NXP Semiconductors
User's Guide

Document Number: MCUXSDKLPC540XXGSUG
Rev. 0, 01/2018

Getting Started with MCUXpresso SDK for LPC540xx

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 LPCXpresso540XX (document MCUXSDKLPC540XXRN). After opening SDK_2.3.0_LPCXpresso54018\docs, consult the MCUXpresso SDK Release Notes Supporting LPC540XX.pdf.

For the latest version of this and other MCUXpresso SDK documents, see the MCUXpresso SDK homepage MCUXpresso-SDK: Software Development Kit for MCUXpresso.

Contents

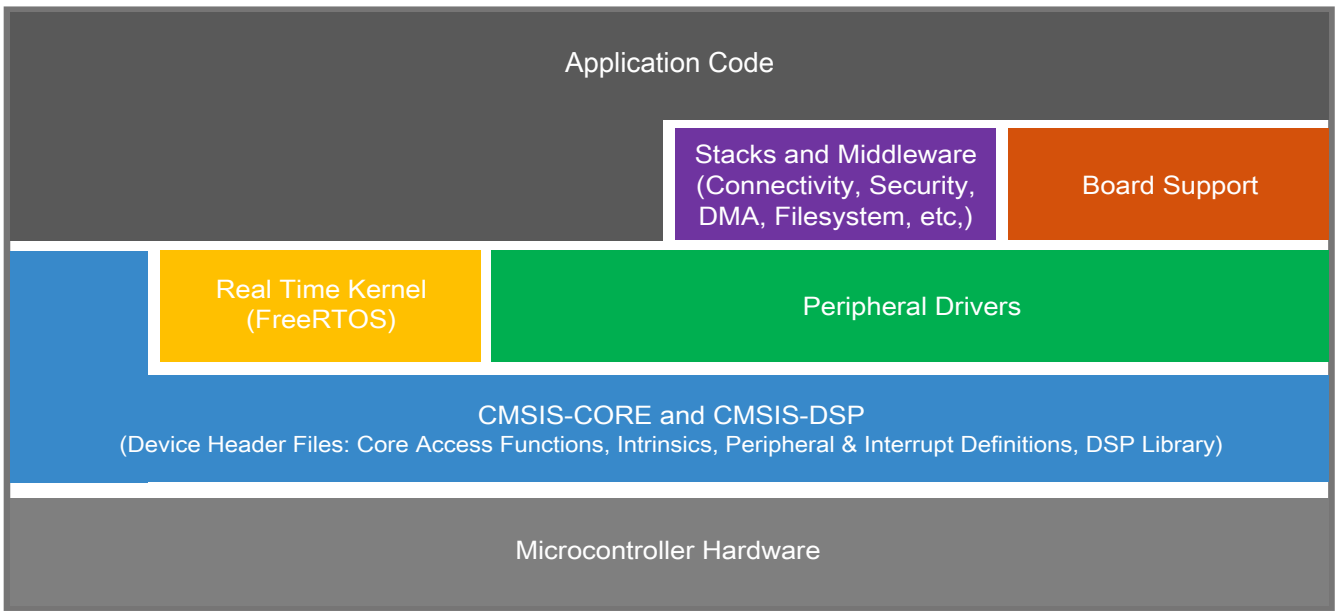

Figure 1. MCUXpresso SDK layers

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 are used (for example, SPI conversion using DMA).
- emwin_examples: Applications that use the emWin GUI widgets.
- 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
- 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 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. We'll discuss the hello_world example (part of the demo_apps folder), but 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:

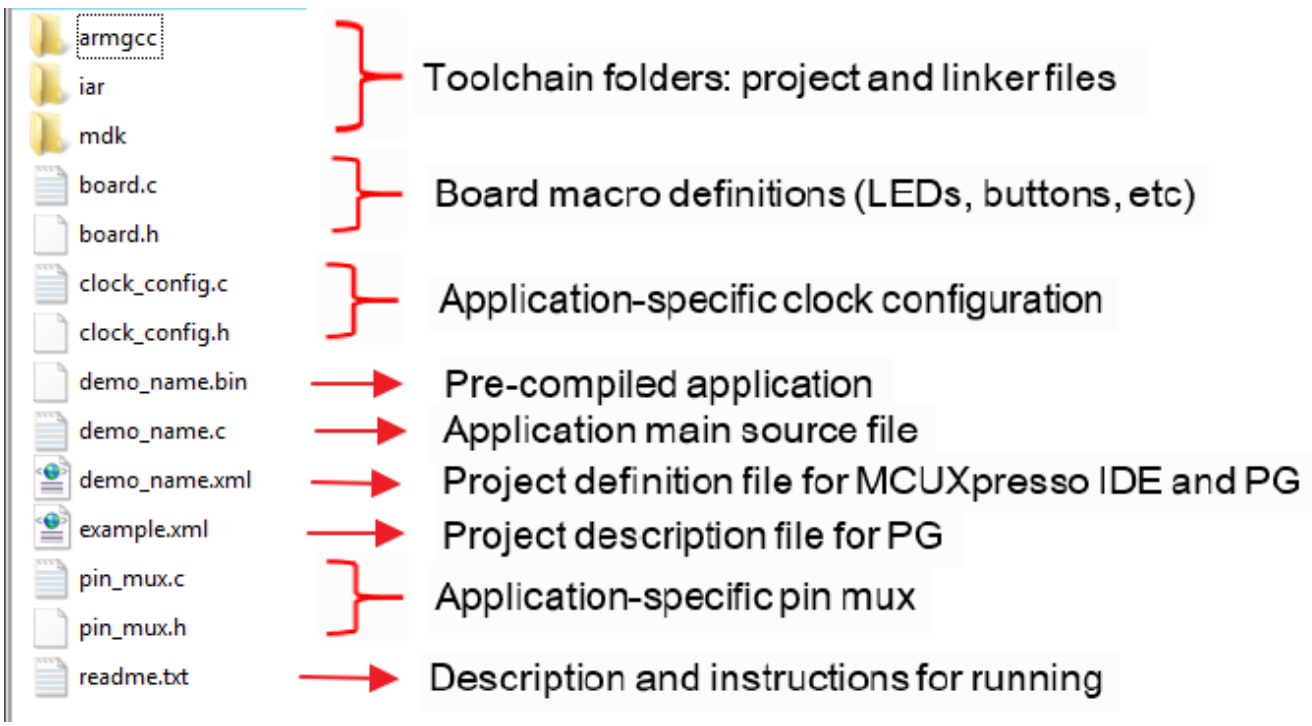

Figure 2. Application folder structure

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 IDE (except MCUXpresso IDE), there are a variety of source files 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 them could have potential impacts on other projects that depend on them.

2.3 Locating Example Application Source Files

When opening an example application in any of the supported IDE (except MCUXpresso IDE), there are a variety of source files 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 them could have potential impacts on other projects that depend on them.

2.3.1 What is the plain load image?

The plain load image is linked to SRAMX. When building, it needs to be programmed to the external onboard flash. After reset, the bootloader starts to run. Then, it checks the image type (which is set in the image header, which resides in the startup code). If the type is plain load image, the bootloader copies the image to SRAMX, then jumps to SRAMX to run.

2.3.2 What is the XIP image?

When an XIP (Execute-in-Place) image is used, the MCU executes instructions and uses data directly from external (quad) SPI flash. When building, it needs to be programmed to the external onboard flash. After reset, the bootloader starts to run. Then, it checks the image type (which is set in the image header, which resides in the startup code). If the type is XIP image, the device executes in the external flash.

2.3.3 What is the SRAM target image?

The SRAM target image is linked to the SRAM address when building. When debugging, the SRAM target, the IDE just loads the image into SRAM to run.

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 LPCXpresso54018 hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Before downloading and running 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 the 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 you are 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 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

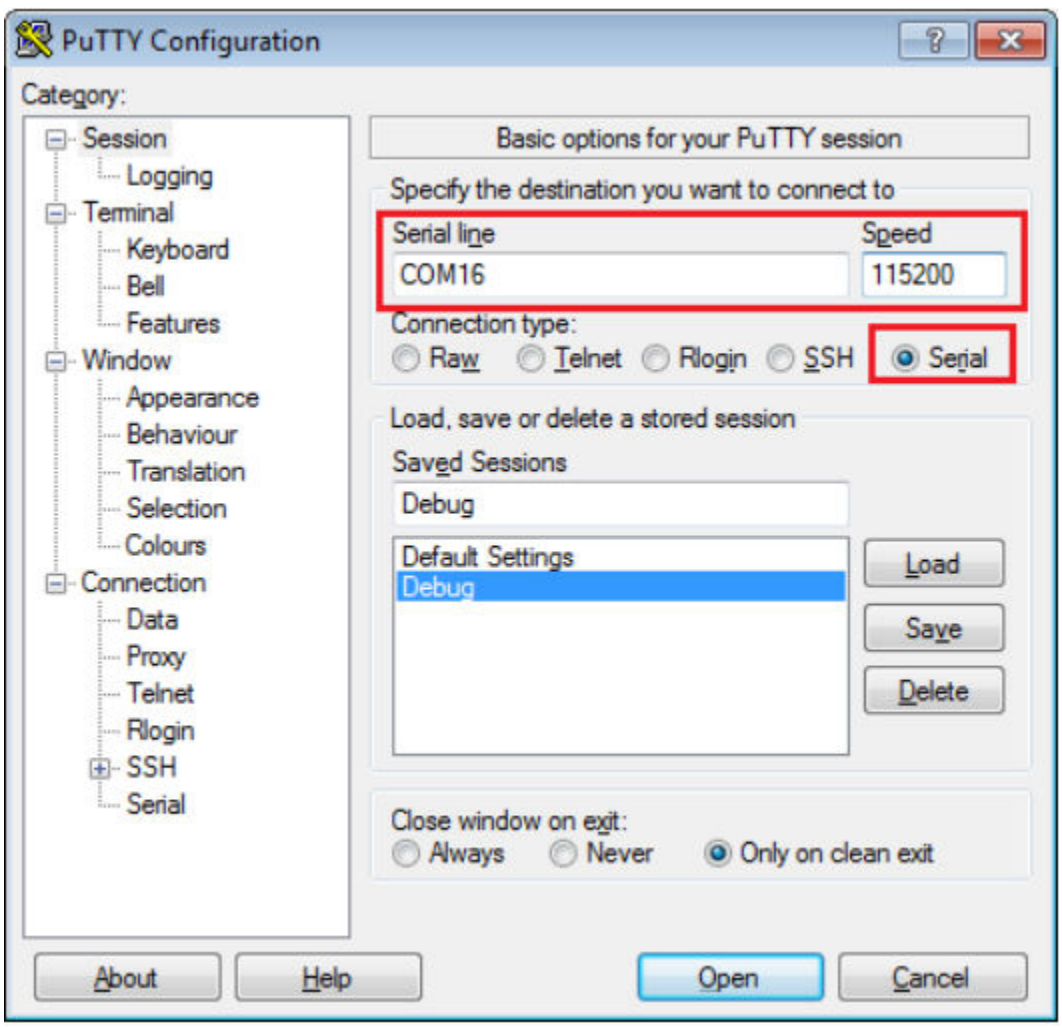

Figure 3. Terminal (PuTTY) configuration

3.1 Build an non-XIP (plain load) example application

The following steps guide you through opening the hello_world example application. These steps may change slightly for other example applications as some of these applications may have additional layers of folders in their path.

1. If not already done, 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 LPCXpresso54018 hardware platform as an example, the hello_world workspace is located in

   <install_dir>/boards/lpcxpresso54018/demo_apps/hello_world/iar/hello_world.eww

2. Select the desired build target from the drop-down. For this example, select the “hello_world – Debug” target.

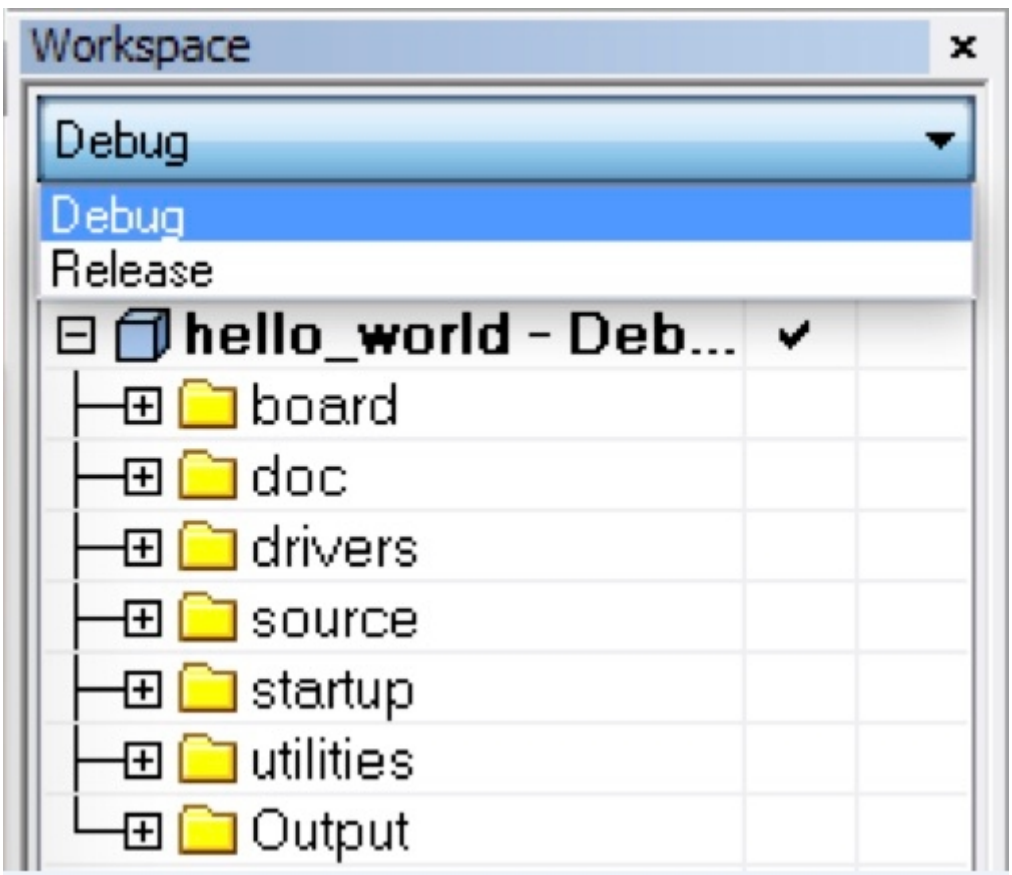

Figure 4. Demo build target selection

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

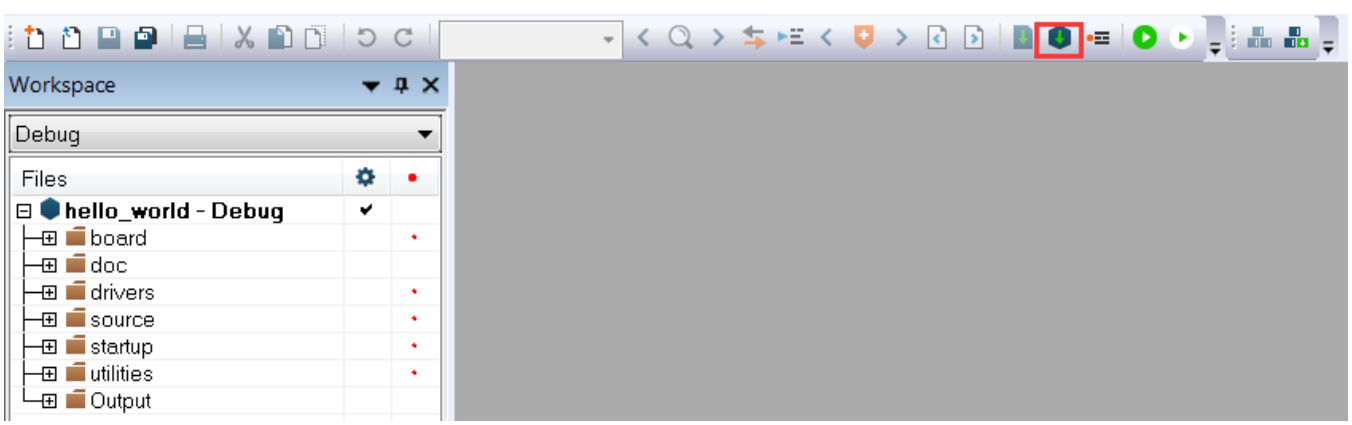

Figure 5. Build the demo application

4. The build completes without errors.

3.2 Run a non-XIP (plain load) example application using CMSIS DAP

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

Figure 6. Download and Debug button

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

NOTE

The application is programmed to the external on board flash, then jumped to SRAM to run

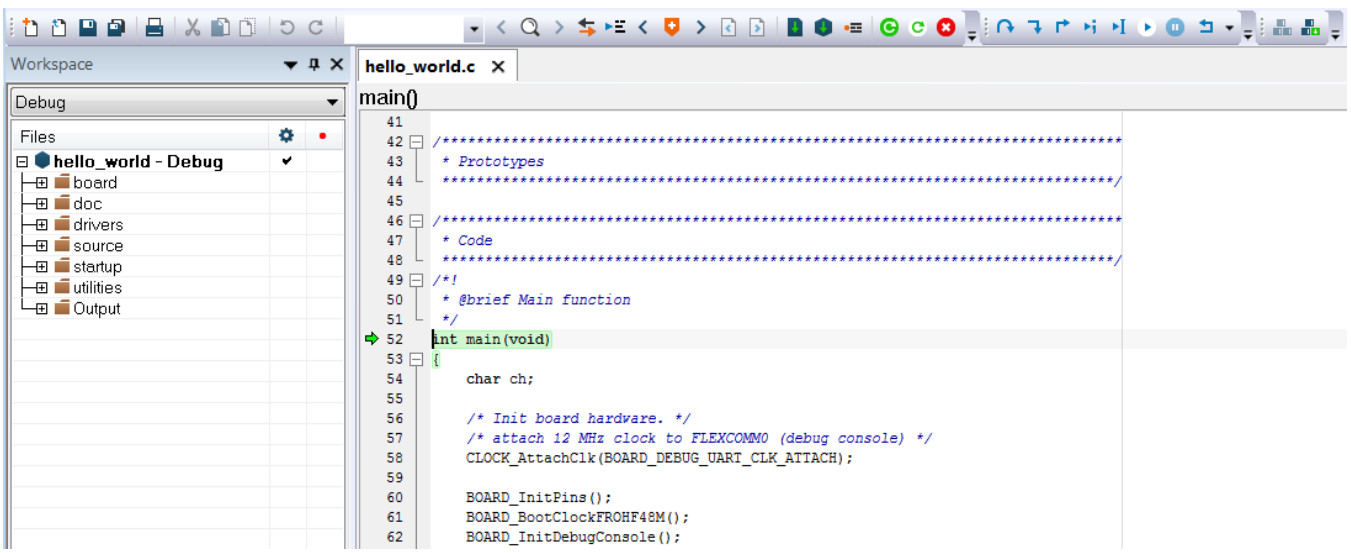

Figure 7. Stop at main() when running debugging

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

Figure 8. Go button

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

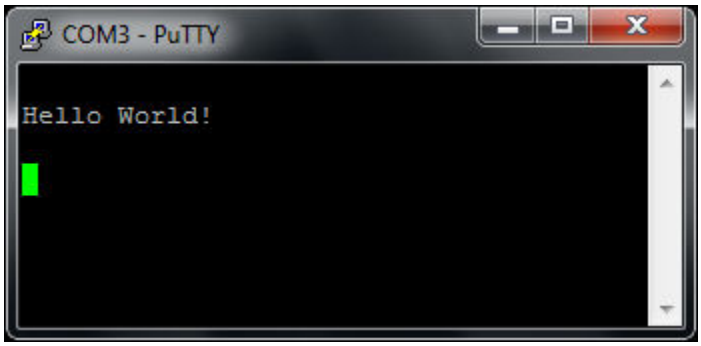

Figure 9. Text display of the hello_world demo

3.3 Run a non-XIP (plain load) example application using J-Link/J-Trace

This section describes how to configure EWARM if using a J-link/J-Trace debug probe, or a probe which has been programmed with J-link firmware.

1. Select “J-Link/J-Trace” and “Use macro file(s)”.

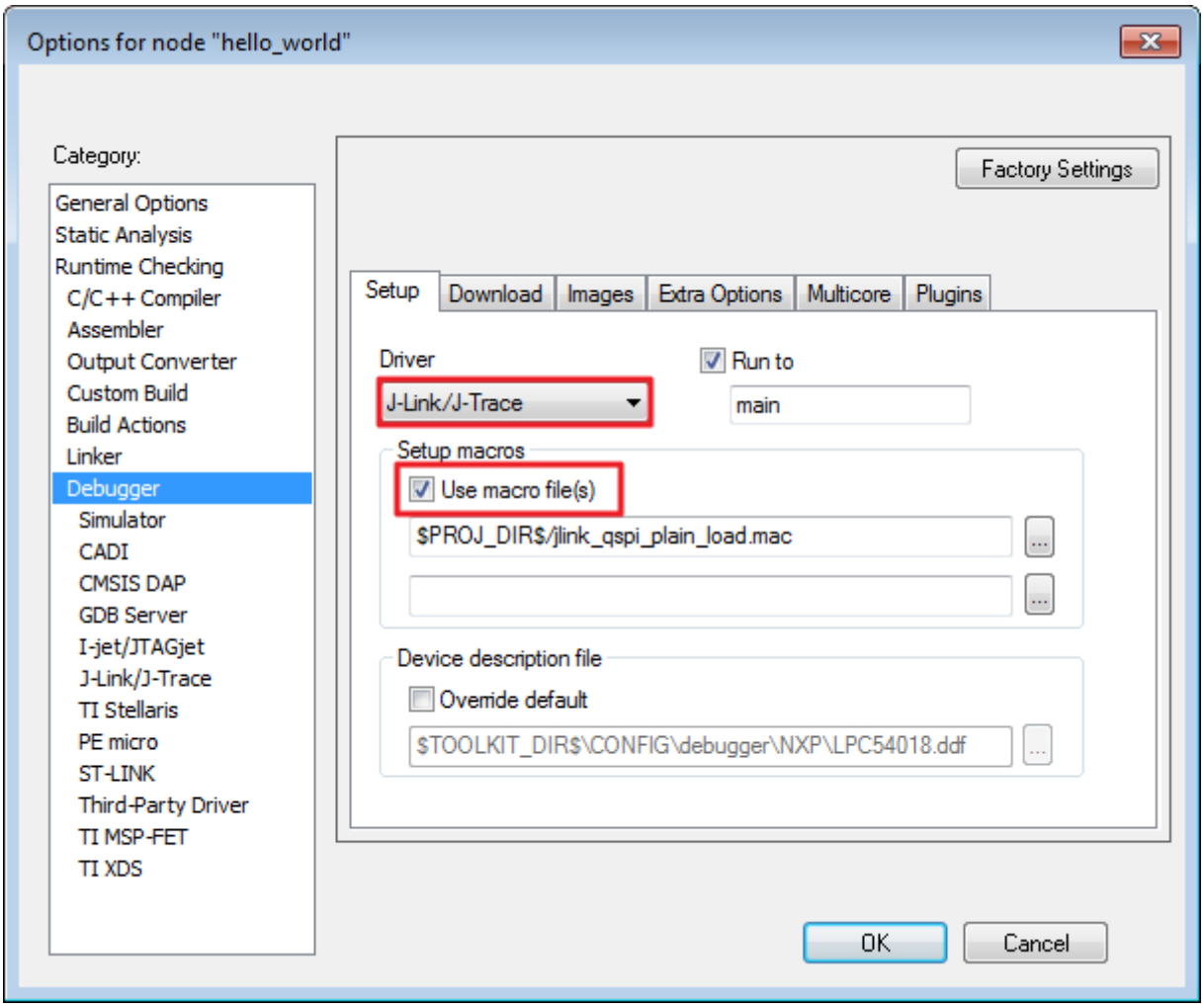

Figure 10. Select "J-Link/J-Trace" and "Use macro file(s)"

2. Unselect "Use flash loader(s)" to use SEGGER J-Link flashloaders. Then, click the "OK" button.

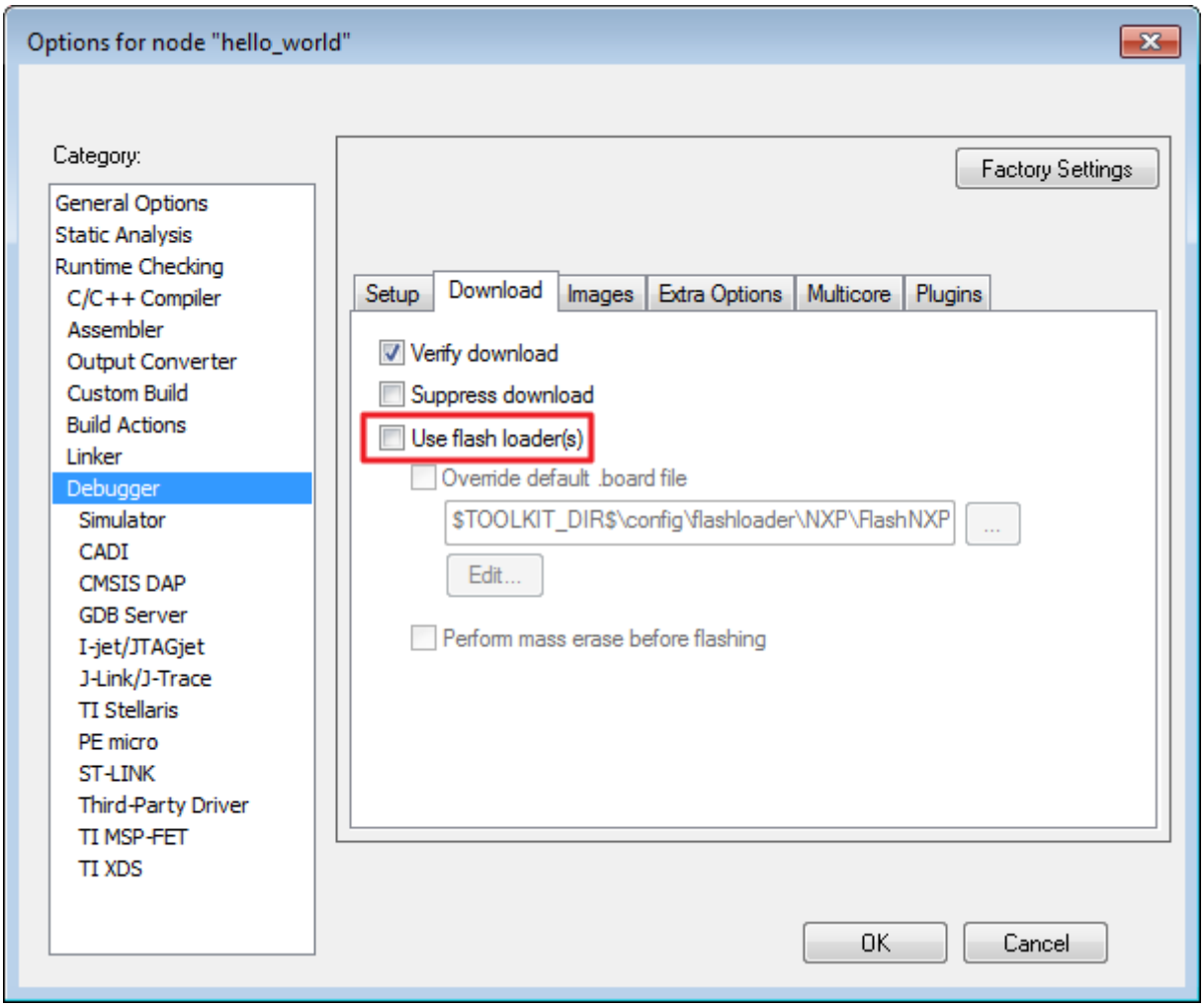

Figure 11. Unselect “Use flash loader(s)”

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

Figure 12. Download and Debug button

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

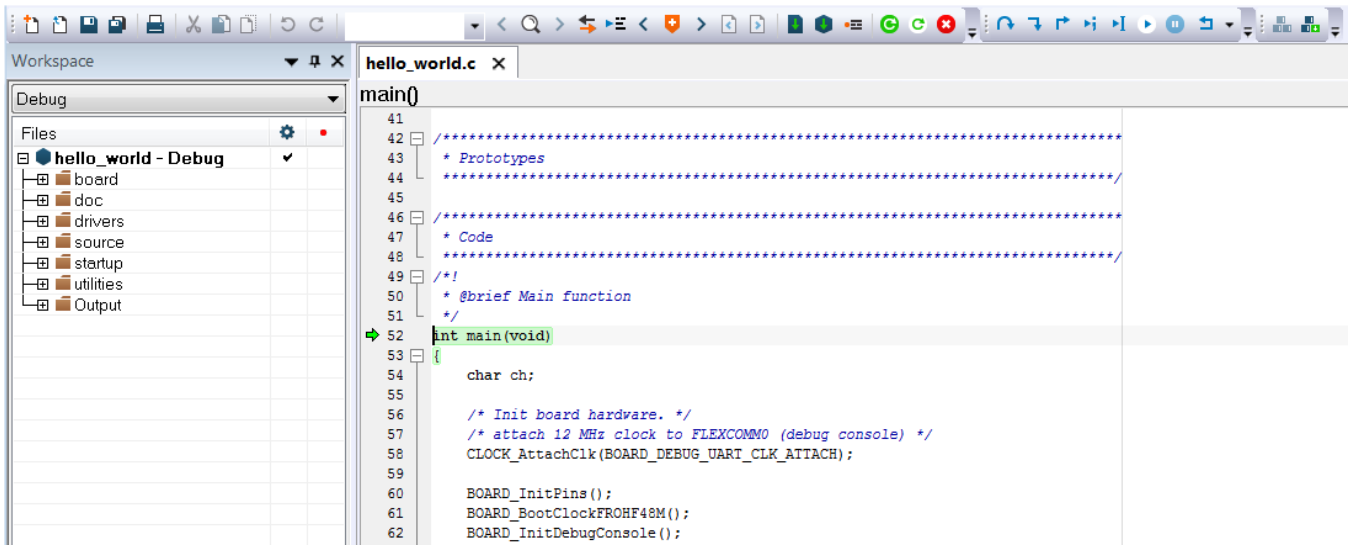

Figure 13. Stop at main() when running debugging

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

Figure 14. Go button

6. 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.

