

Getting Started with MCUXpresso SDK for MIMXRT1170-EVK

1 Overview

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

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MIMXRT1170-EVK* (document MCUXSDKMIMXRT117XRN).

For more details about MCUXpresso SDK, refer to [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#).

Contents

1	Overview.....	1
2	MCUXpresso SDK board support package folders.....	2
3	Run a demo using MCUXpresso IDE.....	3
4	Run a demo application using IAR.....	15
5	Run a demo using Keil® MDK/μVision.	21
6	Run a demo using Arm® GCC.....	26
7	MCUXpresso Config Tools.....	36
A	How to determine COM port.....	37
B	Default debug interfaces.....	38
C	How to add or remove boot header for XIP targets.....	40



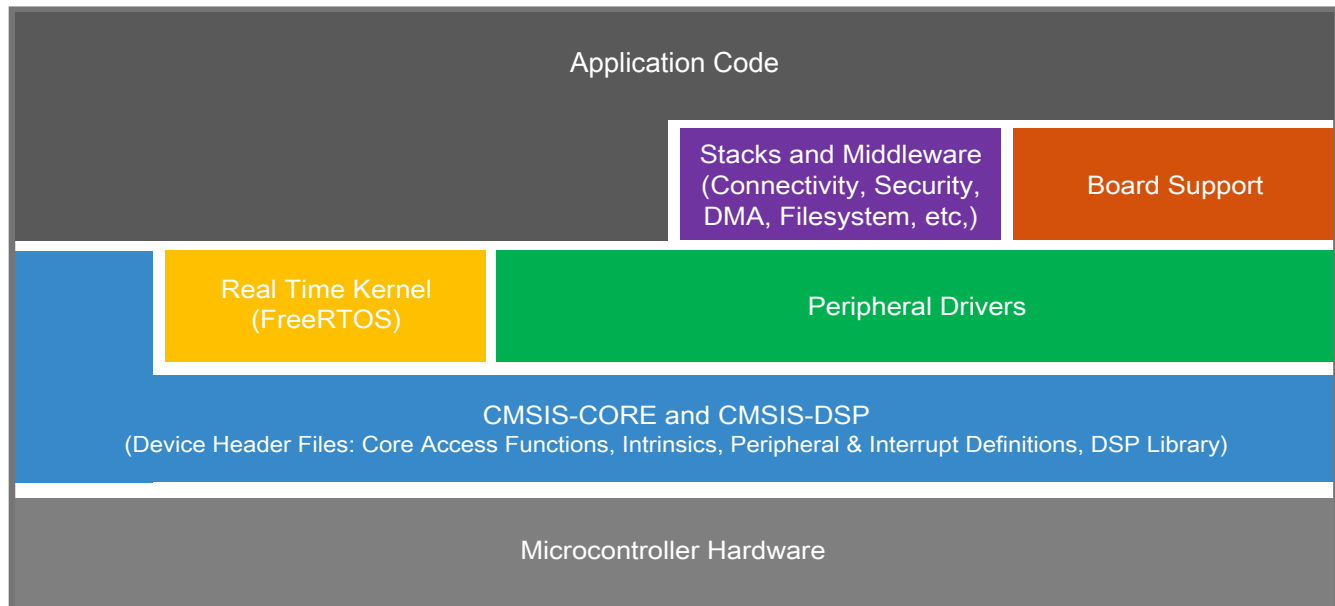


Figure 1. MCUXpresso SDK layers

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):

- **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).
- **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
- **wireless_examples**: Applications that use the Zigbee and OpenThread stacks.

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*.

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:

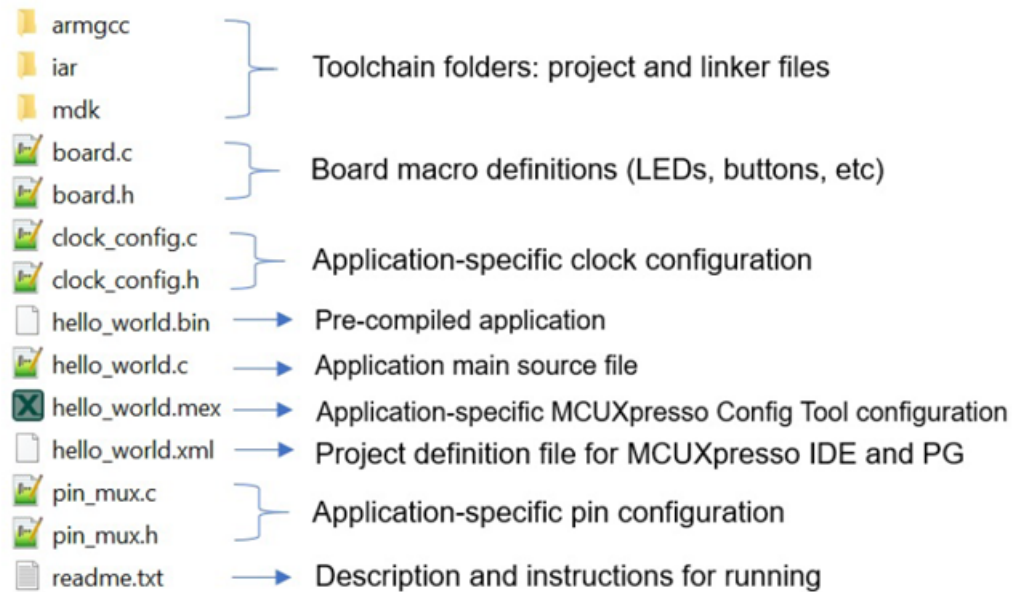


Figure 2. Application folder structure

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>/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
- `devices/<device_name>/project` Project template used in CMSIS PACK new project creation

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.

3 Run a demo using MCUXpresso IDE

NOTE

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

Run a demo using MCUXpresso IDE

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 MIMXRT1170-EVK 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.

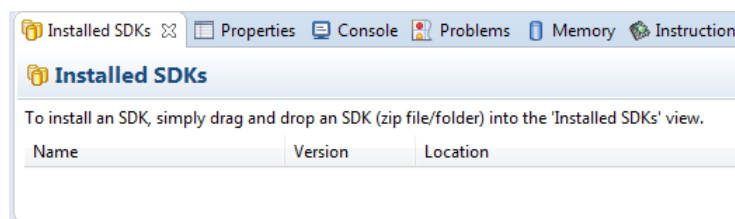


Figure 3. Install an SDK

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



Figure 4. Import an SDK example

3. In the window that appears, select **MIMXRT1176xxxxx**. Then, select **evkmimxrt1170** and click **Next**.

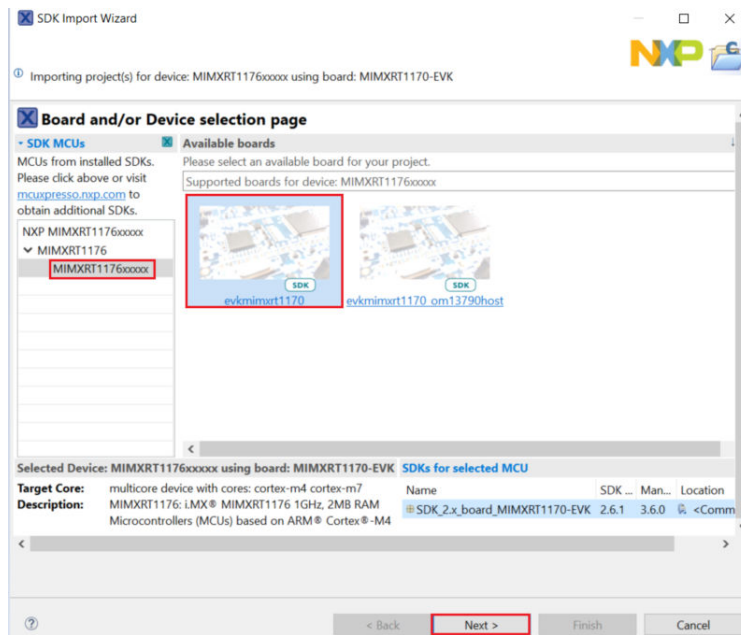


Figure 5. Select MIMXRT1170-EVK board

- Expand the demo_apps folder and select hello_world. Then, click Next.

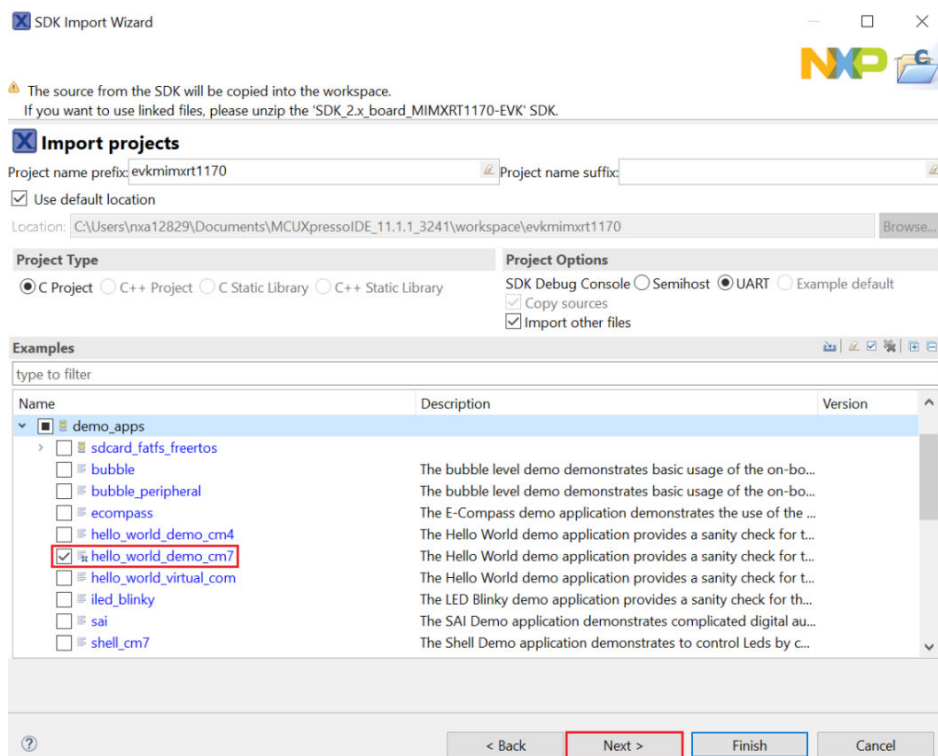


Figure 6. Select hello_world

- Ensure **Redlib: Use floating point version of printf** is selected if the example prints floating point numbers 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 **Finish**.

Run a demo using MCUXpresso IDE

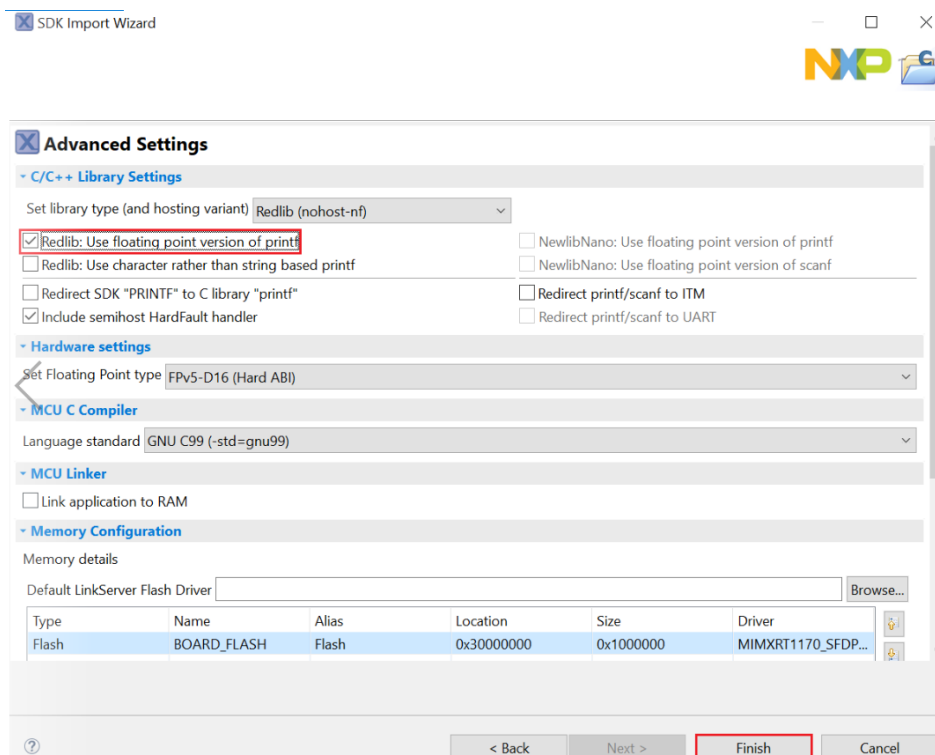


Figure 7. Select Use floating point version of printf

3.3 Run an example application

To download and run the application, perform these steps:

1. See [Default debug interfaces](#) to determine the debug interface that comes loaded on your specific hardware platform.
 - If using J-Link with either a standalone debug pod or OpenSDA, 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 <http://www.keil.com/download/docs/408>.
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 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

