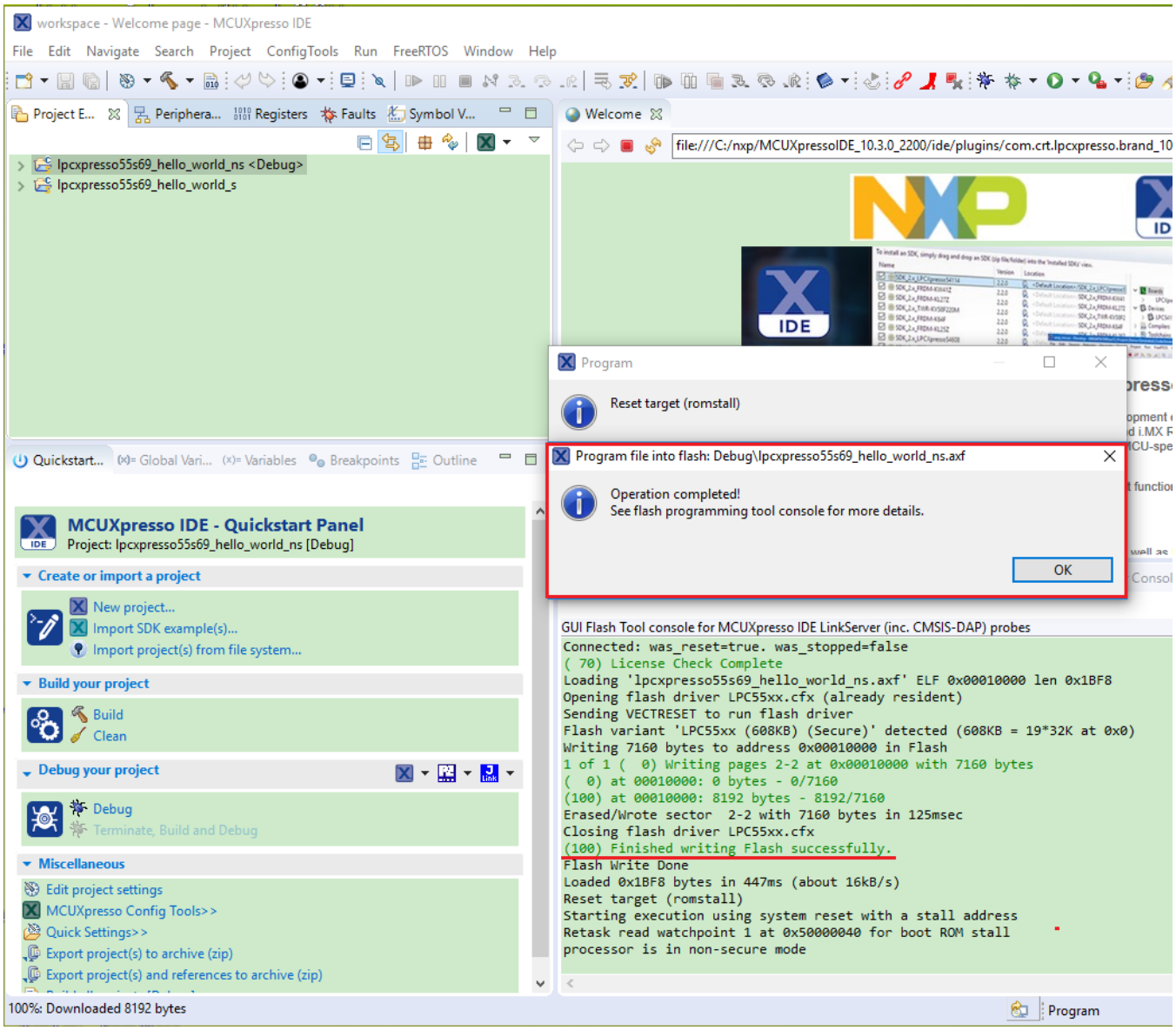

Figure 35. Load `lpcxpresso55s69_hello_world_ns.axf`

After loading the non-secure application, press **RESET** on board to release the device connect. Then, highlight the `lpcxpresso55s69_trustzone_examples_hello_world_s` project (TrustZone master project) in the Project Explorer. In the Quickstart Panel, click **lpcxpresso55s69_trustzone_examples_hello_world_s [Debug]** to launch the second debug session.

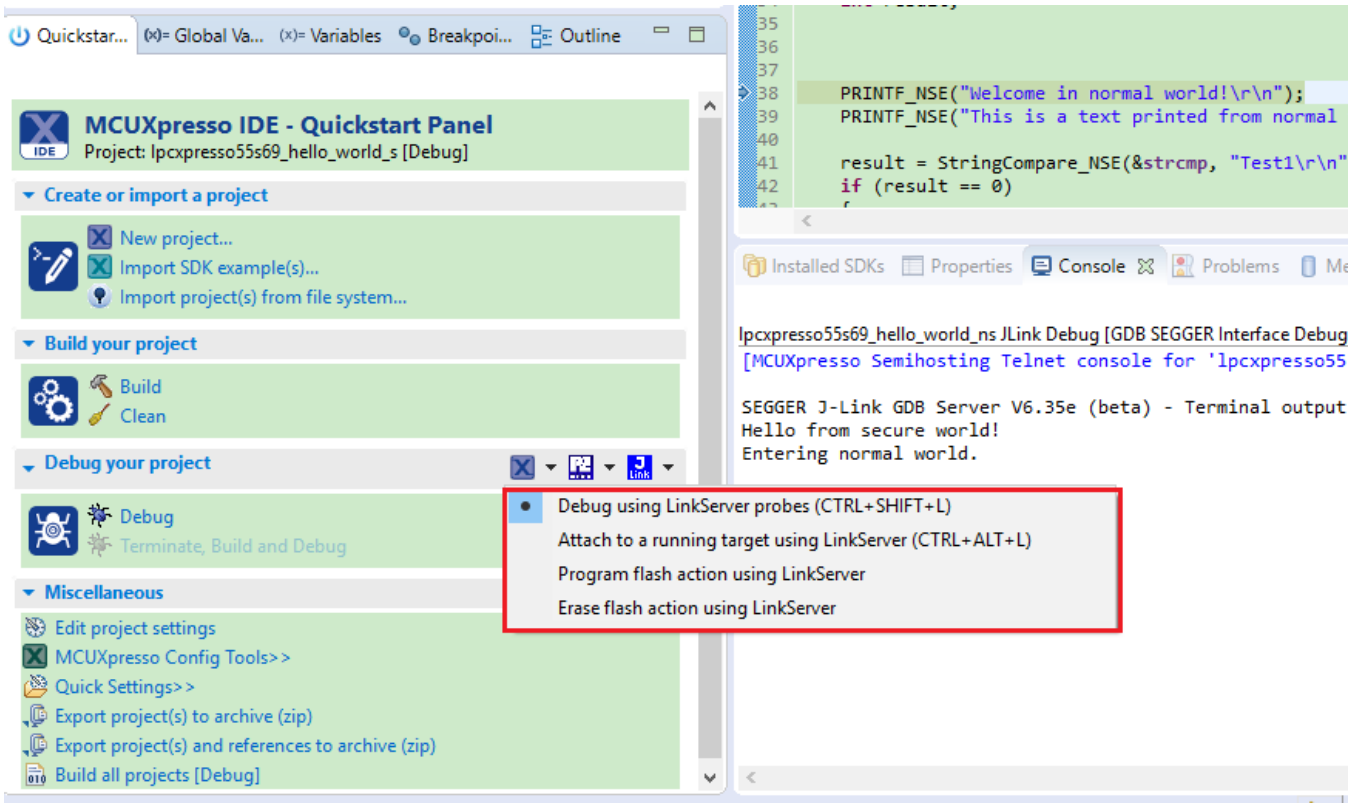

Figure 36. Debug `lpcxpresso55s69_hello_world_s` case

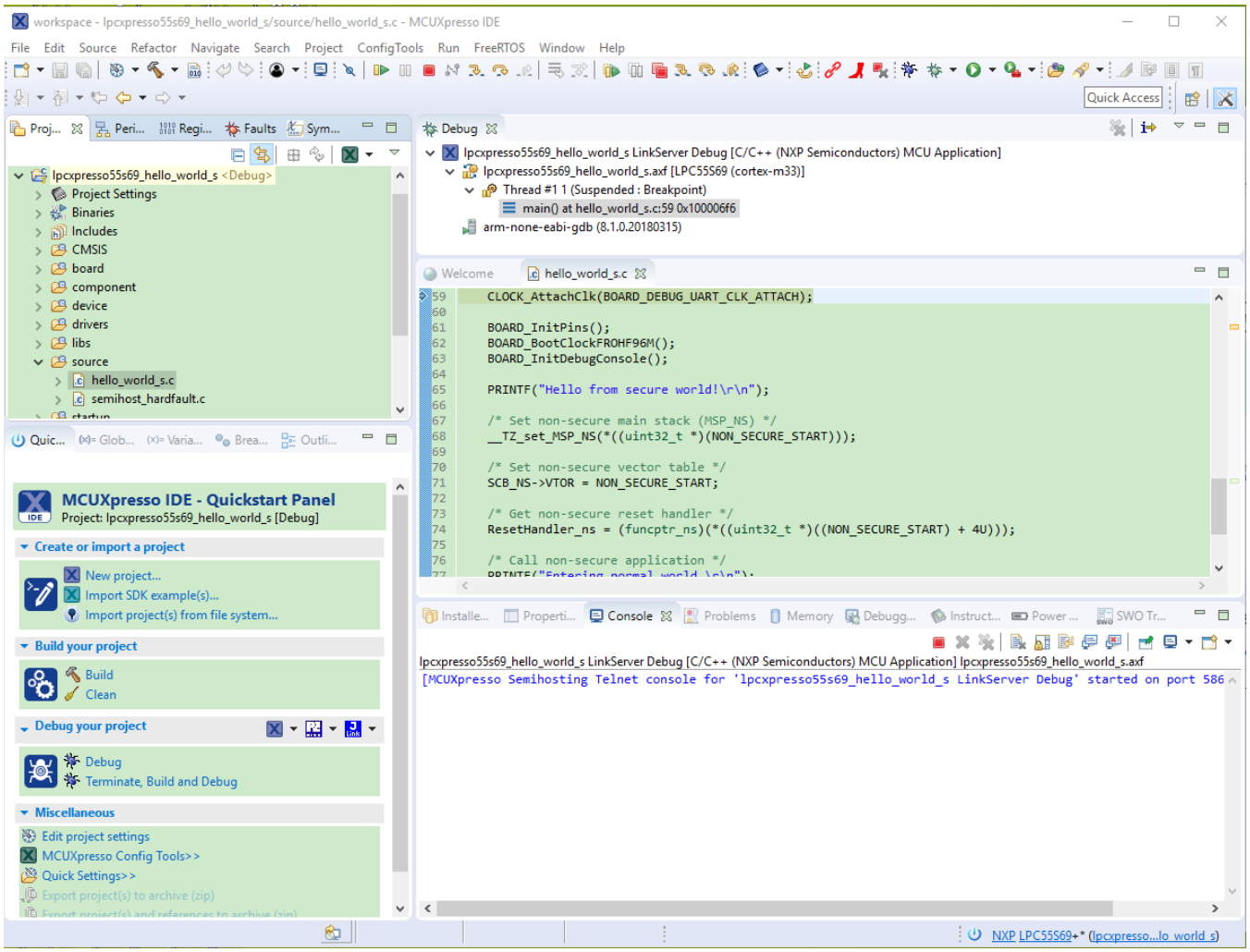

Figure 37. Debug the `hello_world_s` project

Start the application by clicking **Resume**. The `hello_world` TrustZone application then starts running, and the secure application starts the non-secure application during run time.