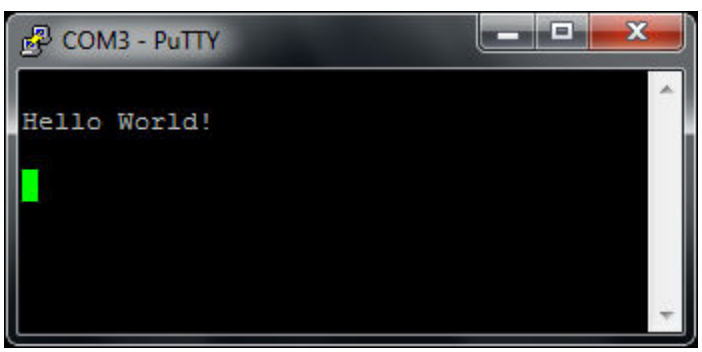

Figure 15. Text display of the hello_world demo

3.4 Build an XIP example application

The following steps guide you through opening the hello_world_qspi_xip example application. These steps may change slightly for other example applications, as some of these applications may have additional layers of folders in their paths.

1. If not already done, 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 LPCXpresso54018 hardware platform as an example, the hello_world workspace is located in

   <install_dir>/boards/lpcxpresso54018/demo_apps/hello_world/iar/hello_world_qspi_xip.eww

2. Select the desired build target from the drop-down. For this example, select the “hello_world_qspi_xip – Debug” target.

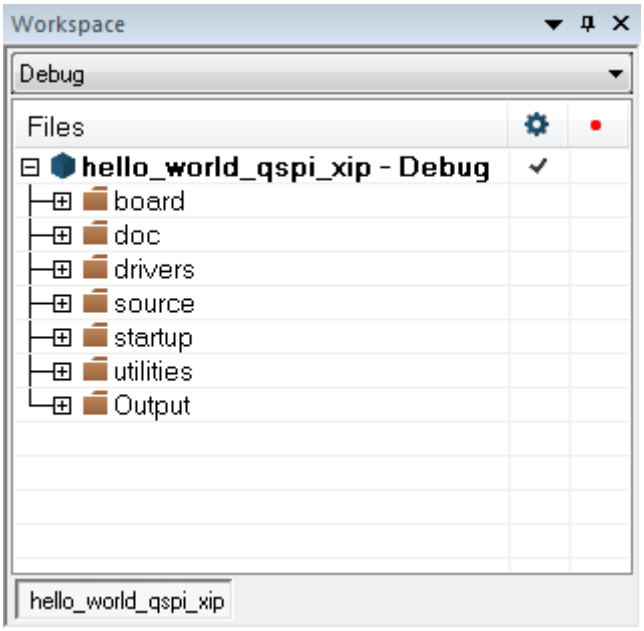

Figure 16. Demo build target selection

3. Select "Reset and halt after bootloader", highlighted in red below.

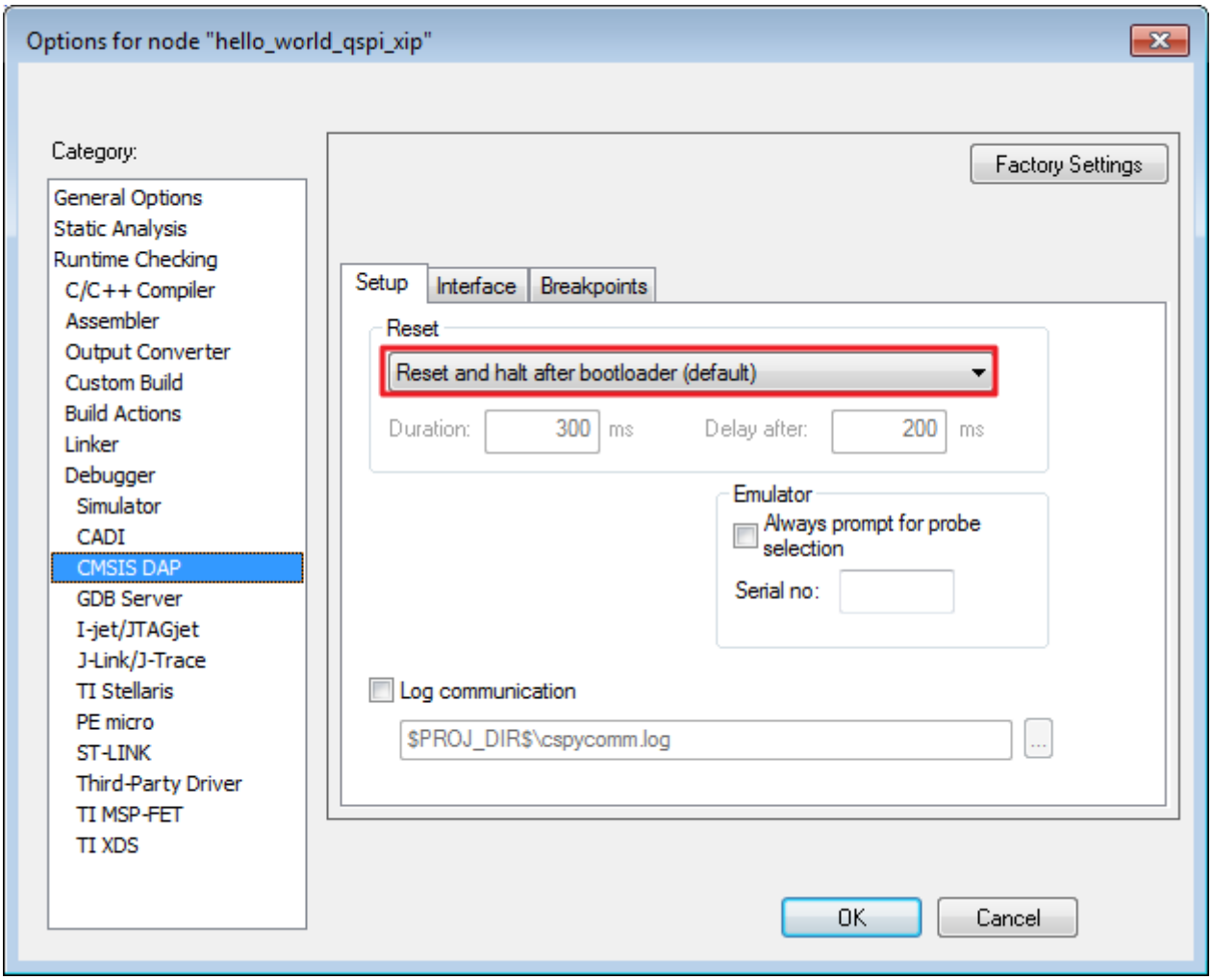

Figure 17. Reset and half after bootloader

4. To build the demo application, click the "Make" button, highlighted in red below.

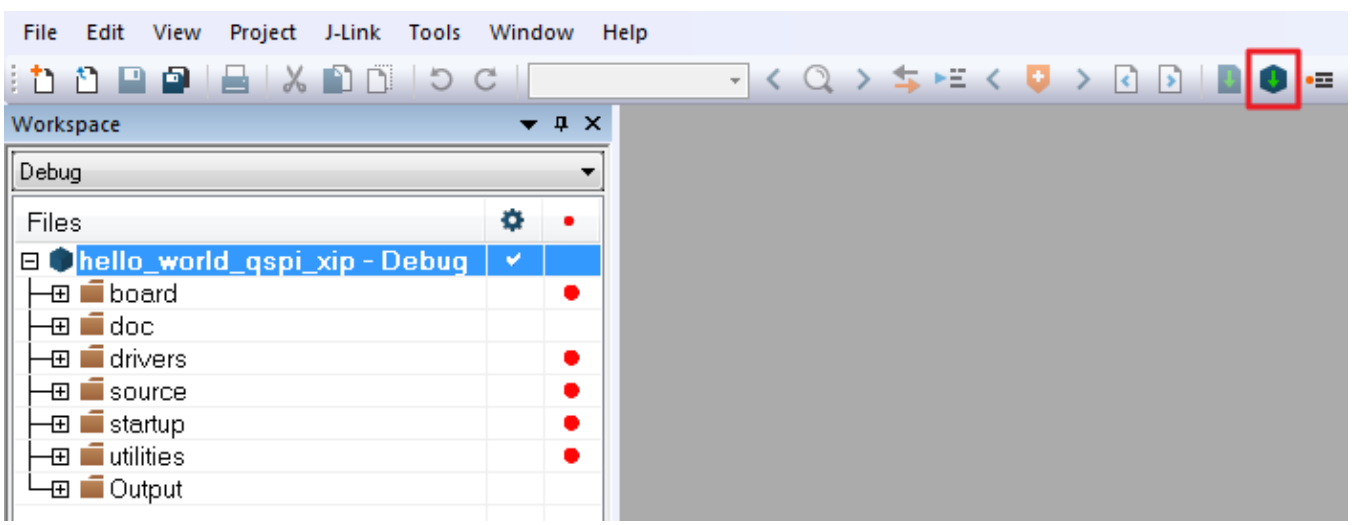

Figure 18. Build the demo application

3.5 Run an XIP example application

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

Figure 19. Download and Debug button

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

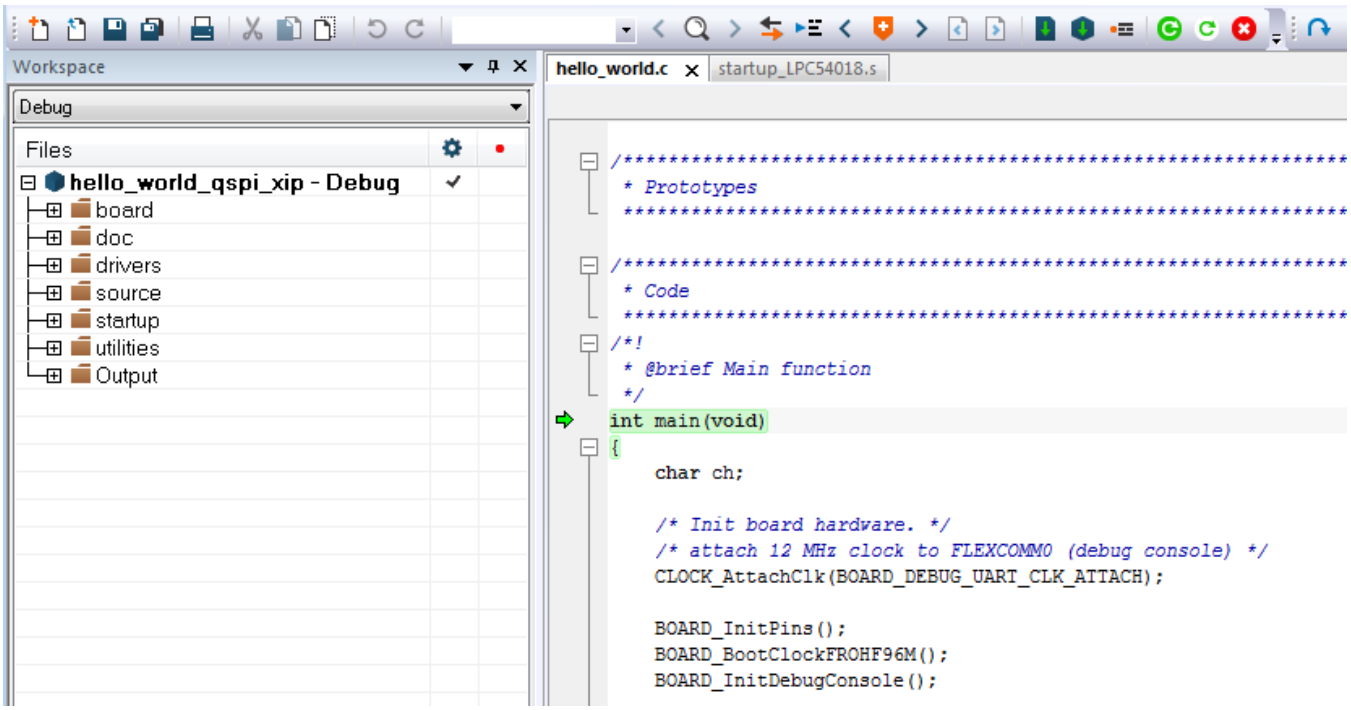

Figure 20. Stop at main() when running debugging

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

Figure 21. Go button

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

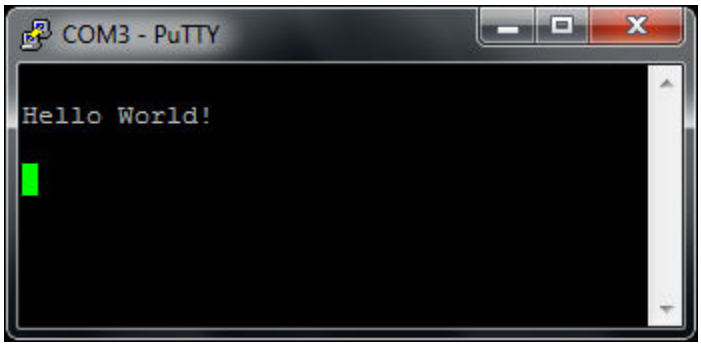

Figure 22. Text display of the hello_world_qspi_xip demo

3.6 Build a SRAM example application using J-Link/J-Trace

The following steps guide you through opening the spifi_ dma_transfer example application. These steps may change slightly for other example applications, as some of these applications may have additional layers of folders in their paths.

1. If not already done, 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 LPCXpresso54018 hardware platform as an example, the spifi_ dma_transfer workspace is located in

   <install_dir>/boards/lpcxpresso54018/driver_examples /spifii/iar/spifi_dma_transfer.eww

2. Select the desired build target from the drop-down. For this example, select the “spifi_ dma_transfer – Debug” target.

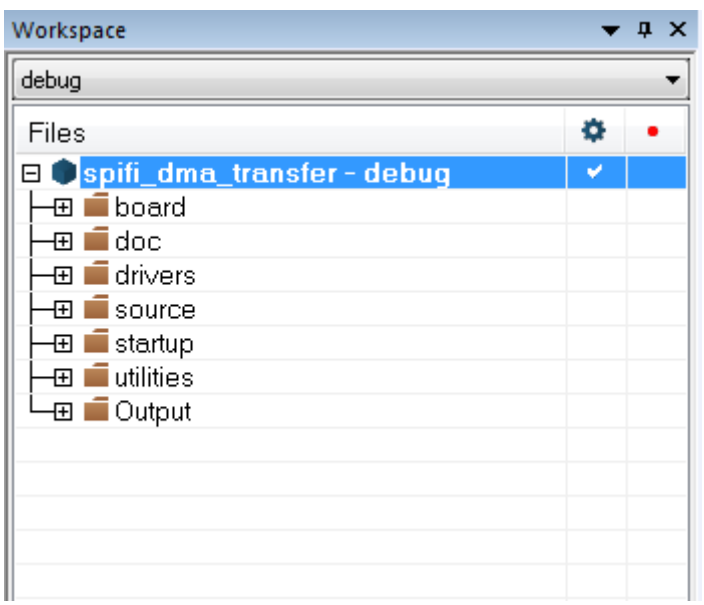

Figure 23. Demo build target selection

3. Select “J-Link/J-Trace” and “Use macro file(s)”.

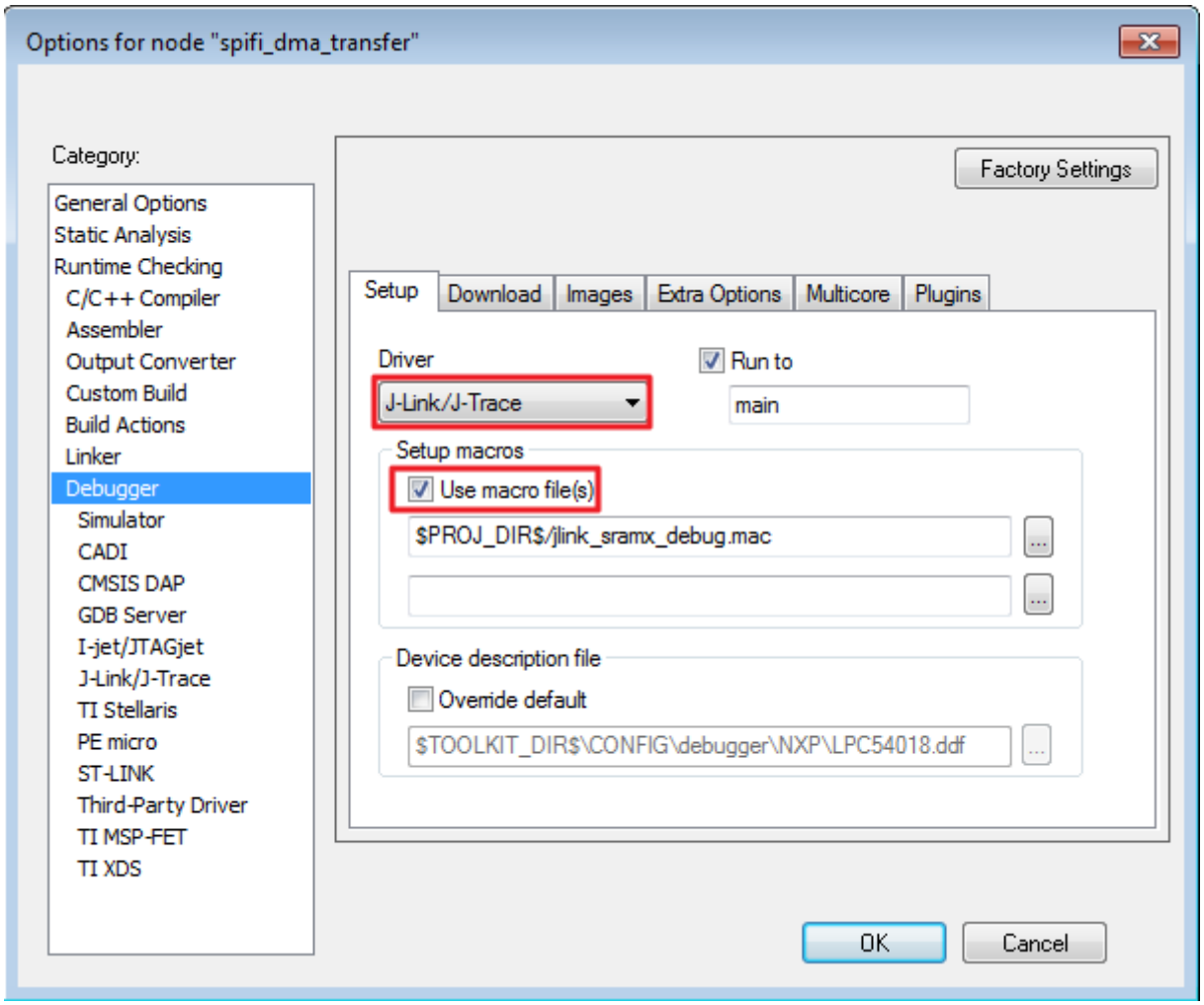

Figure 24. Select “J-Link/J-Trace” and “Use macro file(s)”

4. Unselect “Use flash loader(s)”. Then, click the "OK” button.

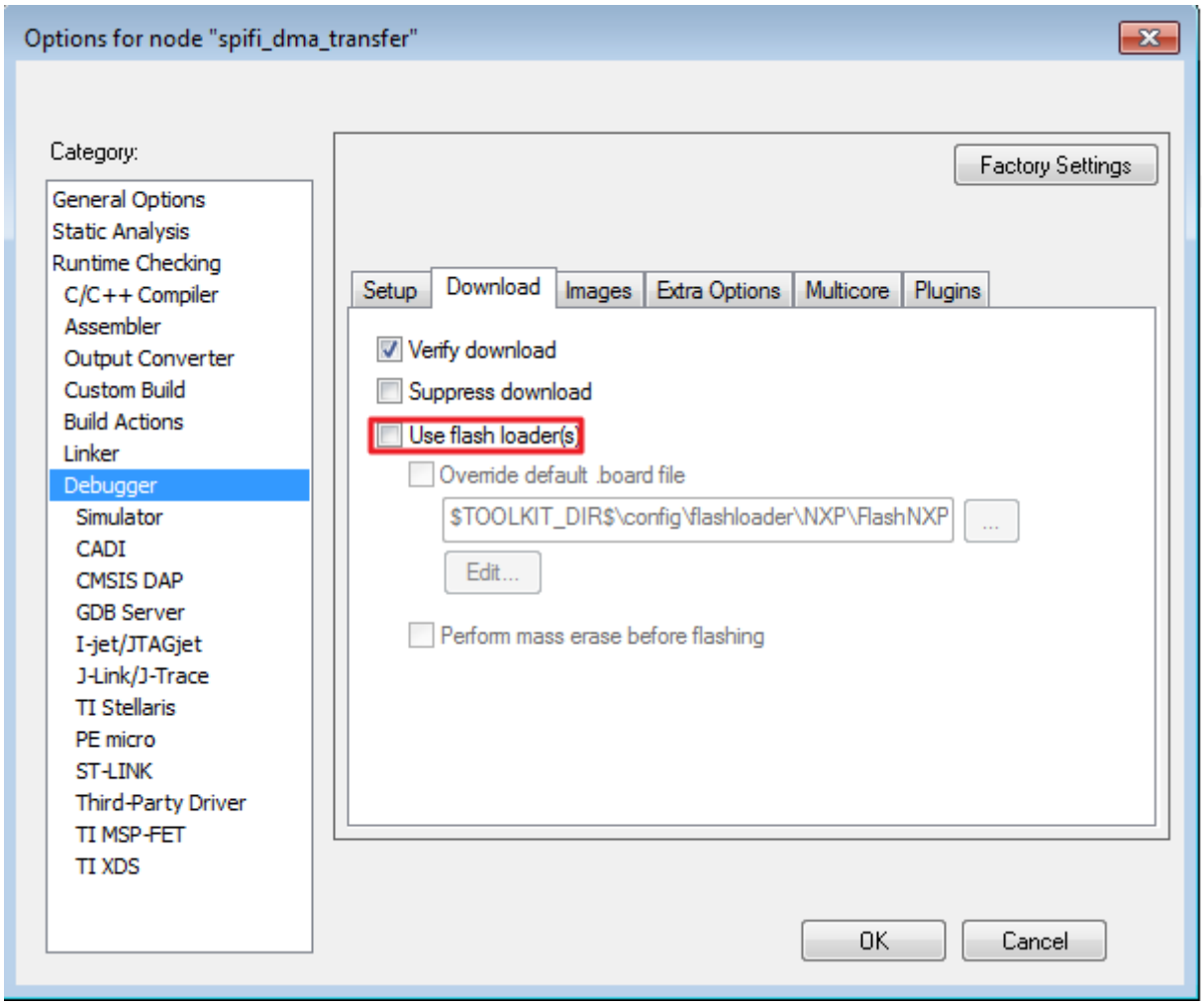

Figure 25. Unselect “Use flash loader(s)”

5. Select “Use command line options”. Then, click the "OK” button.

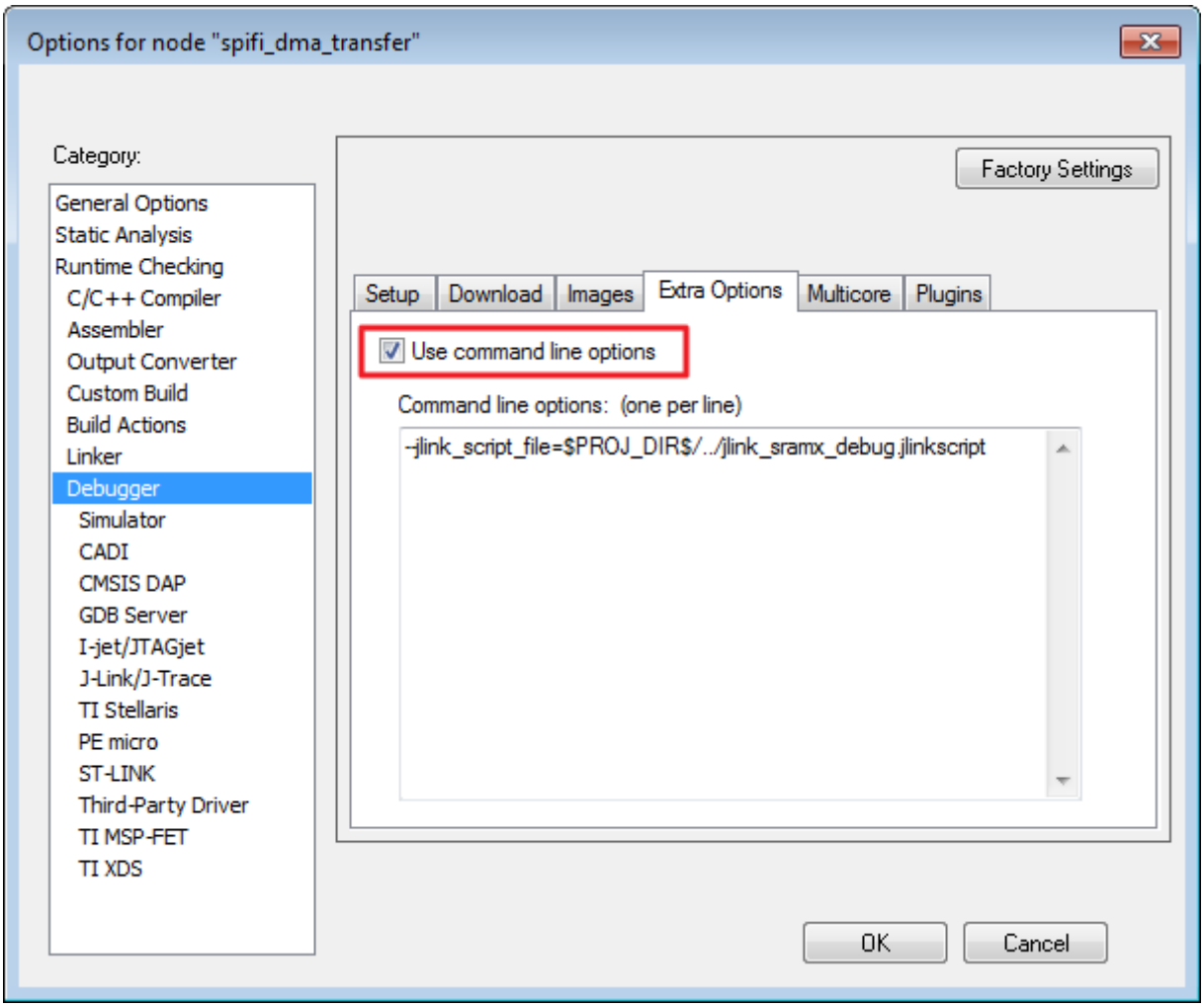

Figure 26. Select "Use command line options"