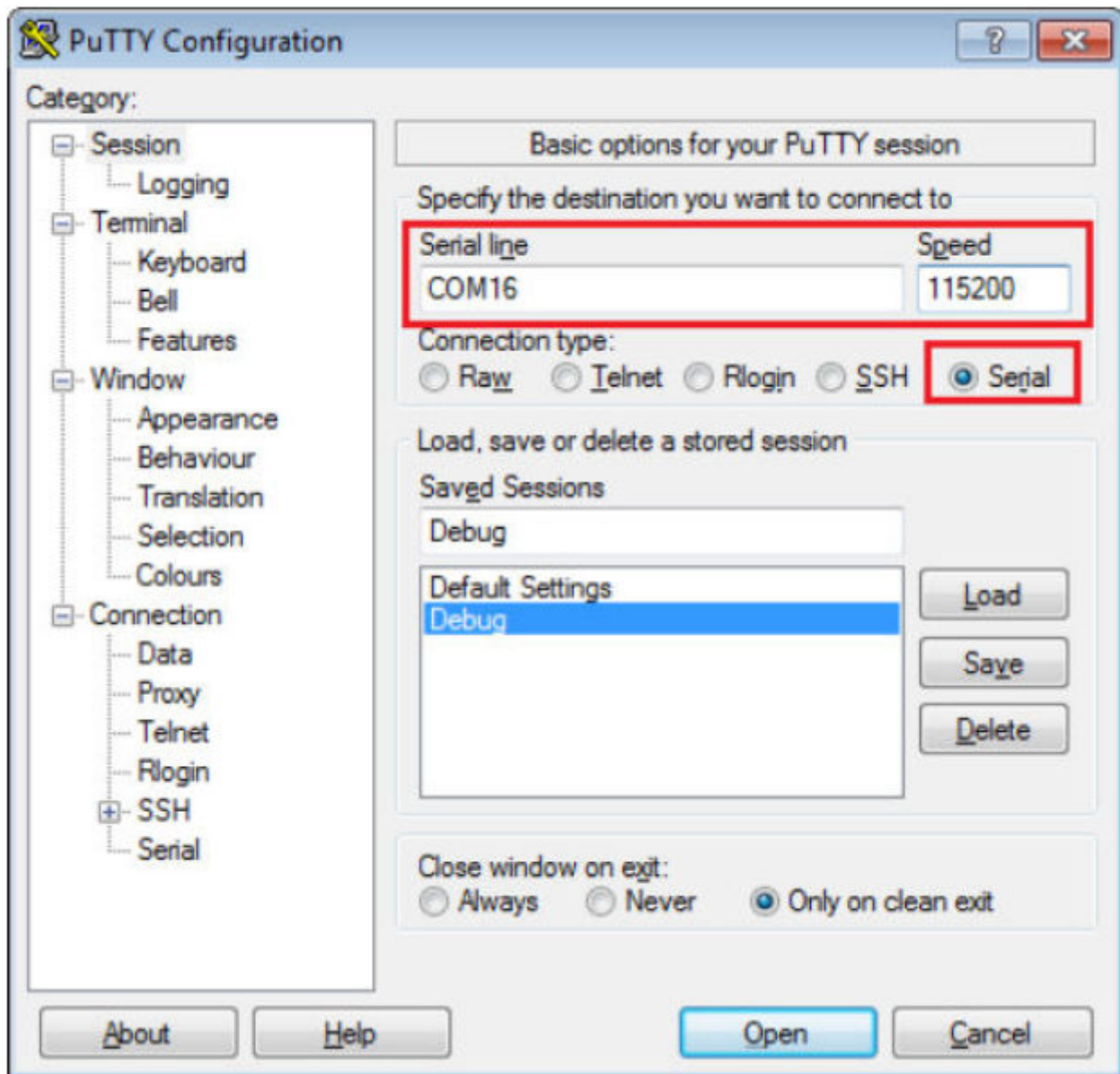


Figure 8. Terminal (PuTTY) configurations

4. On the Quickstart Panel, click on Debug 'evkmimxrt1170_demo_apps_hello_world' [Debug].

Run a demo using MCUXpresso IDE

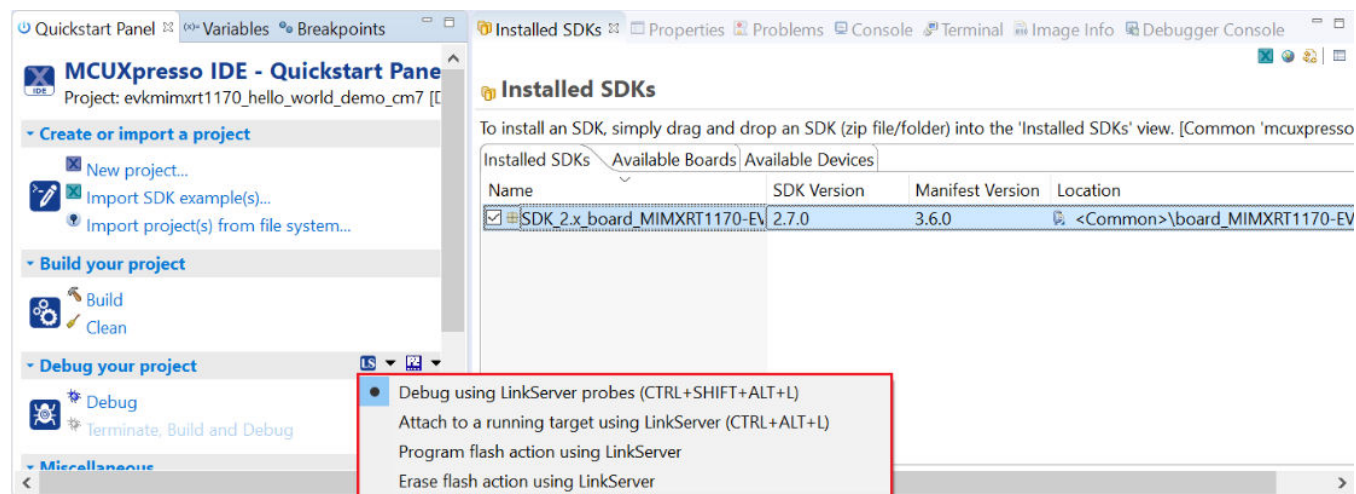


Figure 9. Debugging hello_world case

- 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.)

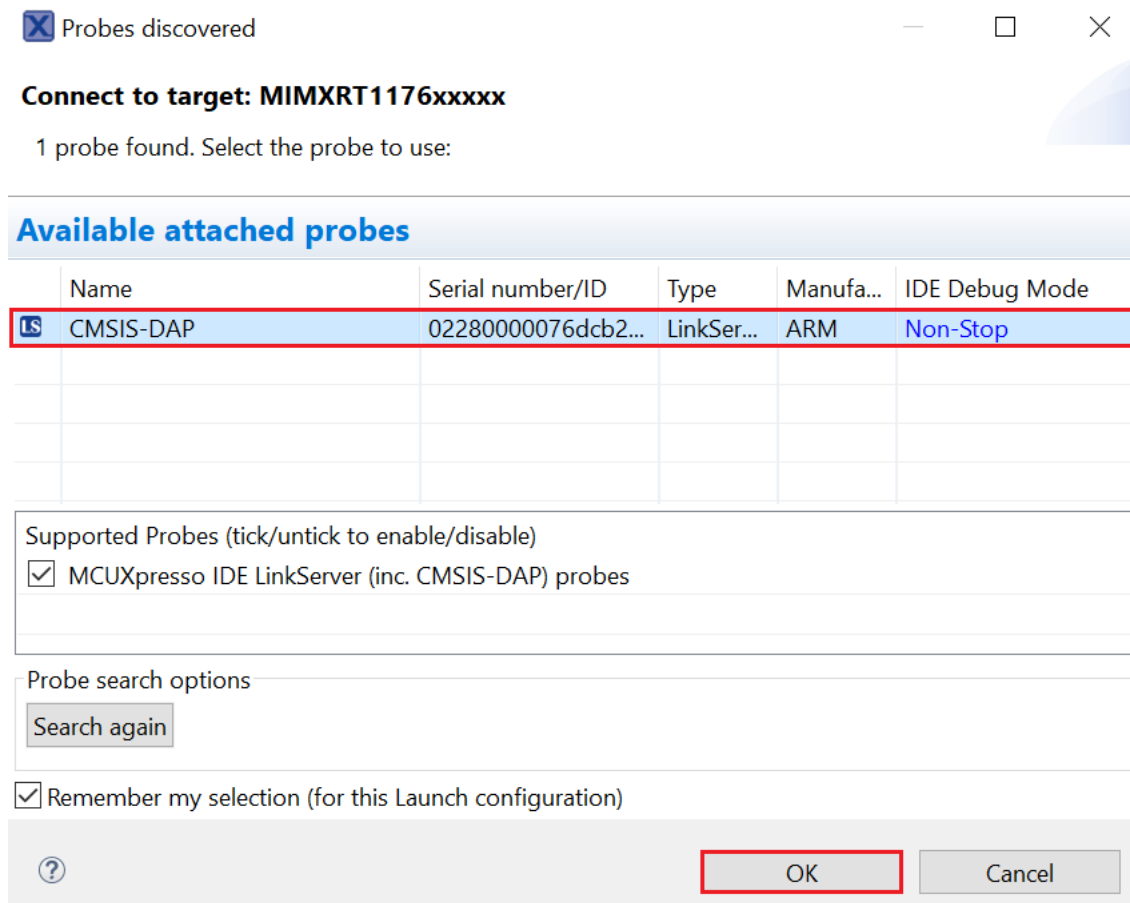
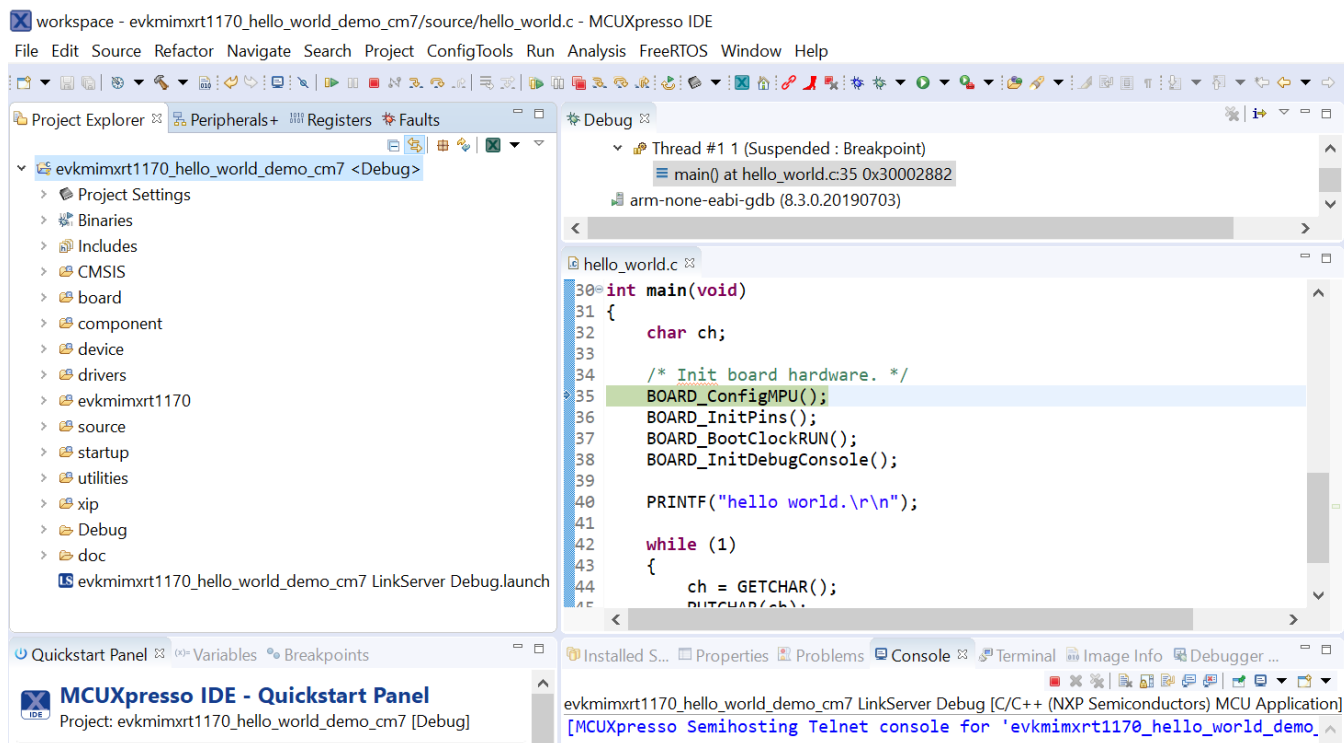


Figure 10. Attached Probes: debug emulator selection

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

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

Error may occur here.

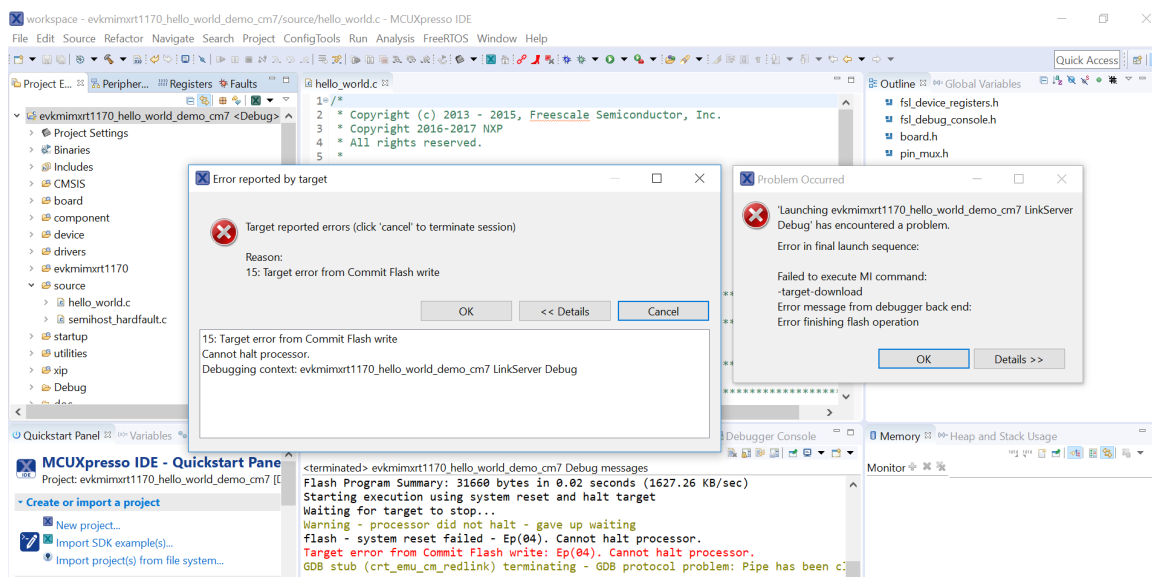


Figure 12. Download error

In this case, double click `evkmimxrt1170_hello_world_demo_cm7 LinkServer Debug.launch`.