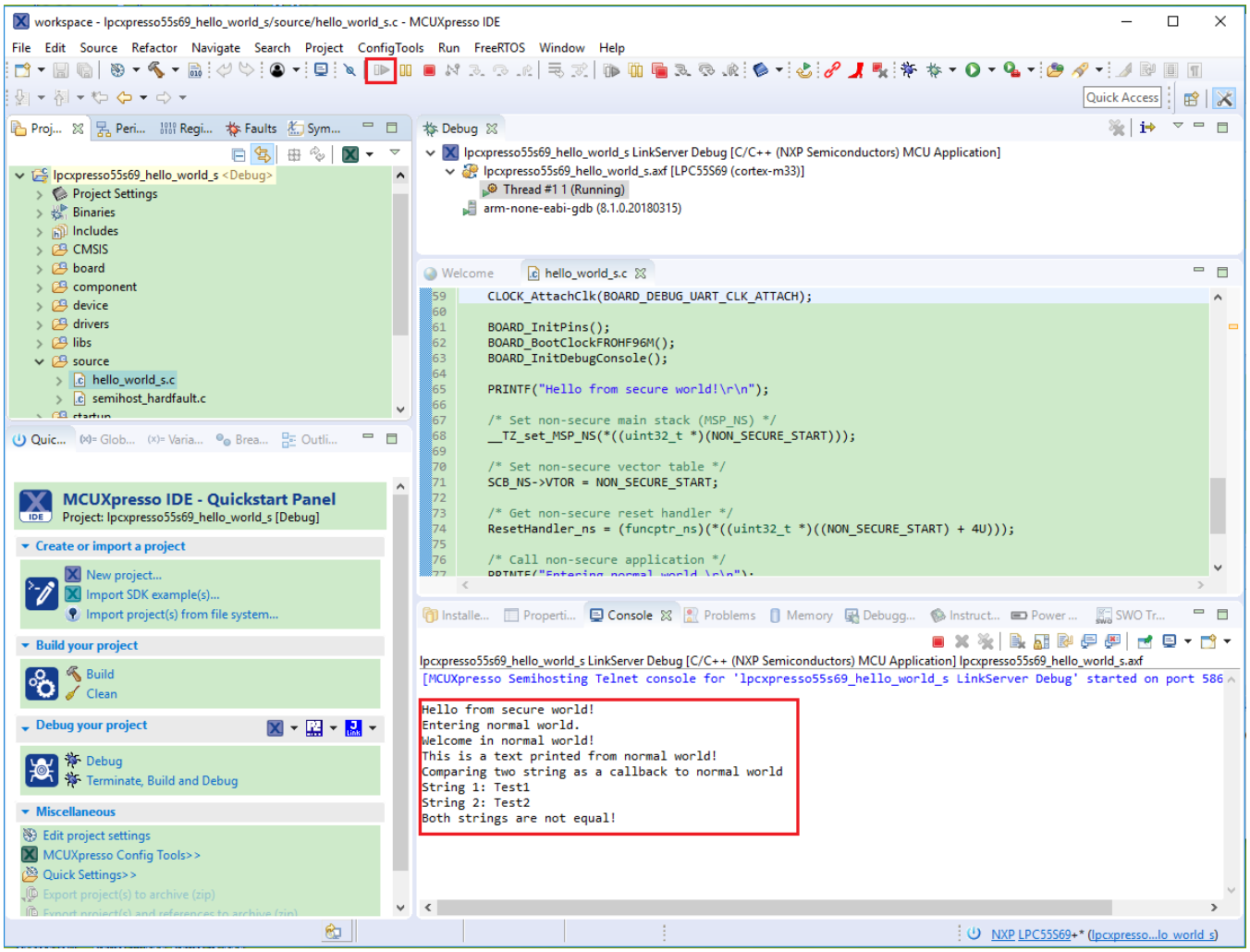

Figure 38. Run Hello World trustzone example and get the message

4 Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK.

NOTE

IAR Embedded Workbench for Arm version 8.32.1 is used in the following example, and the IAR toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes* (document ID: MCUXSDKRN).

4.1 Build an example application

Do the following steps to 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 LPCXpresso55S69 hardware platform as an example, the `hello_world` workspace is located in:

   `<install_dir>/boards/lpcxpresso55s69/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.

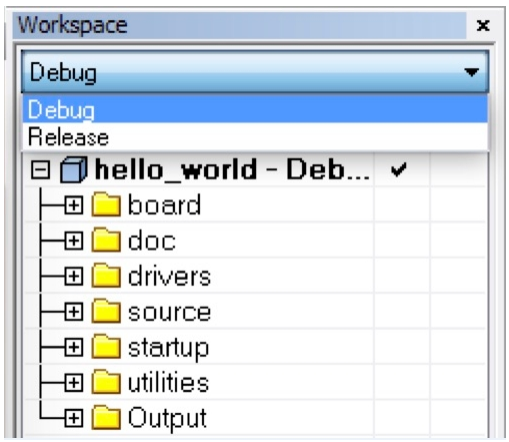

Figure 39. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red, as shown in Figure 40.

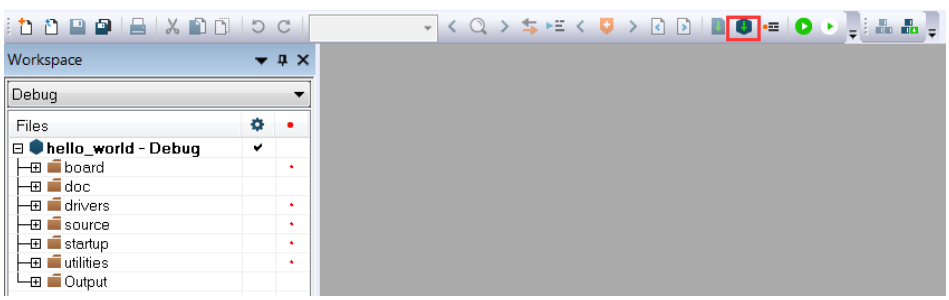

Figure 40. Build the demo application

4. The build completes without errors.

4.2 Run an example application

To download and run the application, perform these steps:

1. See the table in Default debug interfaces to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with CMSIS-DAP/mbed/DAPLink interfaces, visit developer.mbed.org/handbook/Windows-serial-configuration and follow the instructions to install the Windows® operating system serial driver. If running on Linux® OS, this step is not required.
   - For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.

2. Connect the development platform to your PC via USB cable.
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 How to determine COM port). Configure the terminal with these settings:
    a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
    b. No parity
    c. 8 data bits
    d. 1 stop bit

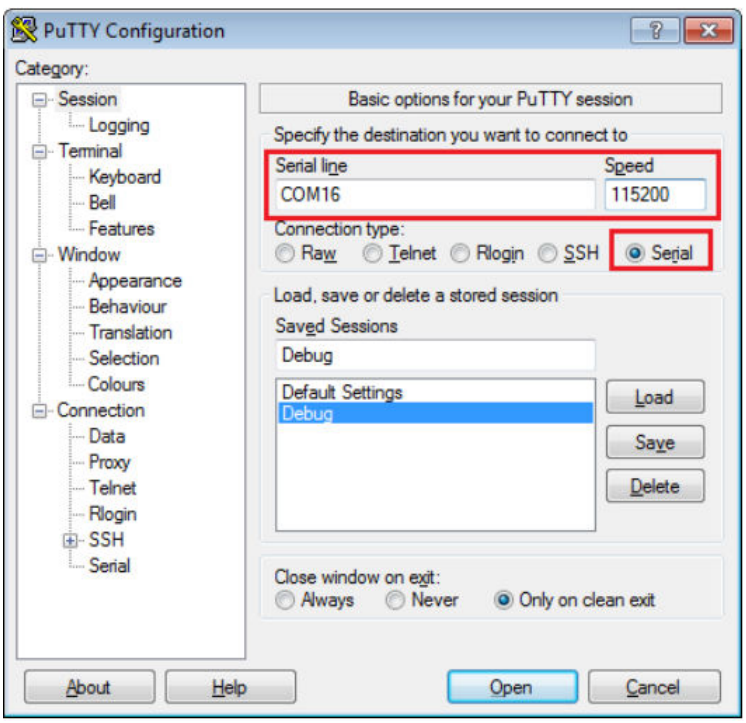

Figure 41. Terminal (PuTTY) configuration

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

Figure 42. Download and Debug button

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

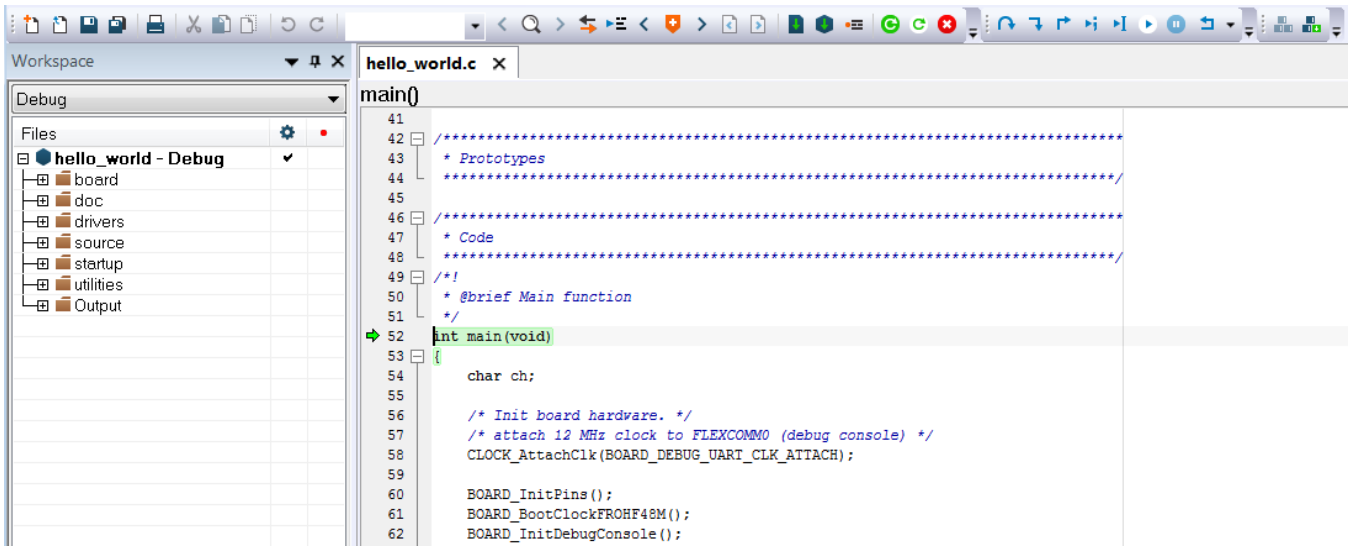

Figure 43. Stop at `main()` when running debugging

6. Run the code by clicking the **Go** button.

Figure 44. Go button

7. The `hello_world` application is now running and a banner is displayed on the terminal. If it does not appear, check your terminal settings and connections.

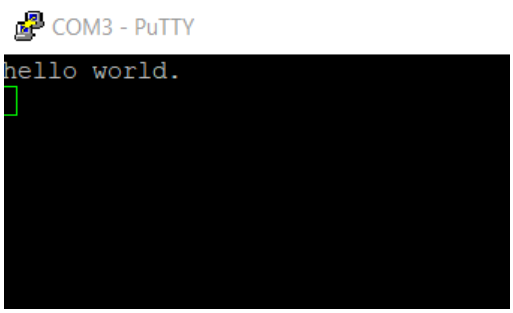

Figure 45. Text display of the `hello_world` demo

4.3 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/lpcxpresso55s69/multicore_examples/hello_world/cm33_core0/iar/hello_world_cm33_core0.eww

<install_dir>/boards/lpcxpresso55s69/multicore_examples/hello_world/cm33_core1/iar/hello_world_cm33_core1.eww

Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm33_core1) first, because the primary core application project (cm33_core0) 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.

4.4 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 46. 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.

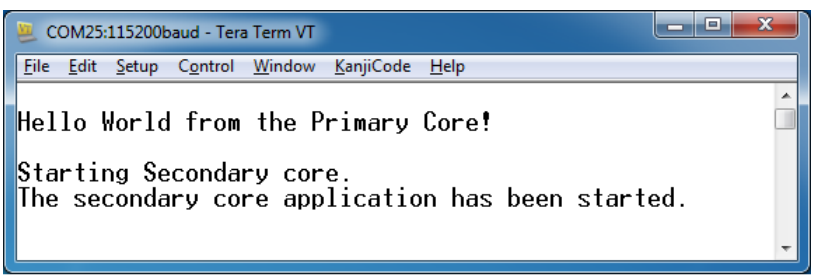

Figure 47. Hello World from primary core message

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.