6. Select the "Core" option and click the "OK" button.

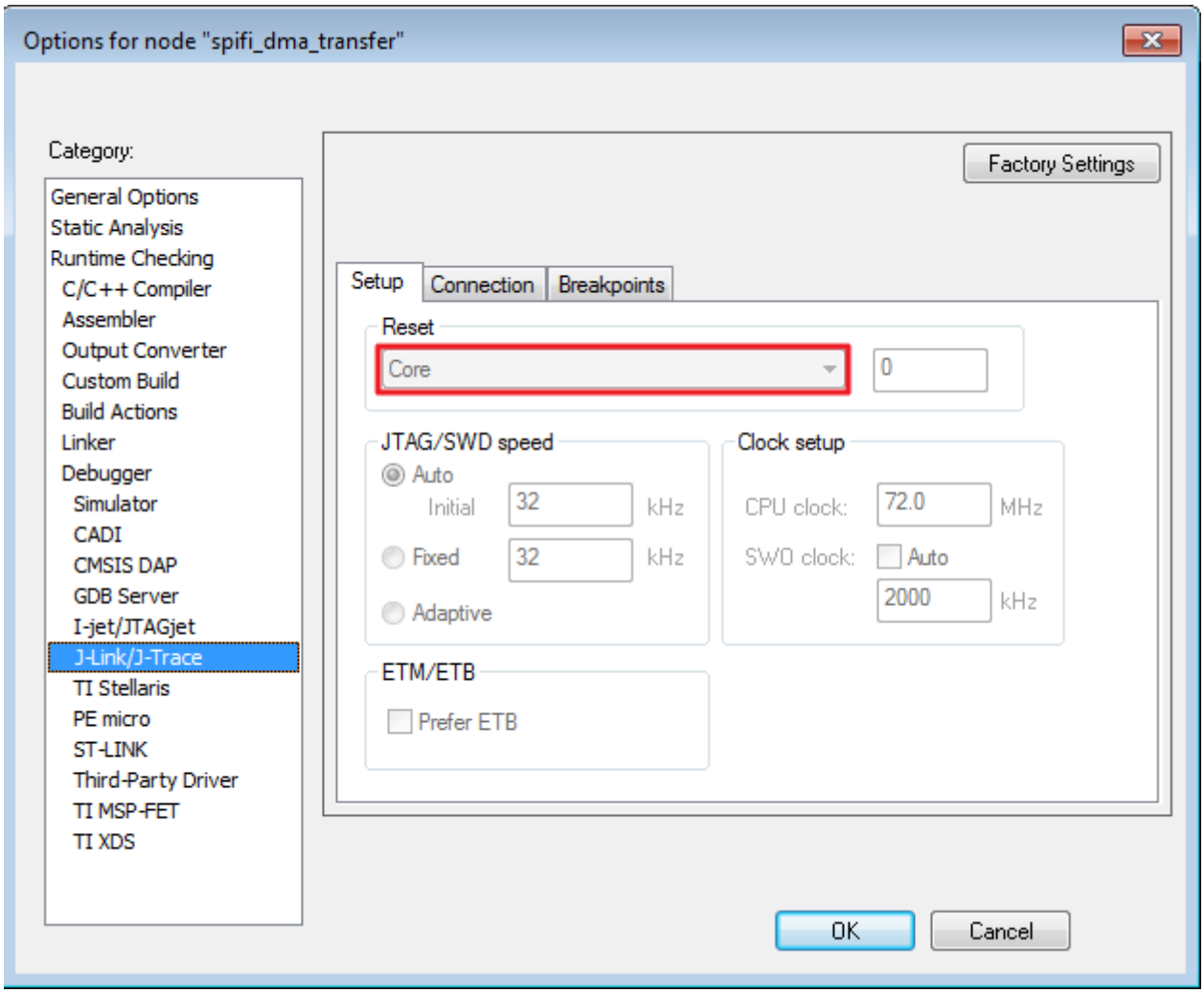

Figure 27. Select "Core" option

7. To build the example application, click the "Make" button, highlighted in red below.

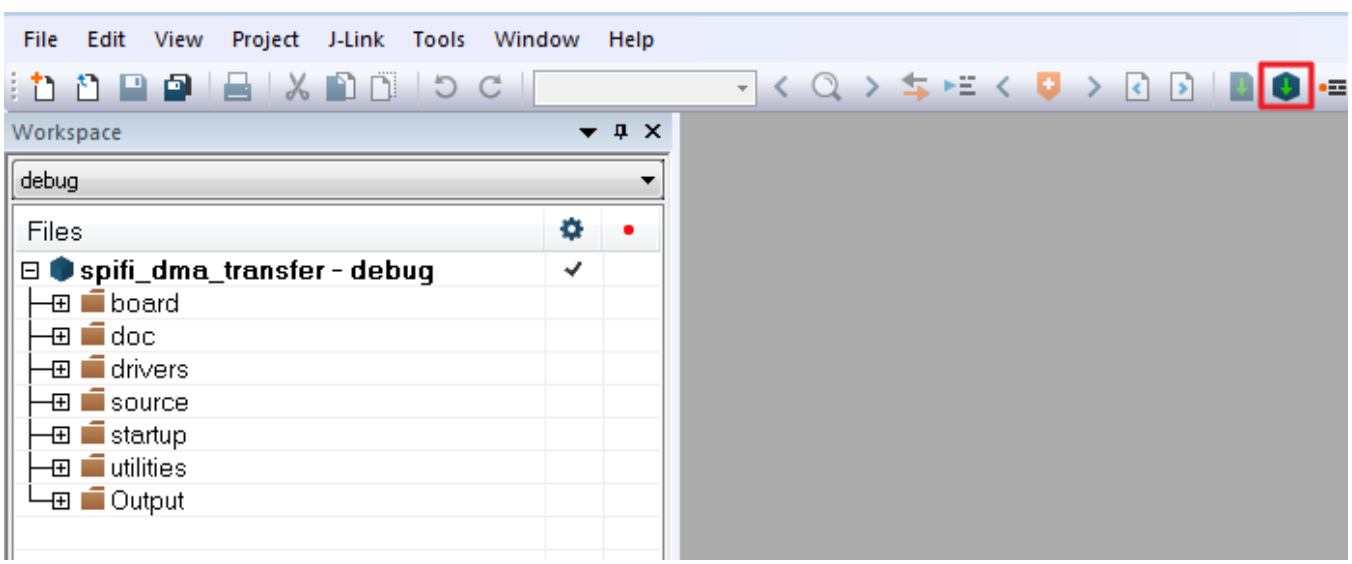

Figure 28. Build the example application

3.7 Run a SRAM example application using J-Link/J-Trace

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

Figure 29. Download and Debug button

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

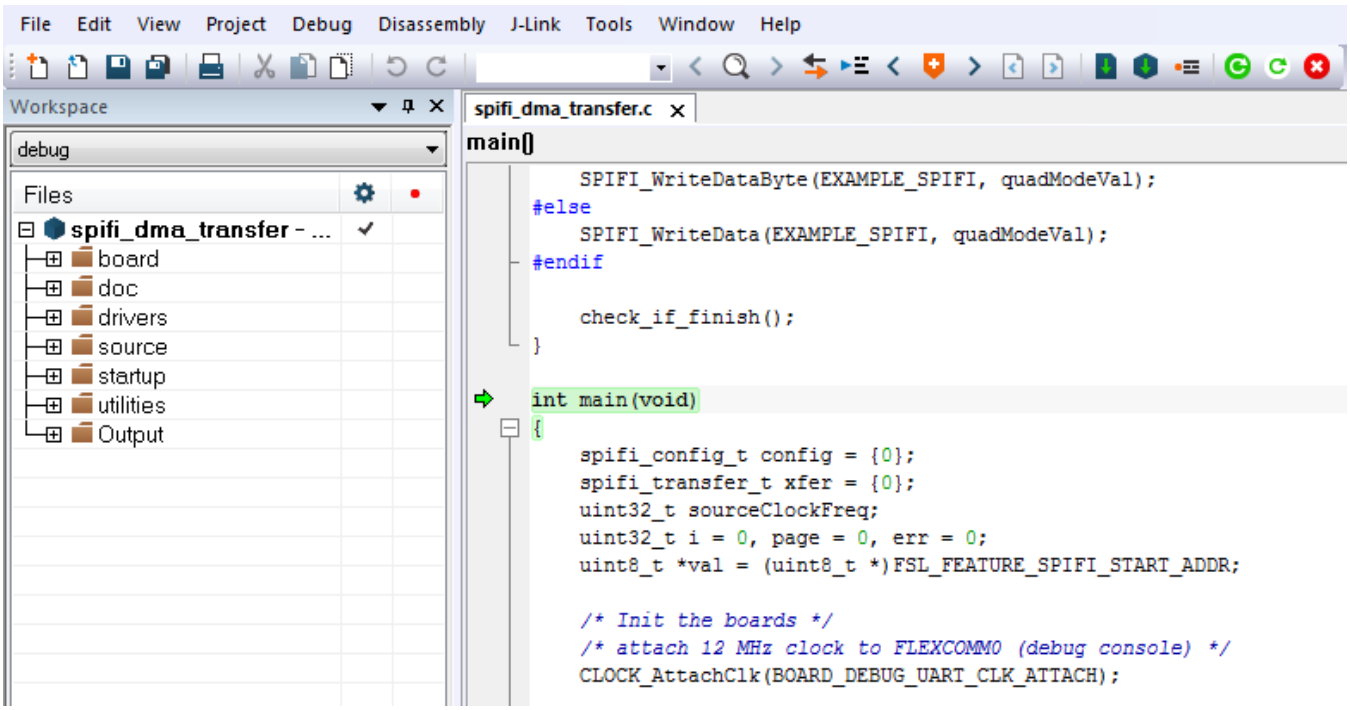

Figure 30. Stop at main() when running debugging

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

Figure 31. Go button

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

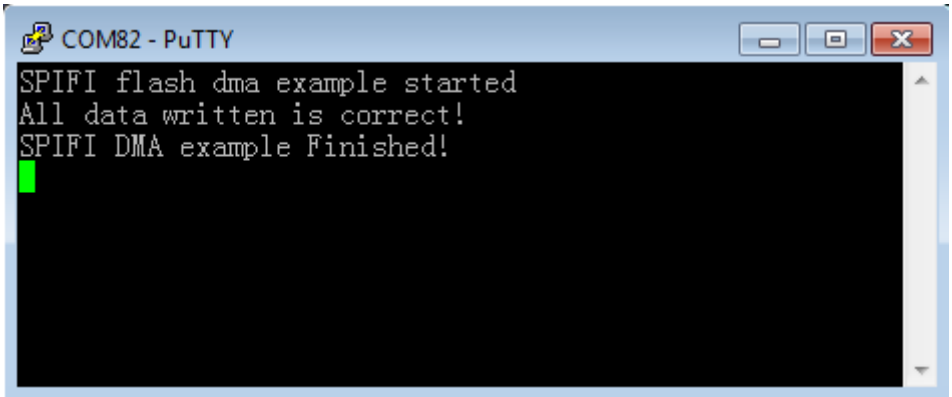

Figure 32. Text display of the spifi_ dma_transfer example

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 LPCXpresso54018 hardware platform is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

Before downloading and running 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 the 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 you are 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 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

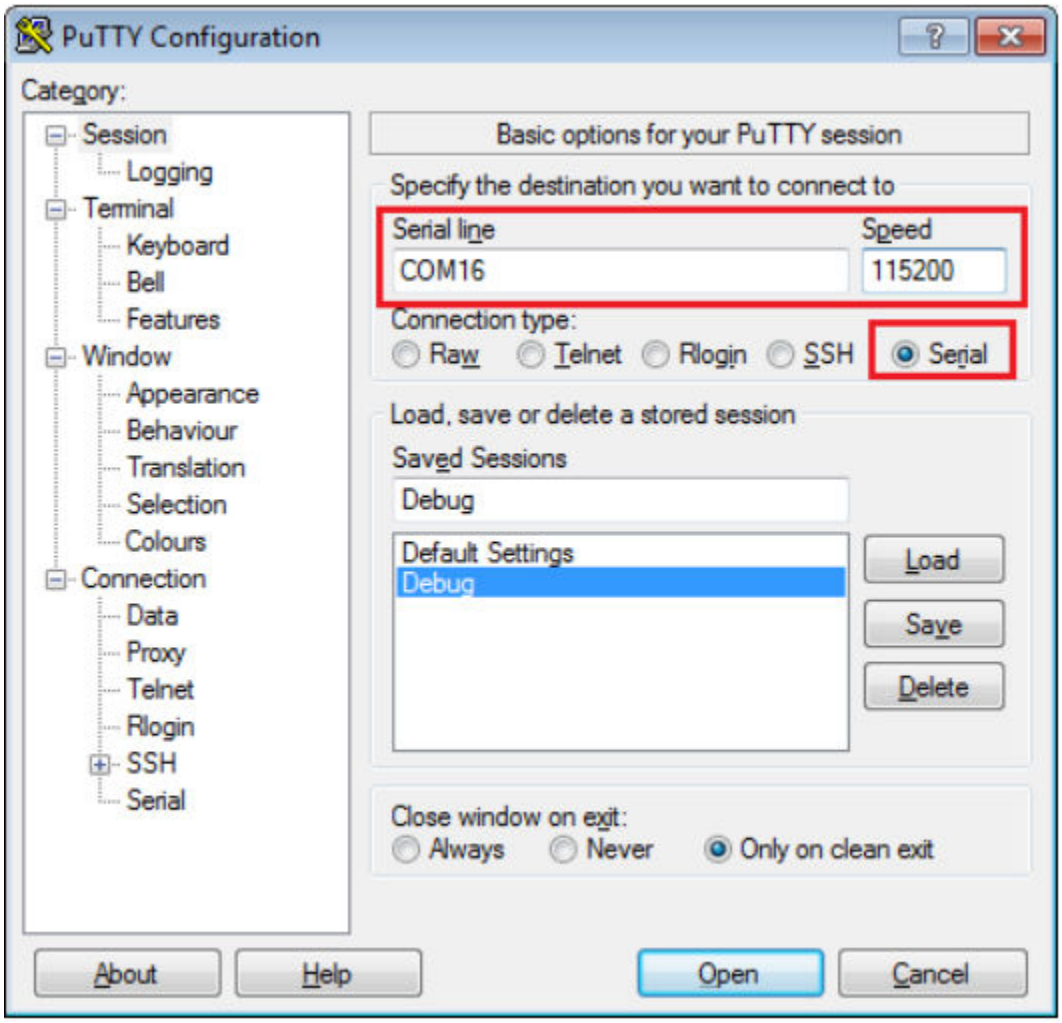

Figure 33. Terminal (PuTTY) configuration

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.

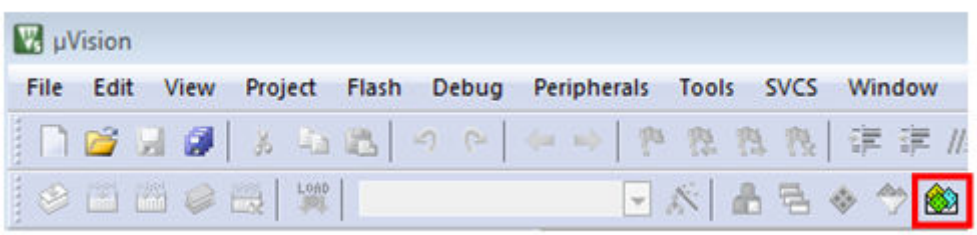

Figure 34. Launch the Pack installer

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

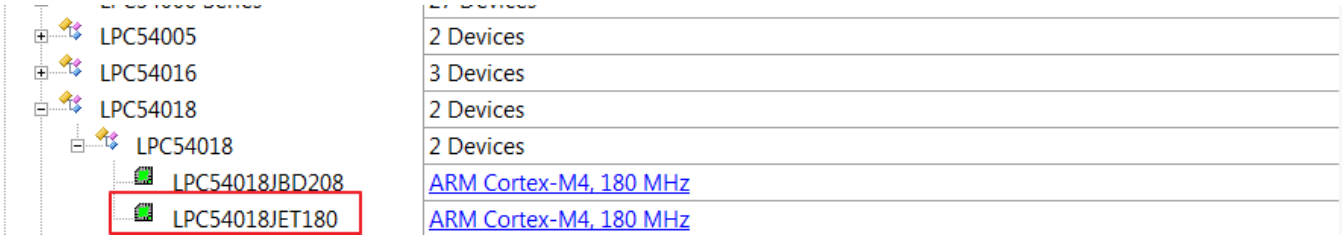

Figure 35. Install CMSIS pack

4.2 Build a non-XIP (plain load) 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/lpcxpresso54018/demo_apps/hello_world/mdk/hello_world.uvmpw
- To build the demo project, select the "Rebuild" button, highlighted in red.

Figure 36. Build the demo

- The build completes without errors.

4.3 Run a non-XIP (plain load) example application

1. To debug the application, click the “Start/Stop Debug Session” button, highlighted in red.

NOTE

The application is only downloaded into the SRAM when debugging. If you need to program the image to external flash, see *Section 4.4, "How to program the non-XIP (plain load) example application to external flash"*.

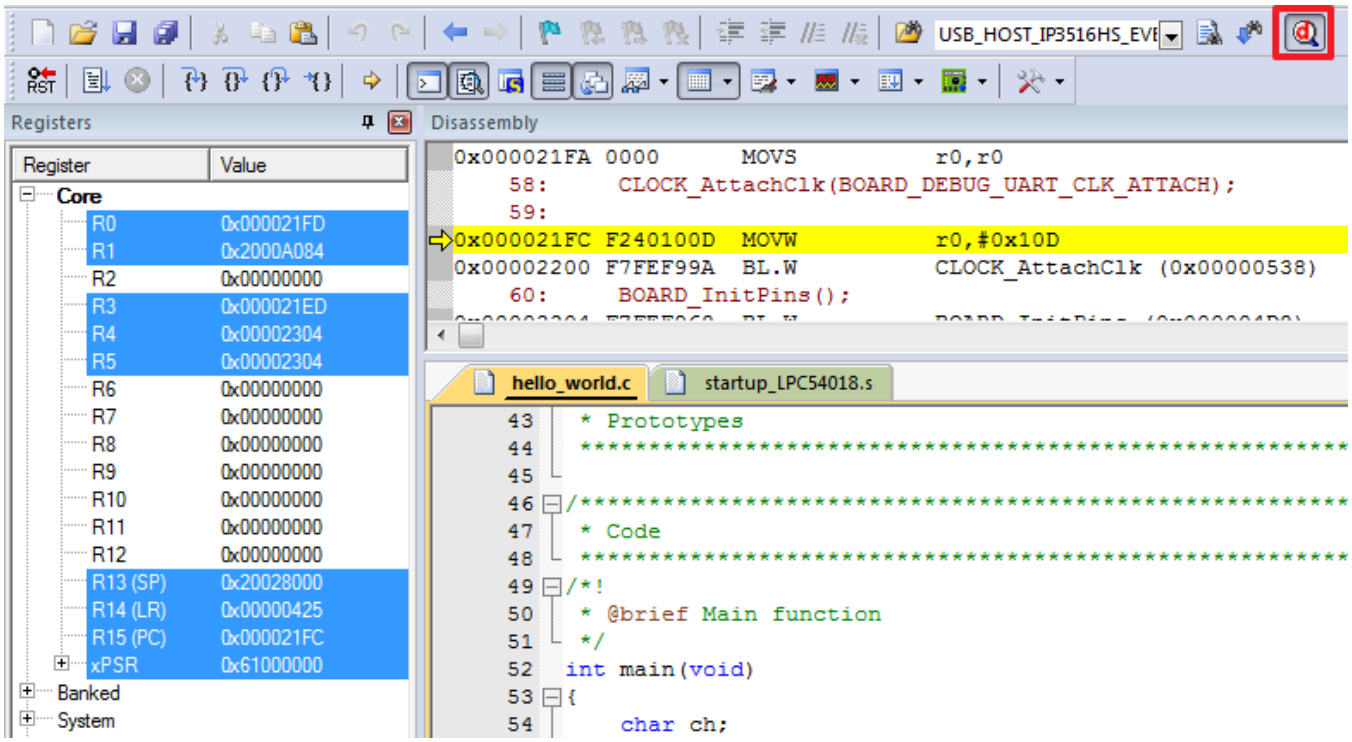

Figure 37. Stop at main() when run debugging

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

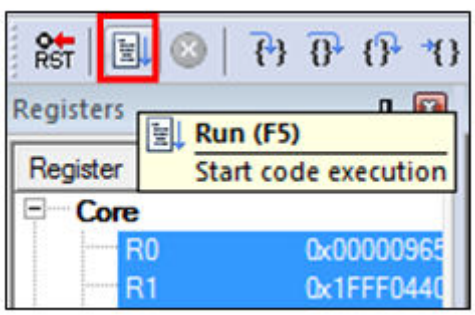

Figure 38. Go button

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.

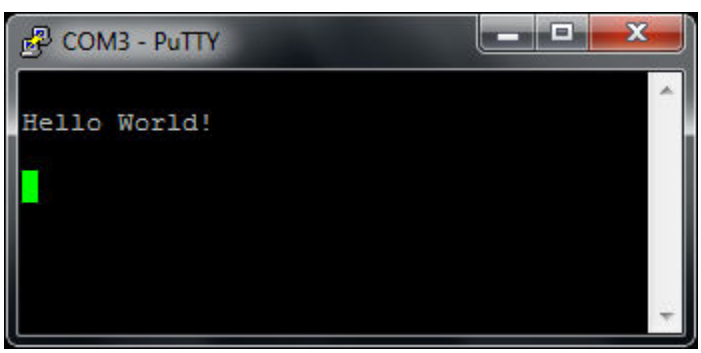

Figure 39. Text display of the hello_world demo

4.4 How to program the non-XIP (plain load) example application to external flash

1. Click the configure target option button and select the “SYSRESETREQ” reset option.

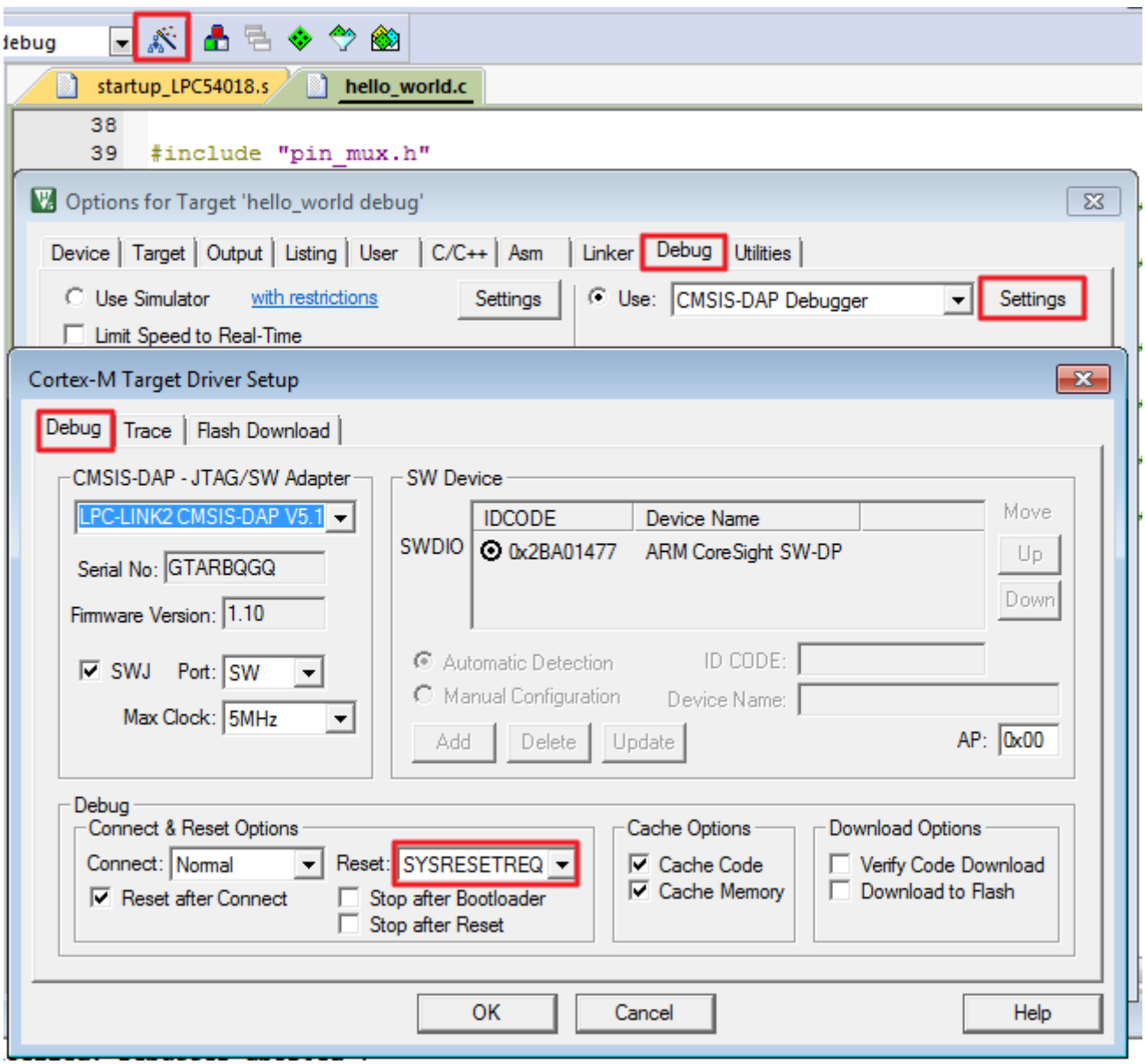

Figure 40. Select “SYSRESETREQ” reset option

2. Select the “Flash Download” option and click the “Add” button.

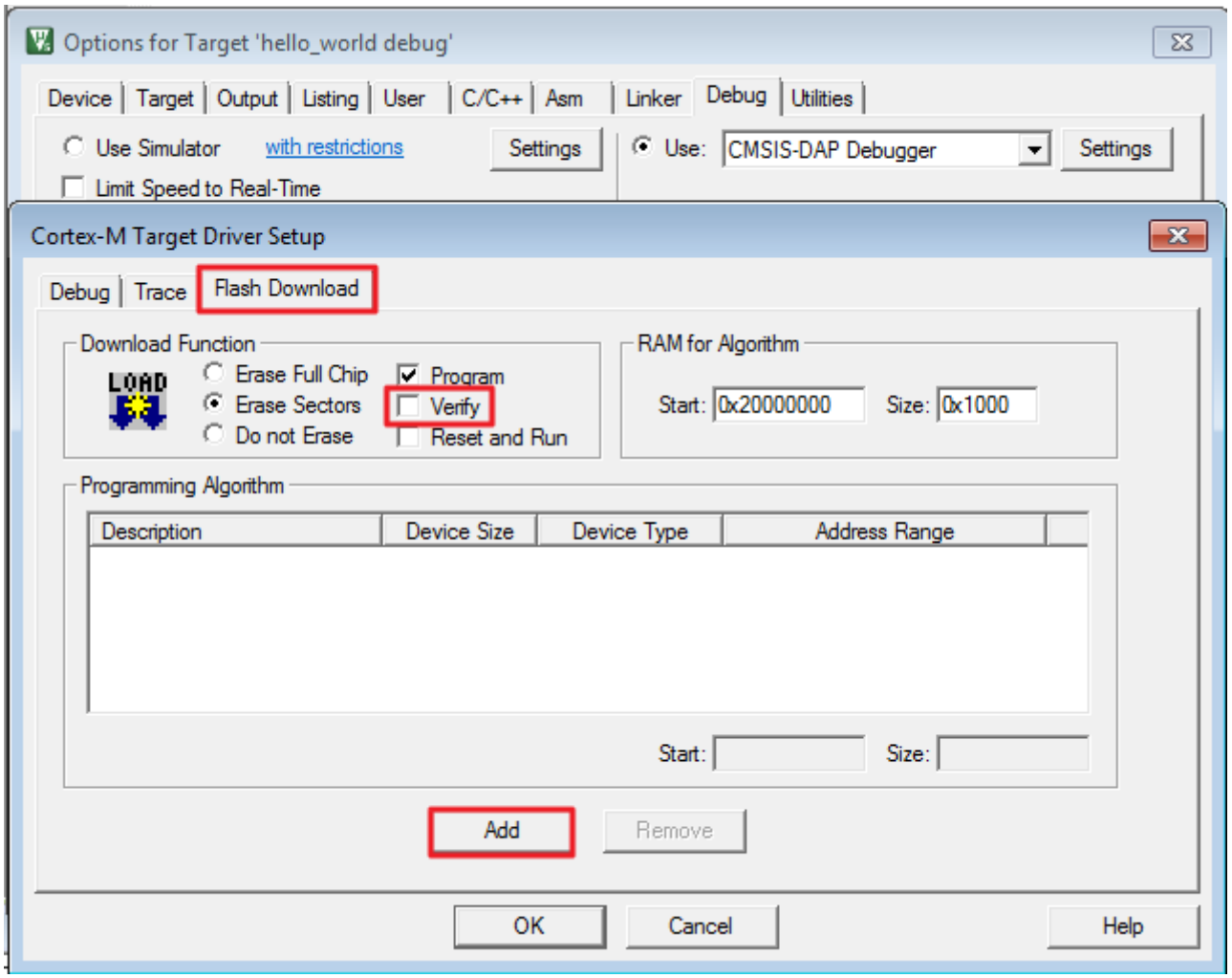

Figure 41. Select “Flash Download” option

3. Select the “LPC540xx W25Q128JVFM SPIF” option. For the LPCXpresso54018 board, select "LPC540xx MX25L12835FM2I" for the LPC54018-IoT-Module. Then, click the “Add” button.

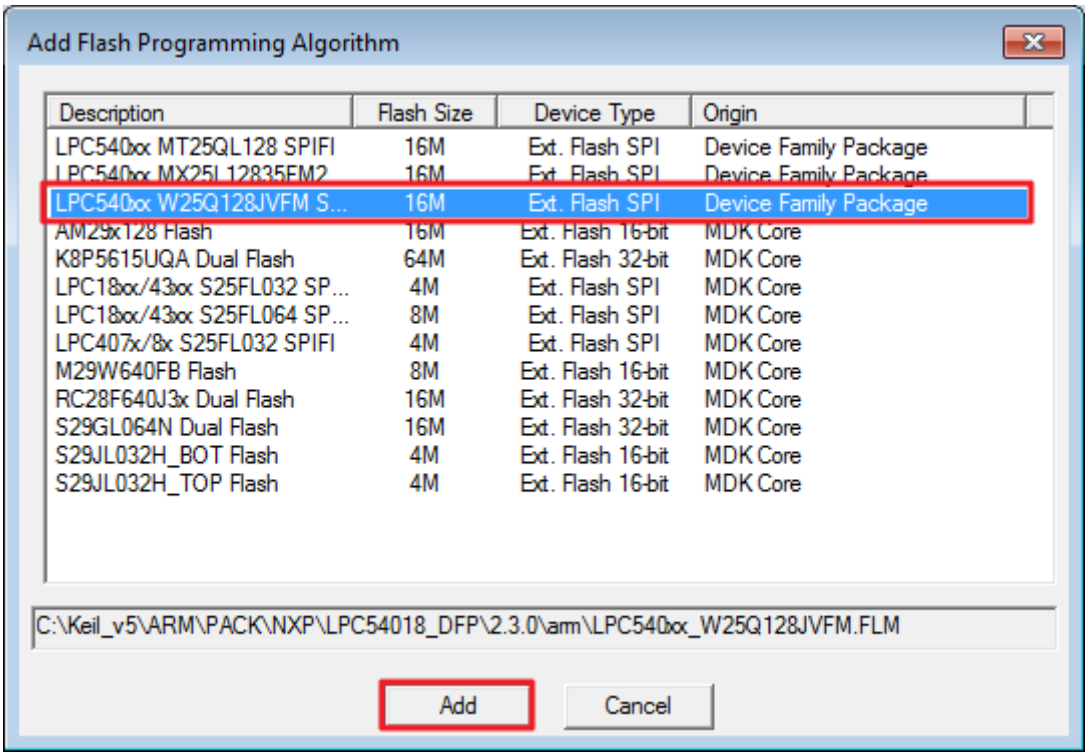

Figure 42. Select "LPC540xx W25Q128JVFM SPIF" option

4. Set 0x00000000 as the start address and click the "OK" button.

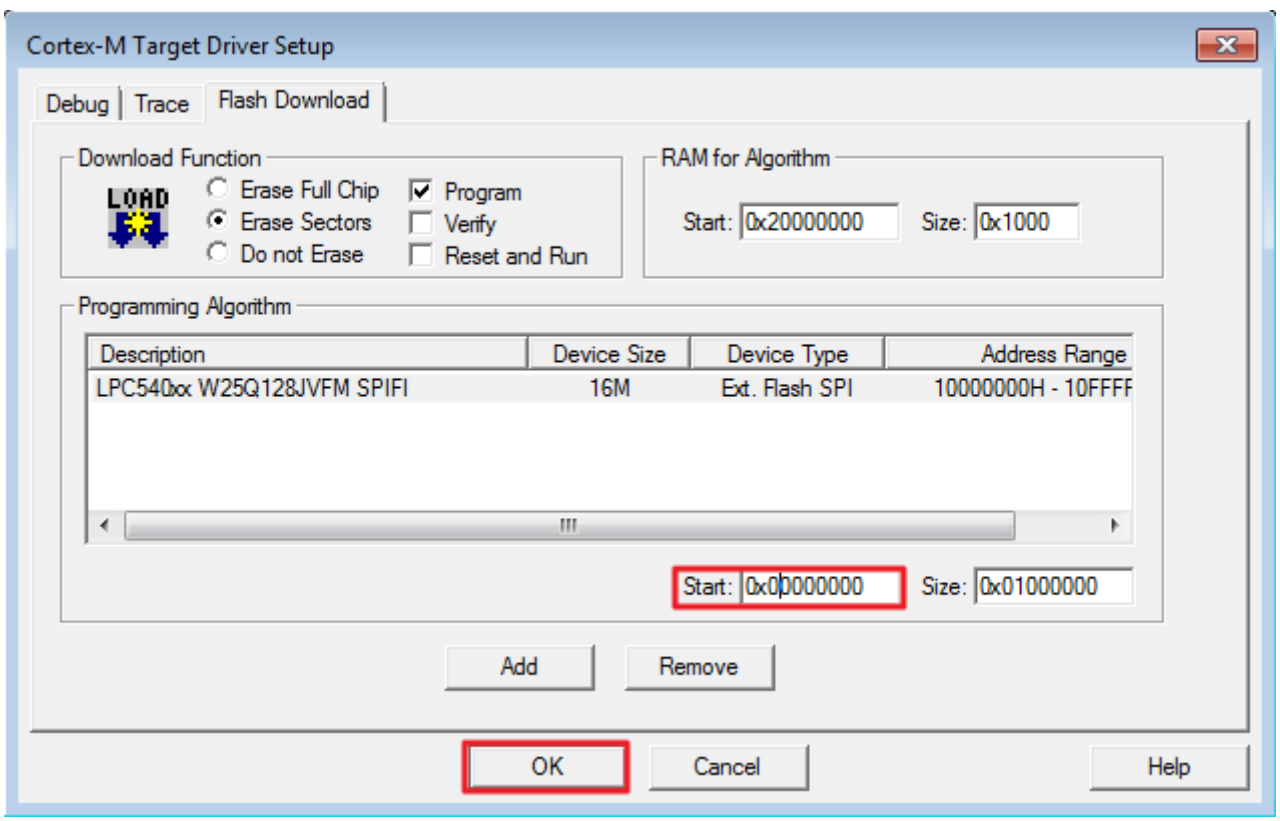

Figure 43. Set 0x00000000 as the start address

5. Click the "LOAD" button.

NOTE

If 'LOAD' fails, press the SW4 button on the board, then repower the board or reset the board (get into ISP mode). Keep pressing SW4 when clicking the 'LOAD' button again to program the application into the external flash.