Run a demo using MCUXpresso IDE

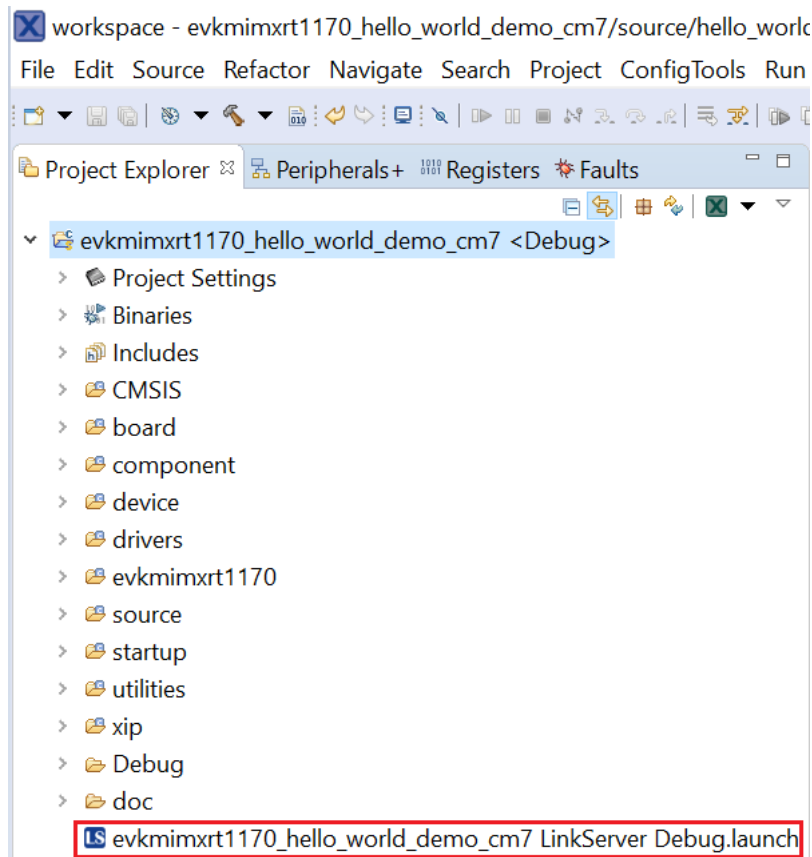


Figure 13. Select Debug launch

Then in **LinkServer Debugger** -> **Reset handling**, select **VECTRESET**. Try again.

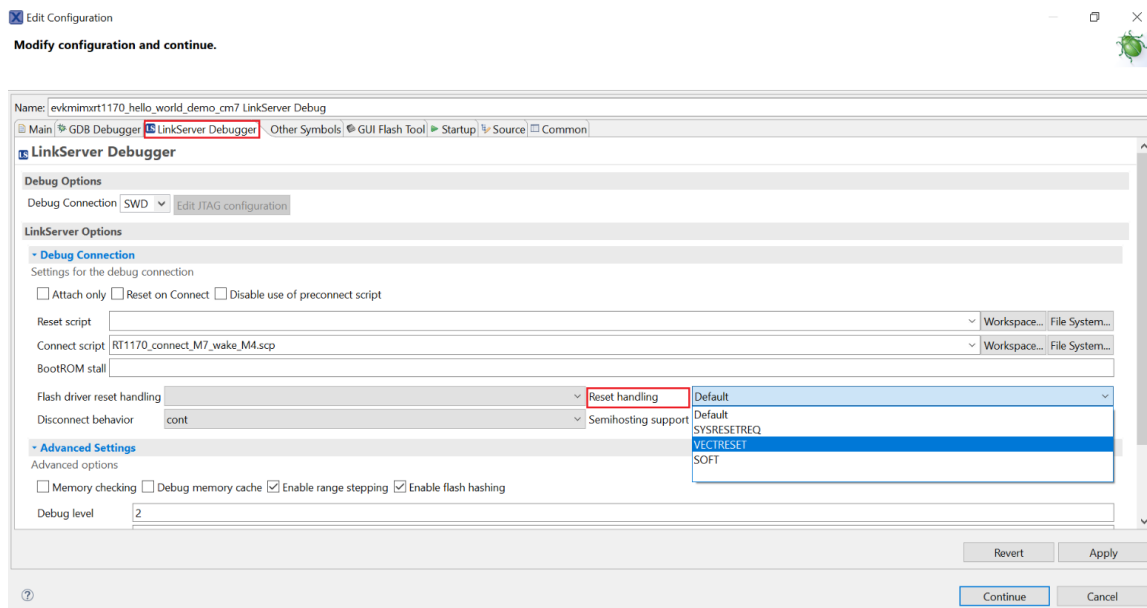


Figure 14. Change Reset Handling

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



Figure 15. 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.

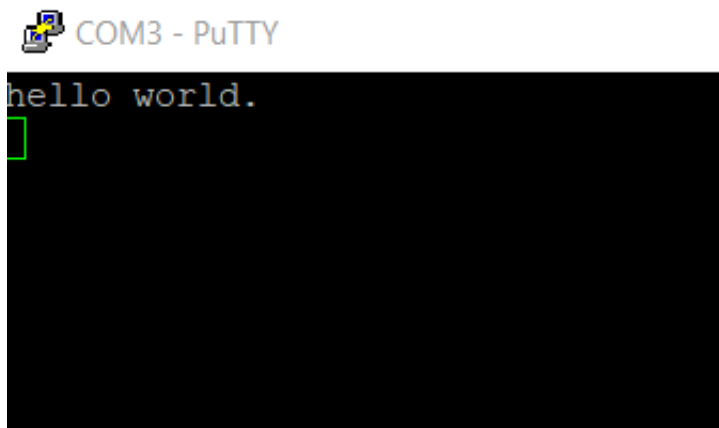


Figure 16. Text display of the `hello_world` demo

3.4 Build a multicore example application

This section describes the steps required to configure MCUXpresso IDE 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 evkmimxrt1170 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 evkmimxrt1170 is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, select **MIMXRT1176xxxxx**. Then, select **evkmimxrt1170** and click **Next**.

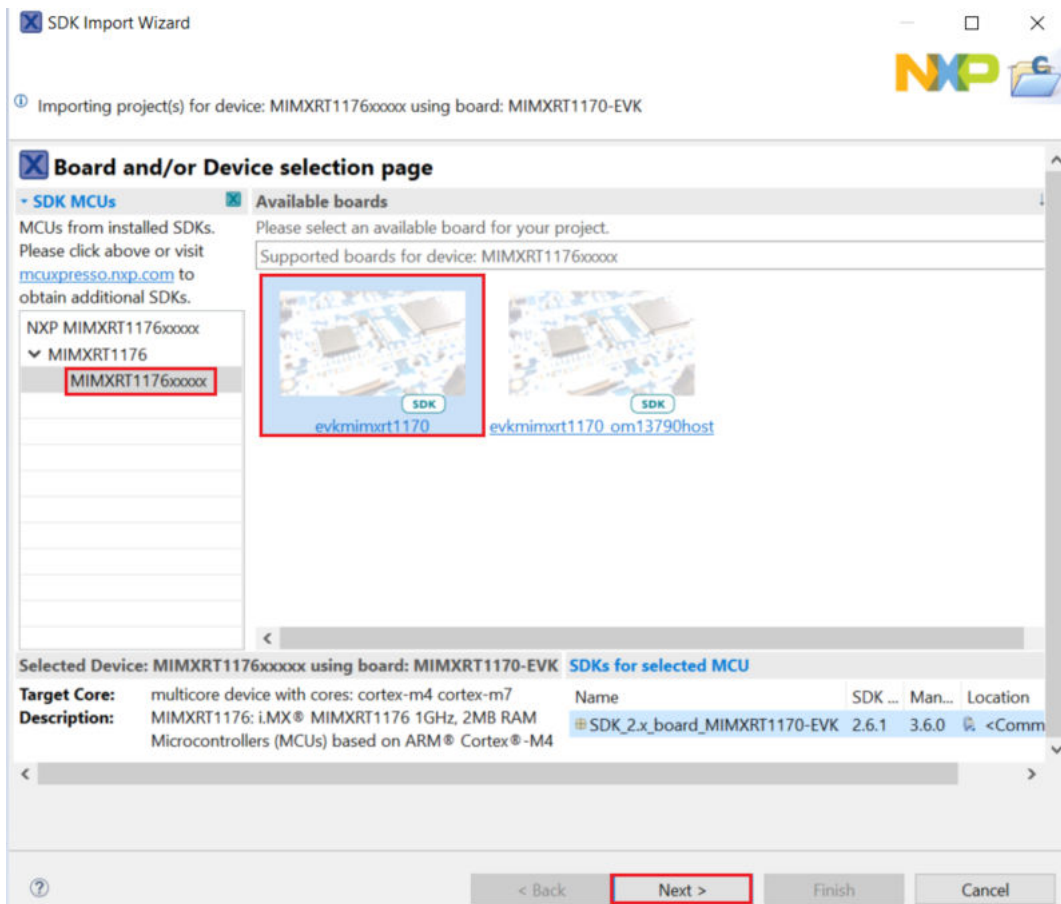


Figure 17. Select the evkmimxrt1170 board

- Expand the `multicore_examples` folder and select `hello_world_cm7`. The `hello_world_cm4` counterpart project is automatically imported with the `cm7` project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.

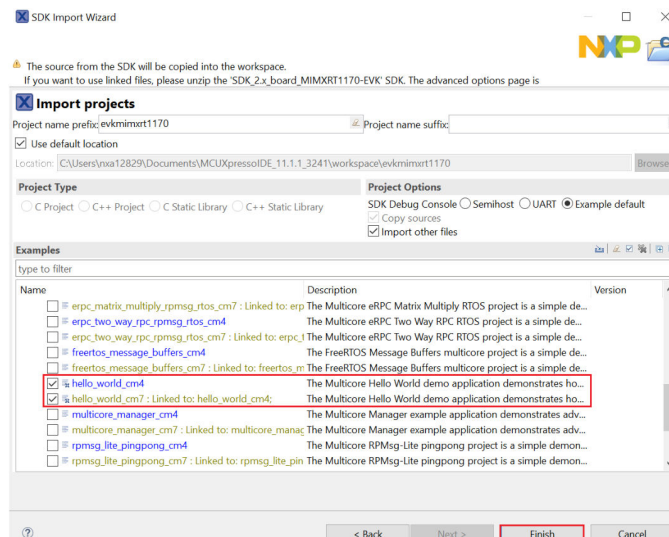


Figure 18. Select the hello_world multicore example

- Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `hello_world_cm7` 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 in Figure 19. For this example, select **Debug**.

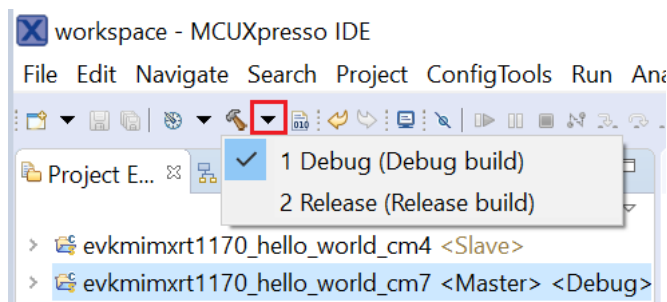


Figure 19. Selection of the build target in MCUXpresso IDE

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 (cm7) also causes the referenced auxiliary core application (cm4) 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 (cm7) build.

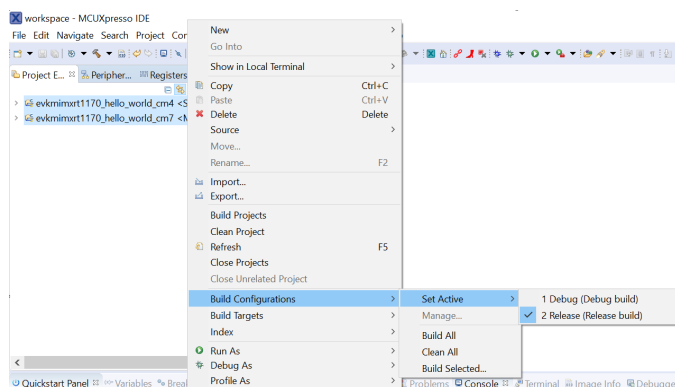


Figure 20. Switching multicore projects into the Release build configuration

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.