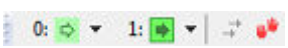

Figure 48. "Stop all cores" and "Start all cores" control buttons

4.5 Build a TrustZone example application

This section describes the particular steps that need to be done in order to build and run a TrustZone application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
<application_name>_ns/iar

<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
<application_name>_s/iar
```

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

```
<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/
hello_world_ns/iar/hello_world_ns.eww

<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/
hello_world_s/iar/hello_world_s.eww

<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/
hello_world_s/iar/hello_world.eww
```

This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since the CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project since CMSE library is not ready.

4.6 Run a TrustZone example application

The secure project is configured to download both secure and non-secure output files, so debugging can be fully managed from the secure project. To download and run the TrustZone application, switch to the secure application project and perform steps 1 – 4 as described in *Section 4.2, Run an example application*. These steps are common for both single core, dual-core, and TrustZone applications in IAR. After clicking **Download and Debug**, both the secure and non-secure image are loaded into the device flash memory, and the secure application is executed. It stops at the `Rest_Hander` function.

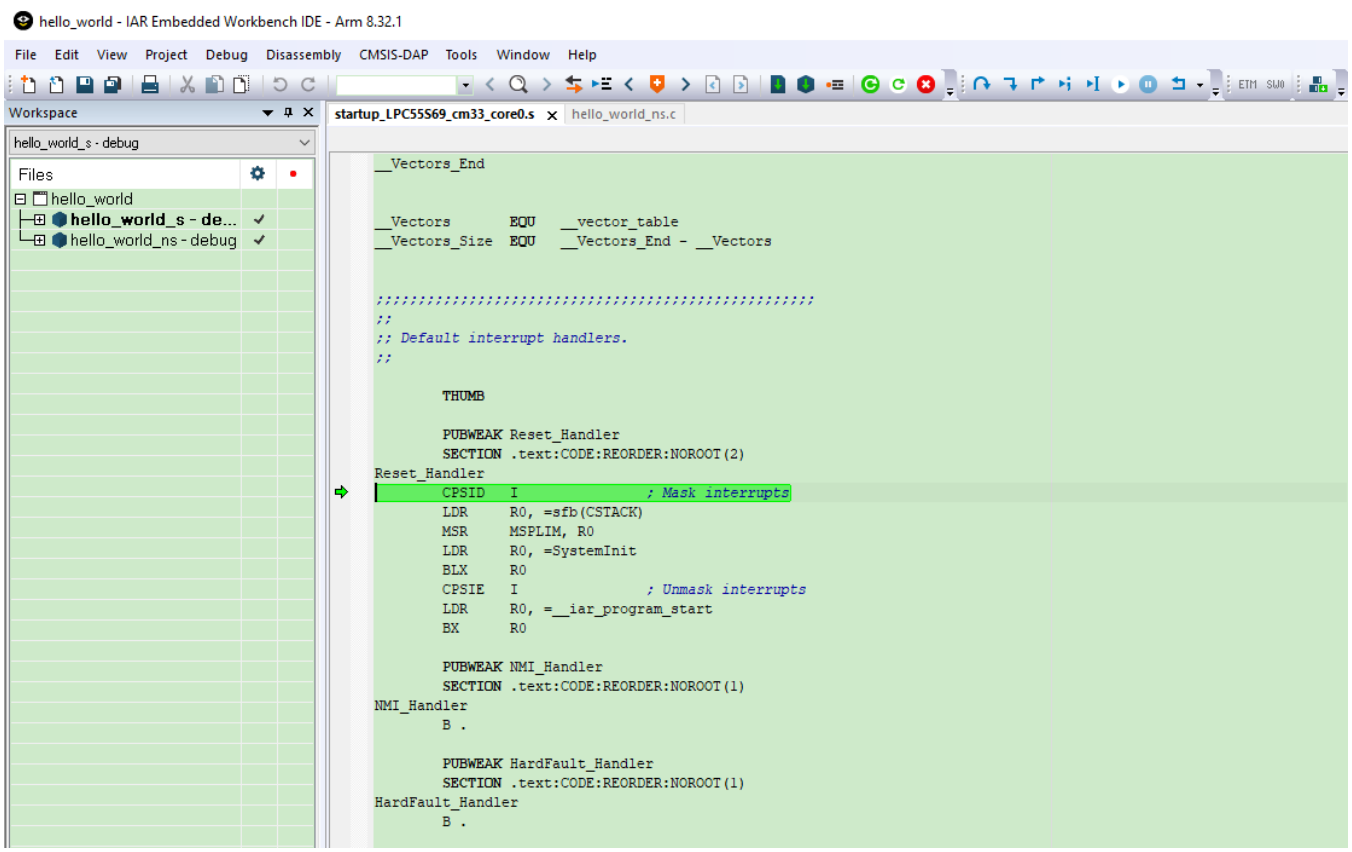

Figure 49. Stop at `Rest_Hander` when running debugging

Run the code by clicking **Go** to start the application.

Figure 50. Go button

The TrustZone `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.

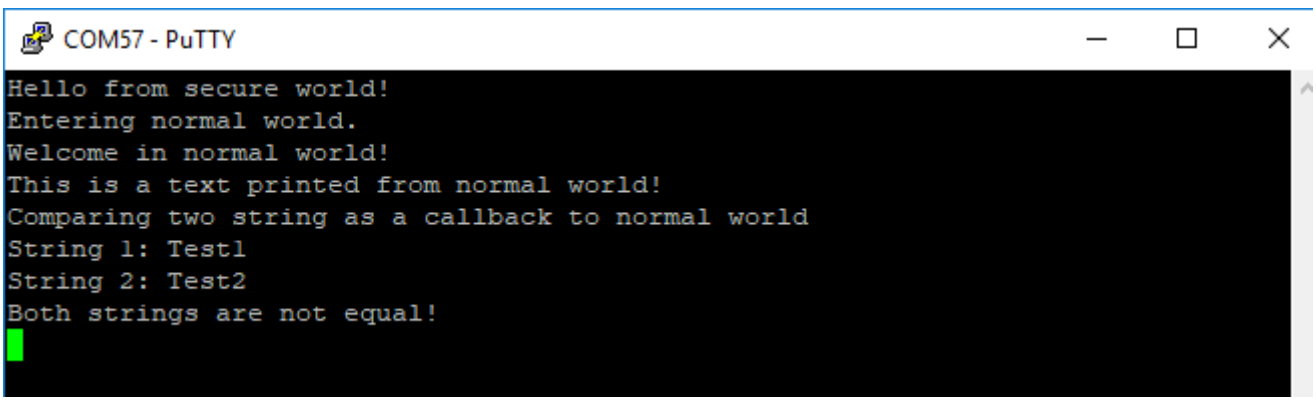

Figure 51. Text display of the `trustzone hello_world` application

5 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 LPCXpresso55S69 hardware platform is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

5.1 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.

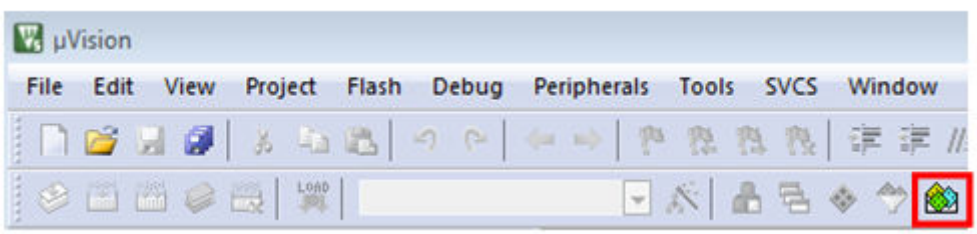

Figure 52. Launch the Pack Installer

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

5.2 Build an example application

1. Open the desired example application workspace in:

   `<install_dir>/boards/<board_name>/<example_type>/<application_name>/mdk`

   The workspace file is named as `<demo_name>.uvmpw`. For this specific example, the actual path is:

   `<install_dir>/boards/lpcxpresso55s69/demo_apps/hello_world/mdk/hello_world.uvmpw`
2. To build the demo project, select **Rebuild**, highlighted in red.

Figure 53. Build the demo

3. The build completes without errors.

5.3 Run an example application

To download and run the application, perform these steps:

1. See the table in Default debug interfaces to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with the CMSIS-DAP/mbed/DAPLink interface, visit mbed Windows serial configuration and follow the instructions to install the Windows operating system serial driver. If running on Linux OS, this step is not required.
   - For boards with a P&E Micro interface, visit www.pemicro.com/support/downloads_find.cfm and download and install the P&E Micro Hardware Interface Drivers package.
   - If using J-Link either a standalone debug pod or OpenSDA, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.
   - If using J-Link either a standalone debug pod or J-link firmware programmed into the on-board debug probe, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.
   - For boards with the OSJTAG interface, install the driver from www.keil.com/download/docs/408.
2. Connect the development platform to your PC via USB cable using OpenSDA USB connector.
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 How to determine COM port). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

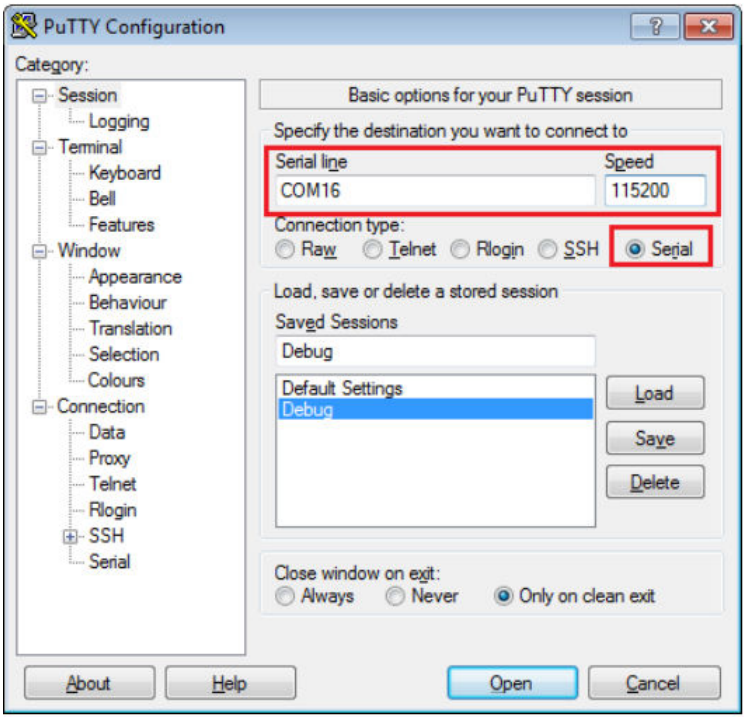

Figure 54. Terminal (PuTTY) configurations

4. In µVision, after the application is built, click the **Download** button to download the application to the target.
5. After clicking the **Download** button, the application downloads to the target and is running. To debug the application, click the **Start/Stop Debug Session** button, highlighted in red.

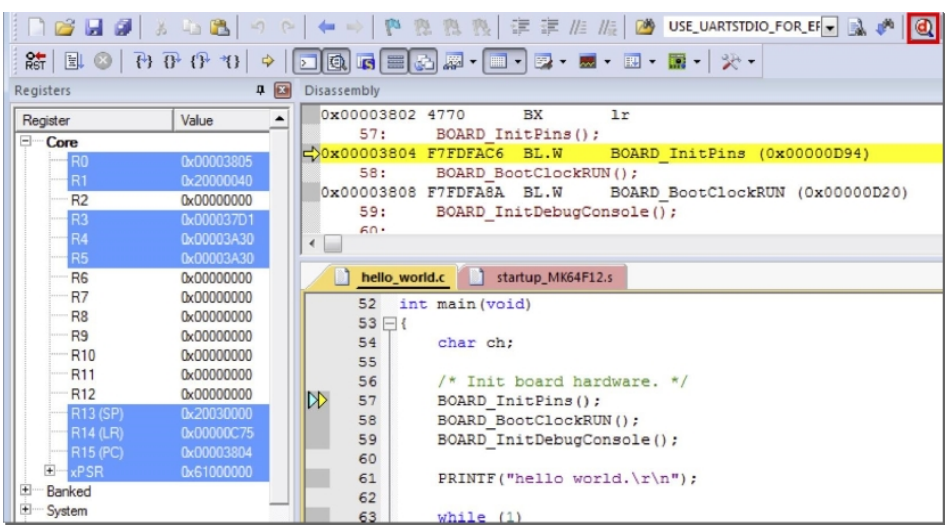

Figure 55. Stop at `main()` when run debugging

6. Run the code by clicking the **Run** button to start the application.

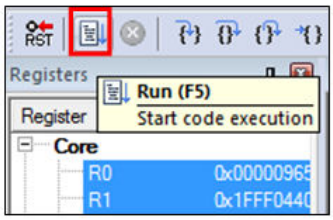

Figure 56. Go button

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

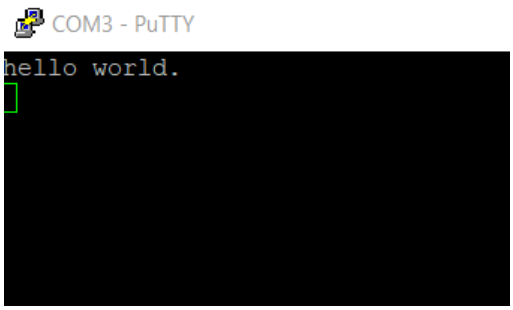

Figure 57. Text display of the hello_world demo

NOTE

Must use J-Link v6.56a or higher version in Keil. When using J-Link on Keil, if the device is blank (flash is erased), first uncomment the line16 in the example/mdk/JLinkSettings.JLinkScript to release the function void ResetTarget(void){}. Then do the download or debug, comment the function void ResetTarget(void){} again.

5.4 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/lpcxpresso55s69/multicore_examples/hello_world/cm33_core0/mdk/hello_world_cm33_core0.uvmpw

<install_dir>/boards/lpcxpresso55s69/multicore_examples/hello_world/cm33_core1/mdk/hello_world_cm33_core1.uvmpw

Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm33_core1) first because the primary core application project (cm33_core0) 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.

5.5 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.

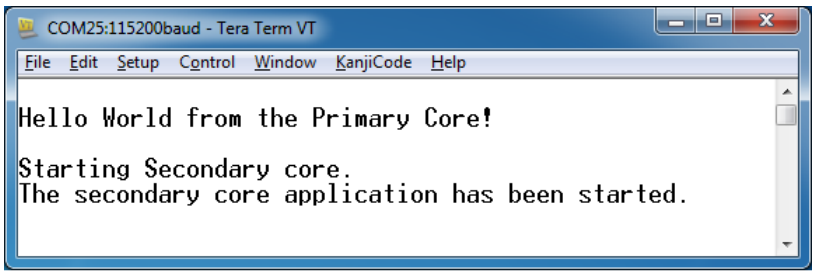

Figure 58. Hello World from primary core message

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.

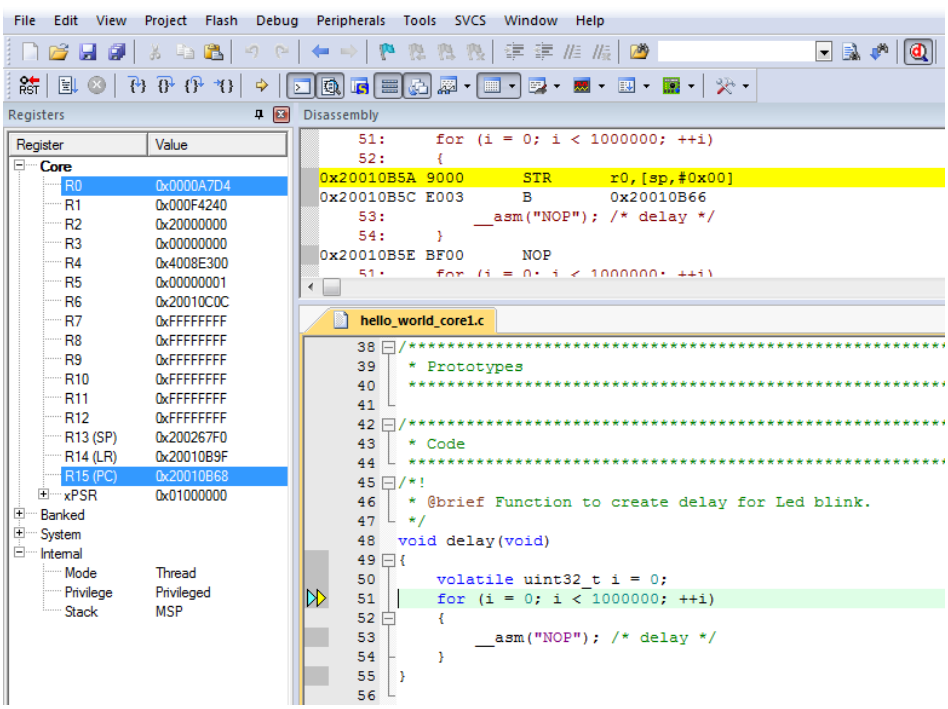

Figure 59. Debugging the auxiliary core application

Arm describes multi-core debugging using the NXP LPC54114 Cortex-M4/M0+ dual-core processor and Keil uVision IDE in Application Note 318 at www.keil.com/appnotes/docs/apnt_318.asp. The associated video can be found here.

5.6 Build a TrustZone example application

This section describes the particular steps that need to be done in order to build and run a TrustZone application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
<application_name>_ns/mdk

<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
<application_name>_s/mdk
```