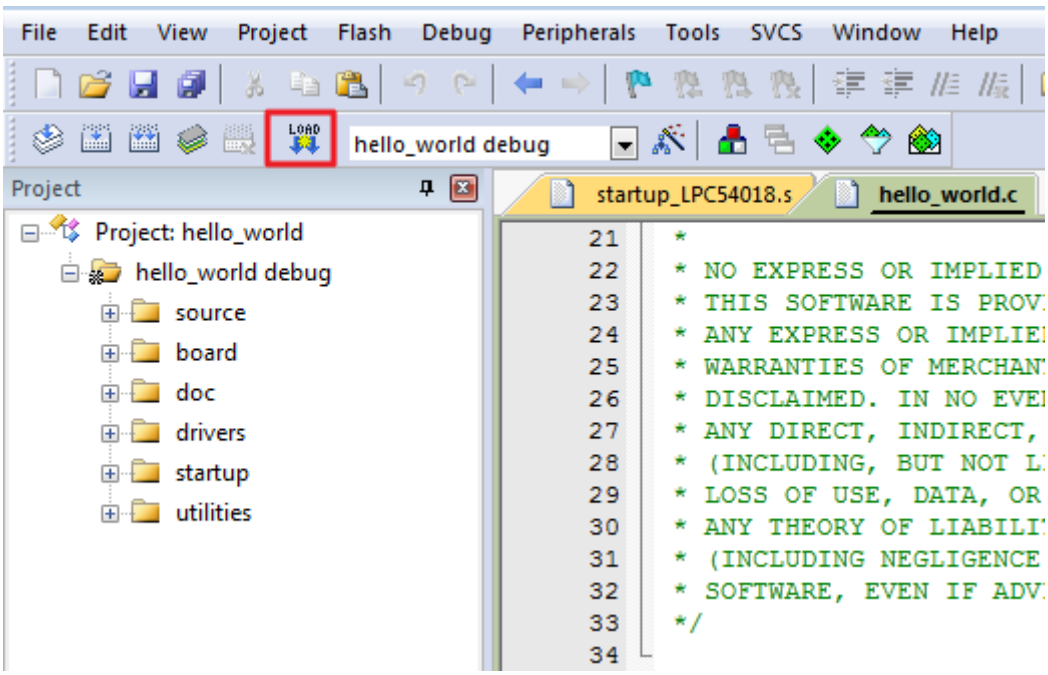

Figure 44. Click the “LOAD” button.

6. Press the reset button on the board to run the example.

   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.

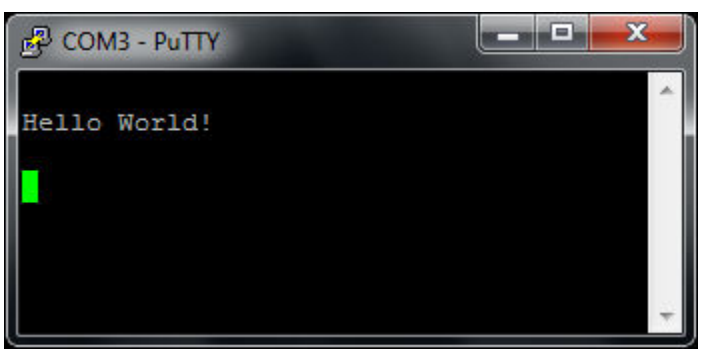

Figure 45. Text display of the hello_world demo

4.5 How to program the non-XIP (plain load) example application to external flash using J-Link

1. Click the configure target option button and select the “Normal” reset option.

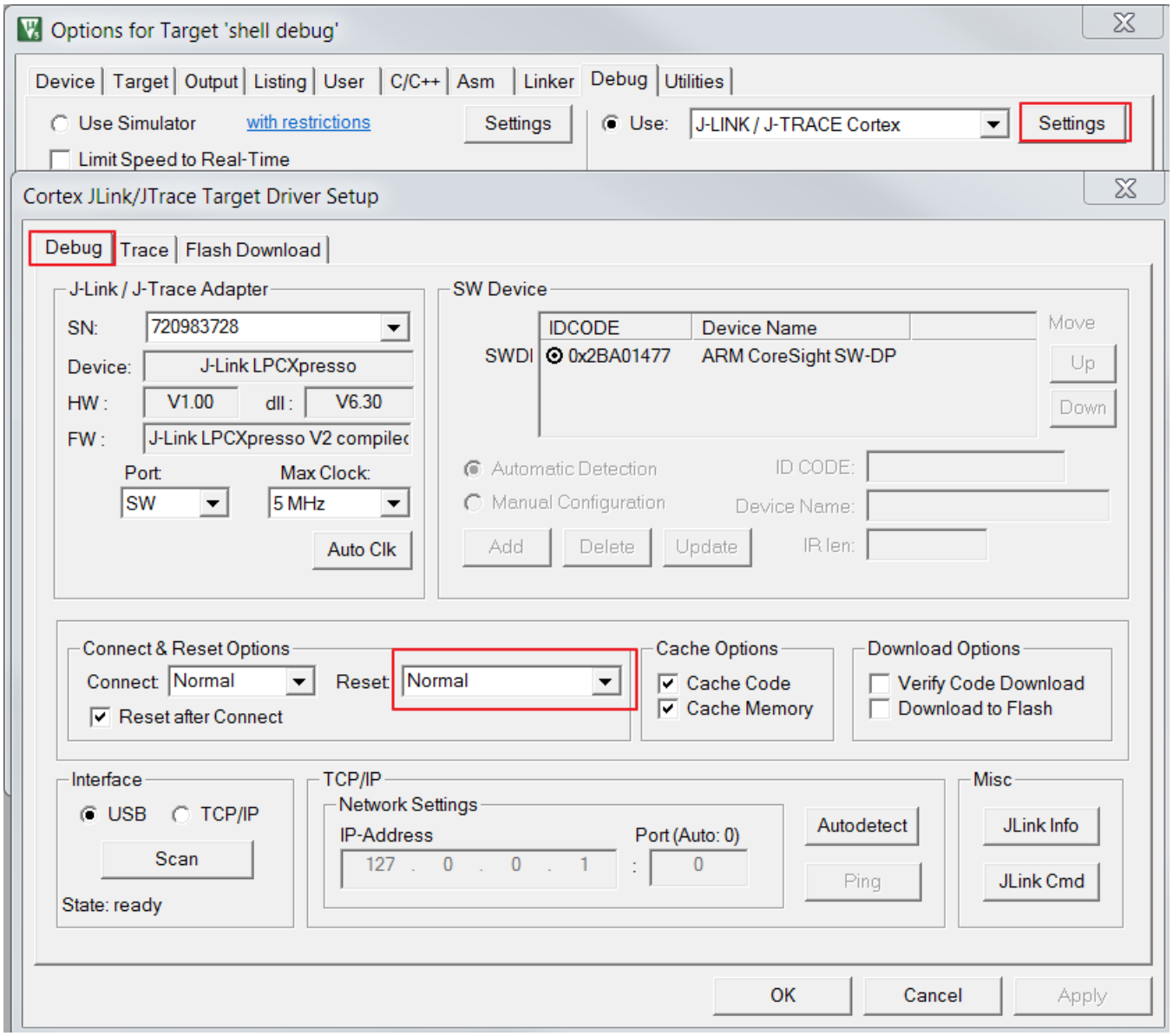

Figure 46. Select "Normal" reset option

2. Select the "Flash Download" option and click the "Add" button.

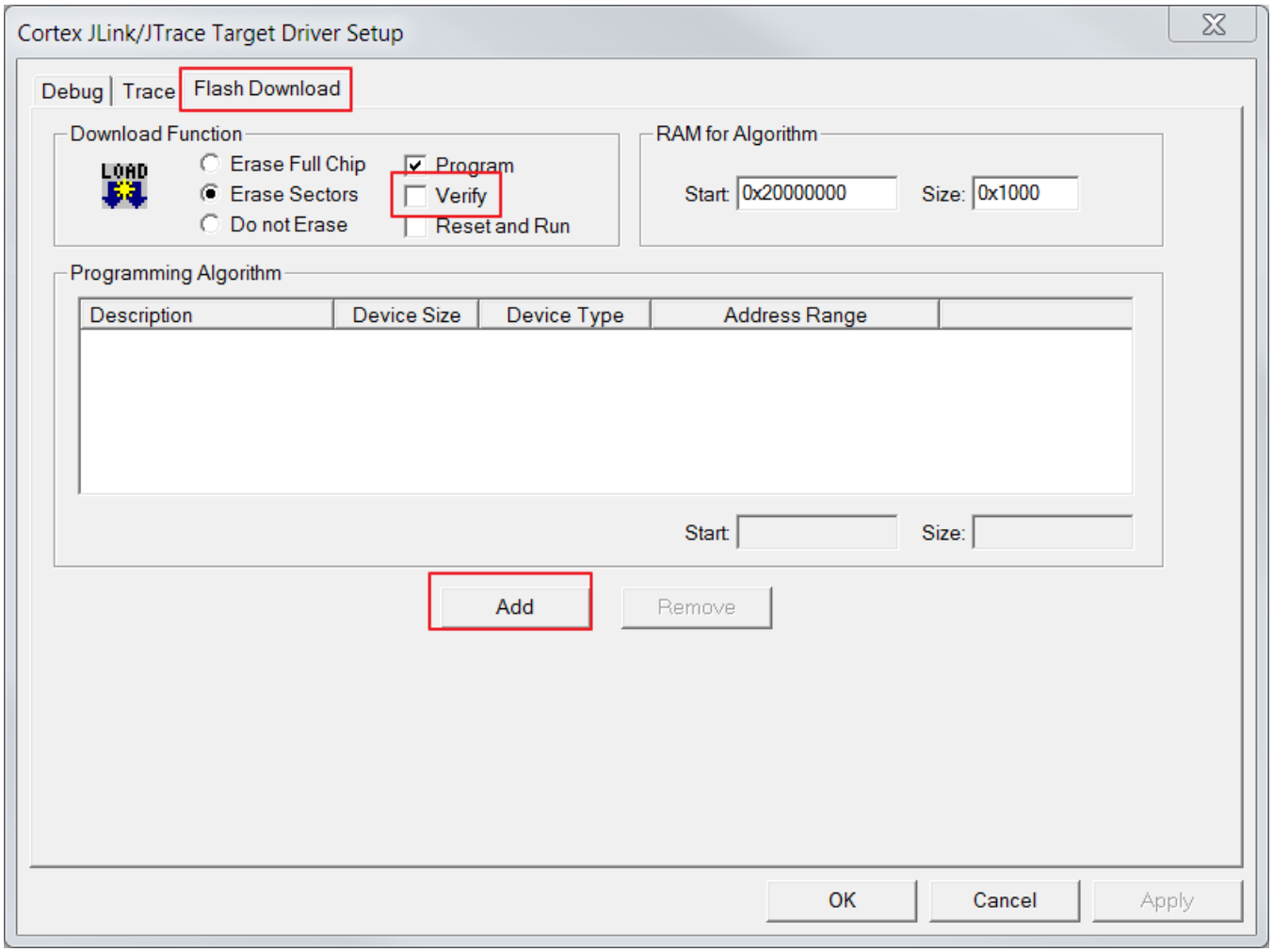

Figure 47. Select “Flash Download” option

3. Select the “LPC540xx W25Q128JVFM SPIF” for the LPCXpresso54018 board. Select "LPC540xx MX25L12835FM2I" for the LPC54018-IoT-Module option, and click the “Add” button.

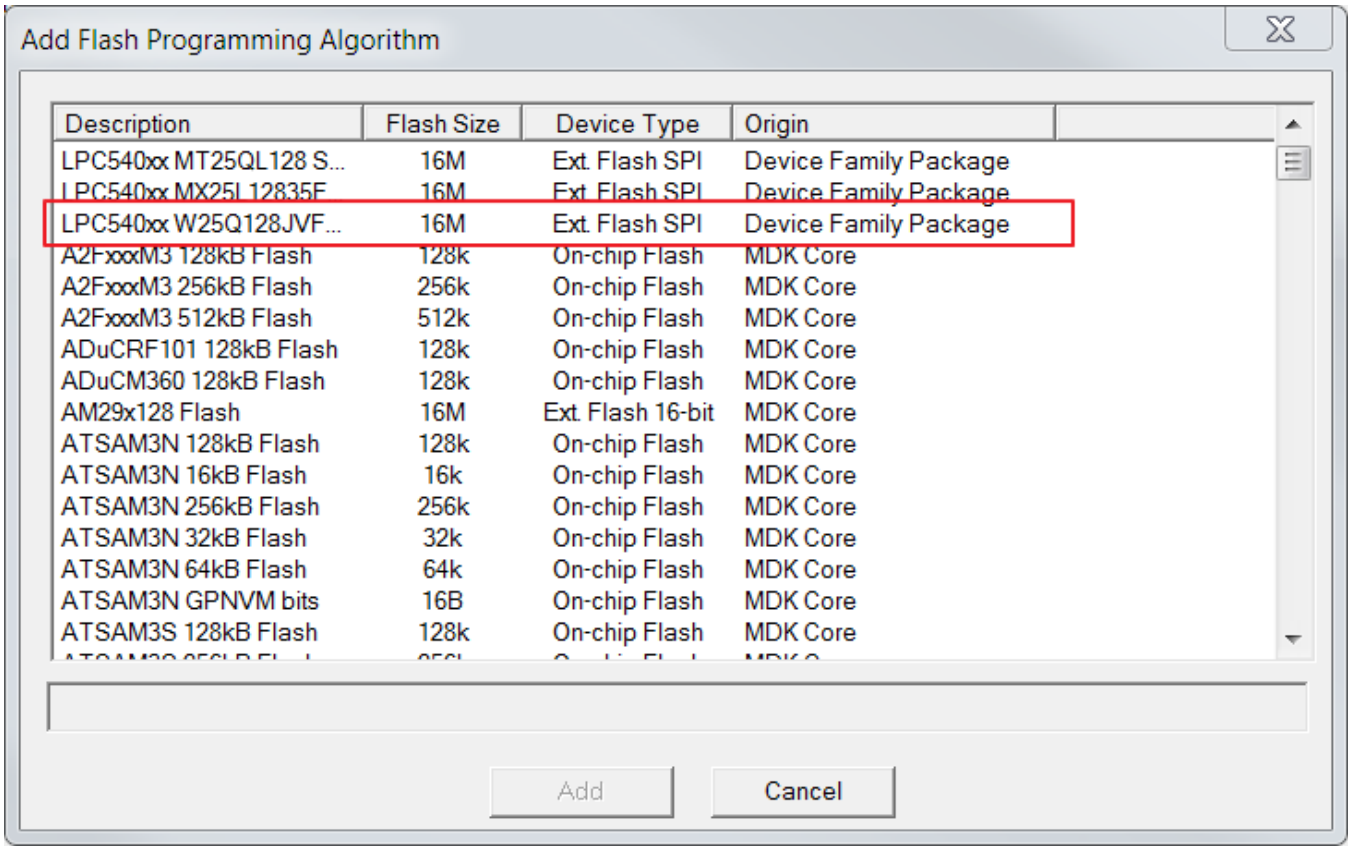

Figure 48. Select “LPC540xx W25Q128JVFM SPIF” option

NOTE

Select "LPC540xx MX25L12835FM2I" for the LPC54018-IoT-Module.

4. Set 0x00000000 as the start address and click the “OK” button.

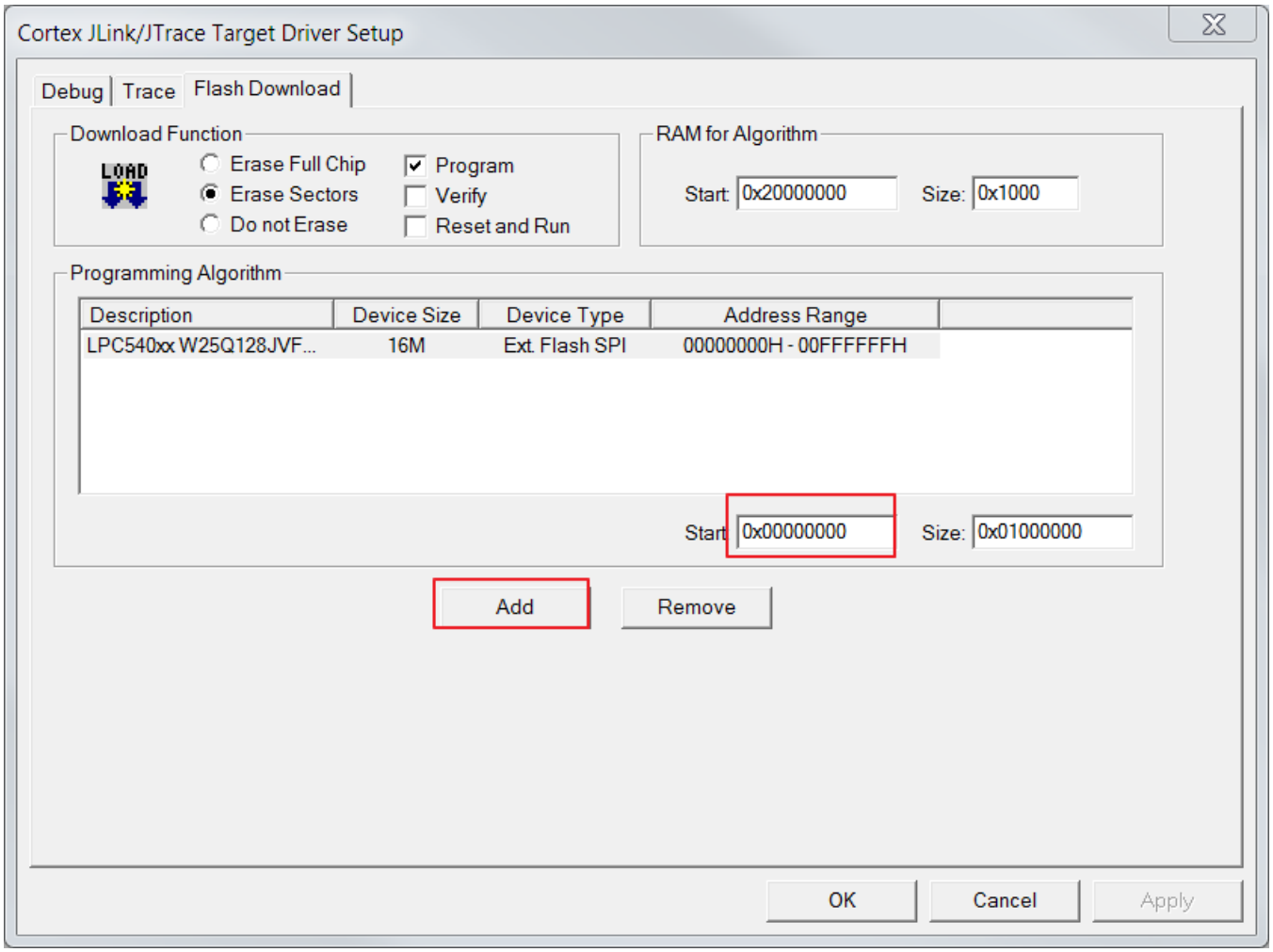

Figure 49. Set 0x00000000 as the start address

5. Click the “LOAD” button.

NOTE

If 'LOAD' fails, press the SW4 button on the board, then repower the board or reset the board (get into ISP mode). Keep pressing SW4 when clicking the 'LOAD' button again to program the application into the external flash.

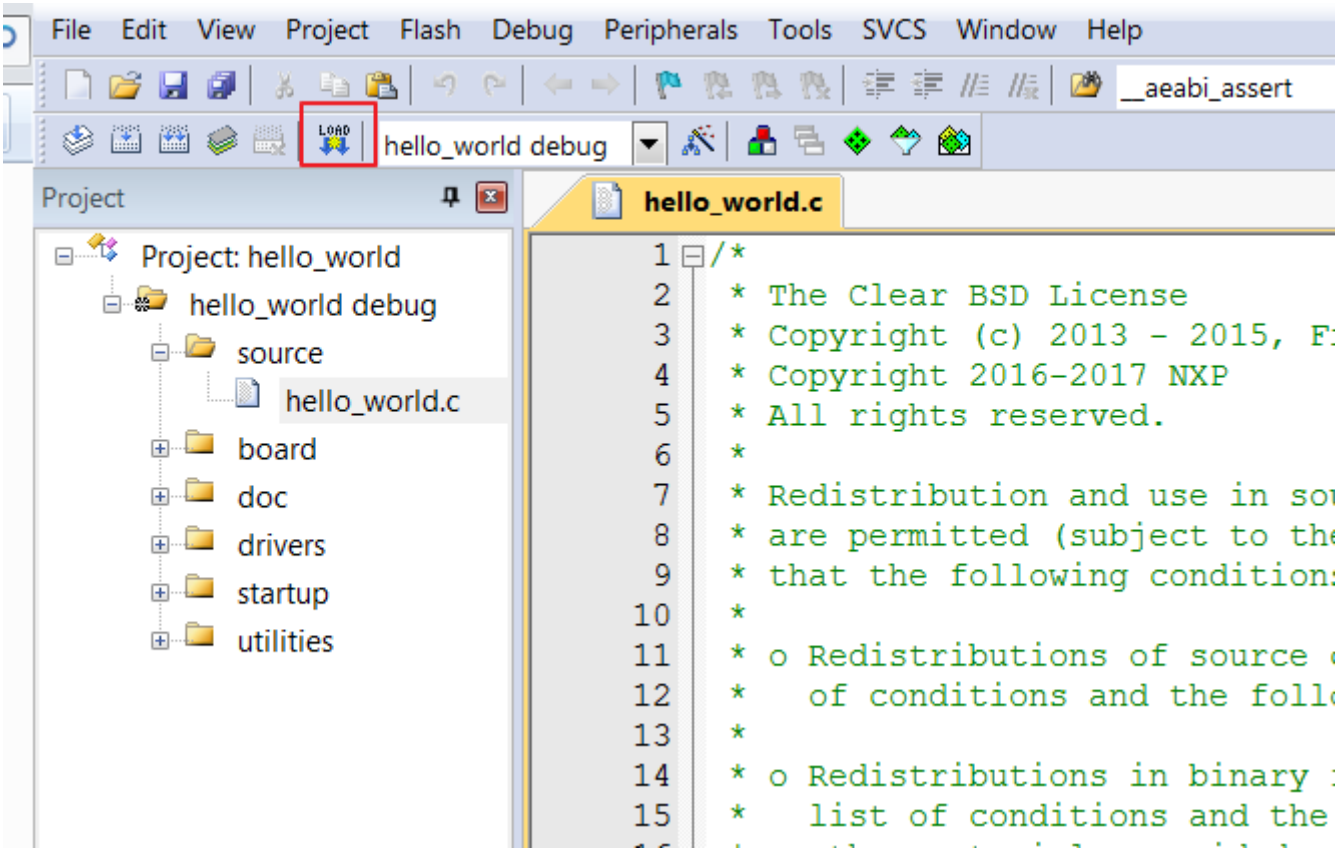

Figure 50. Click the "LOAD" button.

6. Press the reset button on the board to run the example.

   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.

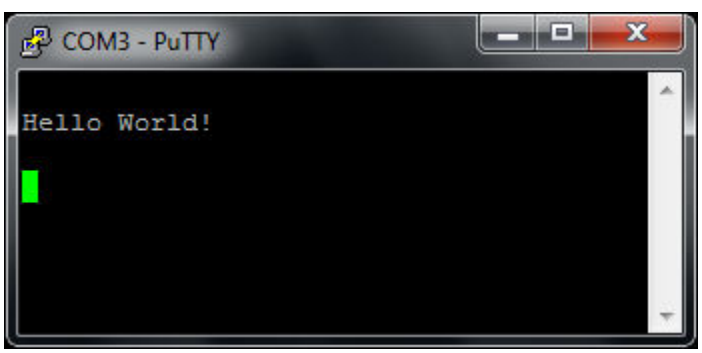

Figure 51. Text display of the hello_world demo

4.6 Build an XIP 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 <demo_name>.uvmpw, so for this specific example, the actual path is:

<install_dir>/boards/lpcxpresso54018/demo_apps/hello_world_qspi_xip/mdk/hello_world_qspi_xip.uvmpw

2. To build the demo project, select the "Rebuild" button, highlighted in red.

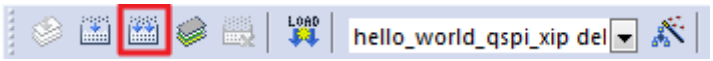

Figure 52. Select “Flash Download” option

3. The build completes without errors

4.7 Run an XIP example application

1. Click the “LOAD” button, highlighted in red.

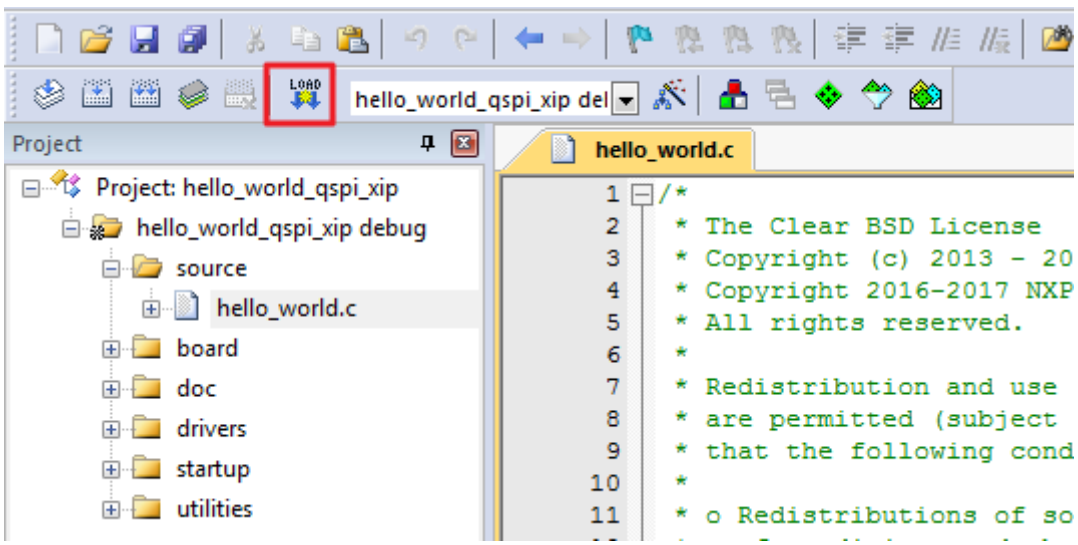

Figure 53. Click "LOAD" button

2. Press the reset button on the board to run the example.

   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.

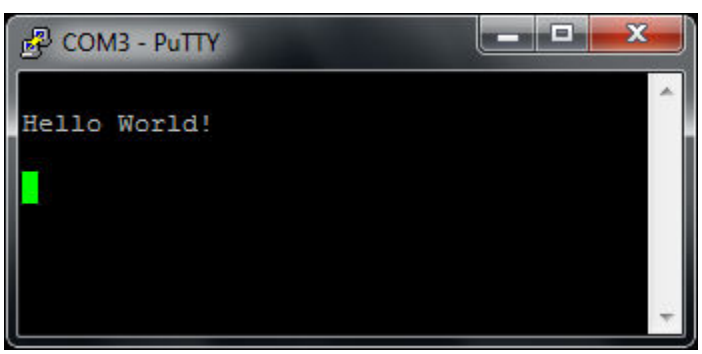

Figure 54. Text display of the hello_world demo

5 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 targeted for the LPCXpresso54018 hardware platform is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

5.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run a 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.

5.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset (in other words, compiler, linker, etc.). The GCC toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes Supporting LPCXpresso54018*. (document MCUXSDKLPC540XXRN).

5.1.2 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 SDK does not utilize 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.

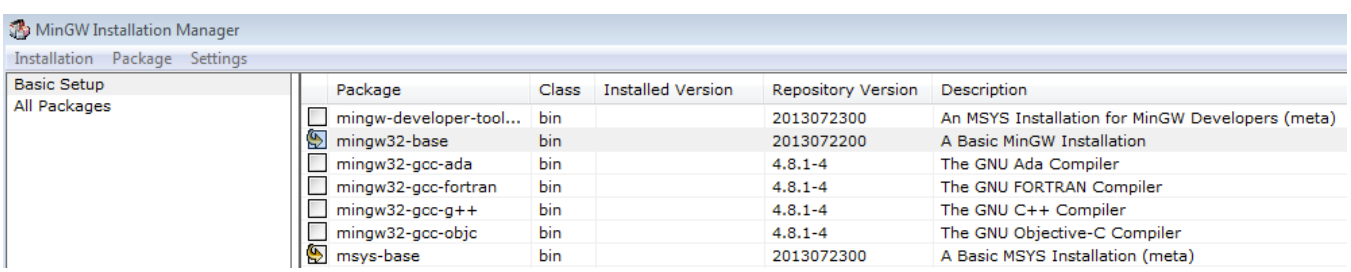

Figure 55. Setup MinGW and MSYS

4. Click “Apply Changes” in the “Installation” menu and follow the remaining instructions to complete the installation.

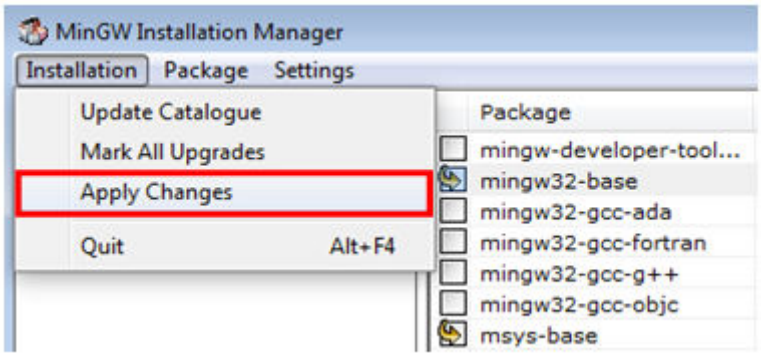

Figure 56. Complete MinGW and MSYS installation

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 does not work.