Run a demo using MCUXpresso IDE

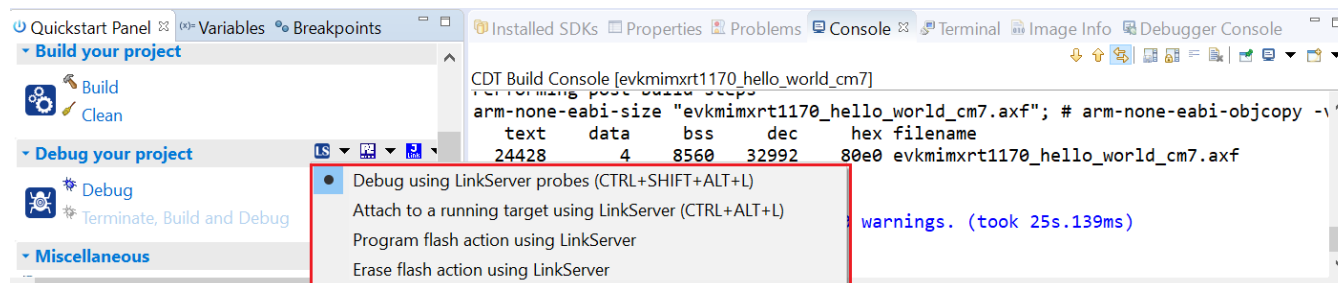


Figure 21. Debug "hello_world_cm7" case

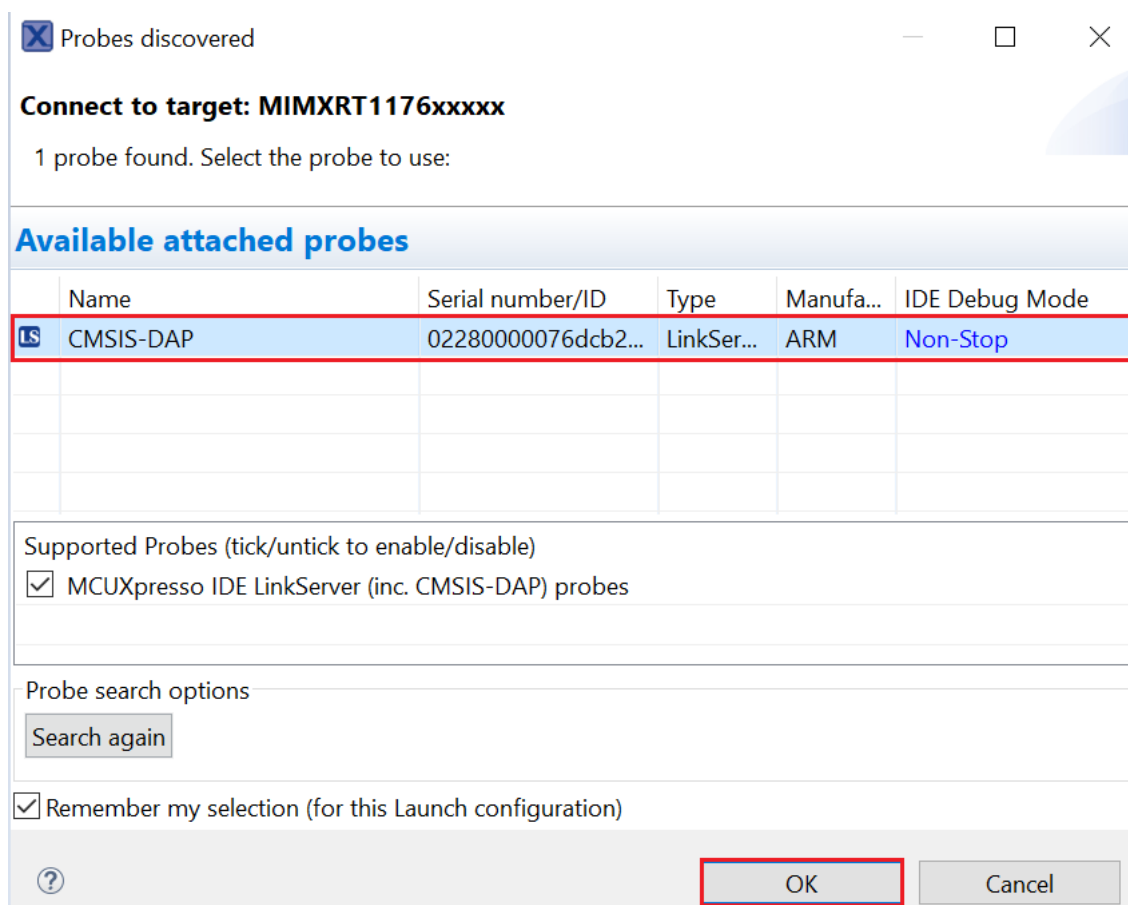


Figure 22. Attached Probes: debug emulator selection

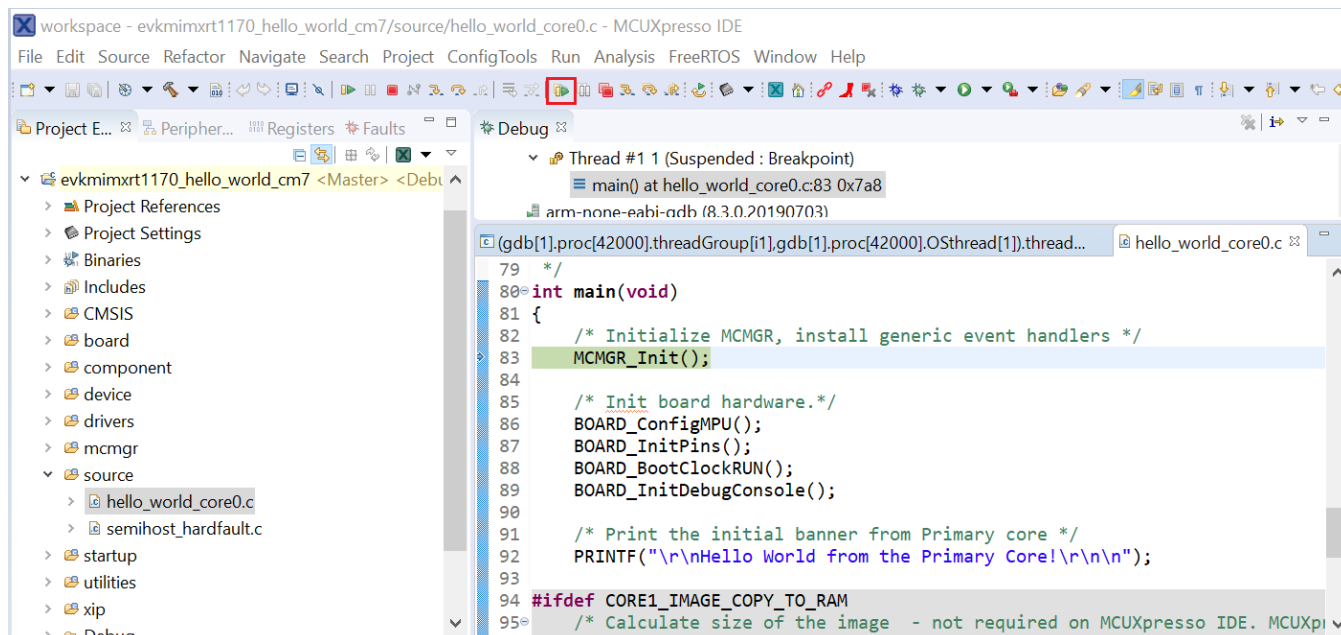


Figure 23. Stop the primary core application at main() when running debugging

After clicking **Resume All Debug sessions**, 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.

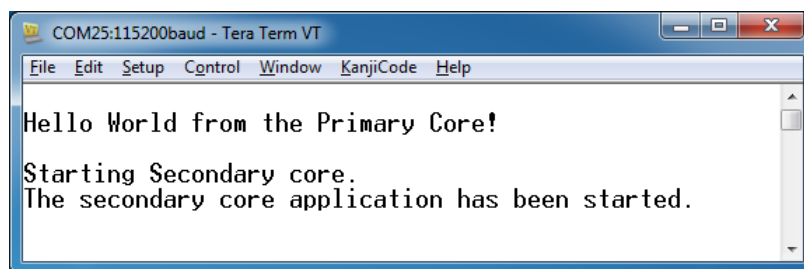


Figure 24. Hello World from the primary core message

4 Run a demo application using IAR

NOTE

Make sure boot mode is at serial download mode (SW1: OFF OFF OFF ON) and reset the board before doing programming or erasing to the external flash on the board.

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 MIMXRT1170-EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

4.1 Build an example application

Do the following steps to build the `hello_world` example application.

Run a demo application using IAR

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>/<core_type>/iar`

Using the MIMXRT1170-EVK hardware platform as an example, the `hello_world` workspace is located in:

`<install_dir>/boards/evkmimxrt1170/demo_apps/hello_world/cm7/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.

There are twelve project configurations (build targets) supported for most MCUXpresso SDK projects:

- **Debug** – Compiler optimization is set to low, and debug information is generated for the executable. The linker file is RAM linker, where text and data section is put in internal TCM.
- **Release** – Compiler optimization is set to high, and debug information is not generated. The linker file is RAM linker, where text and data section is put in internal TCM.
- **ram_0x1400_debug** – Project configuration is same as the debug target. The linker file is RAM_0x1400 linker, where text is put in ITCM with offset 0x1400 and data put in DTCM.
- **ram_0x1400_release** – Project configuration is same as the release target. The linker file is RAM_0x1400 linker, where text is put in ITCM with offset 0x1400 and data put in DTCM.
- **sdram_debug** – Project configuration is same as the debug target. The linker file is SDRAM linker, where text is put in internal TCM and data put in SDRAM.
- **sdram_release** – Project configuration is same as the release target. The linker file is SDRAM linker, where text is put in internal TCM and data put in SDRAM.
- **sdram_txt_debug** – Project configuration is same as the debug target. The linker file is SDRAM_txt linker, where text is put in SDRAM and data put in OCRAM.
- **sdram_txt_release** – Project configuration is same as the release target. The linker file is SDRAM_txt linker, where text is put in SDRAM and data put in OCRAM.
- **flexspi_nor_debug** – Project configuration is same as the debug target. The linker file is flexspi_nor linker, where text is put in flash and data put in TCM.
- **flexspi_nor_release** – Project configuration is same as the release target. The linker file is flexspi_nor linker, where text is put in flash and data put in TCM.
- **flexspi_nor_sdram_release** – Project configuration is same as the release target. The linker file is flexspi_nor_sdram linker, where text is put in flash and data put in SDRAM.
- **flexspi_nor_sdram_debug** – Project configuration is same as the debug target. The linker file is flexspi_nor_sdram linker, where text is put in flash and data put in SDRAM.

For some examples need large data memory, only `sdram_debug` and `sdram_release` targets are supported.

For this example, select **hello_world – debug**.

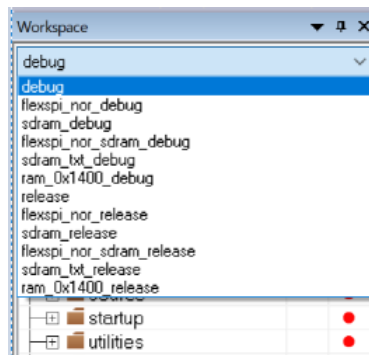


Figure 25. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in [Figure 26](#).

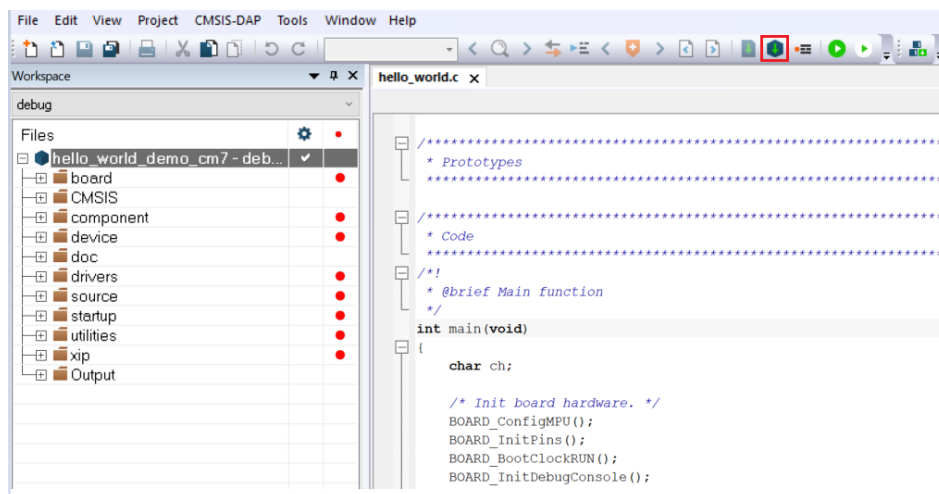


Figure 26. Build the demo application

4. The build completes without errors.

4.2 Run an example application

To download and run the application, perform these steps:

1. This board supports the CMSIS-DAP/mbed/DAPLink debug probe by default. Visit os.mbed.com/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.
2. Connect the development platform to your PC via USB cable. Connect the USB cable to J11 and make sure SW1[1:4] is **0010b**.
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