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

```
<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/
hello_world_ns/mdk/hello_world_ns.uvmpw

<install_dir>/boards/lpcxpresso55s16/trustzone_examples/hello_world/hello_world_s/iar/
hello_world_s.eww

<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/
hello_world_s/mdk/hello_world.uvmpw
```

This project `hello_world.uvmpw` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

Build both applications separately by clicking **Rebuild**. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because CMSE library is not ready.

5.7 Run a TrustZone example application

The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

To download and run the TrustZone application, switch to the secure application project and perform steps 1 – 5 as described in *Section 6.3, "Run an example application"*. These steps are common for single core, dual-core, and TrustZone applications in µVision. After clicking **Download and Debug**, both the secure and non-secure image are loaded into the device flash memory, and the secure application is executed. It stops at the `Rest_Hander` function.

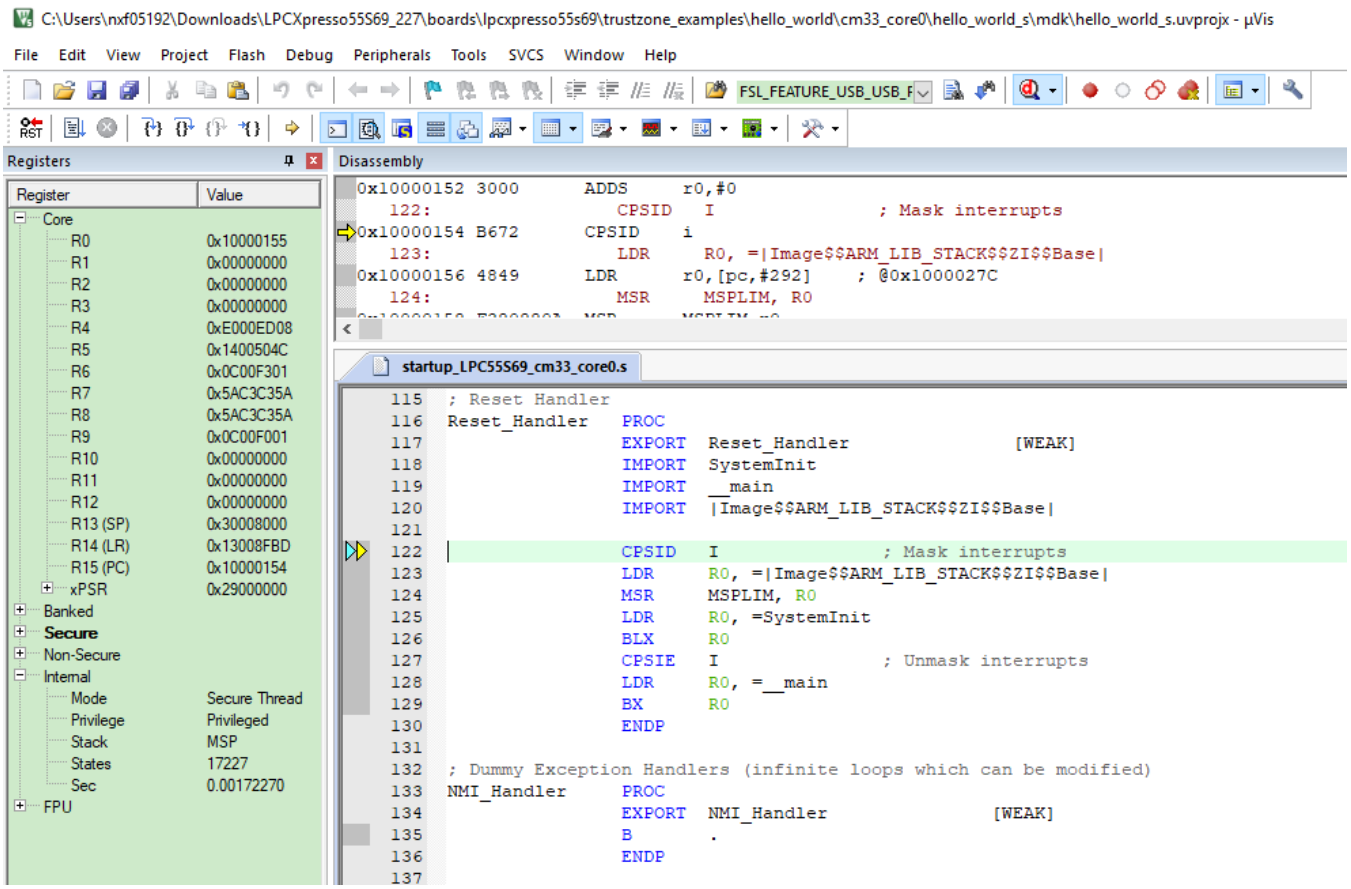

Figure 60. Stop at `Rest_Hander` when running debugging

Run the code by clicking **Run** to start the application.

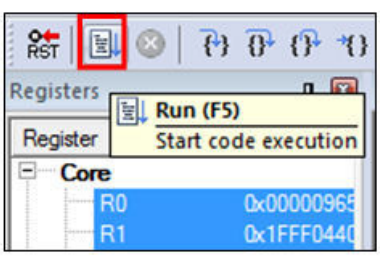

Figure 61. Go button

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

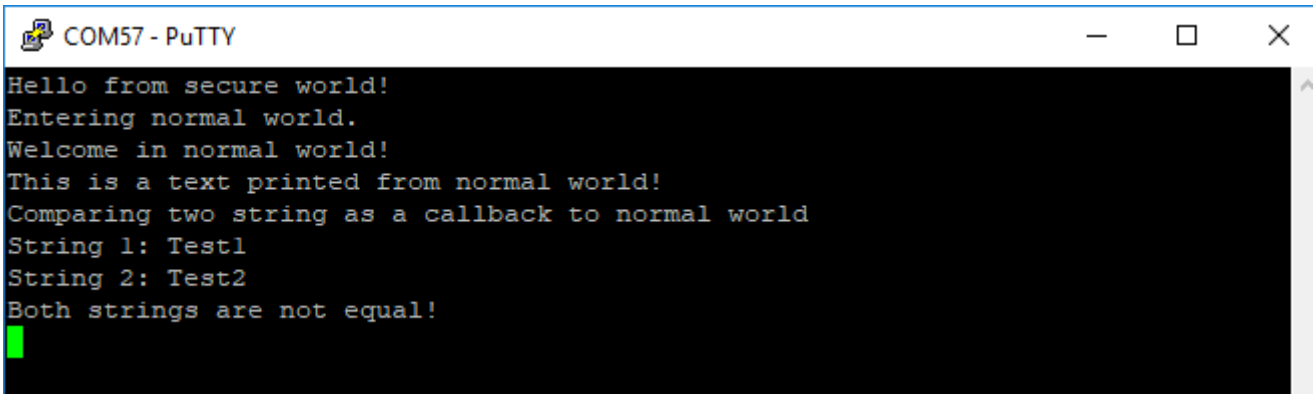

Figure 62. Text display of the trustzone `hello_world` application

6 Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The hello_world demo application is targeted for the LPCXpresso55S69 hardware platform which is used as an example.