NOTE

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

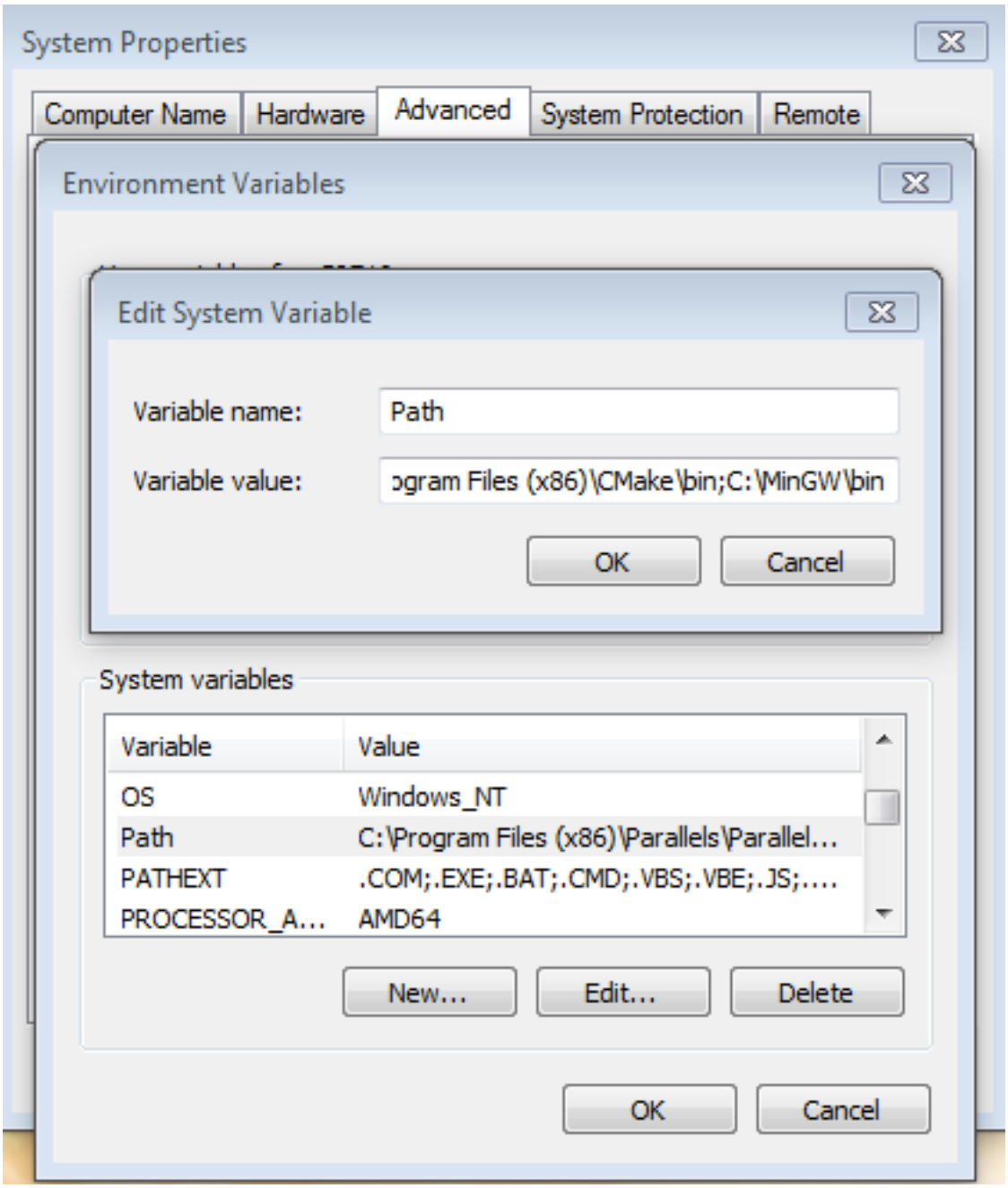

Figure 57. Add Path to systems environment

5.1.3 Add a new system environment variable for ARMGCC_DIR

Create a new *system* environment variable and name it 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\6 2017q2

Reference the installation folder of the GNU ARM GCC Embedded tools for the exact path name of your installation.

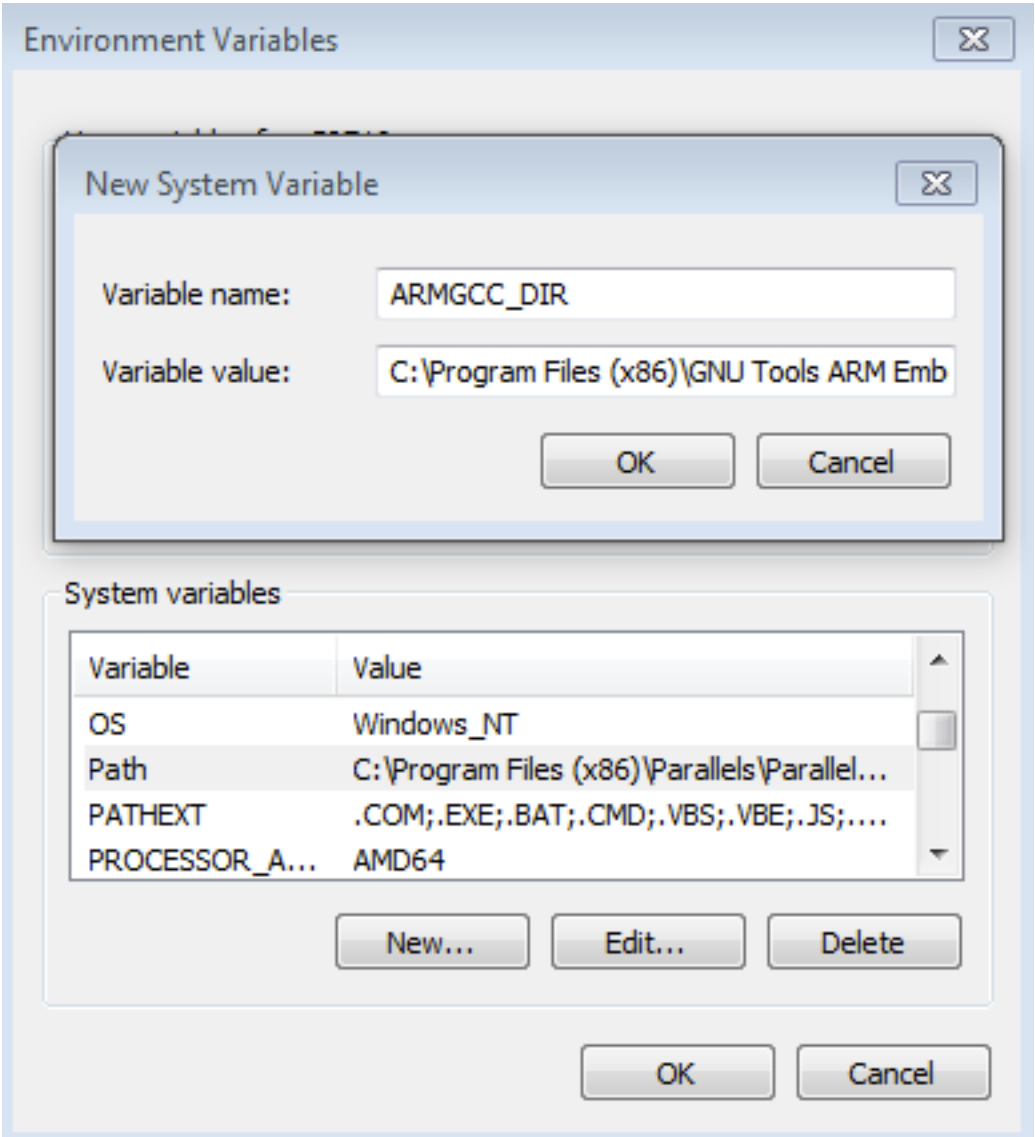

Figure 58. Add ARMGCC_DIR system variable

5.1.4 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.

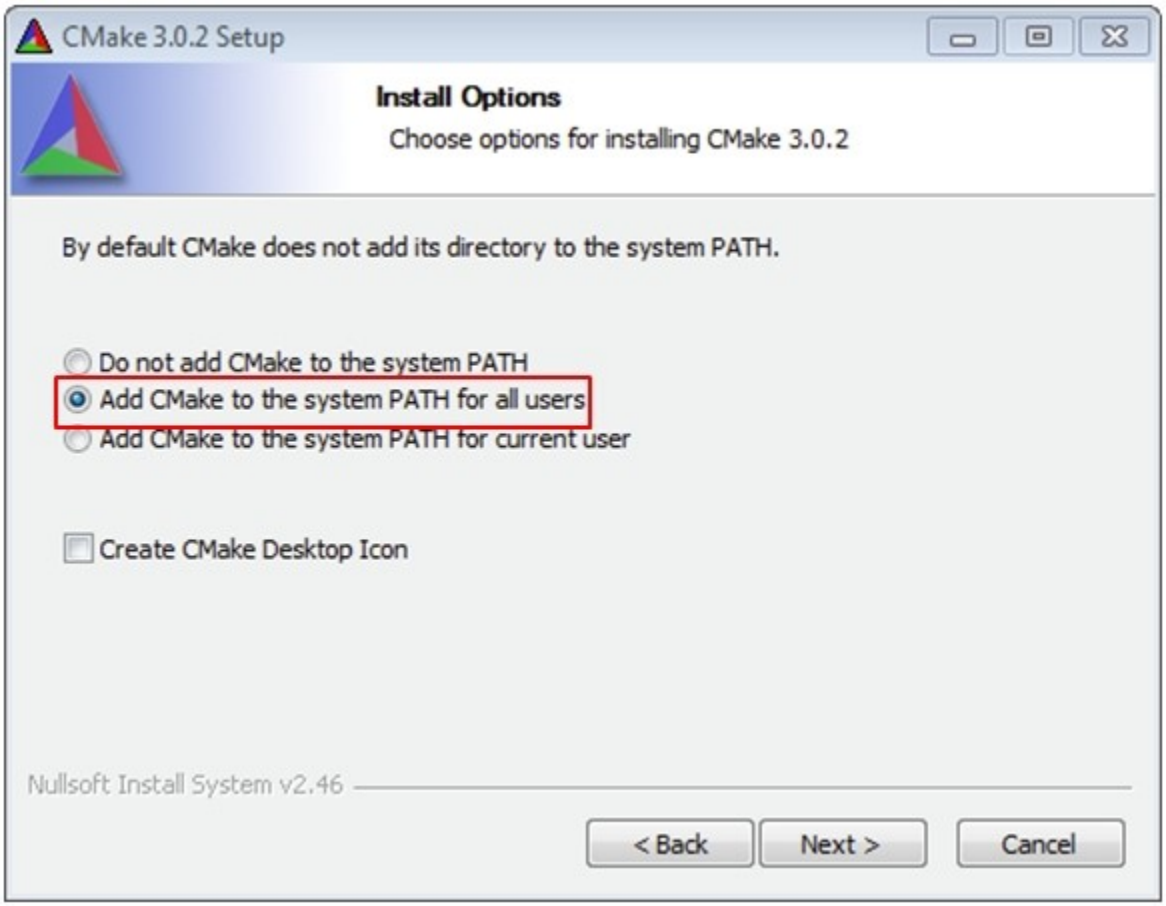

Figure 59. Install CMake

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.

5.2 Build an non-XIP (plain load) 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”.

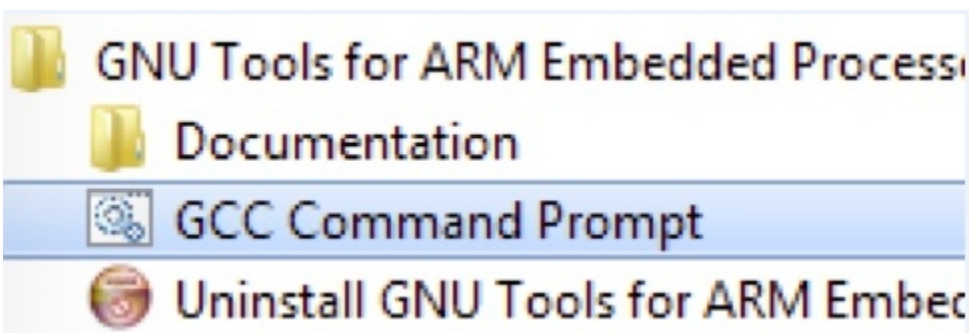

Figure 60. Launch command prompt

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/lpcxpresso54018/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 the "build_debug.bat" file in Windows Explorer to perform the build. The output is shown in this figure:

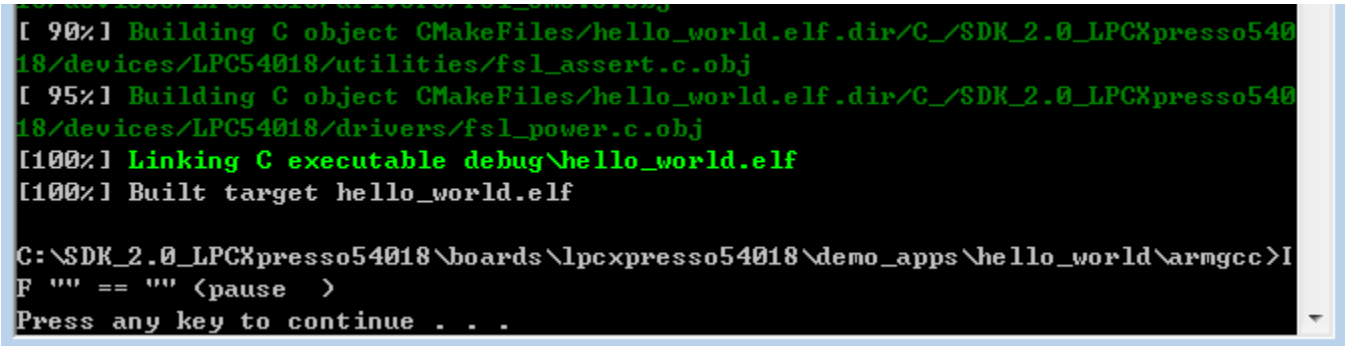

Figure 61. hello_world demo build successful

5.3 Run an non-XIP (plain load) example application

This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, two things must be done:

- Make sure that either:
  - The LPC-Link2 interface on your board is programmed with the J-Link firmware. To determine if your board supports LPC-Link2, see Appendix B. For instructions on reprogramming the LPC-Link2 interface, see Appendix C. If your board does not support LPC-Link2, then a standalone J-Link pod or standalone LPC-Link2 (OM13054) pod is required.
  - You have a standalone J-Link pod that is connected to the debug interface of your board. Note that some hardware platforms require hardware modification in order to function correctly with an external debug interface.

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 Link2 USB connector (named Link for some boards) and the PC USB connector. If you are connecting for the first time, allow about 30 seconds for the devices to enumerate.
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 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

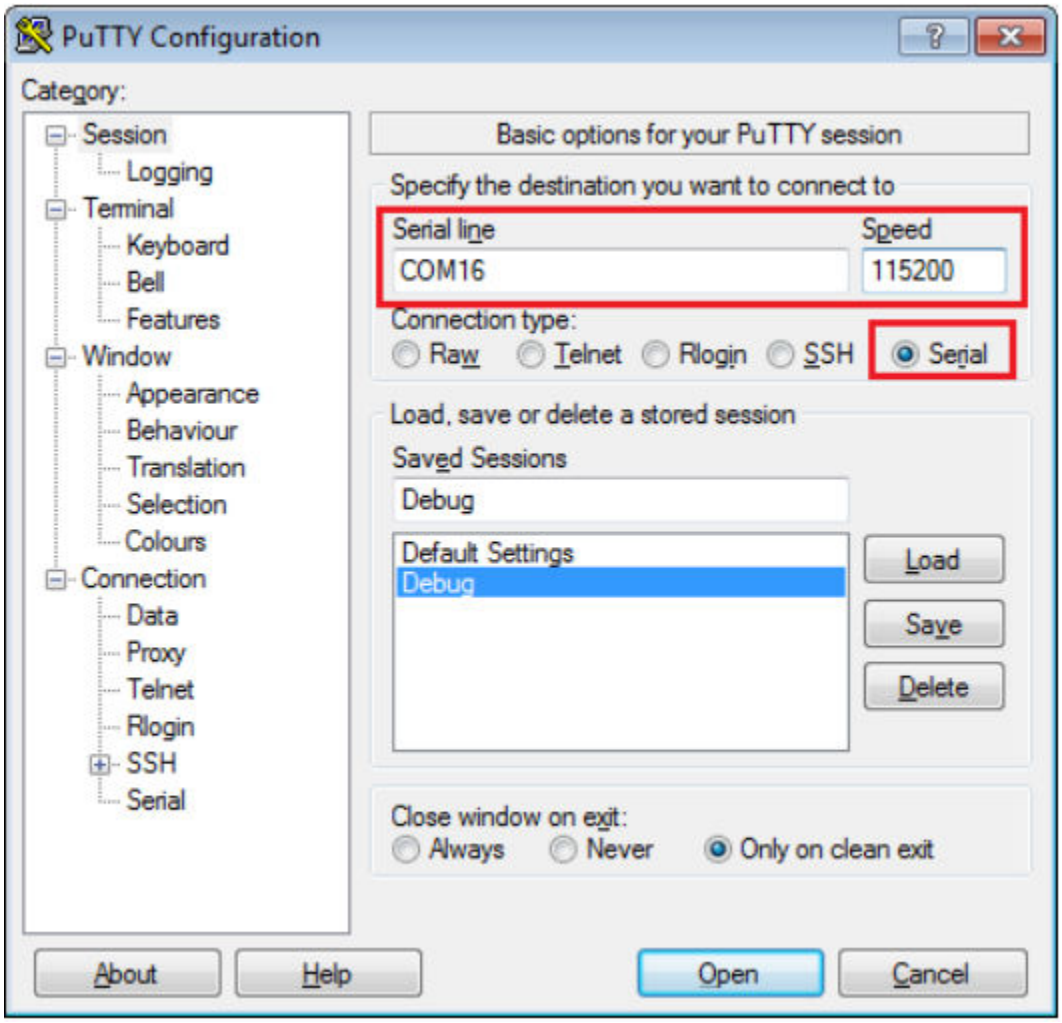

Figure 62. Terminal (PuTTY) configurations

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 the Cortex-M4
5. After it is connected, the screen should resemble this figure:

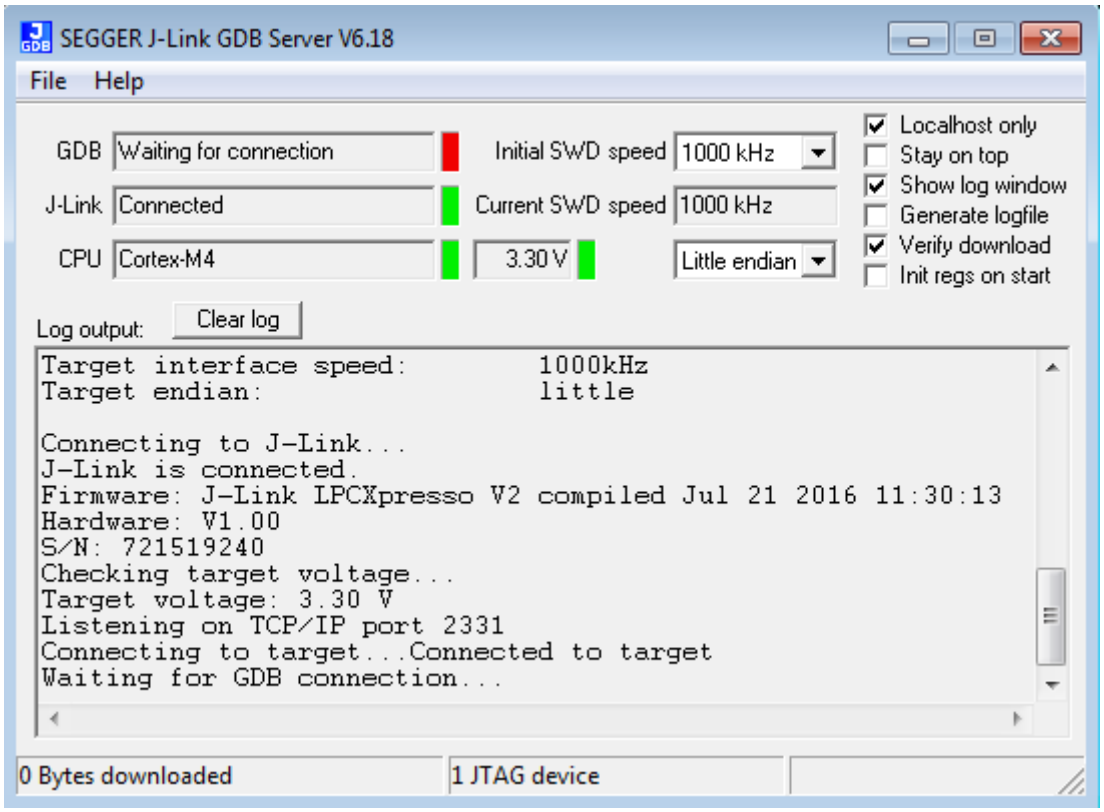

Figure 63. SEGGER J-Link GDB Server screen after successful connection

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".

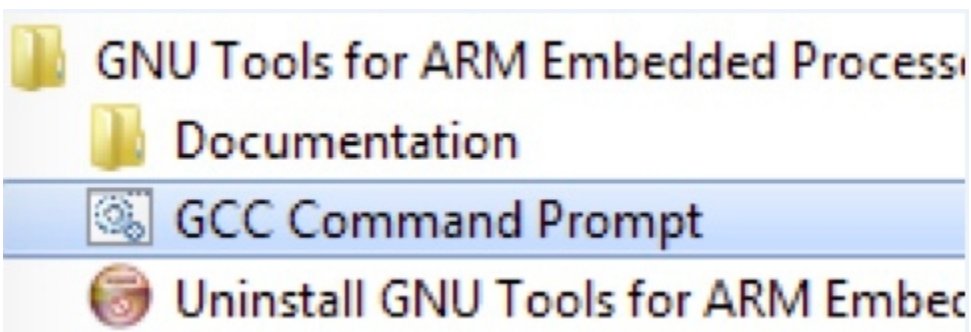

Figure 64. Launch command prompt

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/lpcxpresso54018/demo_apps/hello_world/armgcc/debug
8. Run the command "arm-none-eabi-gdb.exe <application_name>.elf". For this example, it is "arm-none-eabi-gdb.exe hello_world.elf".

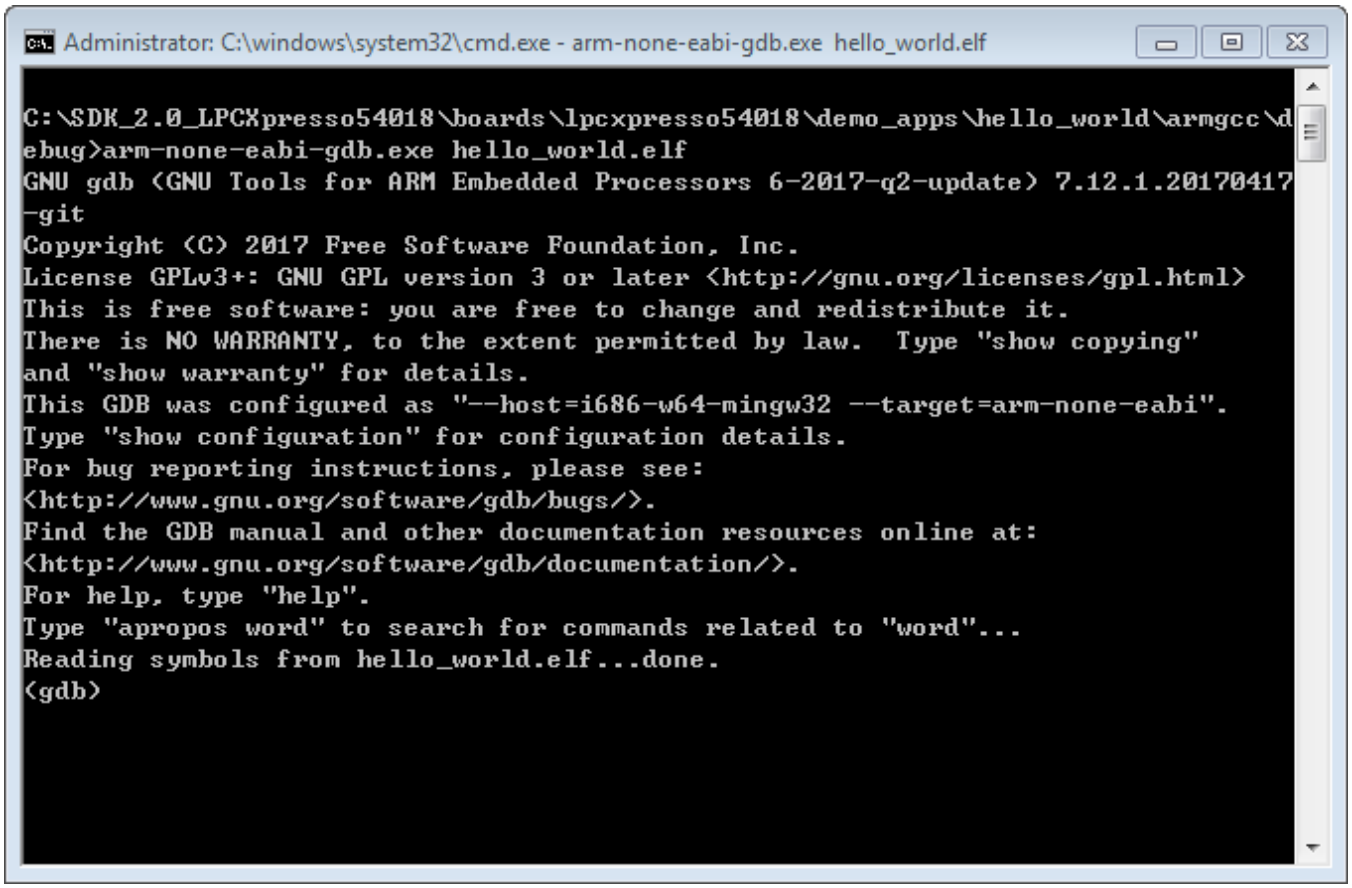

Figure 65. Run arm-none-eabi-gdb

9. Run these commands:
   a. "target remote localhost:2331"
   b. "monitor reset"
   c. "monitor go"
   d. "monitor halt"
   e. "load"
   f. "monitor reg pc=(0x4)"
   g. "monitor reg msp=(0x0)"
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 is not true, check your terminal settings and connections.