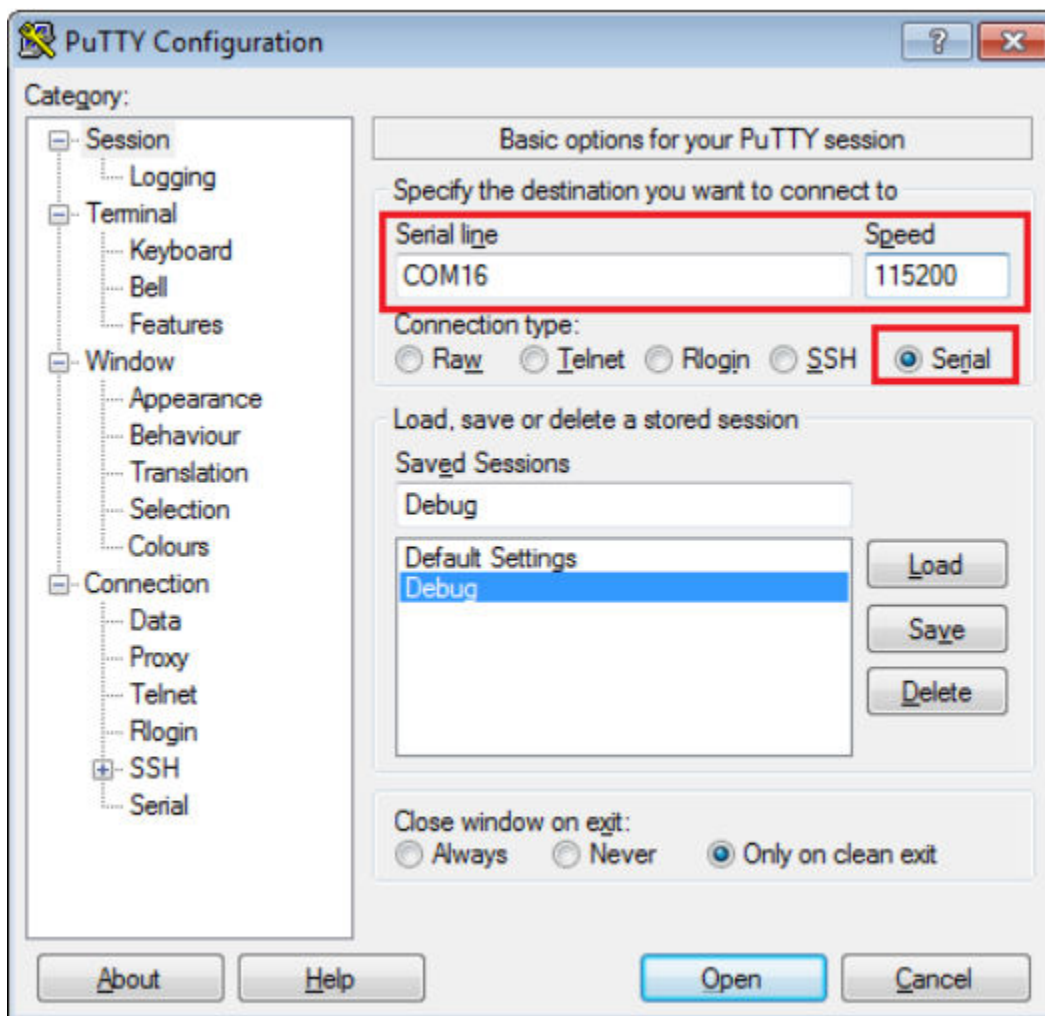


Figure 27. Terminal (PuTTY) configuration

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



Figure 28. Download and Debug button

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

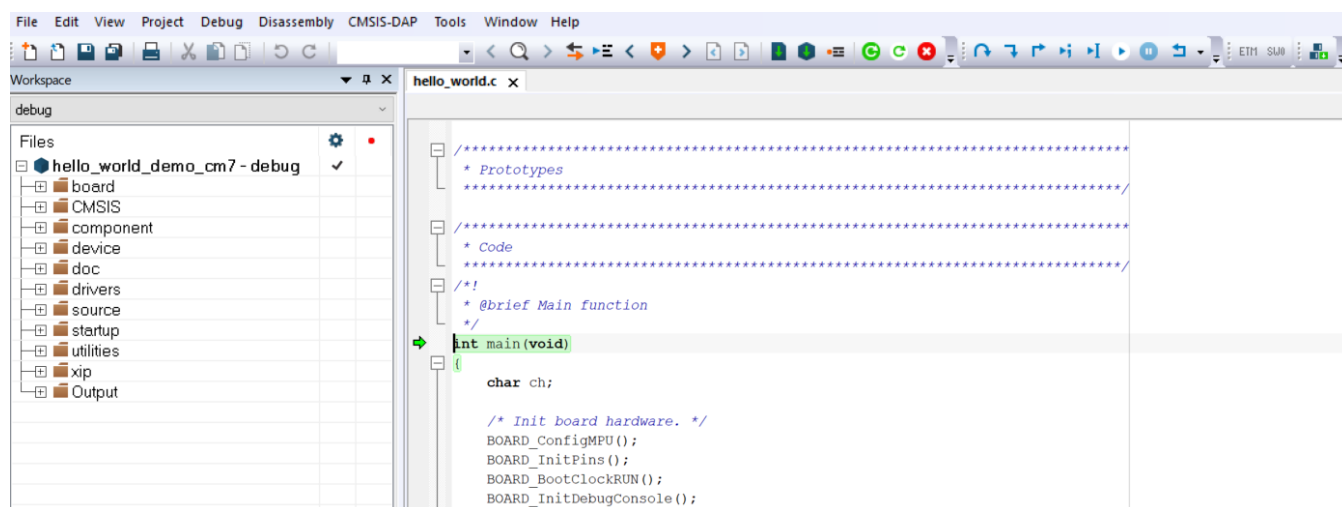


Figure 29. Stop at `main()` when running debugging

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

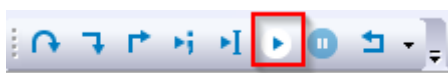


Figure 30. Go button

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

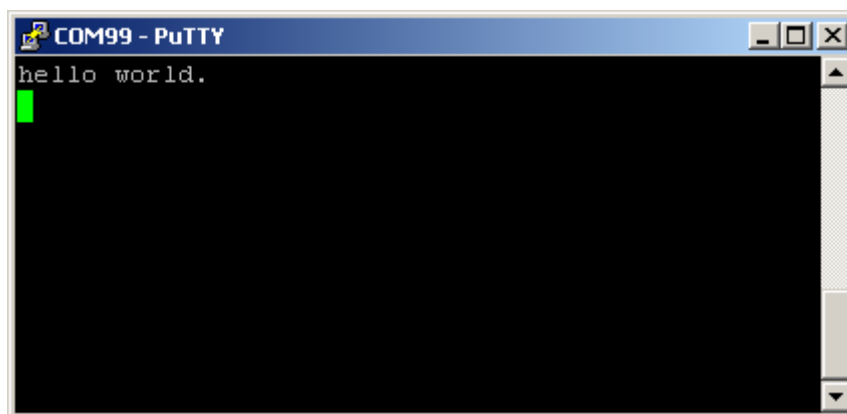


Figure 31. Text display of the `hello_world` demo

4.3 Build a multicore example application

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

```
<install_dir>/boards/evkmimxrt1170/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/evkmimxrt1170/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww
```

Run a demo application using IAR

```
<install_dir>/boards//evkmimxrt1170/multicore_examples/hello_world/cm7/iar/  
hello_world_cm7.eww
```

Build both applications separately by clicking the **Make** button. It is requested to build the application for the auxiliary core (cm4) first, because the primary core application project (cm7) 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.

Because the auxiliary core runs always from RAM, debug and release RAM targets are present in the project only. When building the primary core project, it is possible to select `flexspi_nor_debug/flexspi_nor_release` Flash targets. When choosing Flash targets the auxiliary core binary is linked with the primary core image and stored in the external SPI Flash memory. During the primary core execution the auxiliary core image is copied from flash into the CM4 RAM and executed.

4.4 Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the Flash memory when `flexspi_nor_debug/flexspi_nor_release` targets are used. 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 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 32. Start all cores button

During the primary core code execution, the auxiliary core code is re-allocated from the SPI 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.

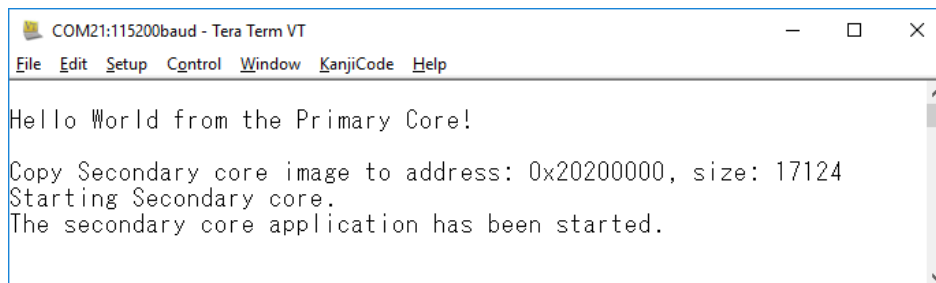


Figure 33. Hello World from primary core message

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.

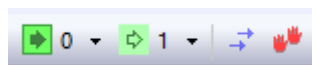


Figure 34. Stop all cores and Start all cores control buttons

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.

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 MIMXRT117x CMSIS pack.

1. Download the MIMXRT1171, MIMXRT1172, MIMXRT1173, MIMXRT1175 and MIMXRT1176 packs .
2. After downloading the DFP, double click to install it.