NOTE

ARMGCC version 7-2018-q2 is used as an example in this document. The latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes for LPCXpresso55S69* (document MCUXSDKLPC55XXRN).

6.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run an MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK. There are many ways to use Arm GCC tools, but this example focuses on a Windows operating system environment.

6.1.1 Install GCC Arm Embedded tool chain

Download and run the installer from developer.arm.com/open-source/gnu-toolchain/gnu-rm. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes* (document ID: MCUXSDKRN).

6.1.2 Install MinGW (only required on Windows OS)

The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third-party C-Runtime DLLs (such as Cygwin). The build environment used by the MCUXpresso SDK does not use the MinGW build tools, but does leverage the base install of both MinGW and MSYS. MSYS provides a basic shell with a Unix-like interface and tools.

1. Download the latest MinGW mingw-get-setup installer from sourceforge.net/projects/mingw/files/Installer/.
2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.

NOTE

The installation path cannot contain any spaces.

3. Ensure that the **mingw32-base** and **msys-base** are selected under **Basic Setup**.

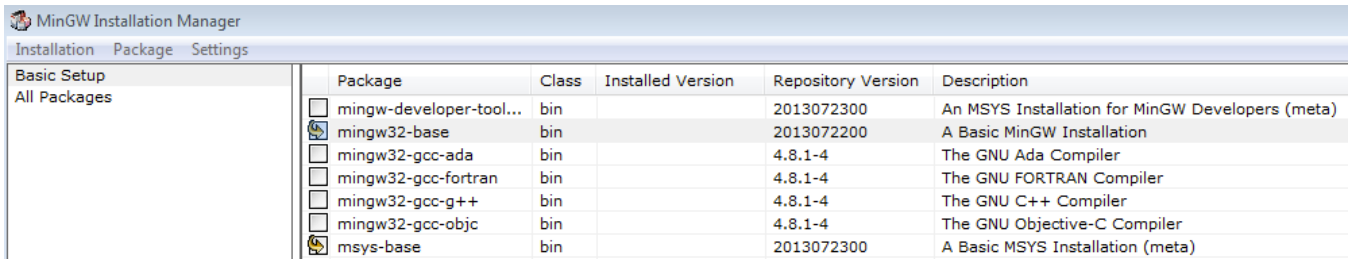

Figure 63. Set up MinGW and MSYS

4. In the **Installation** menu, click **Apply Changes** and follow the remaining instructions to complete the installation.

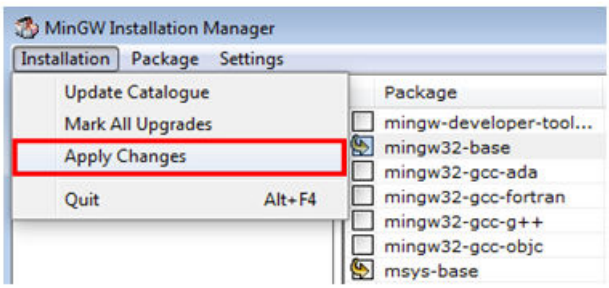

Figure 64. Complete MinGW and MSYS installation

5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

   `<mingw_install_dir>\bin`

   Assuming the default installation path,`C:\MinGW`, an example is shown below. If the path is not set correctly, the toolchain will not not work.

NOTE

If you have `C:\MinGW\msys\x.x\bin` in your PATH variable (as required by Kinetis SDK 1.0.0), remove it to ensure that the new GCC build system works correctly.

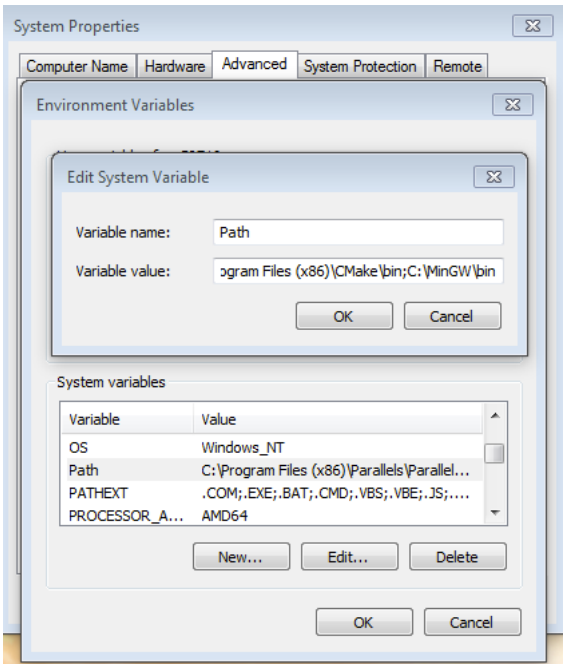

Figure 65. Add Path to systems environment

6.1.3 Add a new system environment variable for `ARMGCC_DIR`

Create a new *system* environment variable and name it as `ARMGCC_DIR`. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

```
C:\Program Files (x86)\GNU Tools ARM Embedded\7-2018-q2
```

See the installation folder of the GNU Arm GCC Embedded tools for the exact path name of your installation.

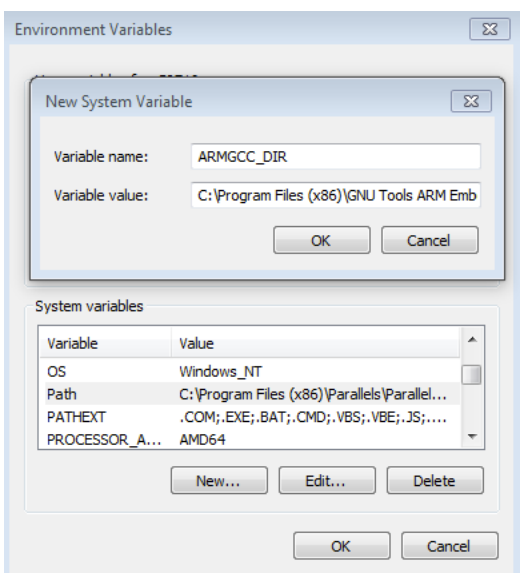

Figure 66. Add `ARMGCC_DIR` system variable

6.1.4 Install CMake

1. Download CMake 3.0.x from www.cmake.org/cmake/resources/software.html.
2. Install CMake, ensuring that the option **Add CMake to system PATH** is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.

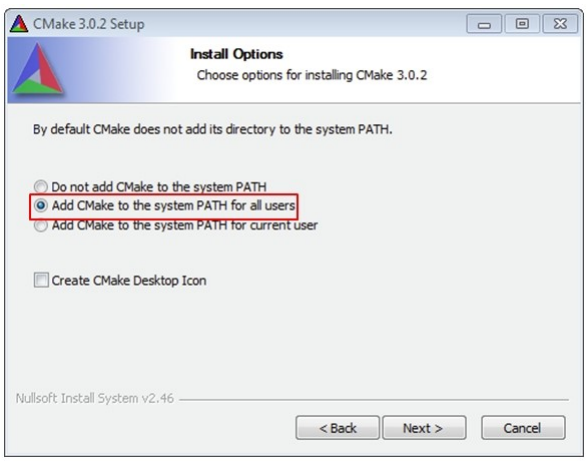

Figure 67. Install CMake

3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure `sh.exe` is not in the Environment Variable PATH. This is a limitation of `mingw32-make`.

6.2 Build an example application

To build an example application, follow these steps.

1. Open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to **Programs** ->**GNU Tools ARM Embedded <version>** and select **GCC Command Prompt**.

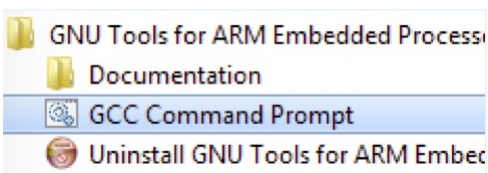

Figure 68. Launch command prompt

2. Change the directory to the example application project directory which has a path similar to the following:

   ```
   <install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc
   ```

   For this example, the exact path is:

   ```
   <install_dir>/examples/lpcxpresso55s69/demo_apps/hello_world/armgcc
   ```

NOTE

To change directories, use the `cd` command.

3. Type **build_debug.bat** on the command line or double click on **build_debug.bat** file in Windows Explorer to build it. The output is shown in this figure:

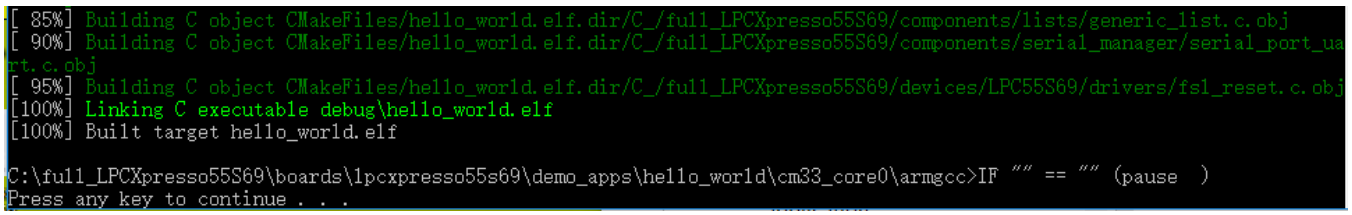

Figure 69. `hello_world` demo build successful

6.3 Run an example application

This section describes steps to run a demo application using J-Link GDB Server application. To complete the set-up check if your board supports OpenSDA in Default debug interfaces

If your board supports OpenSDA

- The OpenSDA interface on your board is pre-programmed with the J-Link OpenSDA firmware.
- For instructions on reprogramming the OpenSDA interface, see Updating OpenSDA firmware.

If your board does not support OpenSDA

- A standalone J-Link pod is required which should be connected to the debug interface of your board.

NOTE

Some hardware platforms require hardware modification in order to function correctly with an external debug interface.

NOTE

J-Link GDB Server application is not supported for TFM examples. Use CMSIS DAP instead of J-Link for flashing and debugging TFM examples.