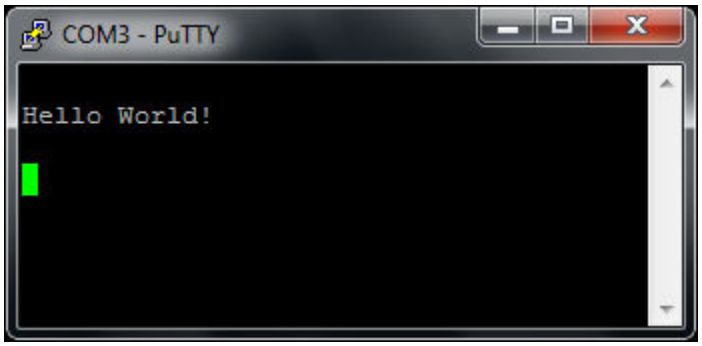

Figure 66. Text display of the hello_world demo

5.4 How to program the non-XIP (plain load) example bin file to external flash

External flash

1. Use the J-FLASH-Lite (version high thatnV6.22) to erase the chip.

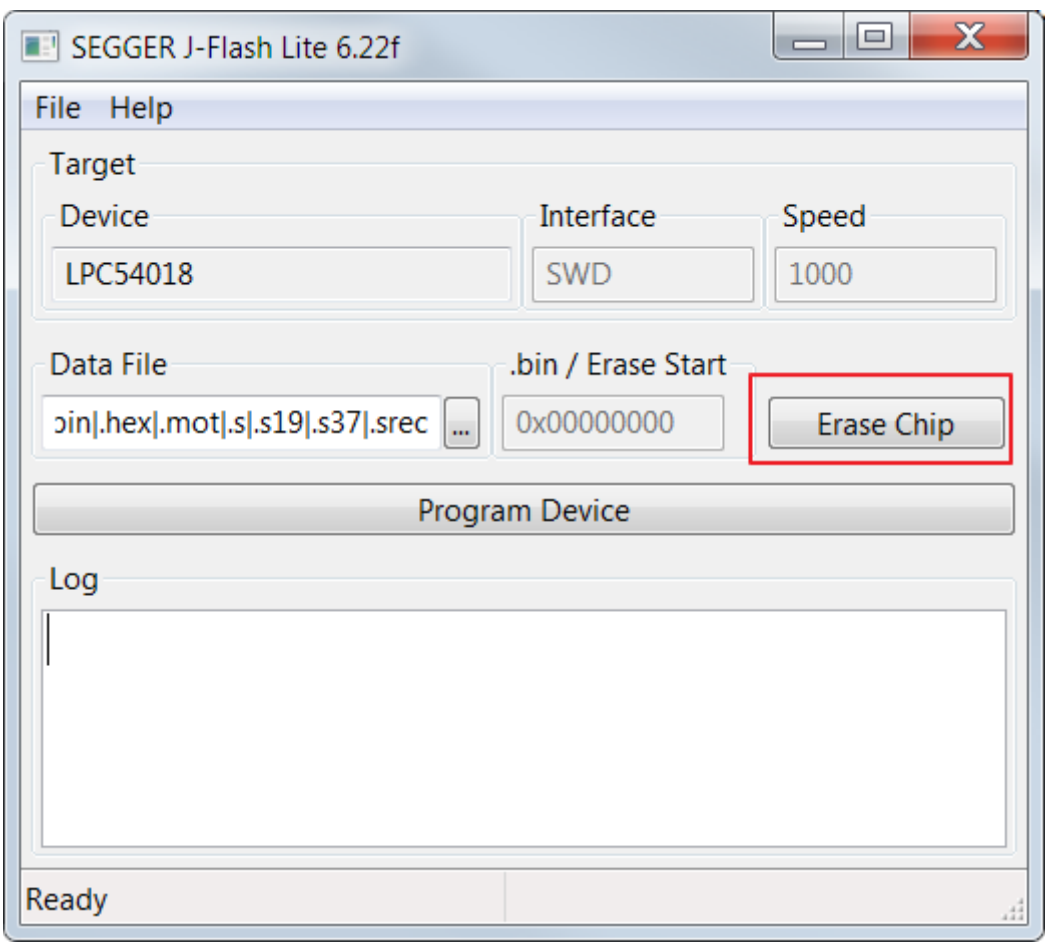

Figure 67. Erase the external flash

2. Wait for the flash erase finish.

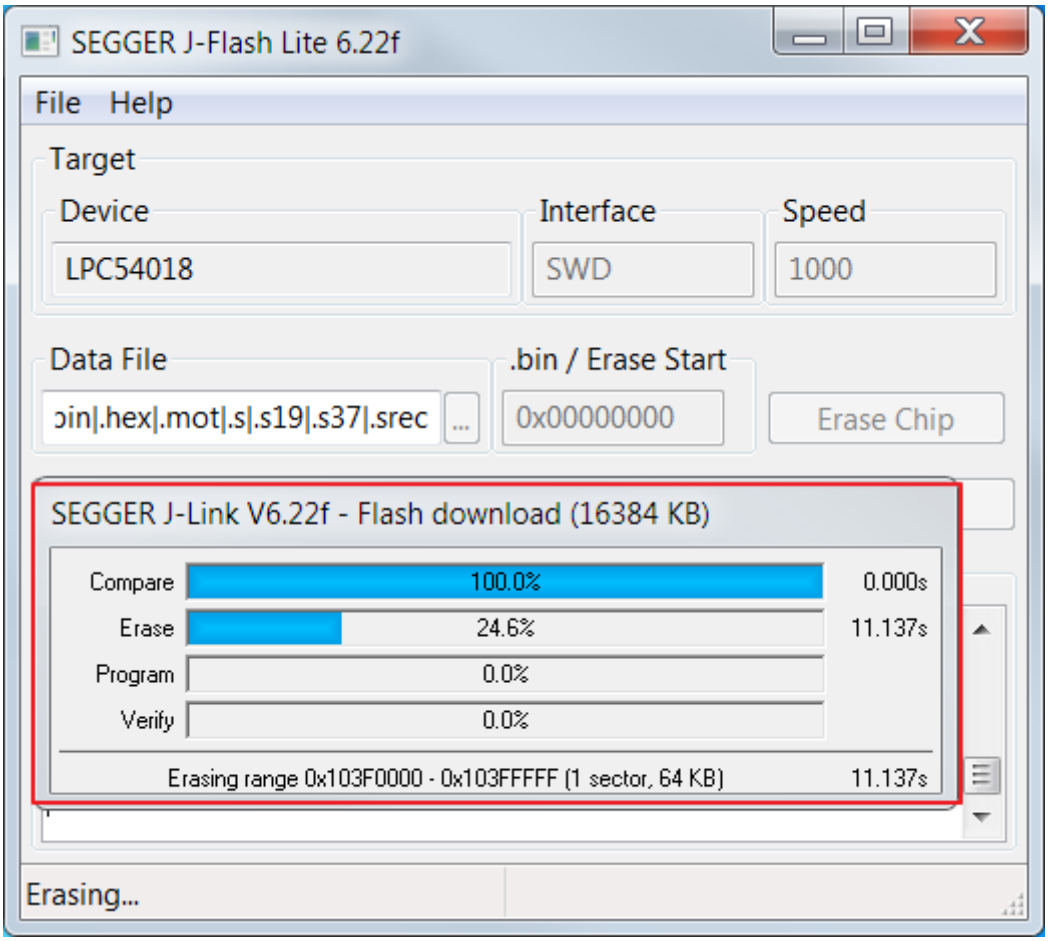

Figure 68. Erase in progress

NOTE

If you cannot erase, press the SW4 button then press the reset button to enter ISP mode. Then, click the "Erase" button again (keep pressing the SW4 button all the time).

3. Click ‘Program Device’ to program the binary file into external flash.

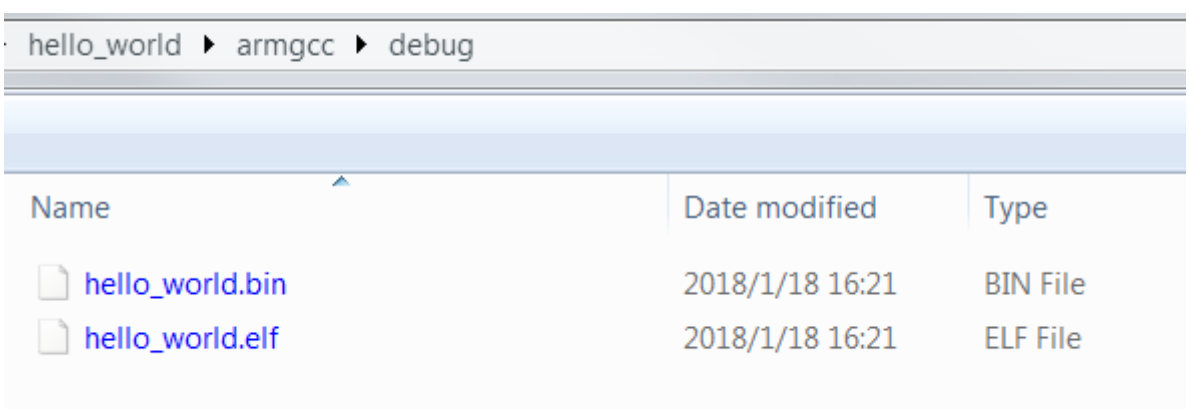

Figure 69. Binary built by armgcc

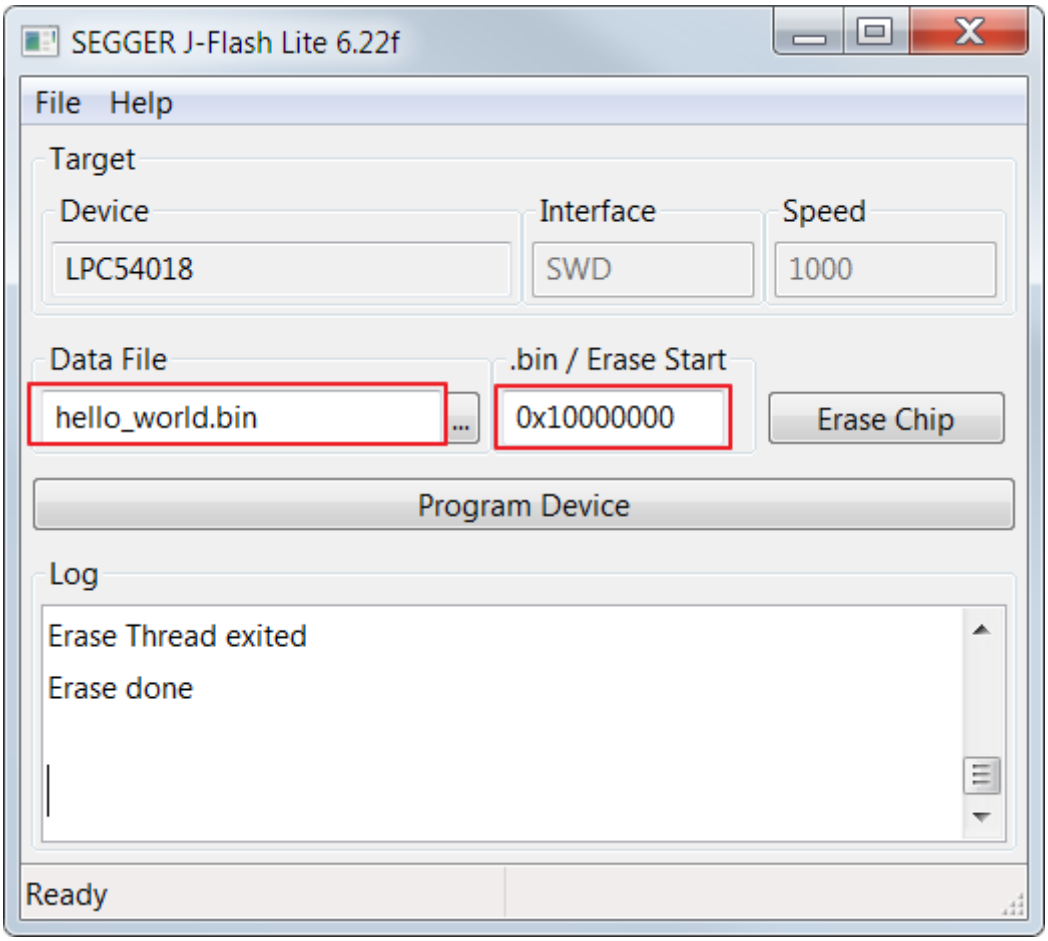

Figure 70. Program the binary to external flash

NOTE

Make sure the '.bin/Erase Start' address is 0x10000000 (external flash base address).

4. After programming, press the reset button on the board to run.

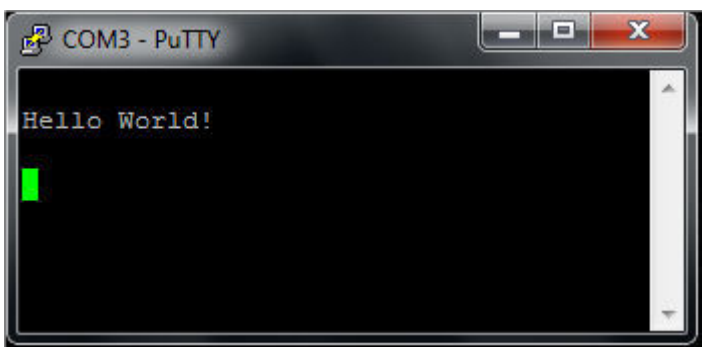

Figure 71. Text display of the hello_world _qspi_xip demo

5.5 Build an XIP example application

1. Following the same steps as *Section 5.3, "Run an non-xip (plain load) example application"*, type “build_debug.bat” on the command line or double click the "build_debug.bat" file in Windows Explorer to perform the build.

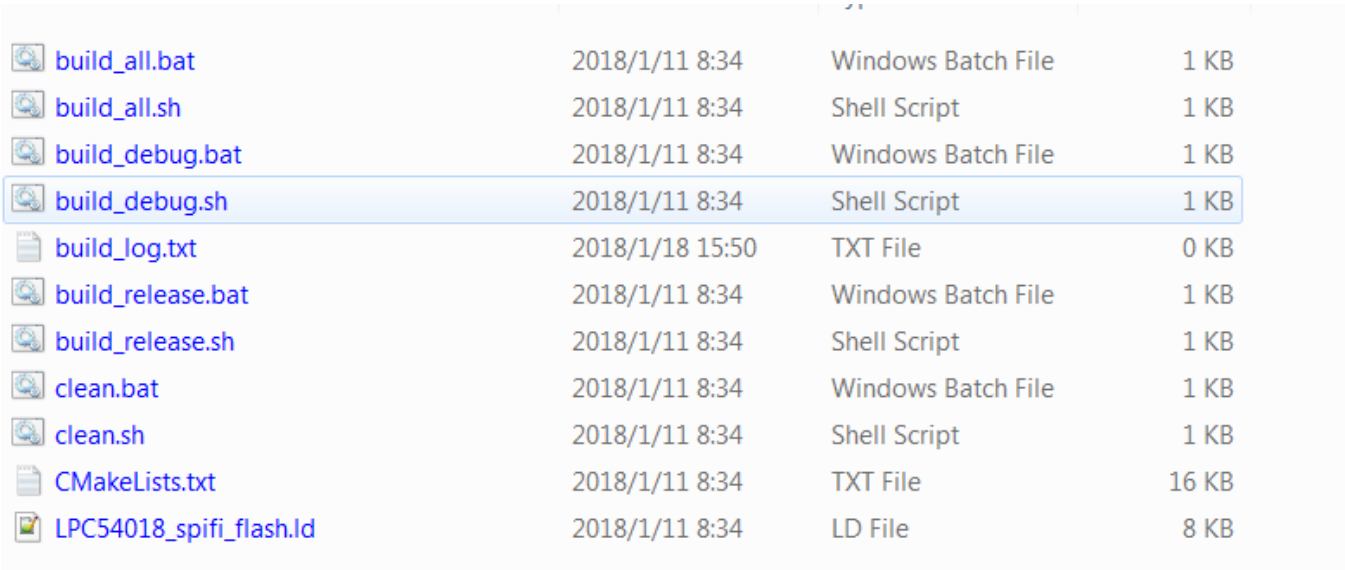

Figure 72. Click build_debug.bat to build the demo

2. The build output is shown in this figure:

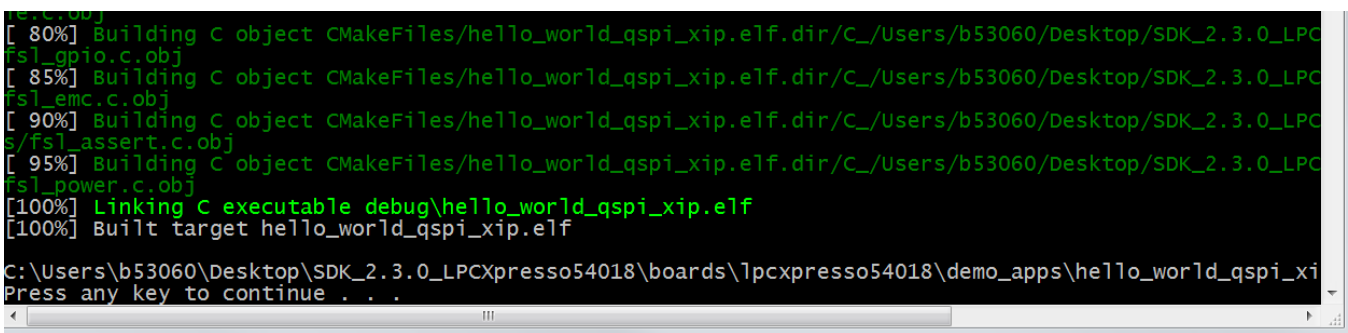

Figure 73. hello_world_qspi_xip demo build successful

5.6 Run an XIP example application

1. Use the J-FLASH-Lite (with version higher than V6.22) to erase the chip.

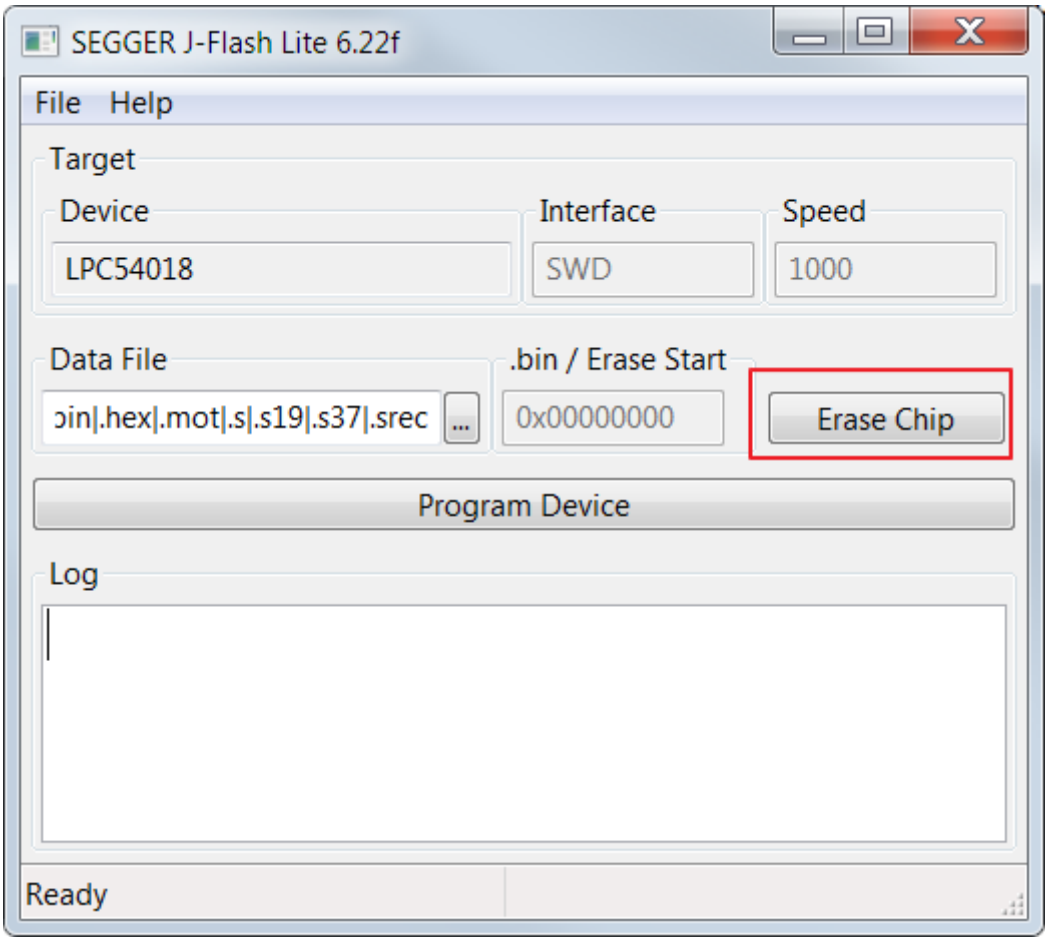

Figure 74. Erase the external flash

2. Wait for the flash erase finish.

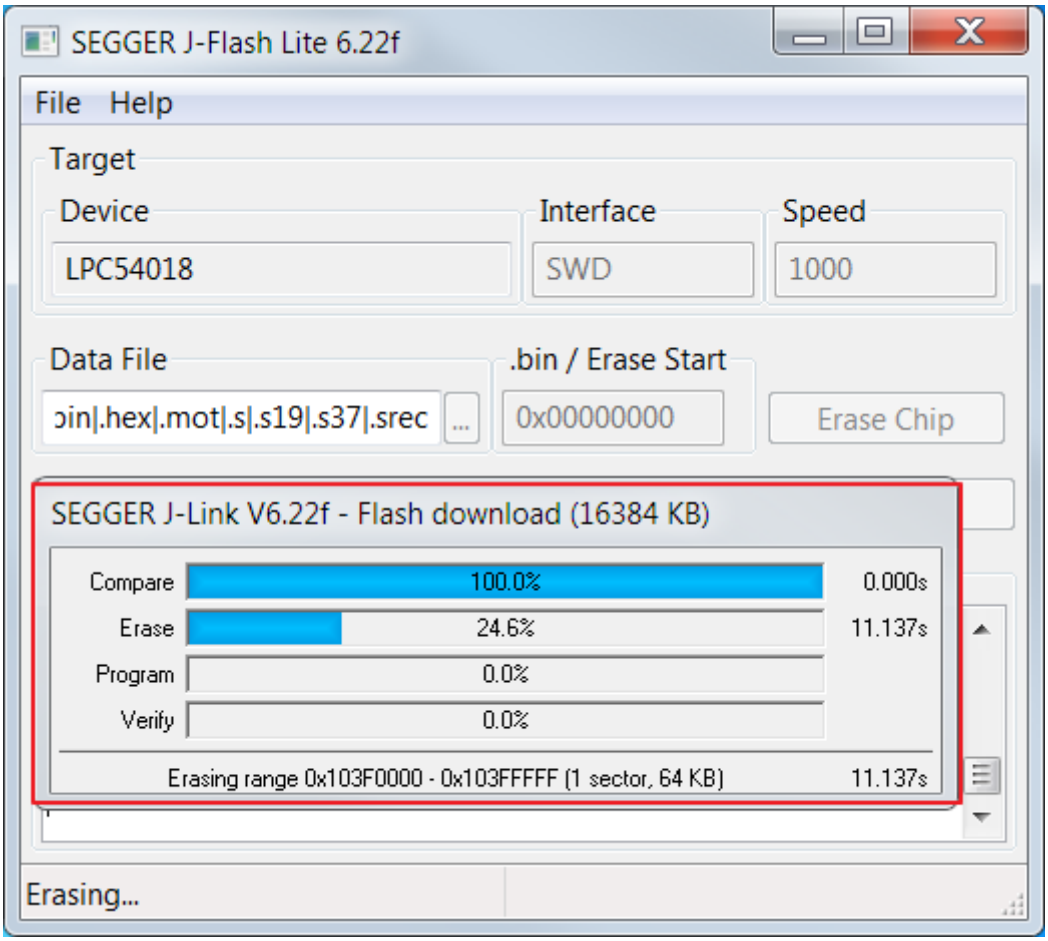

Figure 75. Erase in progress

NOTE

If you cannot erase, press the SW4 button then press the reset button to enter ISP mode. Then, click erase again (keep pressing the SW4 button all the time).

3. Program the binary file into external flash.

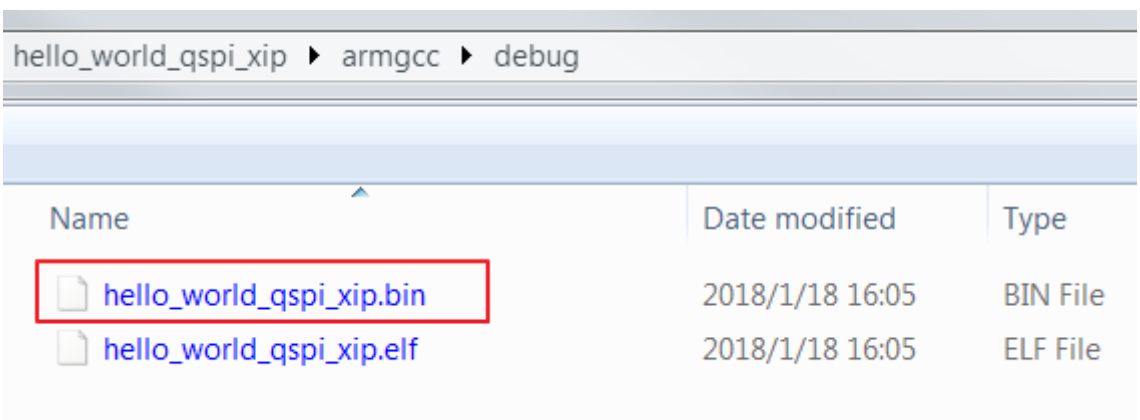

Figure 76. Binary built by armgcc

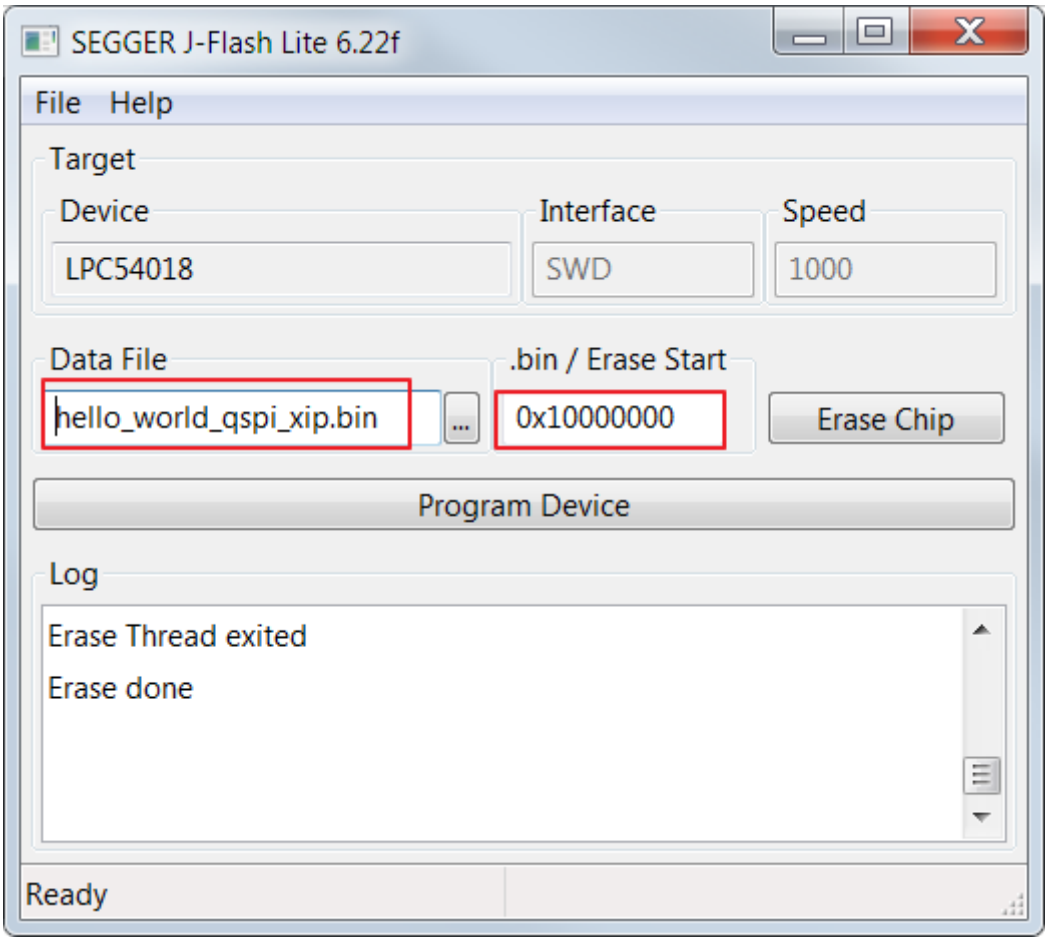

Figure 77. Program the binary to external flash

NOTE

Make sure the '.bin/Erase Start' address is 0x10000000 (the external flash base address).

4. After programming, press the reset button to run.

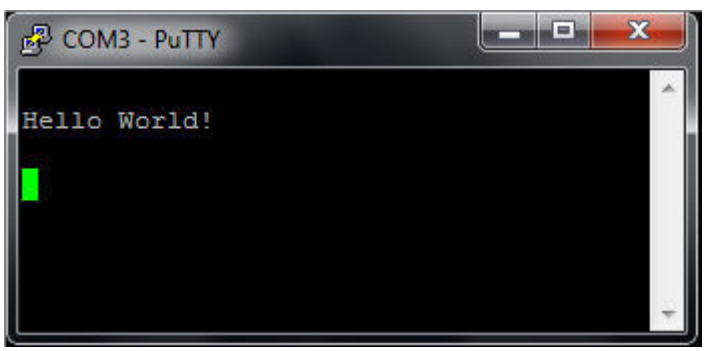

Figure 78. Text display of the hello_world_qspi_xip demo

6 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 LPCXpresso54018 hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

6.1 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 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.

6.2 Build a non-XIP (plain load) 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.

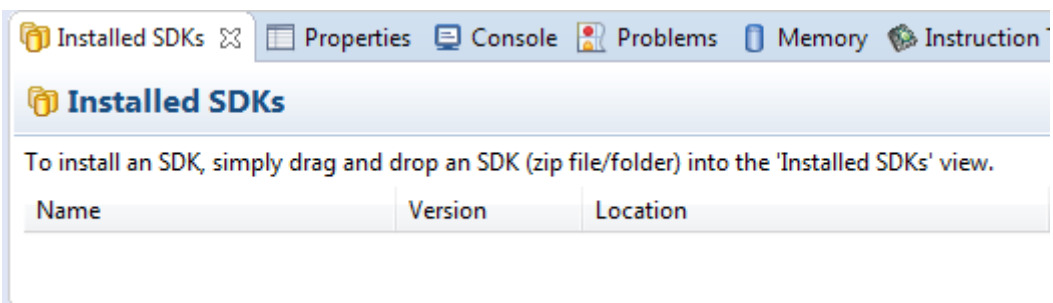

Figure 79. Install an SDK

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

Figure 80. Import an SDK example

3. In the window that appears, expand the “LPC540xx folder and select “LPC54018" . Then, select “lpcxpresso54018” and click the “Next” button.