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/evkmimxrt1170/demo_apps/hello_world/cm7/mdk/hello_world.uvmpw
```

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

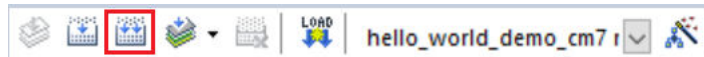


Figure 35. Build the demo

3. The build completes without errors.

5.3 Run an example application

To download and run the application, perform these steps:

1. This board supports the CMSIS-DAP/mbd/DAPLink debug probe by default. Visit os.mbed.com/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.
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 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

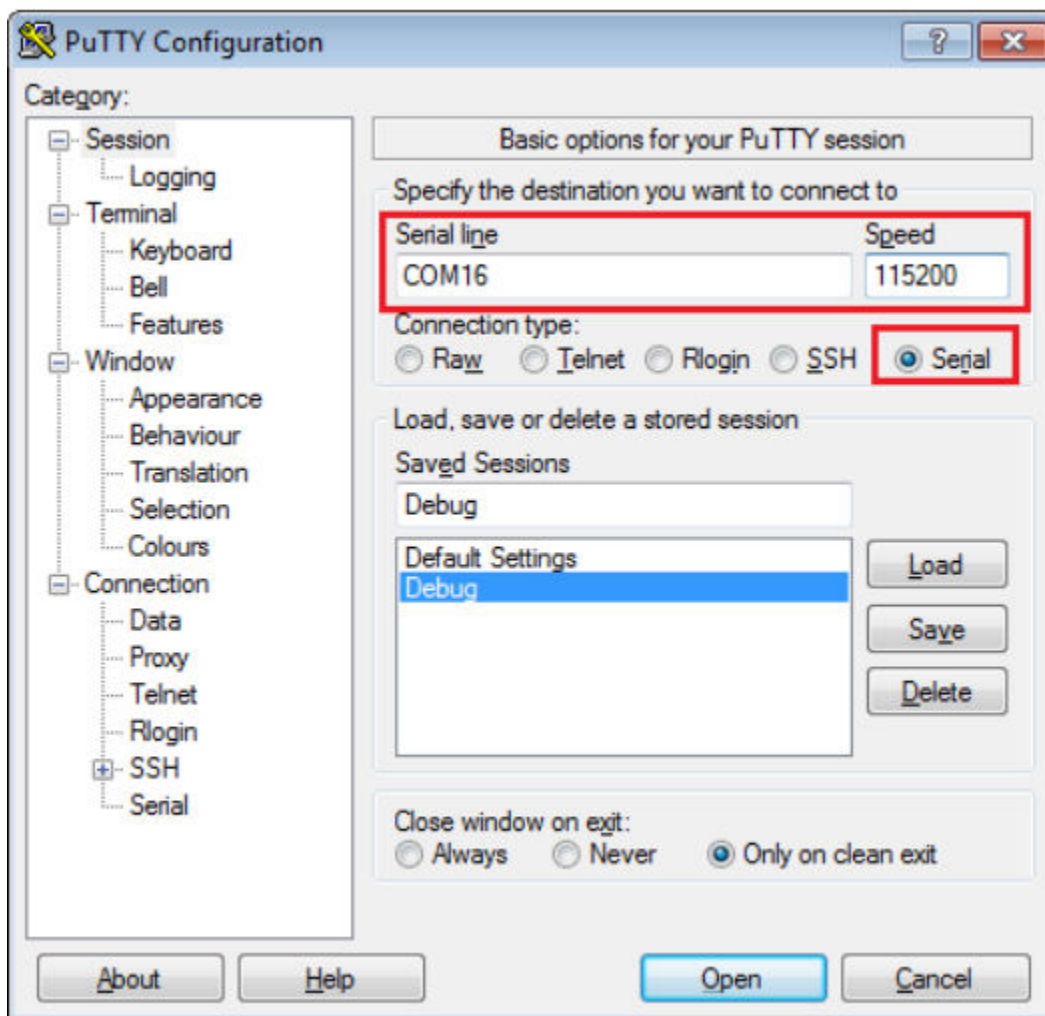


Figure 36. Terminal (PuTTY) configurations

4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button, highlighted in red in [Figure 37](#). If using **J-Link** as the debugger, click **Project option > Debug > Settings > Debug > Port**, and select **SW**.

NOTE

It will need the jlinkscript
(evkmimxrt1170_connect_cm4_cm4side.jlinkscript) to establish a debug session to CM4. And in MDK, it expects a jlinkscript named JLinkSettings.JLinkScript in the folder where the uVision project files are located. Please refer to Segger Wiki for more information: https://wiki.segger.com/Keil_MDK-ARM.

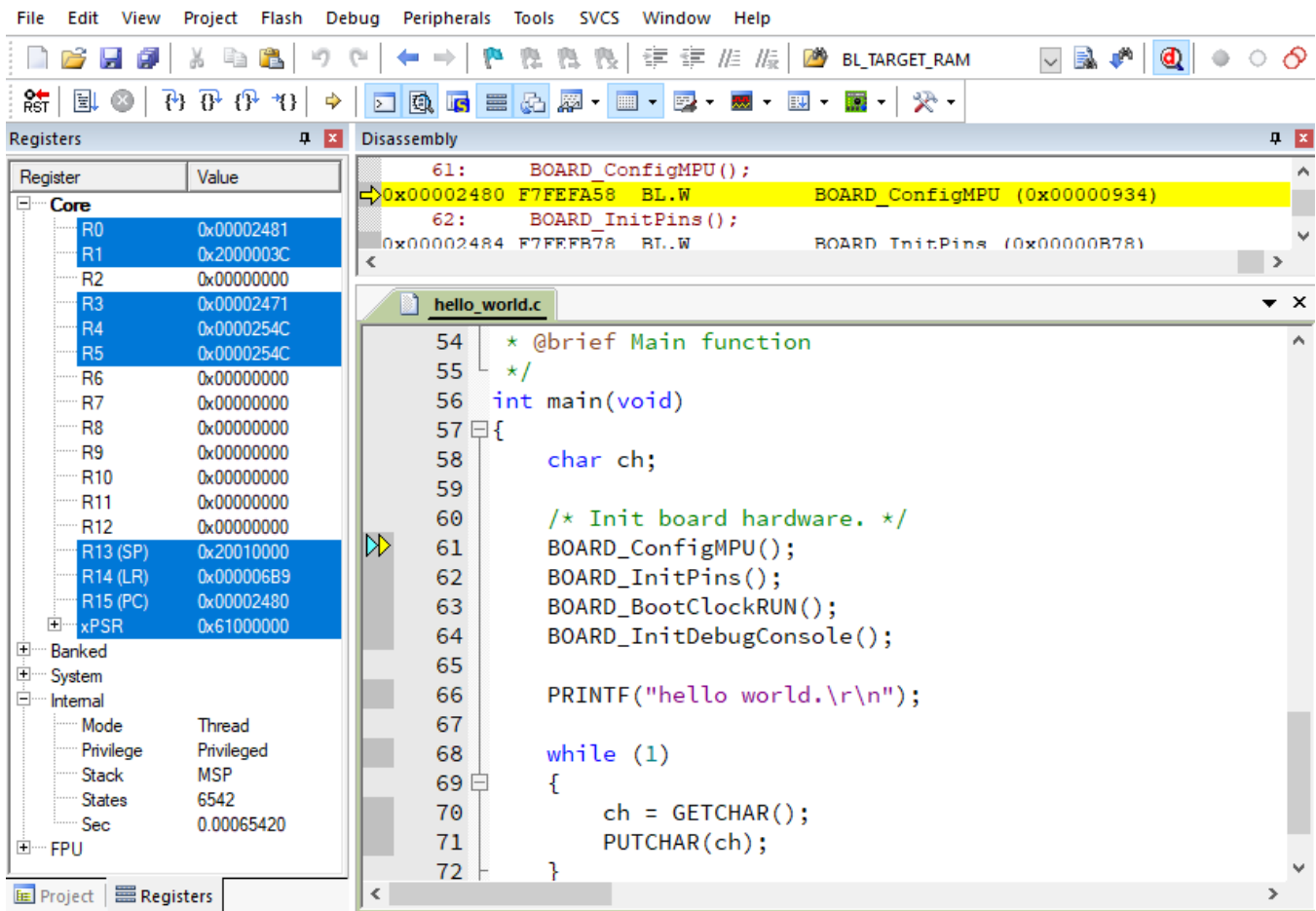


Figure 37. Stop at `main()` when run debugging

5. Run the code by clicking **Run** to start the application, as shown in Figure 38.

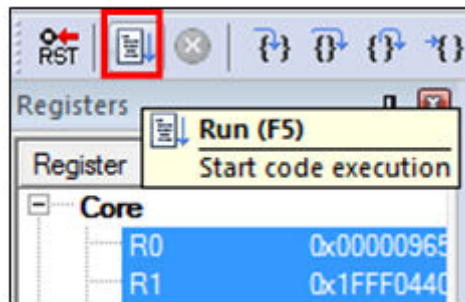


Figure 38. Run button

The `hello_world` application is now running and a banner is displayed on the terminal, as shown in Figure 39. If this is not true, check your terminal settings and connections.

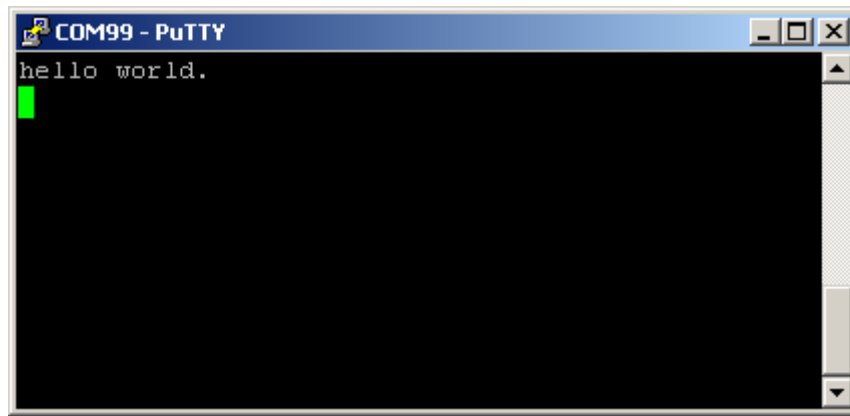


Figure 39. Text display of the `hello_world` demo

5.4 Build a multicore example application

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

```
<install_dir>/boards/evkmimxrt1170/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/evkmimxrt1170/multicore_examples/hello_world/cm4/mdk/  
hello_world_cm4.uvmpw
```

```
<install_dir>/boards/evkmimxrt1170/multicore_examples/hello_world/cm7/mdk/  
hello_world_cm7.uvmpw
```

Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm4) first, because the primary core application project (cm7) 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.

Because the auxiliary core runs always from RAM, debug and release RAM targets are present in the project only. When building the primary core project, it is possible to select `flexspi_nor_debug/flexspi_nor_release` Flash targets. When choosing Flash targets the auxiliary core binary is linked with the primary core image and stored in the external SPI Flash memory. During the primary core execution the auxiliary core image is copied from flash into the CM4 RAM and executed.

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 flash memory. After clicking **Run**, the primary core application is executed. During the primary core code execution, the auxiliary core code is re-allocated from the SPI 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.

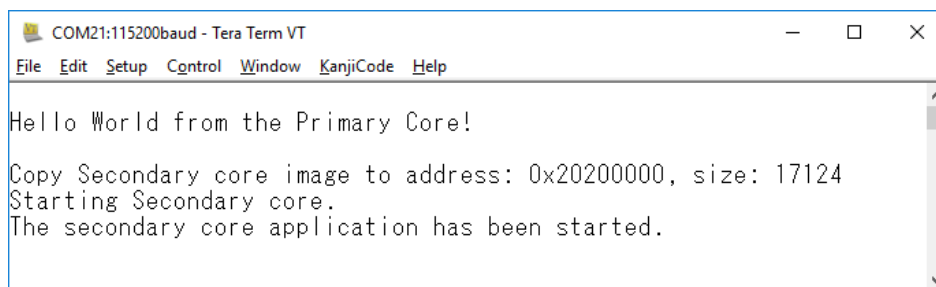


Figure 40. Hello World from primary core message

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 **Start/Stop Debug Session**. After doing this, the second debug session is opened and the auxiliary core application can be debugged.

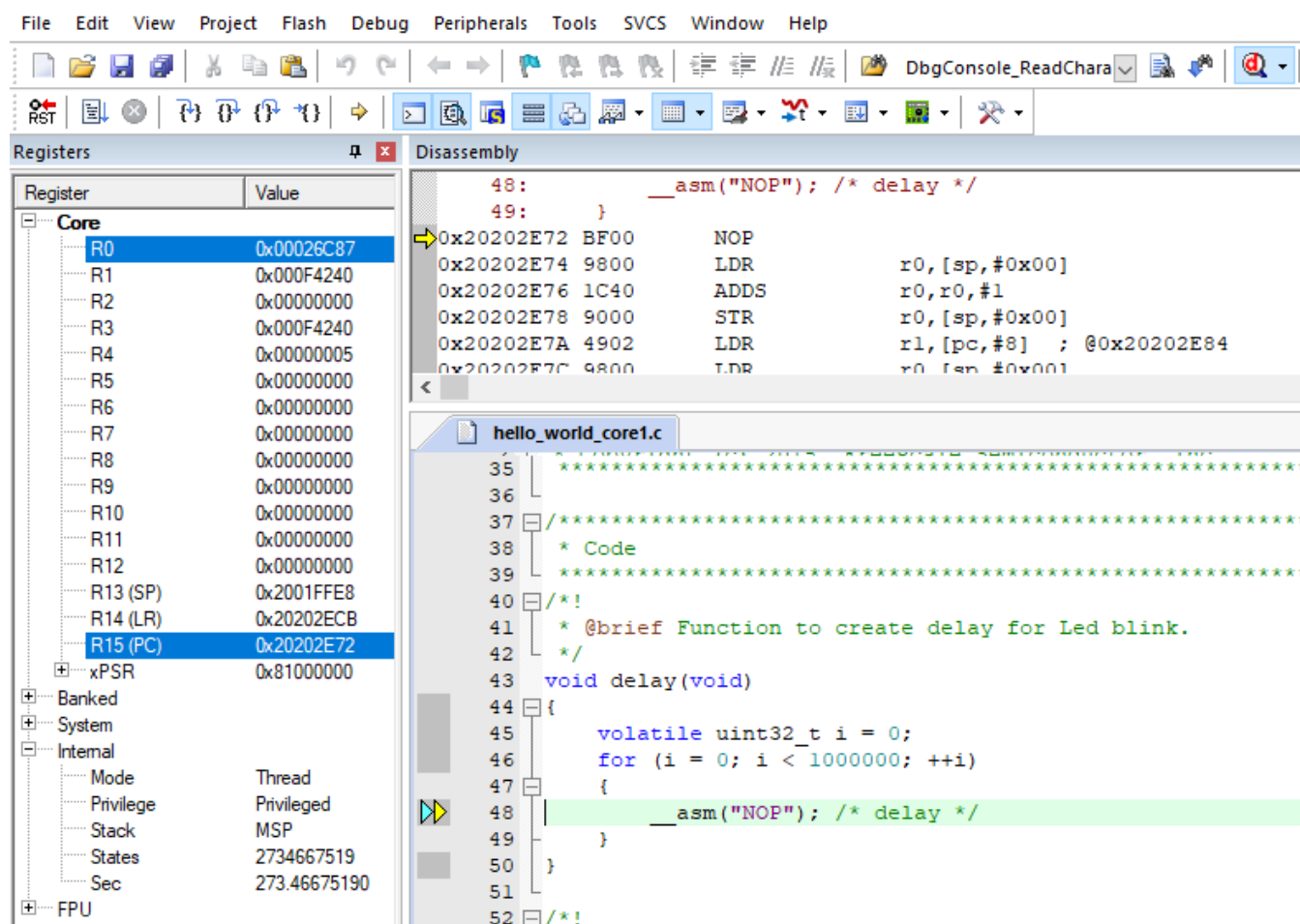


Figure 41. Second debugging auxiliary core application

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 which is used as an example.

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 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 *MCUXpresso SDK Release Notes Supporting MIMXRT1170-EVK* (document MCUXSDKMIMXRT117X).

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**.

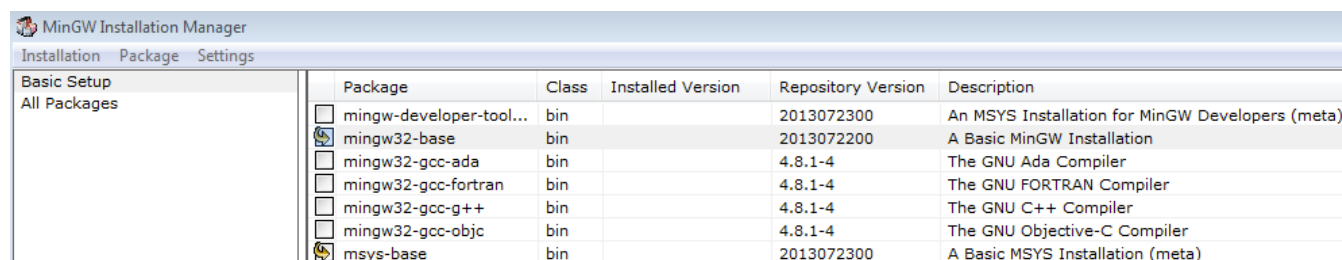


Figure 42. Set up MinGW and MSYS

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

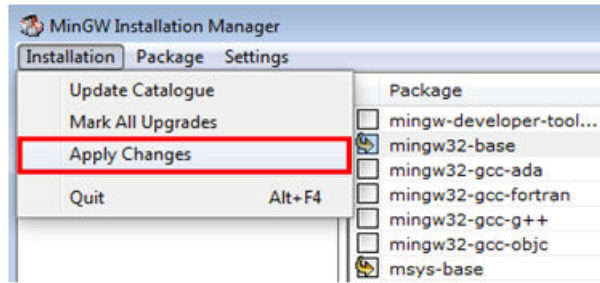


Figure 43. Complete MinGW and MSYS installation

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.

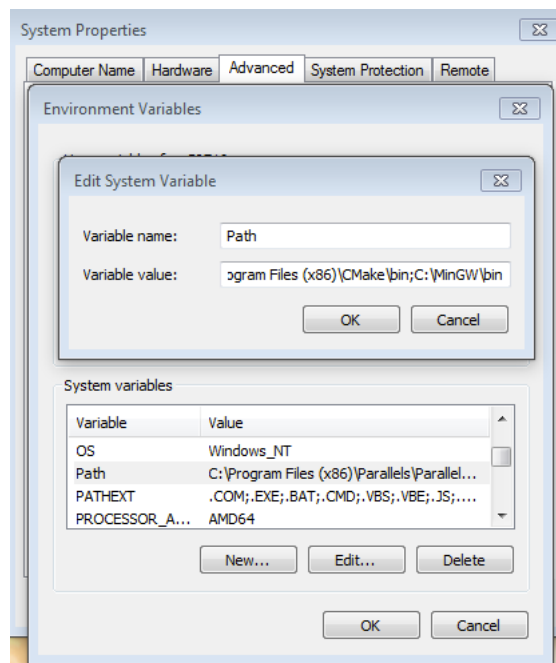


Figure 44. Add Path to systems environment

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\8 2018-q4-major
```