After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector. If using a standalone J-Link debug pod, connect it to the SWD/JTAG connector of the board.
2. 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 How to determine COM port). 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

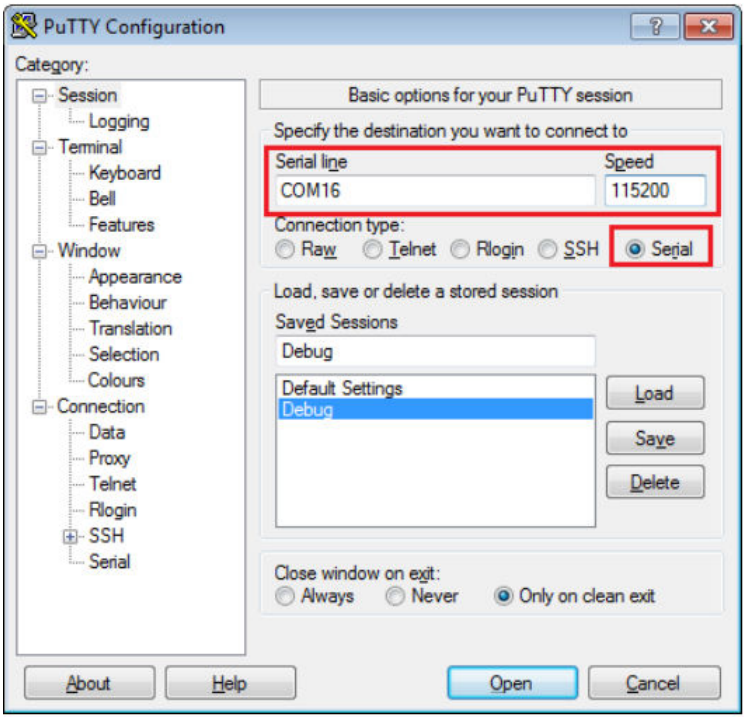

Figure 70. Terminal (PuTTY) configurations

3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting **Programs** -> **SEGGER** -> **J-Link <version> J-Link GDB Server**.
4. Modify the settings as shown below. The target device selection chosen for this example is **LPC55S69**.
5. After it is connected, the screen should look like this figure:

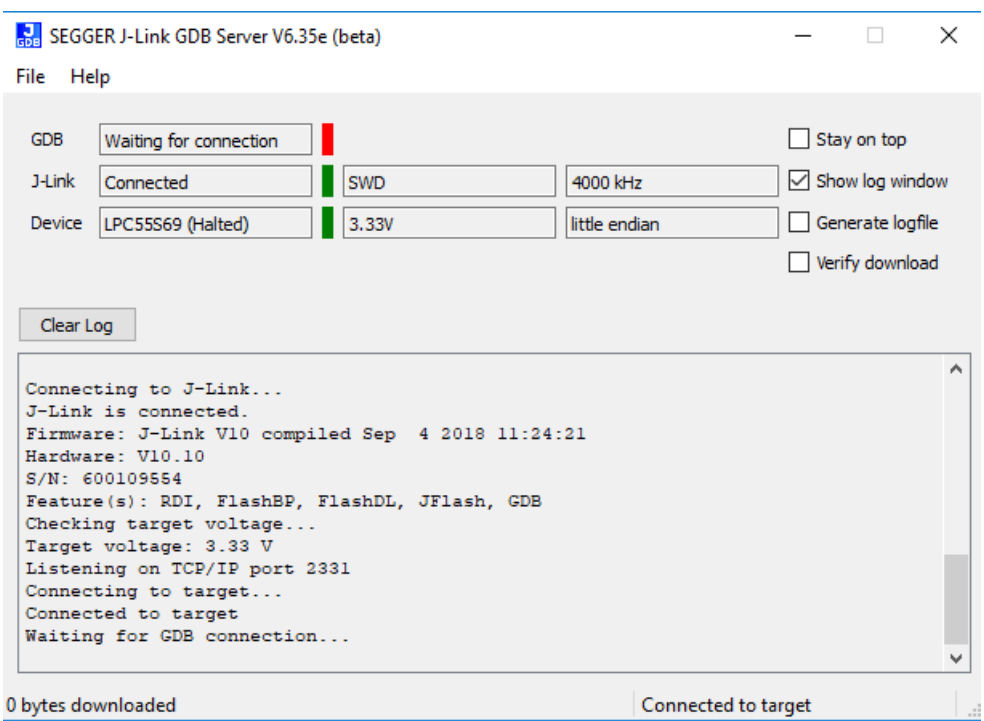

Figure 71. SEGGER J-Link GDB Server screen after successful connection

6. If not already running, open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to **Programs -> GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.

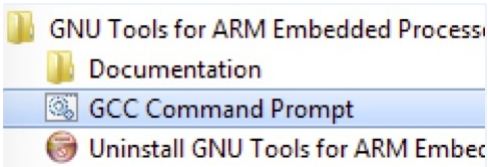

Figure 72. Launch command prompt

7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

   `<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/debug`

   `<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/release`

   For this example, the path is:

   `<install_dir>/boards/lpcxpresso55s69/demo_apps/hello_world/armgcc/debug`
8. Run the `arm-none-eabi-gdb.exe <application_name>.elf` command. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.

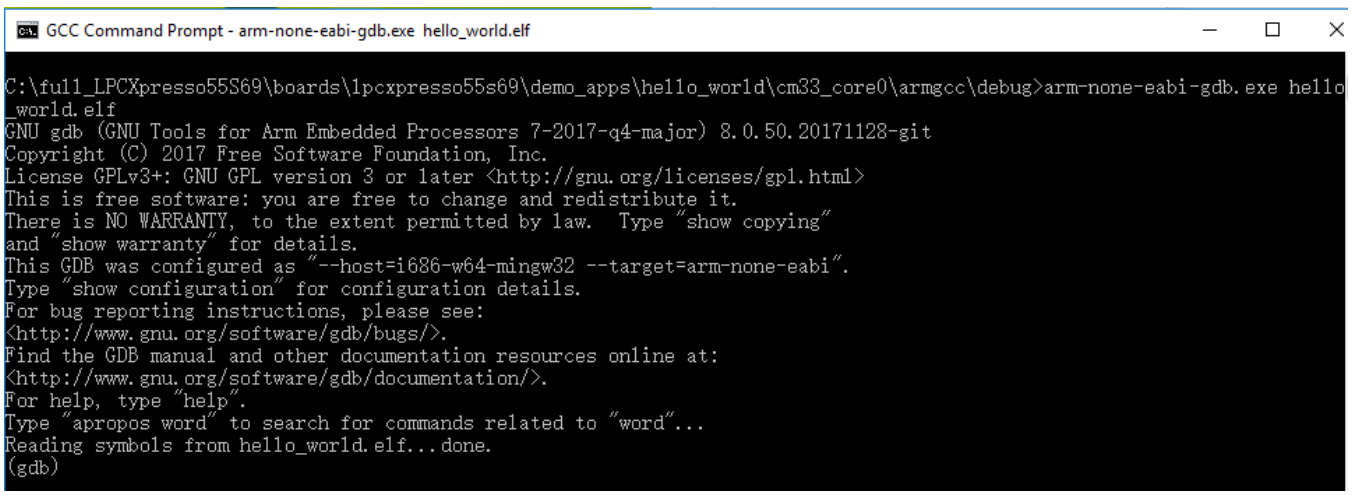

Figure 73. Run `arm-none-eabi-gdb`

9. Run these commands:
   a. `target remote localhost:2331`
   b. `monitor reset`
   c. `monitor halt`
   d. `load`
10. The application is now downloaded and halted at the reset vector. Execute the `monitor go` command to start the demo application.

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

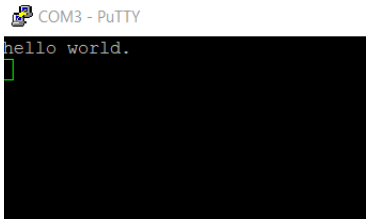

Figure 74. Text display of the hello_world demo

6.4 Build a multicore example application

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

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

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

<install_dir>/boards/lpcxpresso55s69/multicore_examples/hello_world/cm33_core0/armgcc/build_debug.bat

<install_dir>/boards/lpcxpresso55s69/multicore_examples/hello_world/cm33_core1/armgcc/build_debug.bat

Build both applications separately following steps for single core examples as described in *Section 6.2 "Build an example application"*.

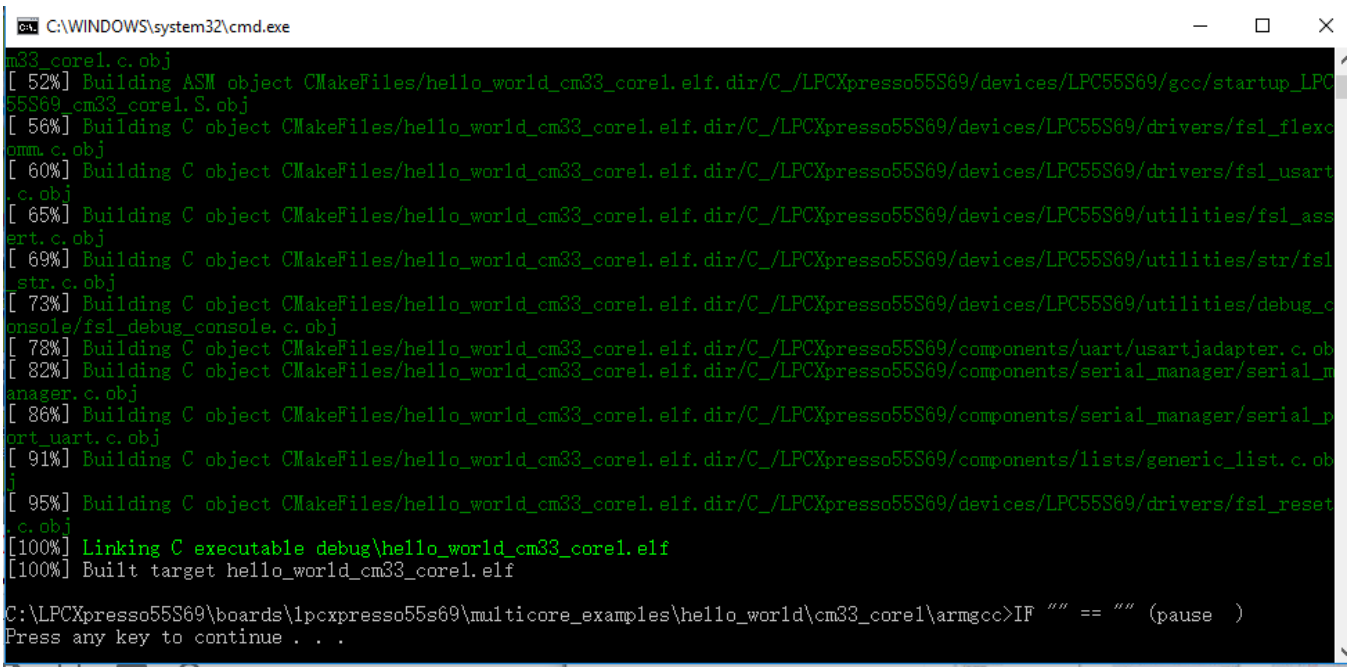

Figure 75. hello_world_cm33_core1 example build successful

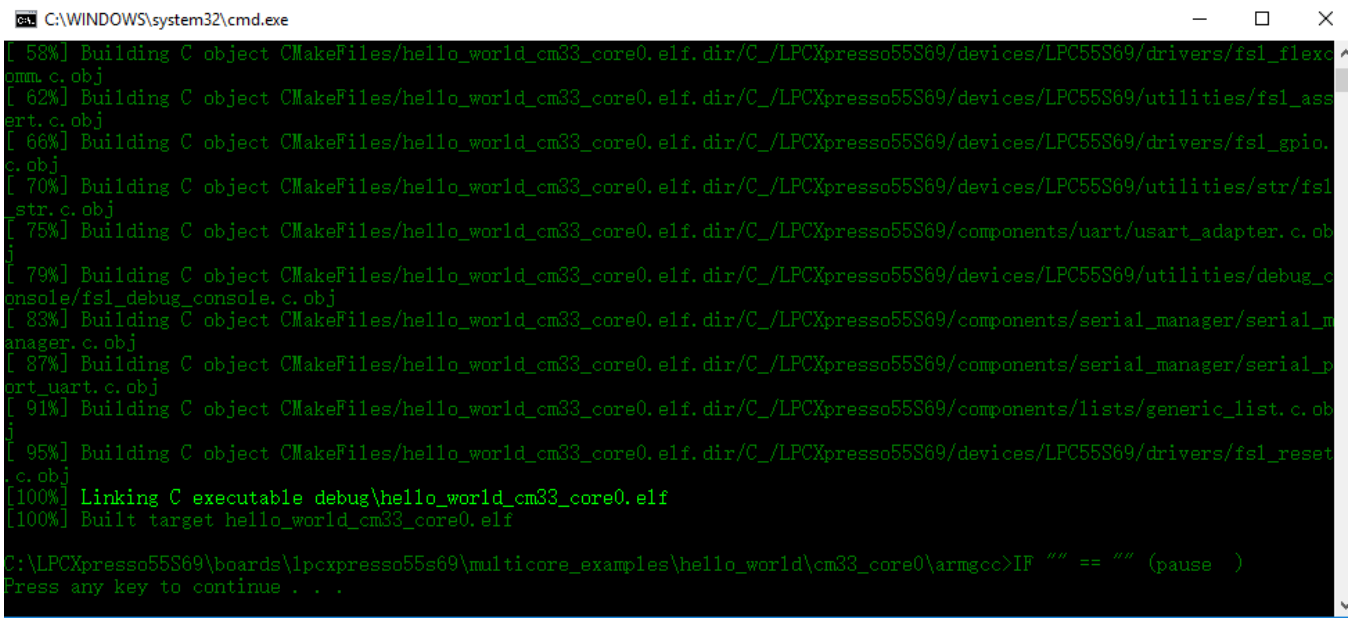

Figure 76. hello_world_cm33_core0 example build successful

6.5 Run a multicore example application

When running a multicore application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single-core application, applies, as described in Run an 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 steps 1 to 10, as described in Run an example application. These steps are common for both single-core and dual-core applications in Arm GCC.