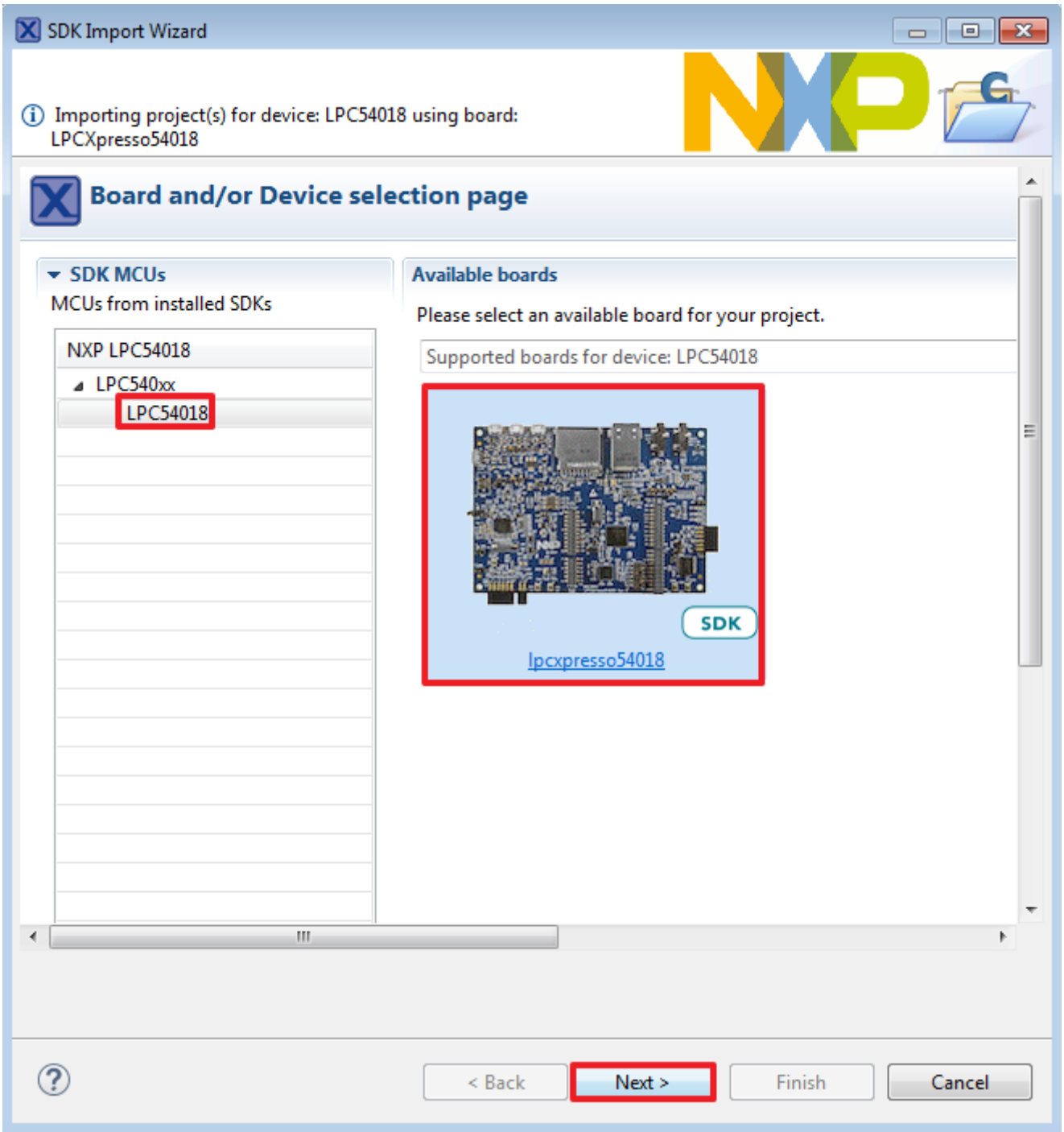

Figure 81. Select LPCXpresso54018 board

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

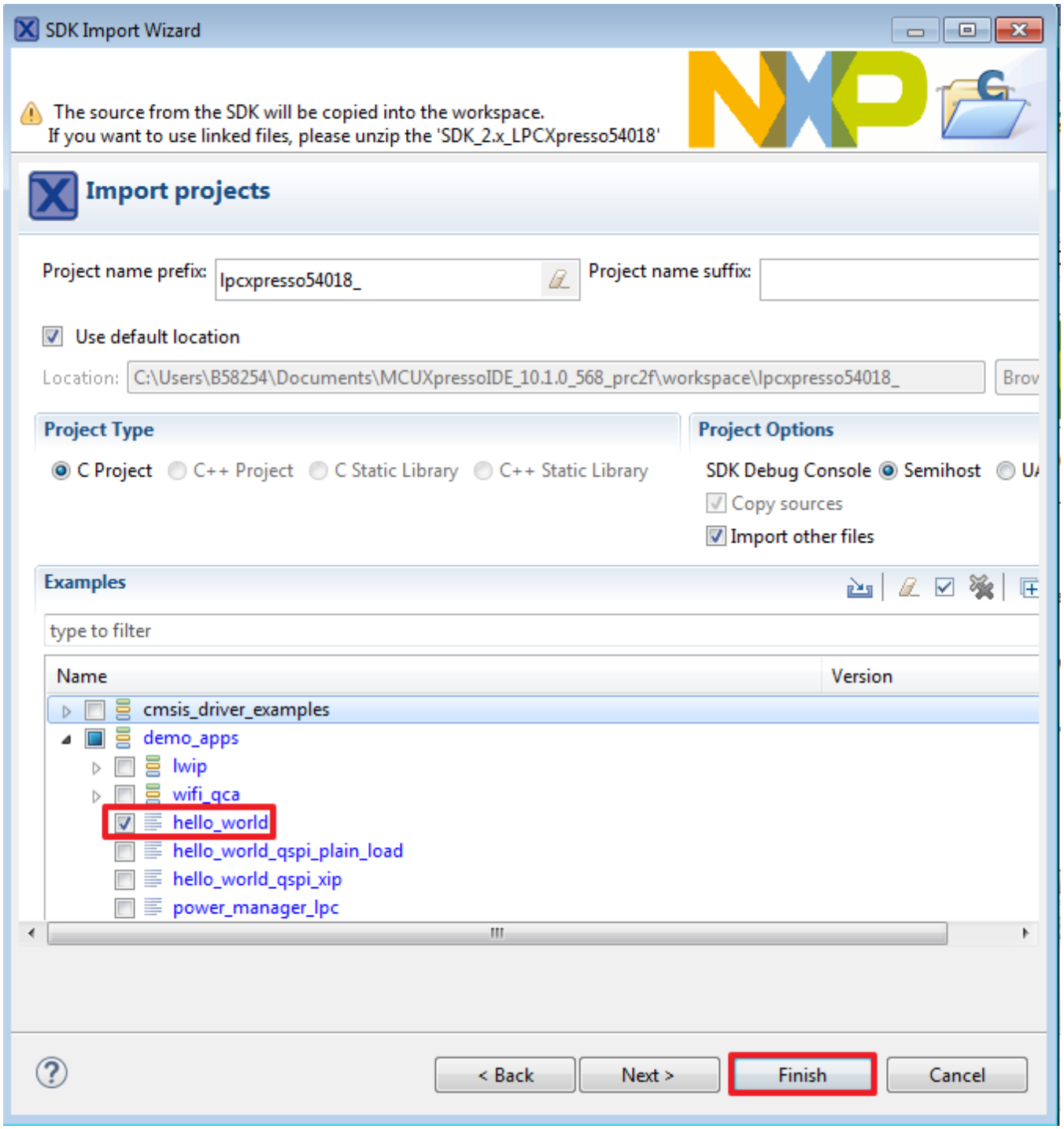

Figure 82. Select "hello_world"

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 adc_basic, adc_burst, adc_dma, and adc_interrupt). Otherwise, there is no need to select it. Click the “Finish” button.

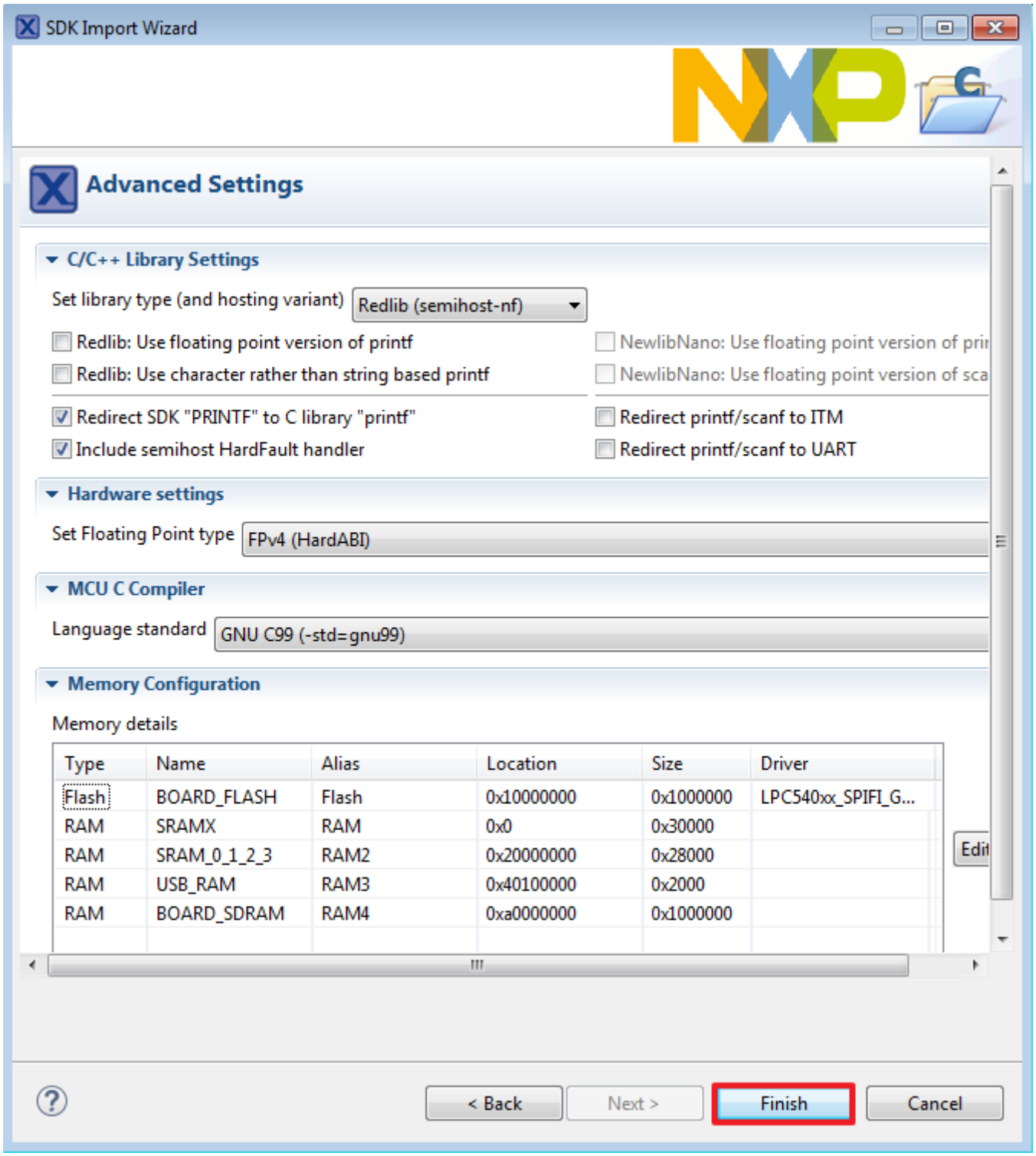

Figure 83. Select "User floating print version of printf"

6.3 Run a non-XIP (plain load) example application

For more information on debug probe support in the MCUXpresso IDE v10.1.1, visit community.nxp.com.

To download and run the application, perform these steps:

1. Reference the table in Appendix B 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.
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 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

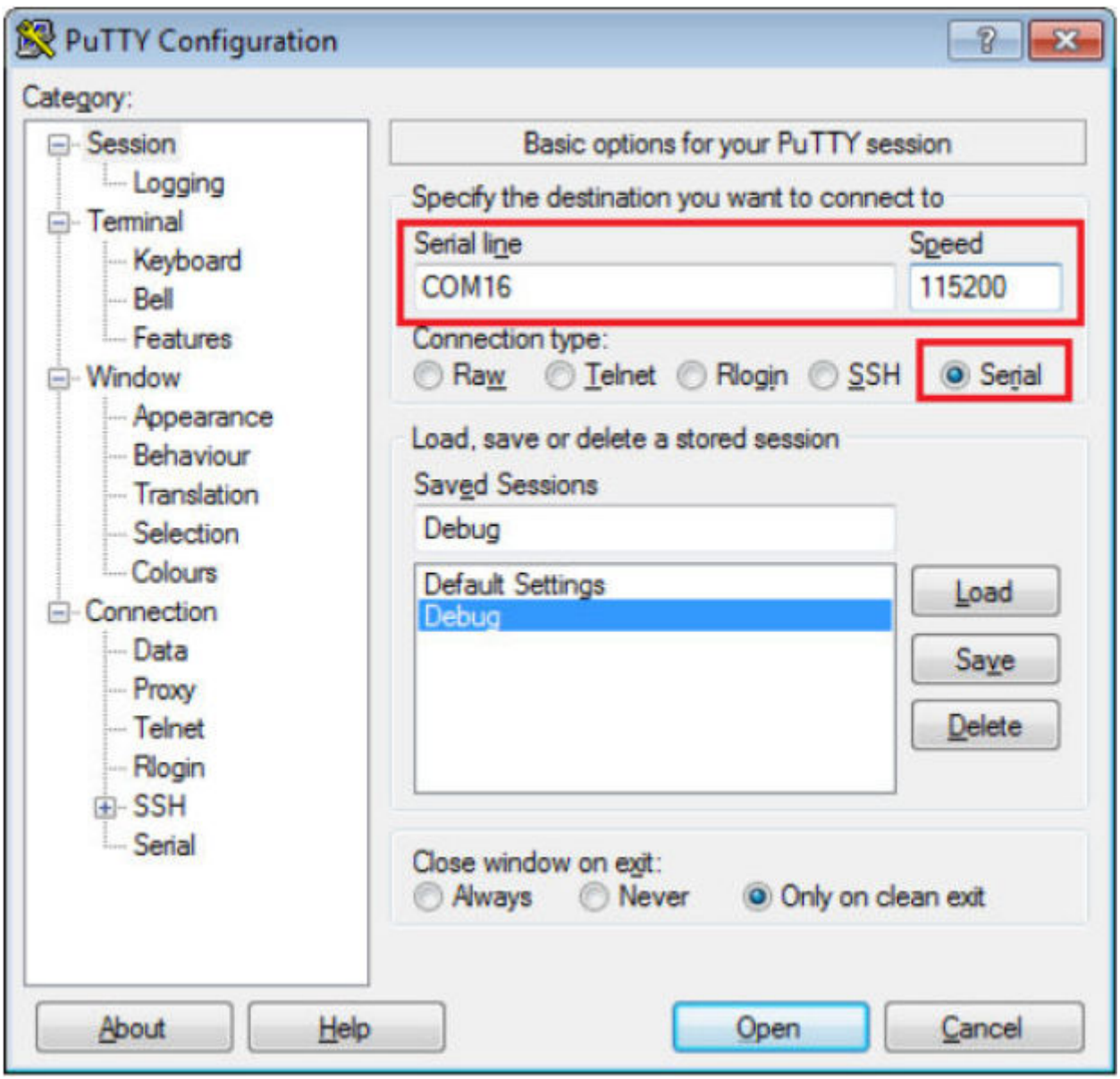

Figure 84. Terminal (PuTTY) configurations

3. On the *Quickstart Panel*, click on "Debug 'lpcxpresso54018_demo_apps_hello_world' [Debug]".

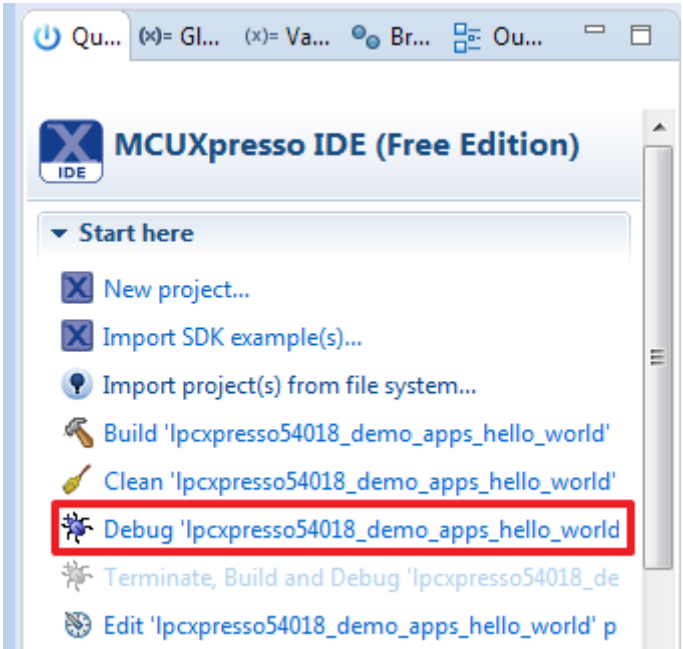

Figure 85. Debug "hello_world" case

4. 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.)

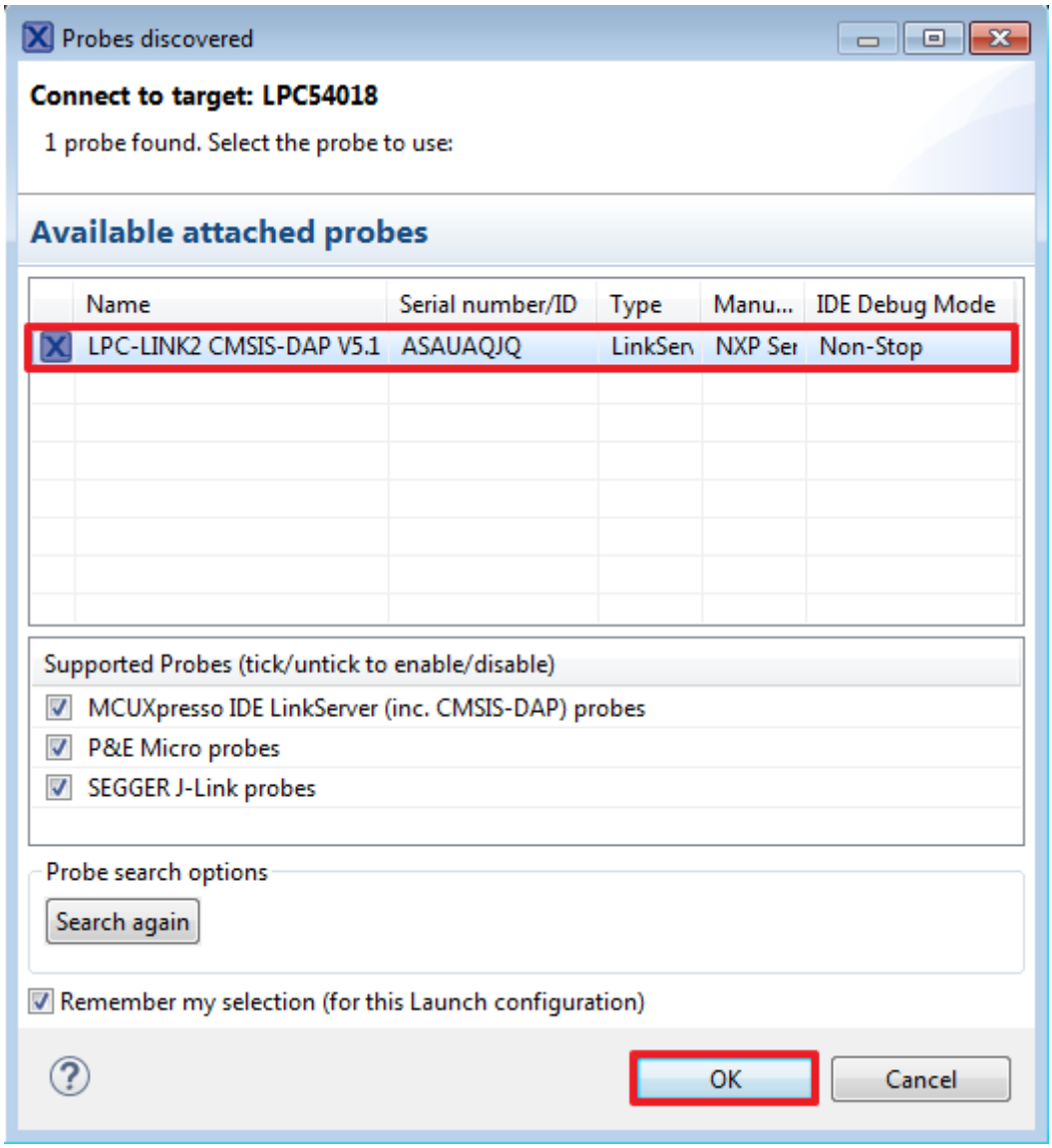

Figure 86. Attached Probes: debug emulator selection

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

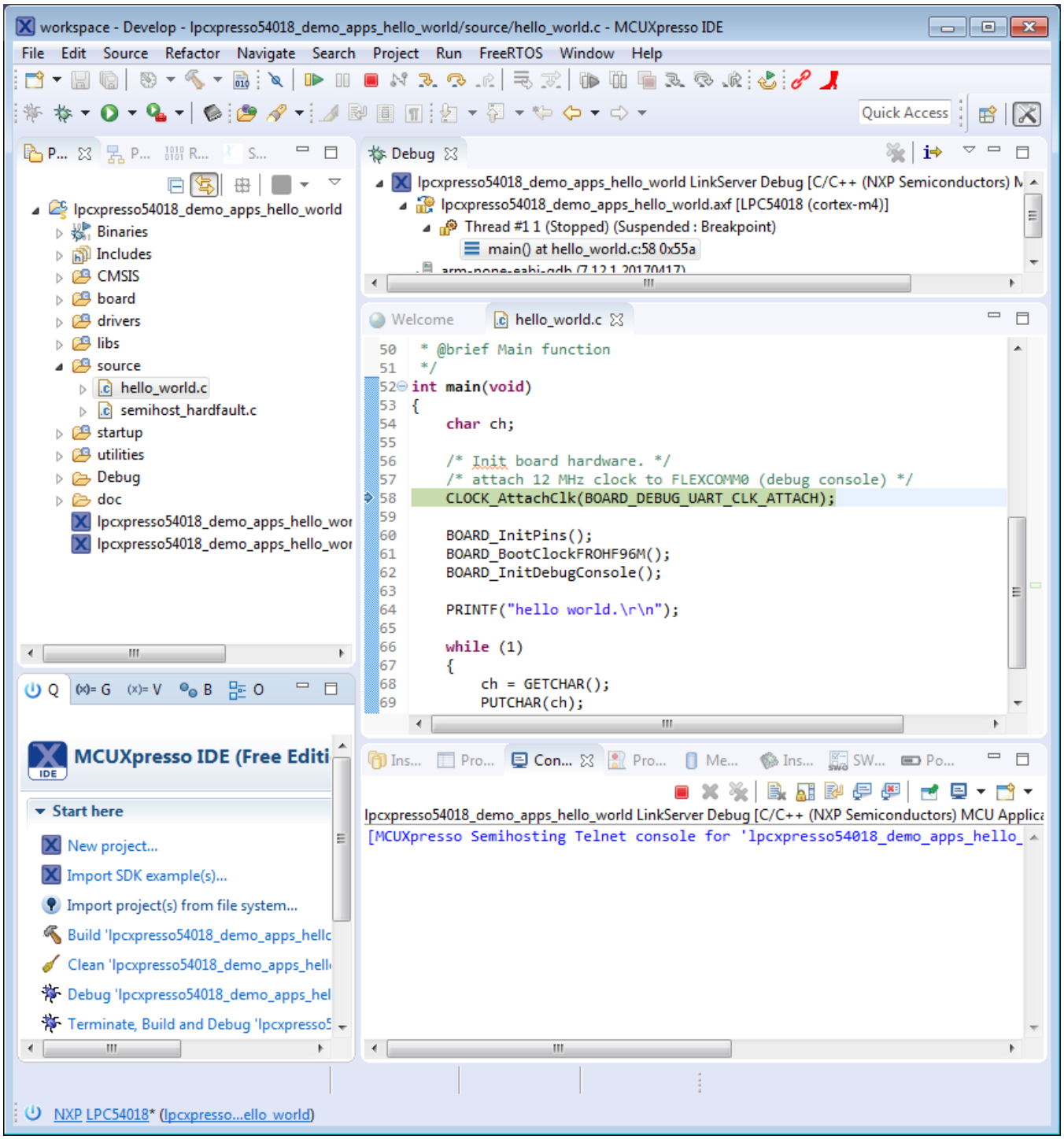

Figure 87. Stop at main() when running debugging

NOTE

The application is only downloaded into the SRAM when debugging. If you need to program the image to external flash, see *Section 6.4, "How to program the non-XIP (plain load) example bin file to external flash"*.

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

Figure 88. Resume button

The hello_world application is now running and a banner is displayed on the console.

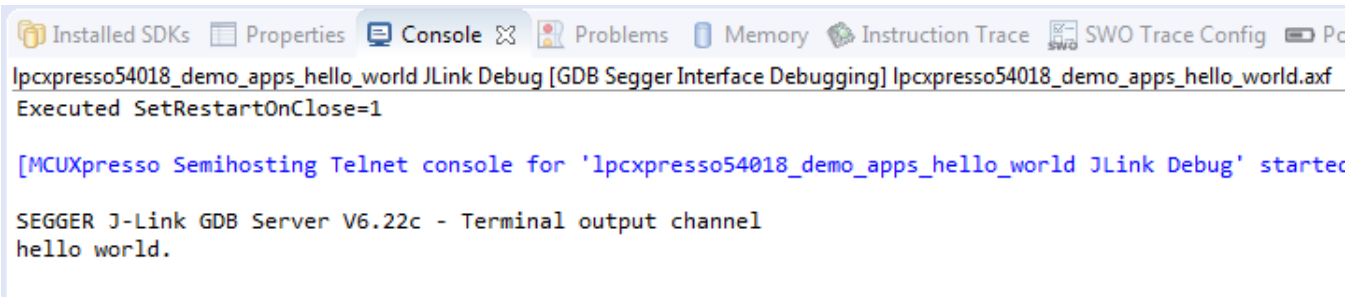

Figure 89. Text display of the hello_world demo

6.4 How to program the non-XIP (plain load) example bin file to external flash

To build an example application, follow these steps.

1. Create a bin file for non-XIP (plain load) demo from *.axf file.

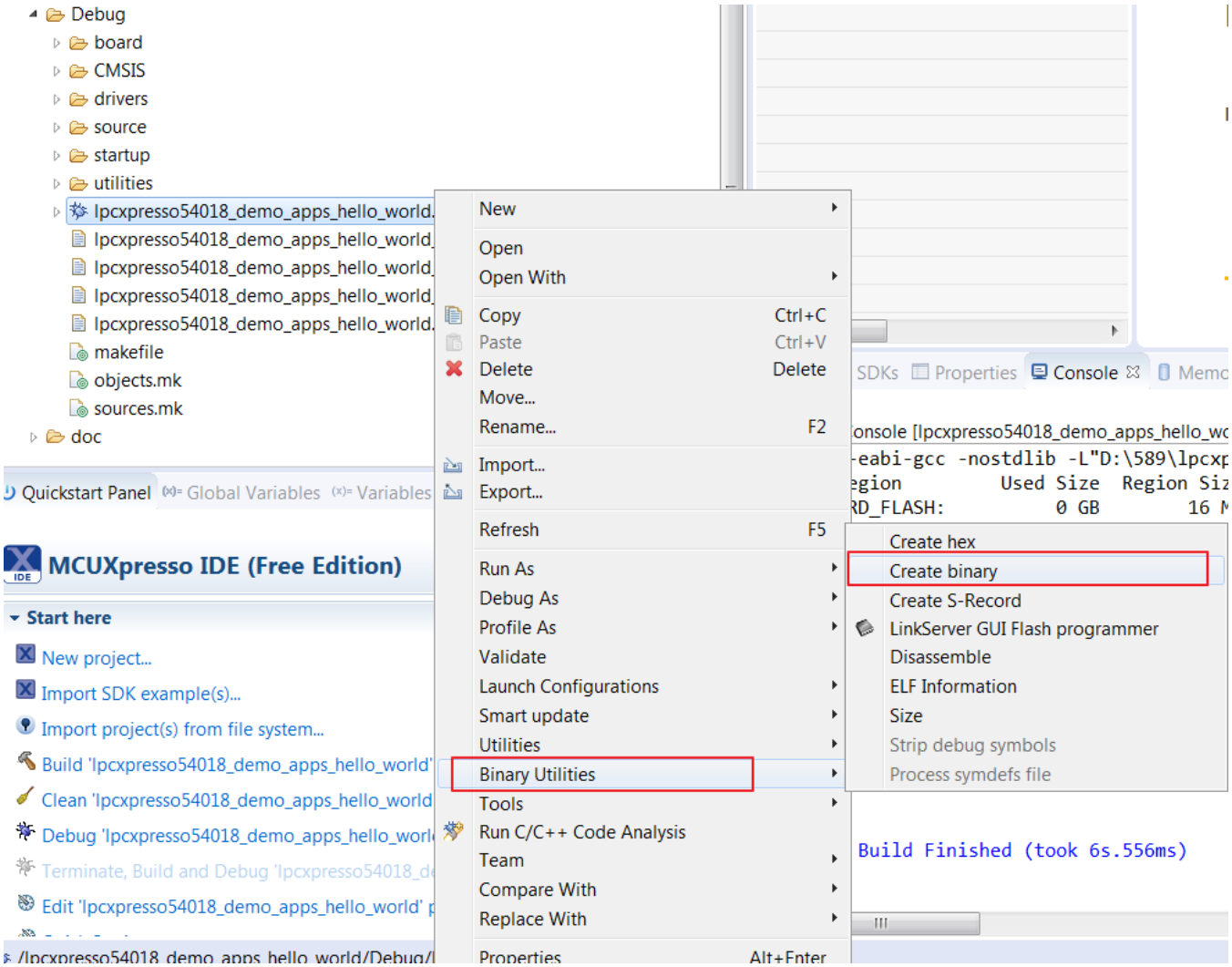

Figure 90. Create a bin file

2. Click the “LinkServer GUI Flash Programmer” button.

Figure 91. Click “LinkServer GUI Flash Programmer” button

3. Select LPC-LINK2 CMSIS-DAP.

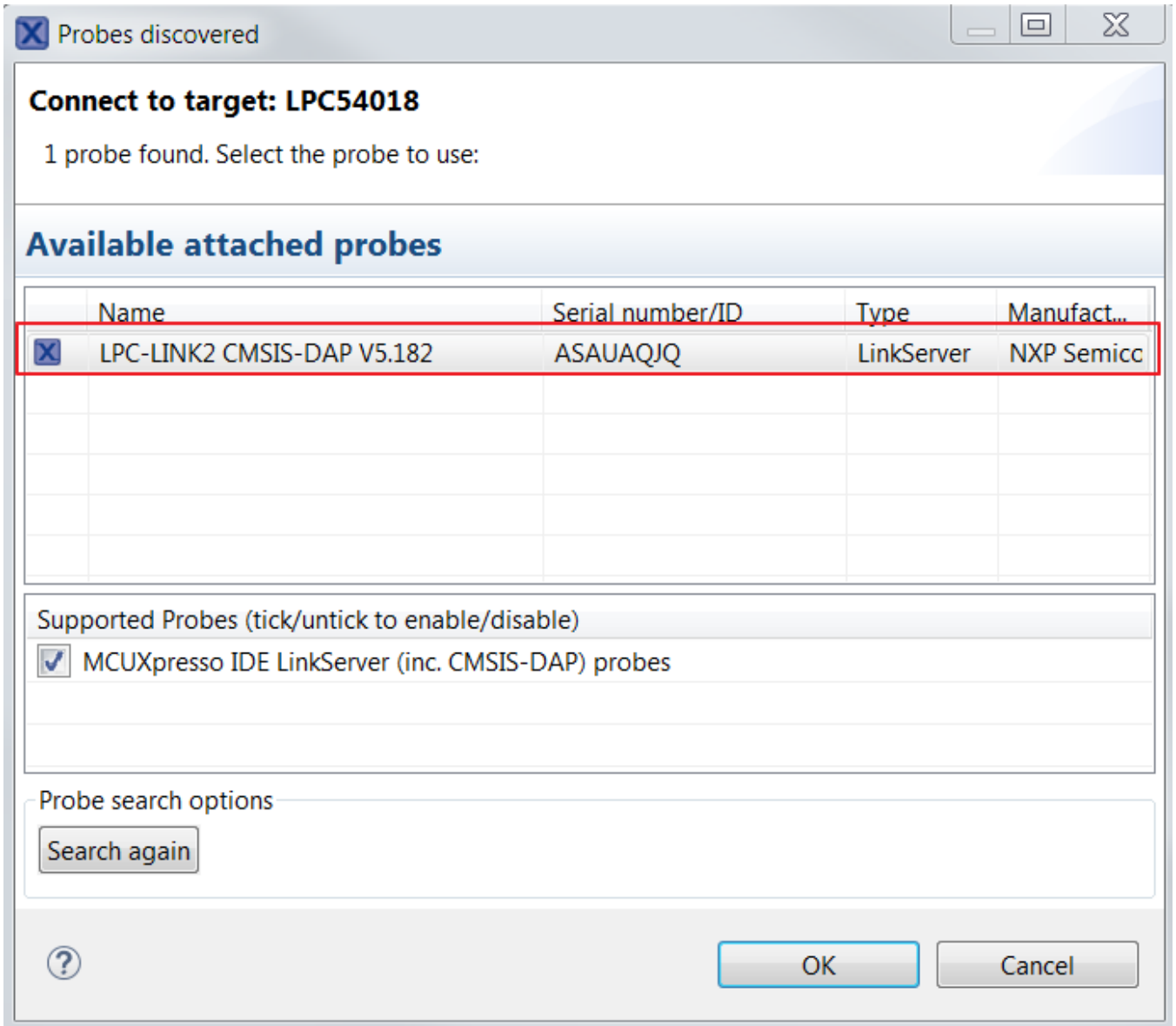

Figure 92. Select LPC-LINK2 CMSIS-DAP

4. Select the flash driver, bin file, and external flash program address, then click the "OK" button to program the bin file to external flash.