Run a demo using Arm® GCC

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

Short path should be used for path setting, you could convert the path to short path by running command `for %I in (.) do echo %~sI` in above path.

```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018~1
C:\PROGRA~2\GNUTOO~1\82018~1
```

Figure 45. Convert path to short path

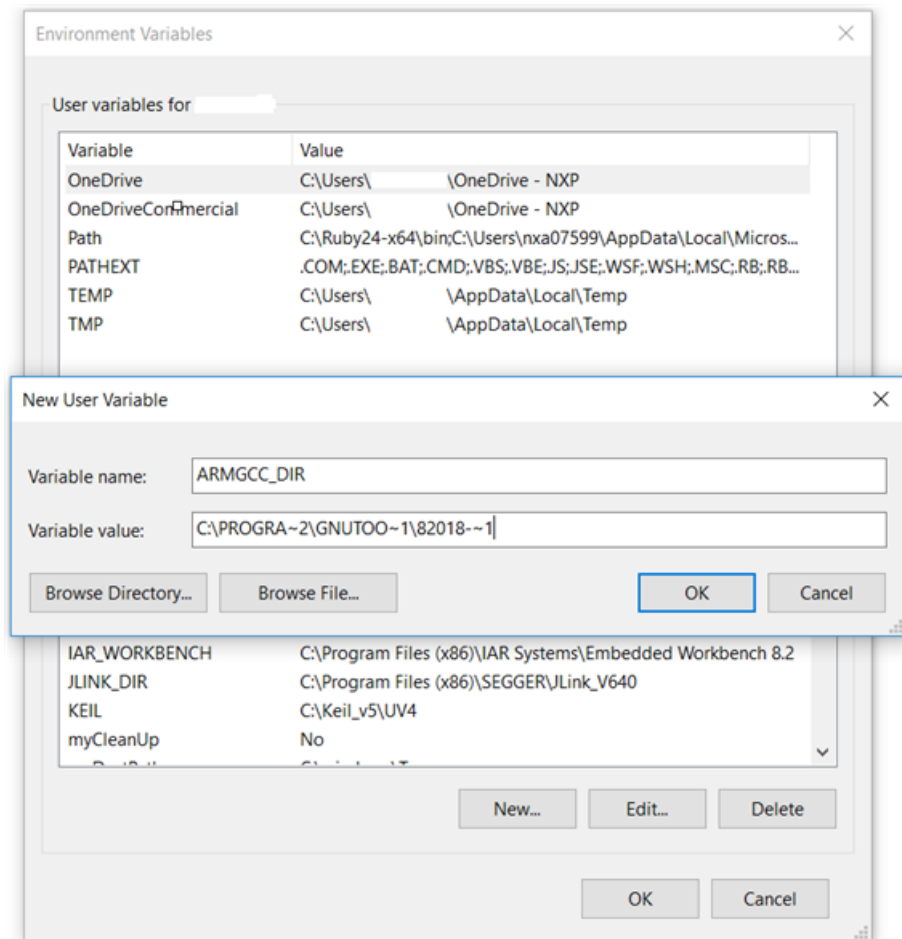


Figure 46. Add ARMGCC_DIR system variable

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.

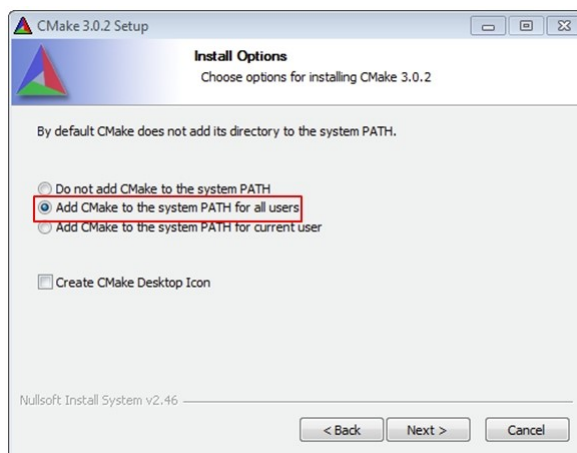


Figure 47. Install CMake

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**.

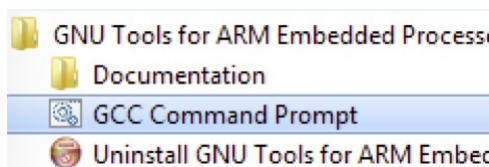


Figure 48. Launch command prompt

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>/<core_type>/<application_name>/armgcc
```

For this example, the exact path is:

```
<install_dir>/examples/evkmimxrt1170/demo_apps/hello_world/cm7/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 as shown in [Figure 49](#).


```

[ 90%] Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/C:/Users/nxa12829/Desktop/rt1170/boards/evkmimxrt1170/xip/evkmimxrt1170_sdram_ini_dcd.c.obj
[ 95%] [100%] Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/C:/Users/nxa12829/Desktop/rt1170/devices/MIMXRT1176/drivers/fsl_anatop.c.obj
Building C object CMakeFiles/hello_world_demo_cm7.elf.dir/C:/Users/nxa12829/Desktop/rt1170/devices/MIMXRT1176/drivers/fsl_anatop_ai.c.obj
Linking C executable debug\hello_world_demo_cm7.elf
[100%] Built target hello_world_demo_cm7.elf
C:\Users\nxa12829\Desktop\rt1170\boards\evkmimxrt1170\demo_apps\hello_world\cm7\armgcc>IF "" == "" (pause )
Press any key to continue . . .

```

Figure 49. 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 perform this exercise, make sure that either:

- The OpenSDA interface on your board is programmed with the J-Link OpenSDA firmware. If your board does not support OpenSDA, then a standalone J-Link pod is required.
- You have a standalone J-Link pod that is 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.

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

1. This board supports the J-Link debug probe. Before using it, install SEGGER software, which can be downloaded from <http://www.segger.com>.
2. 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, also connect it to the SWD/JTAG connector of the board.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
 - a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit

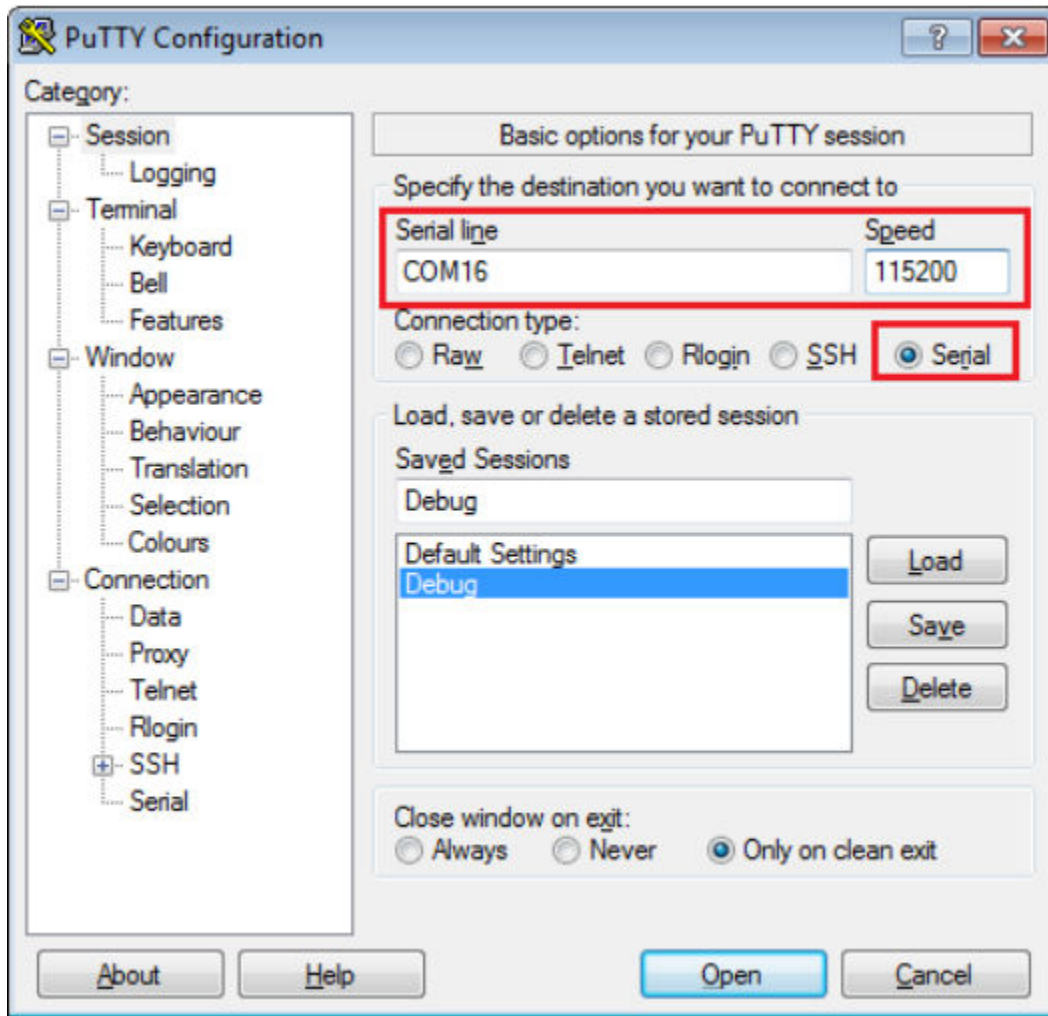


Figure 50. Terminal (PuTTY) configurations

4. Open the J-Link GDB Server application. Go to the SEGGER install folder. For example, *C:\Program Files(x86)\SEGGER\JLink_Vxxx*. Open the command windows. For Debug and Release targets, use the `JLinkGDBServer.exe` command. For the `sdram_debug`, `sdram_release`, `flexspi_nor_sdram_debug`, and `flexspi_nor_sdram_release` targets, use the `JLinkGDBServer.exe-jlinkscriptfile <install_dir>/boards/evkmimxrt1170/demo_apps/hello_world/cm7/evkmimxrt1170_connect_cm4_cm7side.jlinkscript` command
5. The target device selection chosen for this example is **MIMXRT1176DVMAA_cm7**.
6. After it is connected, the screen should resemble [Figure 51](#).

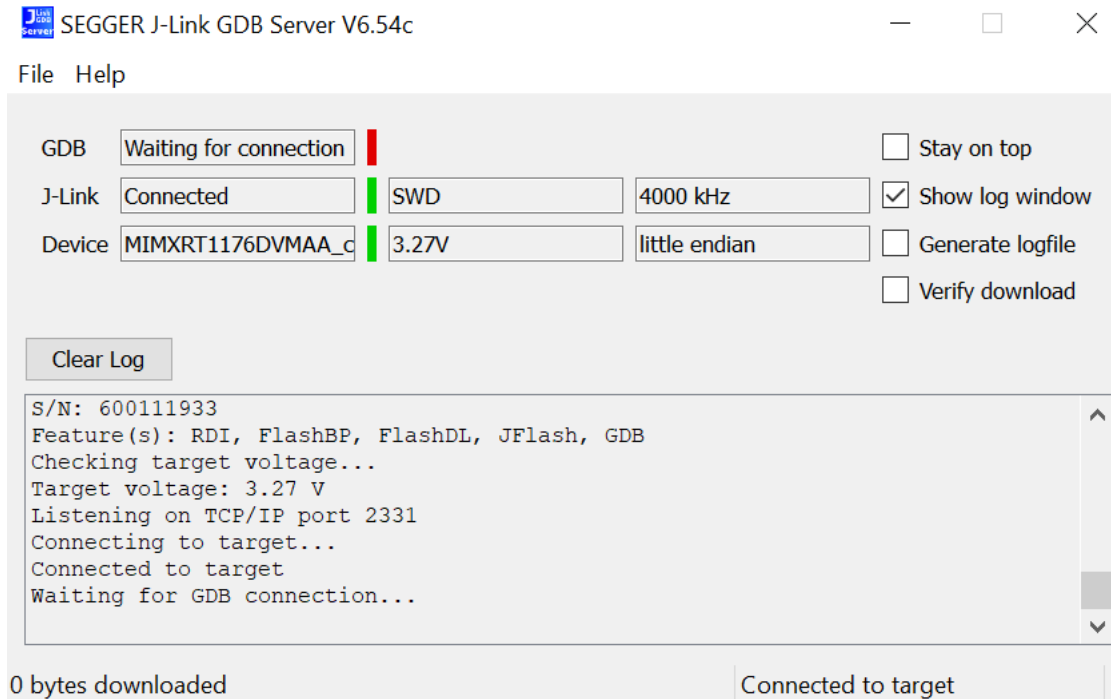


Figure 51. SEGGER J-Link GDB Server screen after successful connection

7. 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**.