Both the primary and the auxiliary image is loaded into the device flash memory. After execution of the “monitor go” command, the primary core application is executed. During the primary core code execution, the auxiliary core code is re-allocated from the flash memory to the RAM, and 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 is not true, check your terminal settings and connections.

```
Select GCC Command Prompt

C:\Program Files (x86)\GNU Tools Arm Embedded\7 2018-q2-update>cd C:\LPCXpresso55S69\boards\lpcxpresso55s69\multicore_ex
amples\hello_world\cm33_core0\armgcc\debug

C:\LPCXpresso55S69\boards\lpcxpresso55s69\multicore_examples\hello_world\cm33_core0\armgcc\debug>arm-none-eabi-gdb.exe h
ello_world_cm33_core0.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2018-q2-update) 8.1.0.20180315-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world_cm33_core0.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000674 in __aeabi_dadd ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0x140 lma 0x0
Loading section .text, size 0x3564 lma 0x140
Loading section .ARM, size 0x8 lma 0x36a4
Loading section .init_array, size 0x4 lma 0x36ac
Loading section .fini_array, size 0x4 lma 0x36b0
Loading section .data, size 0x64 lma 0x36b4
Loading section .m0code, size 0x2164 lma 0x72000
Start address 0x1f4, load size 22652
Transfer rate: 650 KB/sec, 3236 bytes/write.
(gdb) monitor go
(gdb) q
A debugging session is active.

        Inferior 1 [Remote target] will be killed.

Quit anyway? (y or n) y

C:\LPCXpresso55S69\boards\lpcxpresso55s69\multicore_examples\hello_world\cm33_core0\armgcc\debug>
```

Figure 77. Loading and running the multicore example

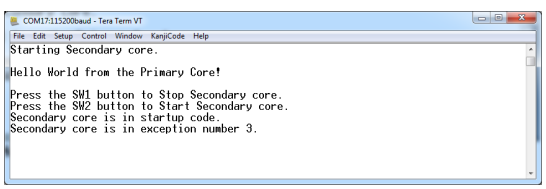

Figure 78. Hello World from primary core message

6.6 Build a TrustZone example application

This section describes the steps to build and run a TrustZone application. The demo application build scripts are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
<application_name>_ns/armgcc

<install_dir>/boards/<board_name>/trustzone_examples/<application_name>/<core_type>/iar/
<application_name>_s/armgcc
```

Run a demo using Arm® GCC

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World GCC build scripts are located in this folder:

```
<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/
hello_world_ns/armgcc/build_debug.bat