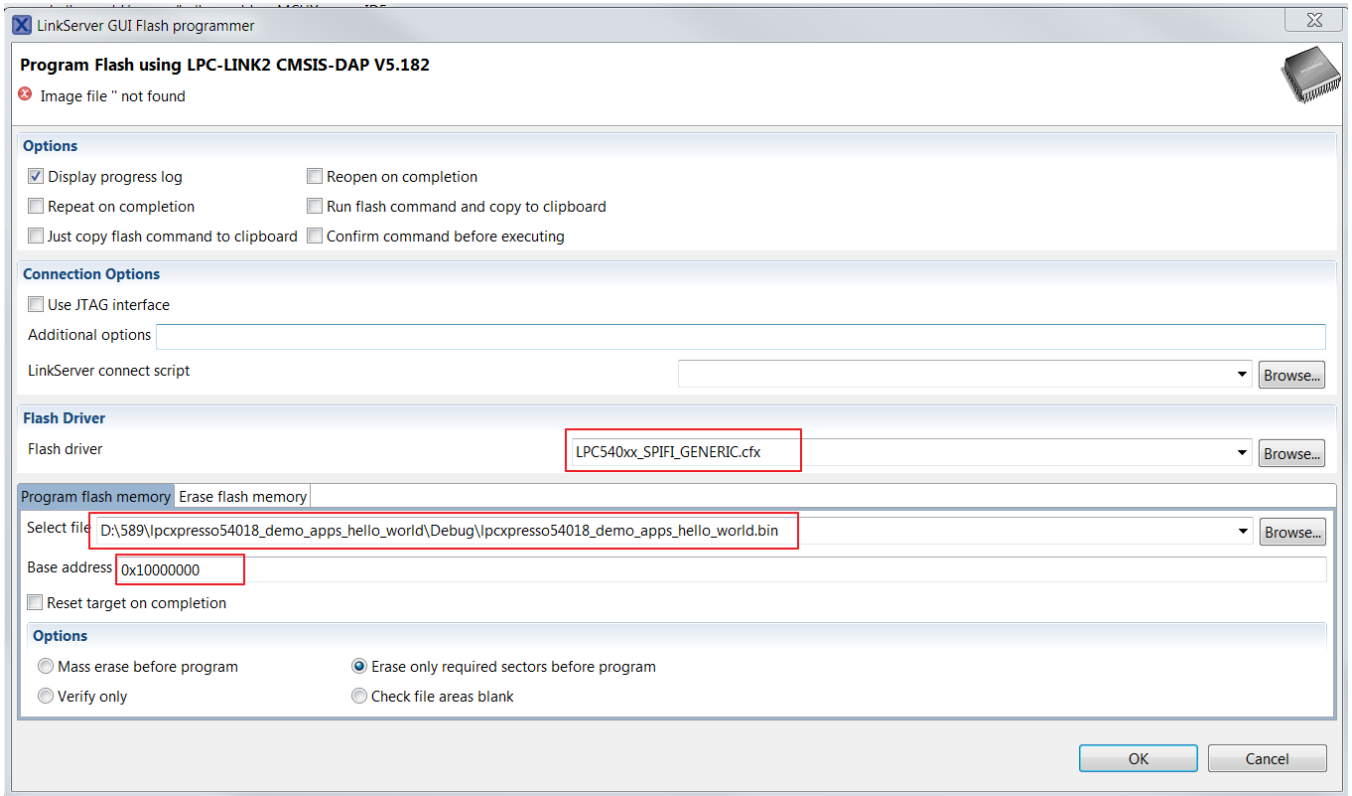

Figure 93. Select the flash driver, bin file, and external flash program address

5. After programming, press the reset button on the board to run the example.

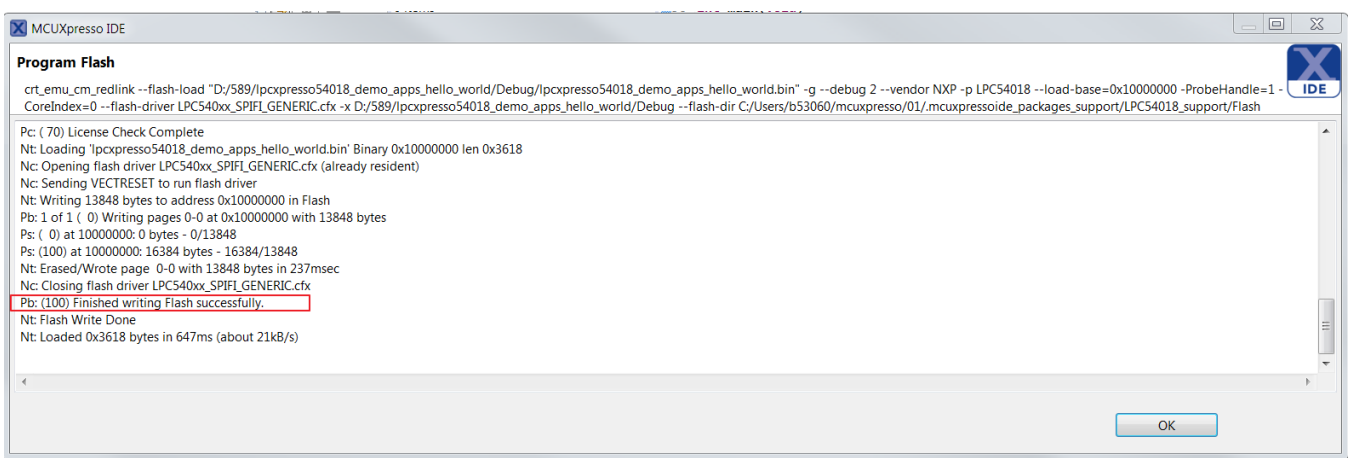

Figure 94. GUI Flash Program result

6.5 Build an XIP 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.

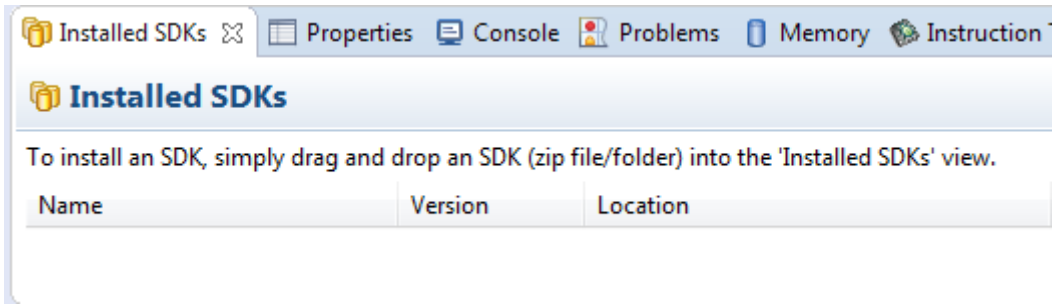

Figure 95. Install an SDK

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

Figure 96. Import an SDK example

3. In the window that appears, expand the “LPC540xx folder and select “LPC54018". Then, select “lpcxpresso54018” and click the “Next” button.

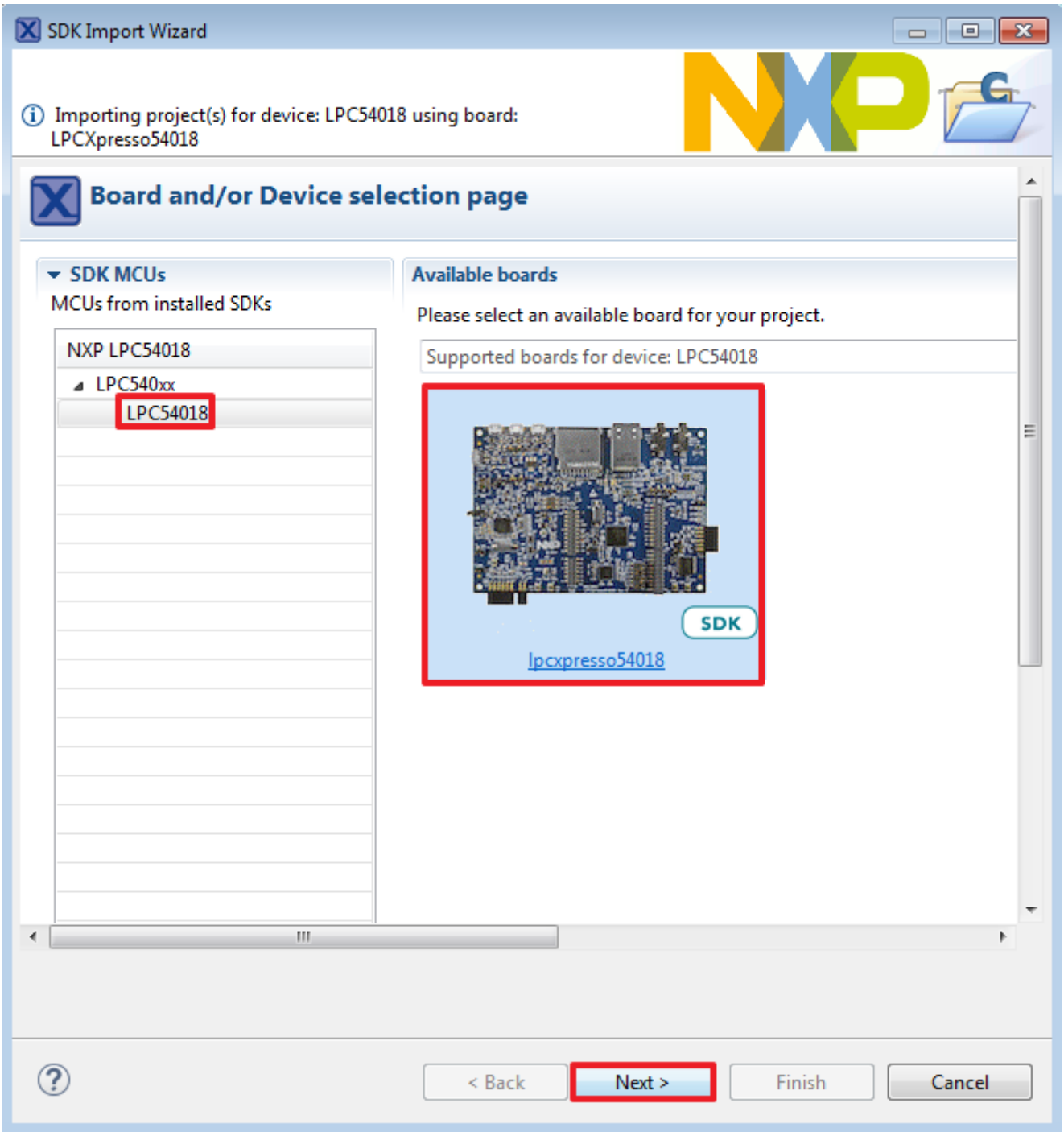

Figure 97. Select LPCXpresso54018 board

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

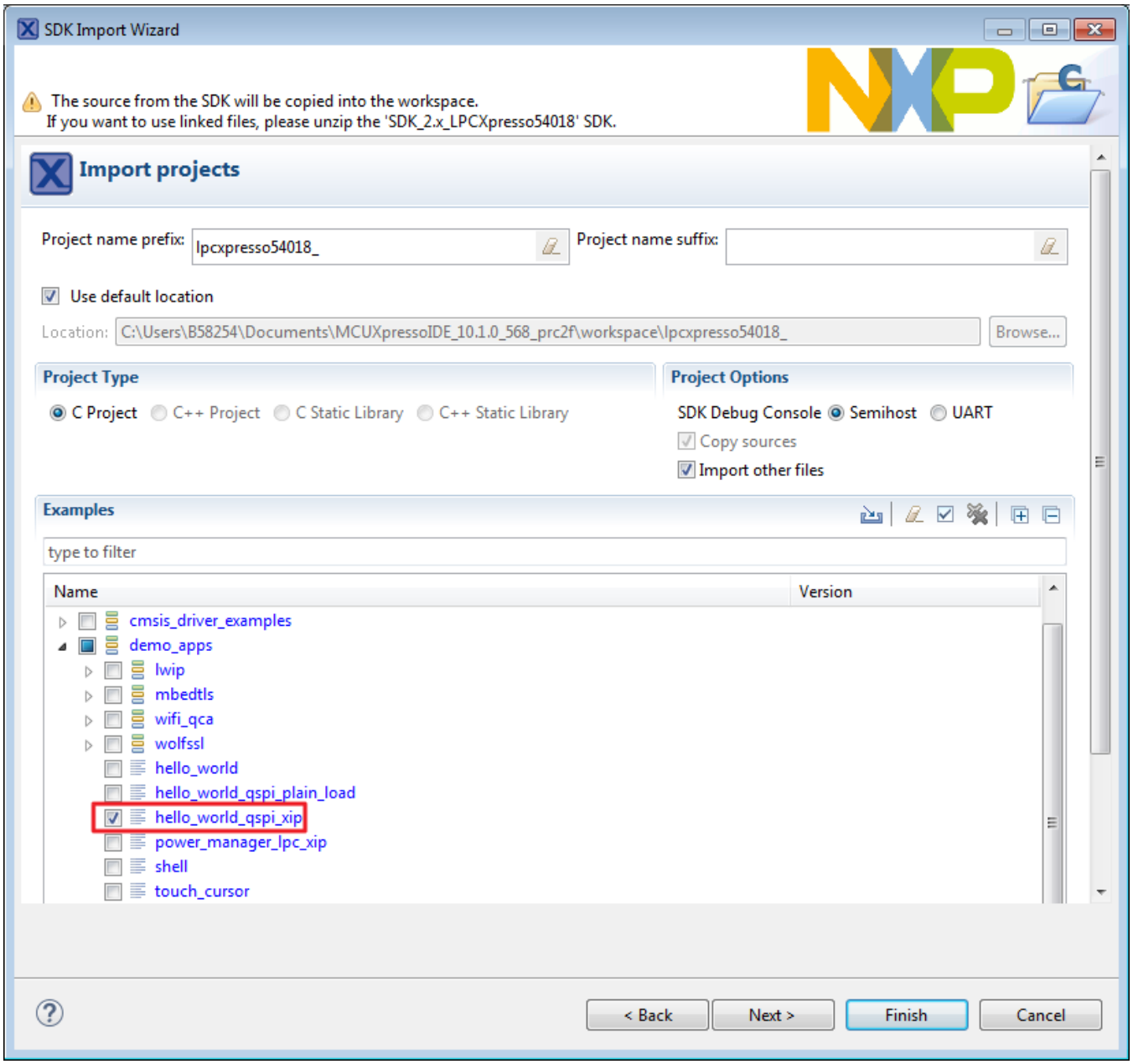

Figure 98. Select "hello_world_qspi_xip"

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 adc_basic, adc_burst, adc_dma, and adc_interrupt). Otherwise, there is no need to select it. Click the “Finish” button.

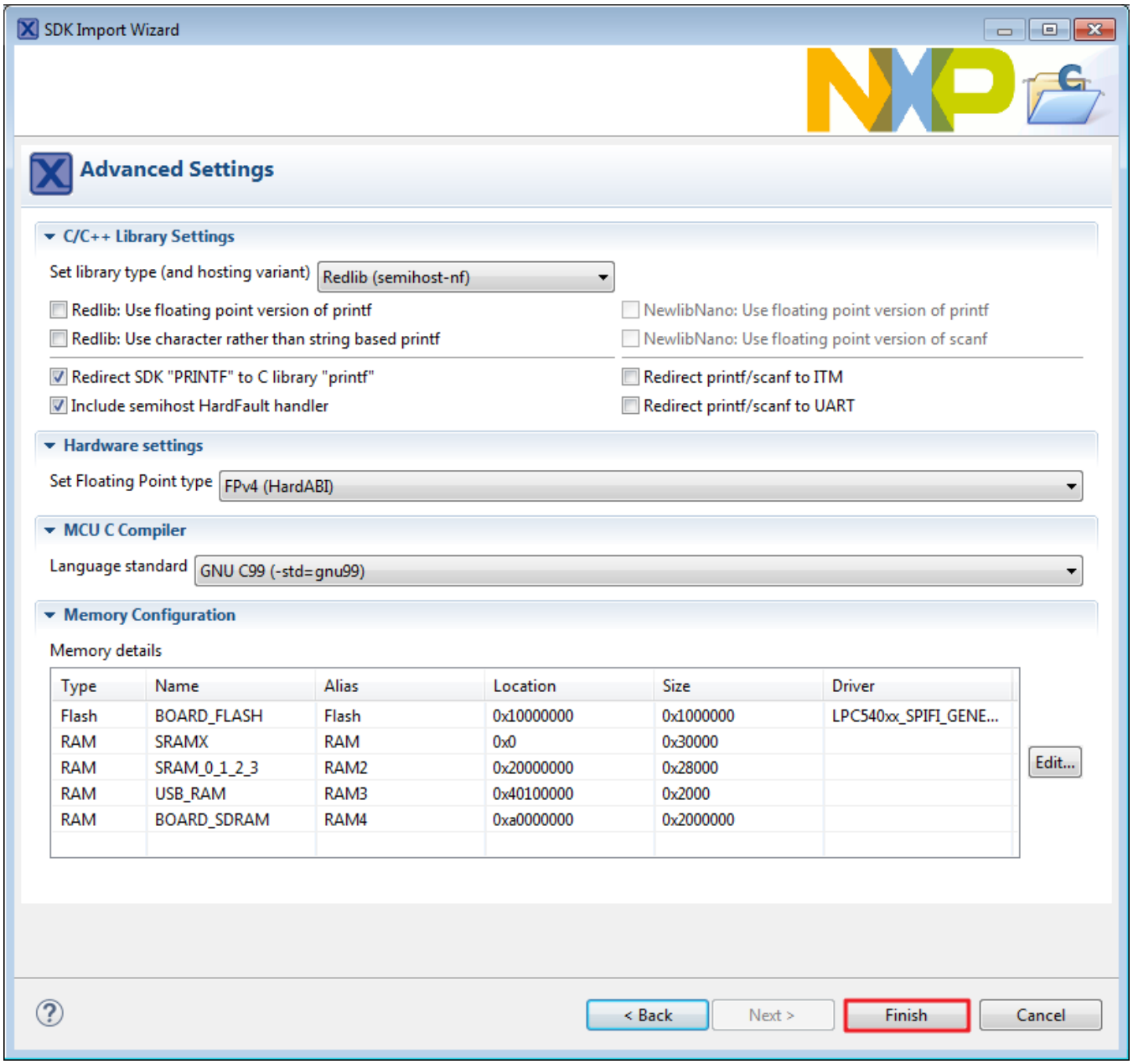

Figure 99. User floating print version of printf

6.6 Run an XIP example application

1. Click the "Debug" button on the tool bar to run the XIP example.

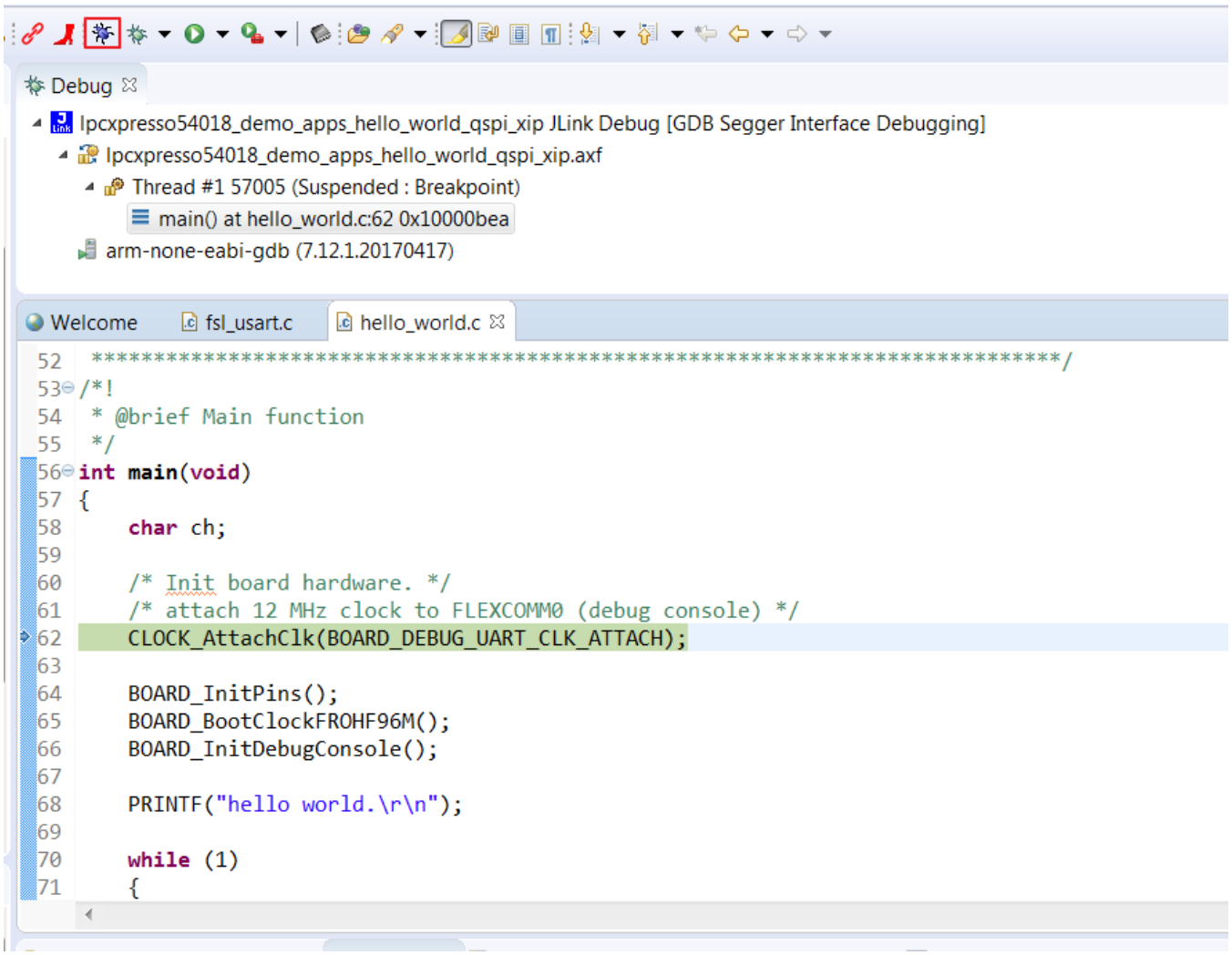

Figure 100. Run an XIP example

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:

Pins tool for configuration of pin routing and pin electrical properties.

Clock tool for system clock configuration.

Project Cloner allows creation of the standalone projects from SDK examples.

MCUXpresso Config Tools can be accessed in the following products:

- **Integrated** in the MCUXpresso IDE. See new perspectives allowing to configure peripherals directly in the IDE.
- **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 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), offers the flexibility to select/change many builds, includes a library, and provides source code options. The source code is organized as software components, categorized as driver, 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 the "New project" option, shown in the below figure.

Figure 101. MCUXpresso IDE Quickstart Panel

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

9 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. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system Start menu and typing “Device Manager” in the search bar, as shown below:

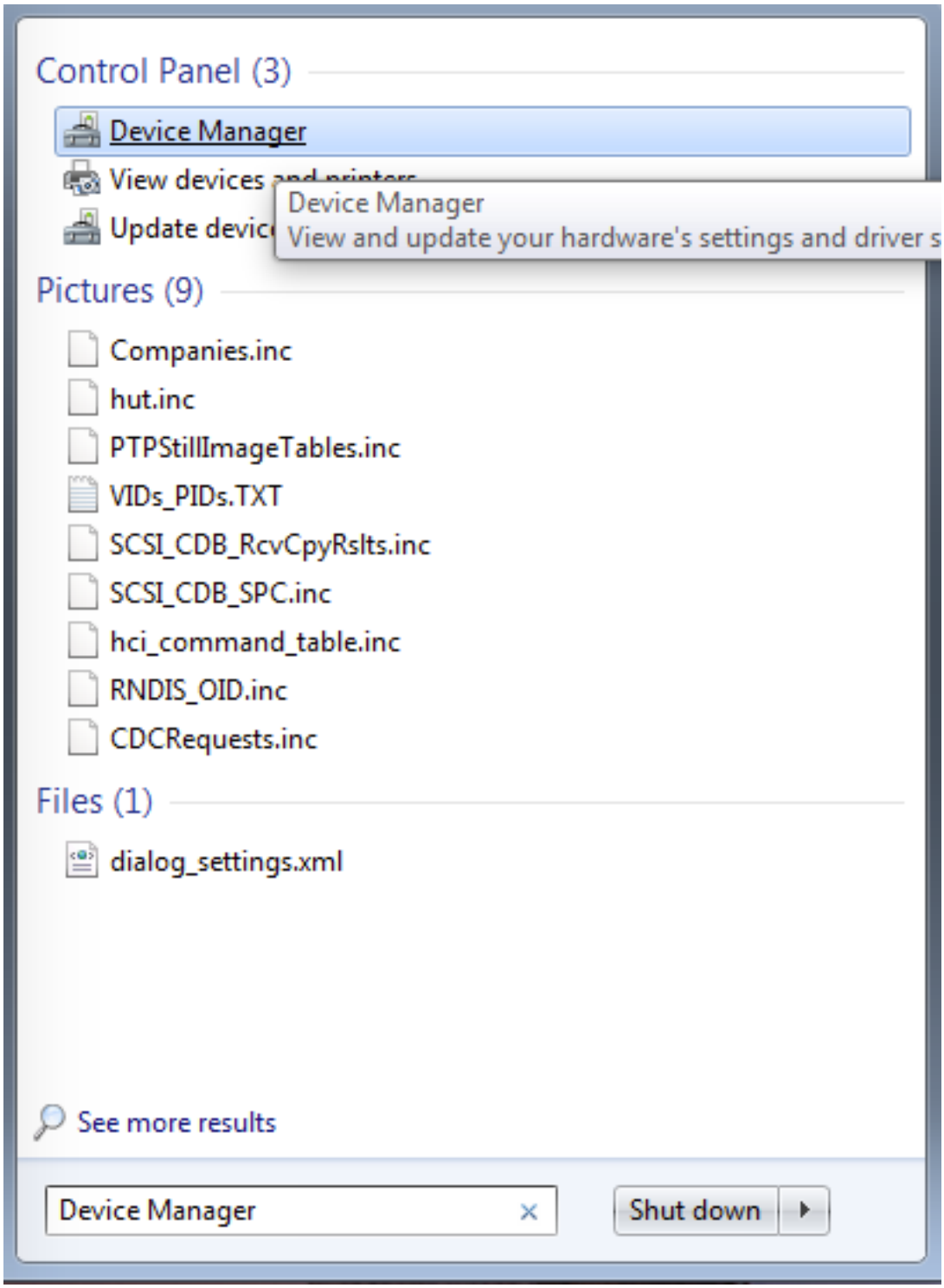

Figure 102. Device manager

2. In the Device Manager, expand the “Ports (COM & LPT)” section to view the available ports. Depending on the NXP board you’re using, the COM port can be named differently:
   a. LPC-Link2

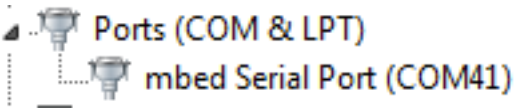

Figure 103. LPC-Link2

10 Appendix B - Default debug interfaces

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

All recent and future NXP hardware platforms support the configurable OpenSDA standard.

NOTE

The 'OpenSDA details' row of the following table is not applicable to LPC.

Table 1. Hardware platforms supported by SDK

Hardware platform	Default interface	OpenSDA details
FRDM-K22F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-K28F	DAPLink	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-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
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0

Table continues on the next page...

Table 1. Hardware platforms supported by SDK (continued)

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-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
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
USB-KW41Z	CMSIS-DAP\DAPLink	OpenSDA v2.1 or greater
USB-KW41Z	CMSIS-DAP\DAPLink	OpenSDA v2.1 or greater
LPC54018 IoT Module	CMSIS-DAP	N/A
LPCXpresso54018	N/A	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
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

11 Appendix C - Updating debugger firmware

11.1 Updating LPCXpresso board firmware

The LPCXpresso hardware platform comes with a CMSIS-DAP-compatible debug interface (known as 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), 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 MCUXSDKLPC540XXGSUG
Revision 0, 01/2018