<install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/hello_world_s/
armgcc/build_debug.bat
```

Build both applications separately, following steps for single core examples as described in *Section 6.2, "Build an example application"*. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because the CMSE library is not ready.

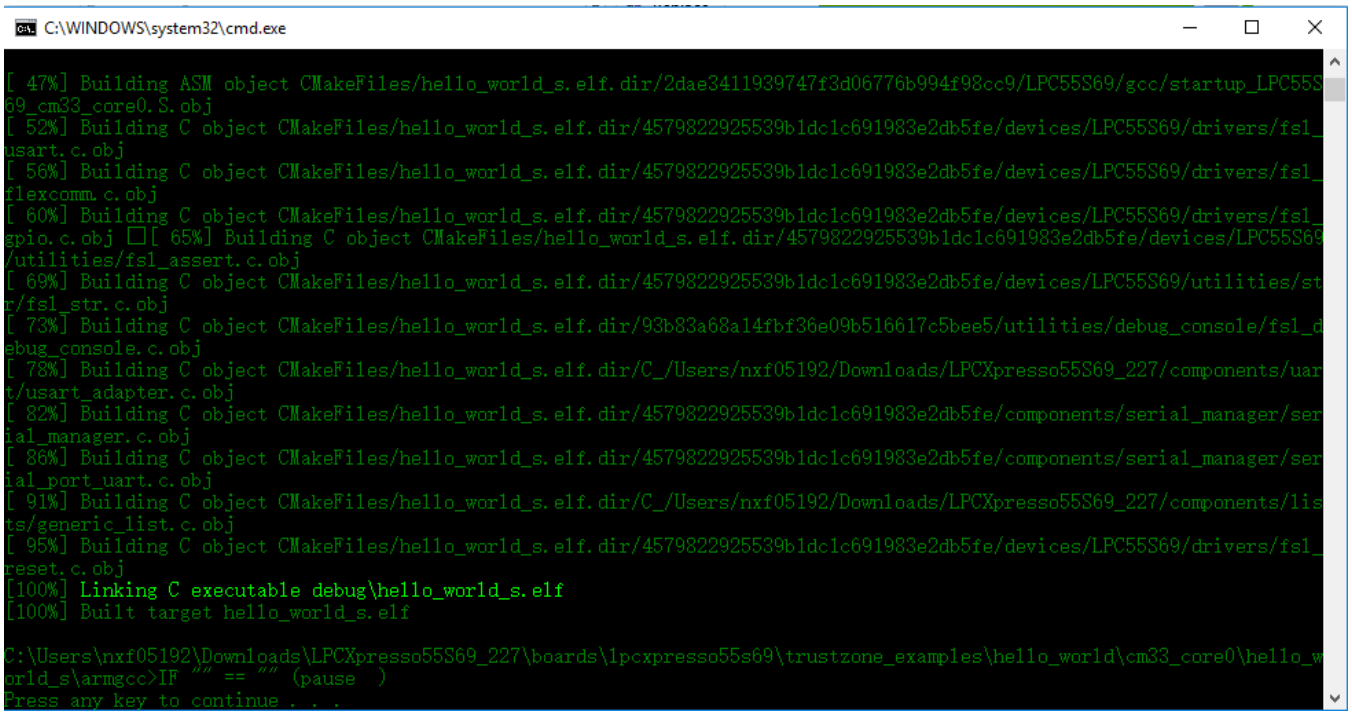

Figure 79. `hello_world_s` example build successful

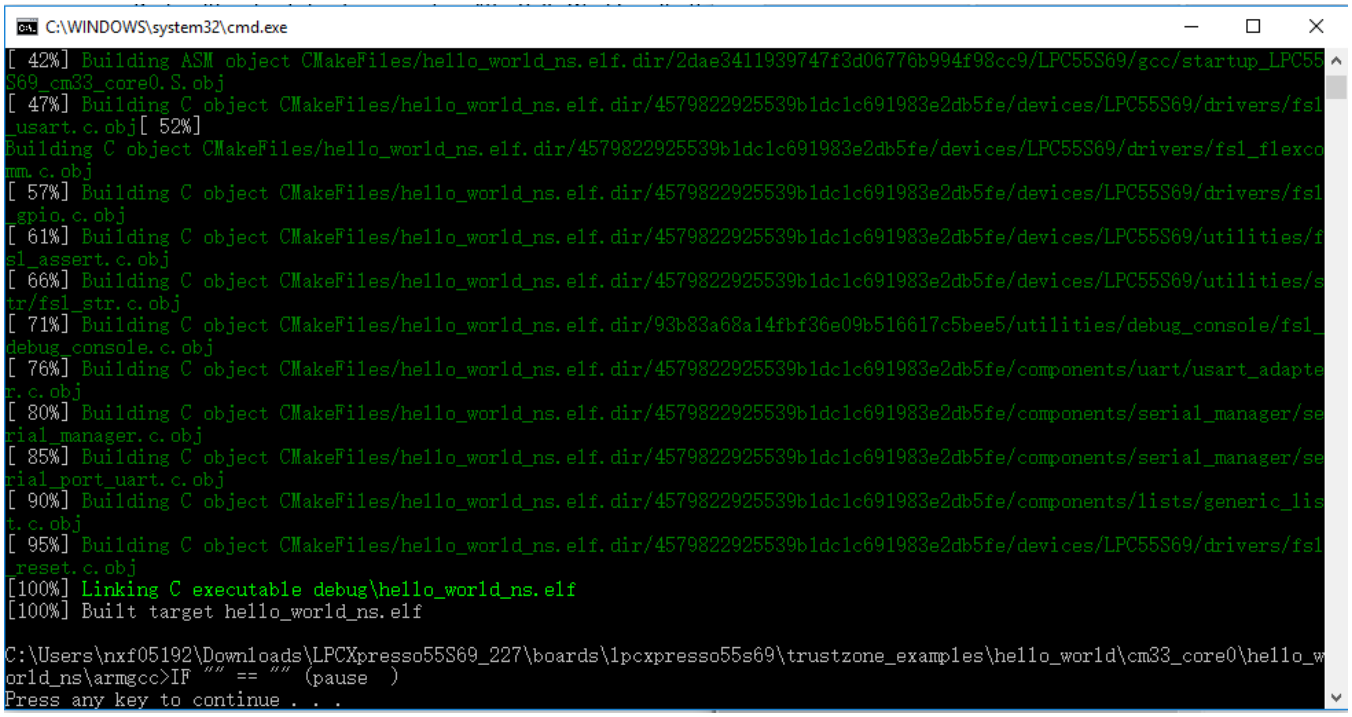

Figure 80. `hello_world_ns` example build successful

6.7 Run a TrustZone example application

When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in *Section 6.3, "Run an example application"*.

To download and run the TrustZone application, perform steps 1 to 10, as described in *Section 5.3, "Run an example application"*. These steps are common for both single core and trustzone applications in Arm GCC.

Then, run these commands:

1. `arm-none-eabi-gdb.exe`
2. `target remote localhost:2331`
3. `monitor reset`
4. `monitor halt`
5. `load <install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/hello_world_ns/armgcc/debug/hello_world_ns.elf`
6. `load <install_dir>/boards/lpcxpresso55s69/trustzone_examples/hello_world/cm33_core0/hello_world_s/armgcc/debug/hello_world_s.elf`

The application is now downloaded and halted at the reset vector. Execute the `monitor go` command to start the demo application.

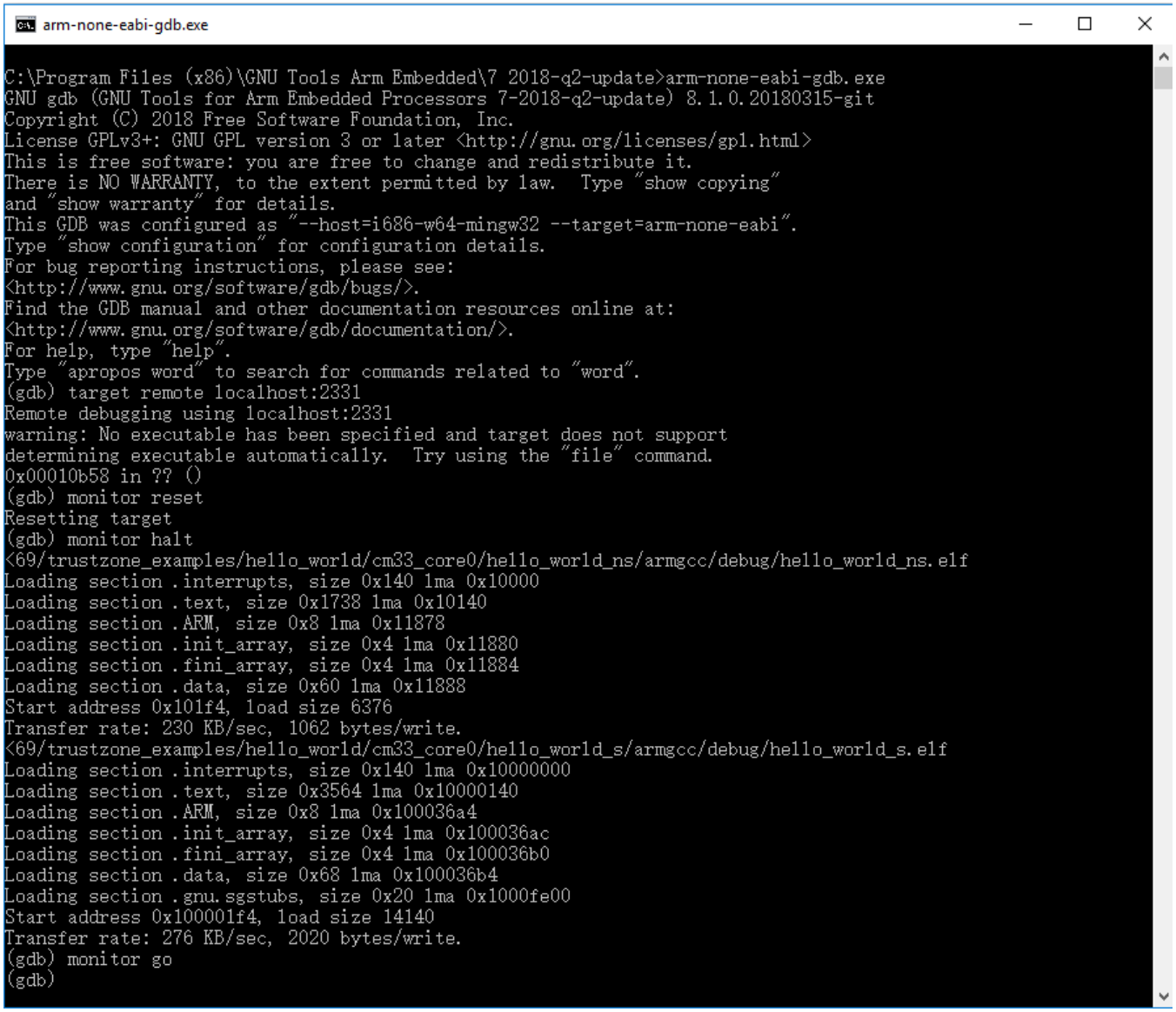

Figure 81. Loading and running the trustzone example

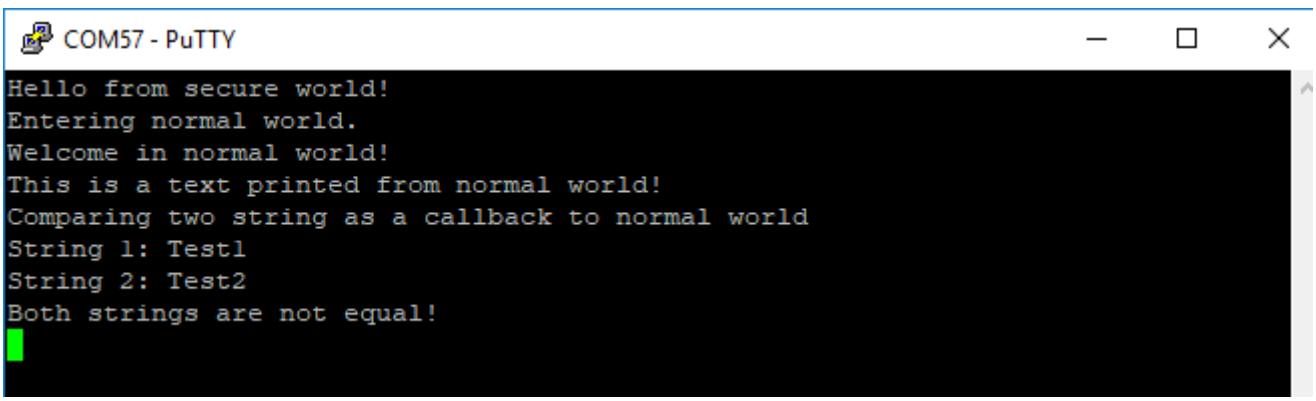

Figure 82. Text display of the trustzone `hello_world` application

7 MCUXpresso Config Tools

MCUXpresso Config Tools can help configure the processor and generate initialization code for the on chip peripherals. The tools are able to modify any existing example project, or create a new configuration for the selected board or processor. The generated code is designed to be used with MCUXpresso SDK version 2.x.

The MCUXpresso Config Tools consist of the following (only pins tool is supported in this release):

Table 1. MCUXpresso Config Tools

Config Tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration	
Peripherals tools	For configuration of other peripherals	
Project Cloner	Allows creation of standalone projects from MCUXpresso SDK examples.	

MCUXpresso Config Tools can be accessed in the following products:

- **Integrated** in the MCUXpresso IDE. Config tools are integrated with both compiler and debugger which makes it the easiest way to begin the development.
- **Standalone version** available for download from www.nxp.com. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.
- **Online version** available on mcuxpresso.nxp.com. Recommended to do a quick evaluation of the processor or use the tool without installation.

Each version of the product contains a specific “Quick Start Guide” document MCUXpresso IDE Config Tools installation folder that can help start your work.

8 MCUXpresso IDE New Project Wizard

MCUXpresso IDE features a new project wizard. The wizard provides functionality for the user to create new projects from the installed SDKs (and from pre-installed part support). It offers user the flexibility to select and change multiple builds. The wizard also includes a library and provides source code options. The source code is organized as software components, categorized as drivers, utilities, and middleware.

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in Figure 83.

Figure 83. MCUXpresso IDE Quickstart Panel

For more details and usage of new project wizard, see the *MCUXpresso_IDE_User_Guide.pdf* in the MCUXpresso IDE installation folder.

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.

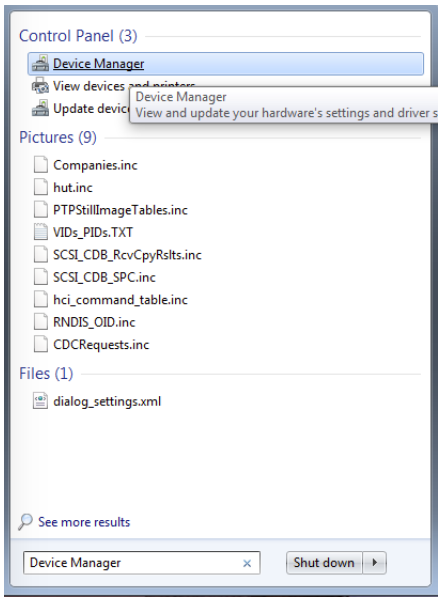

Figure A-1. Device Manager

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. **OpenSDA – CMSIS-DAP/mbed/DAPLink** interface:

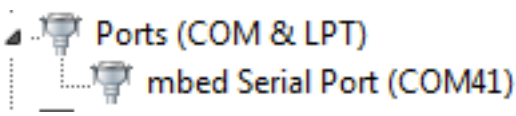

Figure A-2. OpenSDA – CMSIS-DAP/mbed/DAPLink interface

   b. **OpenSDA – P&E Micro**:

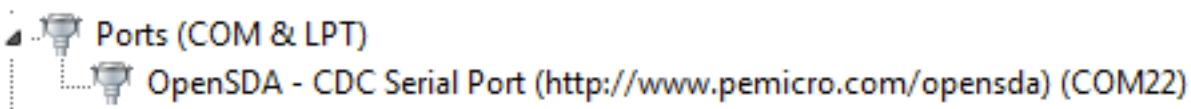

Figure A-3. OpenSDA – P&E Micro

   c. **OpenSDA – J-Link**:

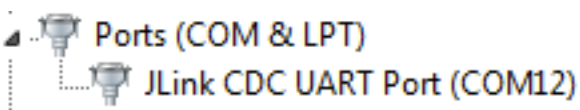

Figure A-4. OpenSDA – J-Link

   d. **P&E Micro OSJTAG**:

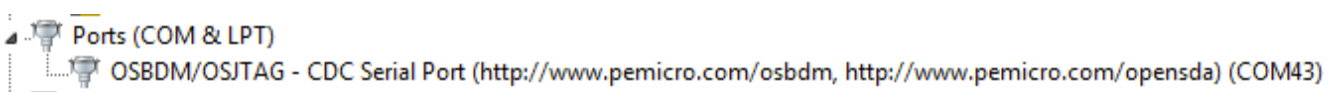

Figure A-5. P&E Micro OSJTAG

   e. **LPC-Link2**:

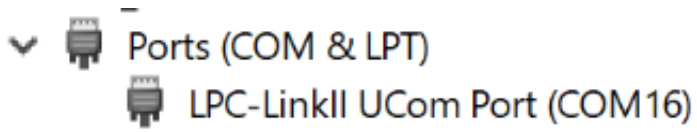

Figure A-6. LPC-Link2

Appendix B Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with a variety of factory programmed debug interface configurations. Table B-1 lists the hardware platforms supported by the MCUXpresso SDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

NOTE

The OpenSDA details column in Table B-1 is not applicable to LPC.

Table B-1. Hardware platforms supported by MCUXpresso SDK

Hardware platform	Default interface	OpenSDA details
EVK-MIMXRT595	CMSIS-DAP	N/A
EVK-MIMXRT685	CMSIS-DAP	N/A
FRDM-K22F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-K28F	DAPLink	OpenSDA v2.1
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2.1
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2.1
FRDM-K32W042	CMSIS-DAP	N/A
FRDM-K64F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
FRDM-K66F	J-Link OpenSDA	OpenSDA v2.1
FRDM-K82F	CMSIS-DAP	OpenSDA v2.1
FRDM-KE15Z	DAPLink	OpenSDA v2.1
FRDM-KE16Z	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.2
FRDM-KL02Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL03Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL25Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL26Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL27Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL28Z	P&E Micro OpenSDA	OpenSDA v2.1
FRDM-KL43Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL46Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL81Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KL82Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KV10Z	CMSIS-DAP	OpenSDA v2.1
FRDM-KV11Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KV31F	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KW24	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2

Table continues on the next page...

Table B-1. Hardware platforms supported by MCUXpresso SDK (continued)

Hardware platform	Default interface	OpenSDA details
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
HVP-KE18F	DAPLink	OpenSDA v2.2
HVP-KV46F150M	P&E Micro OpenSDA	OpenSDA v1
HVP-KV11Z75M	CMSIS-DAP	OpenSDA v2.1
HVP-KV58F	CMSIS-DAP	OpenSDA v2.1
HVP-KV31F120M	P&E Micro OpenSDA	OpenSDA v1
JN5189DK6	CMSIS-DAP	N/A
LPC54018 IoT Module	N/A	N/A
LPCXpresso54018	CMSIS-DAP	N/A
LPCXpresso54102	CMSIS-DAP	N/A
LPCXpresso54114	CMSIS-DAP	N/A
LPCXpresso51U68	CMSIS-DAP	N/A
LPCXpresso54608	CMSIS-DAP	N/A
LPCXpresso54618	CMSIS-DAP	N/A
LPCXpresso54628	CMSIS-DAP	N/A
LPCXpresso54S018M	CMSIS-DAP	N/A
LPCXpresso55s16	CMSIS-DAP	N/A
LPCXpresso55s28	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
TWR-K21D50M	P&E Micro OSJTAG	N/AOpenSDA v2.0
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K24F120M	CMSIS-DAP/mbed	OpenSDA v2.1
TWR-K60D100M	P&E Micro OSJTAG	N/A
TWR-K64D120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K64F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K80F150M	CMSIS-DAP	OpenSDA v2.1
TWR-K81F150M	CMSIS-DAP	OpenSDA v2.1
TWR-KE18F	DAPLink	OpenSDA v2.1
TWR-KL28Z72M	P&E Micro OpenSDA	OpenSDA v2.1
TWR-KL43Z48M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KL81Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KL82Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KM34Z75M	P&E Micro OpenSDA	OpenSDA v1.0

Table continues on the next page...

Table B-1. Hardware platforms supported by MCUXpresso SDK (continued)

Hardware platform	Default interface	OpenSDA details
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV11Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV31F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV46F150M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV58F220M	CMSIS-DAP	OpenSDA v2.1
TWR-KW24D512	P&E Micro OpenSDA	OpenSDA v1.0
USB-KW24D512	N/A External probe	N/A
USB-KW41Z	CMSIS-DAP\DAPLink	OpenSDA v2.1 or greater

Appendix C Updating debugger firmware

C.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 **Debug**). 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**, and then the 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 in again.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MCUXSDKLPC55XXGSUG
Revision 0, 12/2019