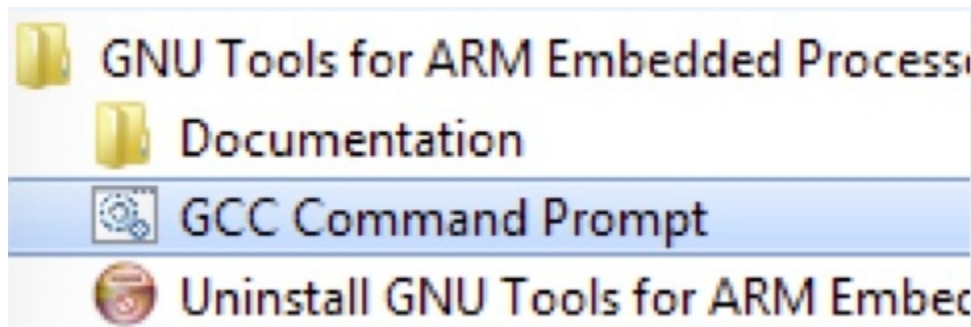


Figure 52. Launch command prompt

8. 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>/cm7/armgcc/debug`

`<install_dir>/boards/<board_name>/<example_type>/<application_name>/cm7/armgcc/release`

For this example, the path is:

`<install_dir>/boards/evkmimxrt1170/demo_apps/hello_world/cm7/armgcc/debug`

9. Run the `arm-none-eabi-gdb.exe <application_name>.elf`. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.



```

C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major>arm-none-eabi-gdb.exe C:\Users\nxa12829\Desktop\rt1170\boards\evkmimxrt1170\demo_apps\hello_world\cm7\armgcc\debug\hello_world_demo_cm7.elf
C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 8-2018-q4-major) 8.2.50.20181213-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 C:\Users\nxa12829\Desktop\rt1170\boards\evkmimxrt1170\demo_apps\hello_world\cm7\armgcc\debug\hello_world_demo_cm7.elf...
(gdb)

```

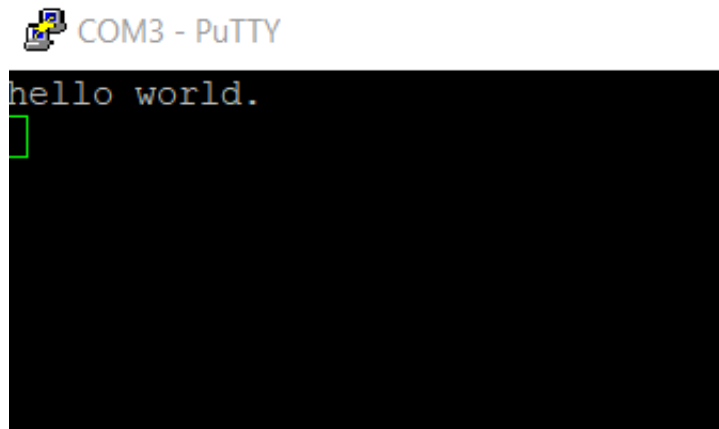
Figure 53. Run `arm-none-eabi-gdb`

10. Run these commands:

- a. `target remote localhost:2331`
- b. `monitor reset`
- c. `monitor halt`
- d. `load`

11. 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 54. 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:

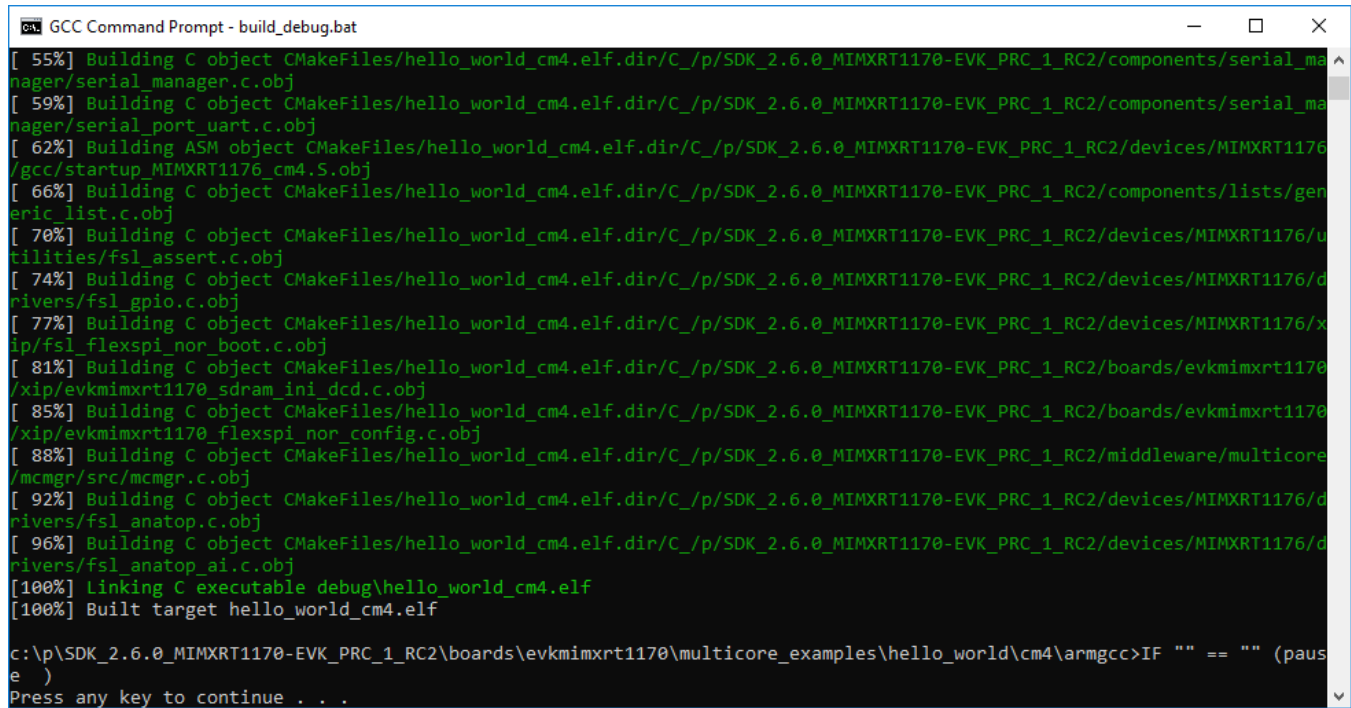
```
<install_dir>/boards/evkmimxrt1170/multicore_examples/<application_name>/<core_type>/armgcc
```

Run a demo using Arm® GCC

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/evkmimxrt1170/multicore_examples/hello_world/cm4/armgcc/build_debug.bat  
  
<install_dir>/boards/evkmimxrt1170/multicore_examples/hello_world/cm7/armgcc/  
build_flexspi_nor_debug.bat
```

Build both applications separately following steps for single core examples as described in [Build an example application](#).



```
GNU GCC Command Prompt - build_debug.bat  
[ 55%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/serial_ma  
nager/serial_manager.c.obj  
[ 59%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/serial_ma  
nager/serial_port_uart.c.obj  
[ 62%] Building ASM object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1176  
/gcc/startup_MIMXRT1176_cm4.S.obj  
[ 66%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/lists/gen  
eric_list.c.obj  
[ 70%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1176/u  
tilities/fsl_assert.c.obj  
[ 74%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1176/d  
rivers/fsl_gpio.c.obj  
[ 77%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1176/x  
ip/fsl_flexspi_nor_boot.c.obj  
[ 81%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/boards/evkmimxrt1170  
/xip/evkmimxrt1170_sdram_ini_dcd.c.obj  
[ 85%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/boards/evkmimxrt1170  
/xip/evkmimxrt1170_flexspi_nor_config.c.obj  
[ 88%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/middleware/multicore  
/mcmgr/src/mcmgr.c.obj  
[ 92%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1176/d  
rivers/fsl_anatop.c.obj  
[ 96%] Building C object CMakeFiles/hello_world_cm4.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1176/d  
rivers/fsl_anatop_ai.c.obj  
[100%] Linking C executable debug\hello_world_cm4.elf  
[100%] Built target hello_world_cm4.elf  
  
c:\p\SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2\boards\evkmimxrt1170\multicore_examples\hello_world\cm4\armgcc>IF "" == "" (paus  
e )  
Press any key to continue . . .
```

Figure 55. hello_world_cm4 example build successful

```

GCC Command Prompt - build_flexspi_nor_debug.bat

[ 57%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/uart/lpuart_adapter.c.obj
[ 60%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/serial_manager/serial_manager.c.obj
[ 64%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/serial_manager/serial_port_uart.c.obj
[ 67%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/components/lists/generic_list.c.obj
[ 71%] Building ASM object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1170/gcc/startup_MIMXRT1170_cm7.S.obj
[ 75%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1170/utilities/fsl_assert.c.obj
[ 78%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1170/drivers/fsl_gpio.c.obj
[ 85%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/boards/evkmimxrt1170/xip/evkmimxrt1170_flexspi_nor_config.c.obj
[ 85%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1170/xip/fsl_flexspi_nor_boot.c.obj
[ 89%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/boards/evkmimxrt1170/xip/evkmimxrt1170_sdram_ini_dcd.c.obj
[ 92%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1170/drivers/fsl_anatop_ai.c.obj
[ 96%] Building C object CMakeFiles/hello_world_cm7.elf.dir/C:/p/SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2/devices/MIMXRT1170/drivers/fsl_anatop.c.obj
[100%] Linking C executable flexspi_nor_debug\hello_world_cm7.elf
[100%] Built target hello_world_cm7.elf

c:\p\SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2\boards\evkmimxrt1170\multicore_examples\hello_world\cm7\armgcc>IF "" == "" (pause)
Press any key to continue . . .

```

Figure 56. hello_world_cm7 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 SPI 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 SPI 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.

```

c:\p\SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2\boards\evkmimxrt1170\multicore_examples\hello_world\cm7\armgcc>cd flexspi_nor_debug

c:\p\SDK_2.6.0_MIMXRT1170-EVK_PRC_1_RC2\boards\evkmimxrt1170\multicore_examples\hello_world\cm7\armgcc>flexspi_nor_debug
>arm-none-eabi-gdb.exe hello_world_cm7.elf
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2019-q3-update\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a
path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 8-2019-q3-update) 8.3.0.20190703-git
Copyright (C) 2019 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_cm7.elf...
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x30004ccc in SCB_EnableDCache () at c:\p\sdk_2.6.0_mimxrt1170-evk_prc_1_rc2\cmsis\include\core_cm7.h:2319
2319      SCB->CCR |= (uint32_t)SCB_CCR_DC_Msk; /* enable D-Cache */
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .flash_config, size 0x200 lma 0x30000400
Loading section .ivt, size 0x30 lma 0x30001000
Loading section .interrupts, size 0x400 lma 0x30002000
Loading section .text, size 0x4f50 lma 0x30002400
Loading section .ARM, size 0x8 lma 0x30007350
Loading section .init_array, size 0x4 lma 0x30007358
Loading section .fini_array, size 0x4 lma 0x3000735c
Loading section .data, size 0x64 lma 0x30007360
Loading section .m0code, size 0x42f4 lma 0x30fc0000
Start address 0x300024b4, load size 39144
Transfer rate: 910 KB/sec, 3558 bytes/write.
(gdb) monitor go
(gdb)

```

Figure 57. Loading and running the multicore example

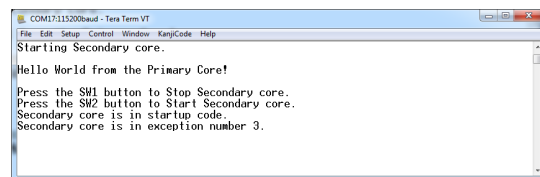







Figure 58. Hello World from primary core message

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.

Table 1 describes the tools included in the MCUXpresso Config Tools.

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	
TEE tool	Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.	
Device Configuration tool	Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to setup various on-chip peripherals prior the program launch.	

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/mcuxpresso. Recommended for customers using IAR Embedded Workbench, Keil MDK μ Vision, 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.

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.

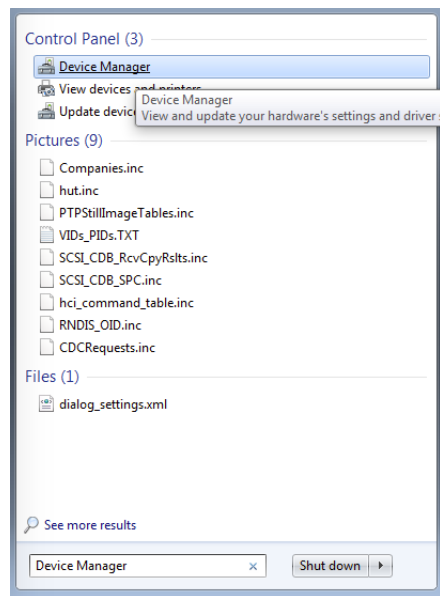


Figure A-1. Device Manager

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.

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 SDK

Hardware platform	Default interface	OpenSDA details
EVK-MC56F83000	P&E Micro OSJTAG	N/A
EVK-MIMXRT595	CMSIS-DAP	N/A
EVK-MIMXRT685	CMSIS-DAP	N/A
FRDM-K22F	CMSIS-DAP/mbd/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/mbd/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/mbd/DAPLink	OpenSDA v2.2
FRDM-KL02Z	P&E Micro OpenSDA	OpenSDA v1.0

Table continues on the next page...

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

Hardware platform	Default interface	OpenSDA details
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/mbd/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbd/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
MIMXRT1170-EVK	CMSIS-DAP	N/A
TWR-K21D50M	P&E Micro OSJTAG	N/A OpenSDA v2.0
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0

Table continues on the next page...

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

Hardware platform	Default interface	OpenSDA details
TWR-K24F120M	CMSIS-DAP/mbd	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
TWR-KM35Z75M	DAPLink	OpenSDA v2.2
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 How to add or remove boot header for XIP targets

The MCUXpresso SDK for i.MX RT1170 provides `flexspi_nor_debug` and `flexspi_nor_release` targets for each example and/or demo which supports XIP (eXecute-In-Place). These two targets add `XIP_BOOT_HEADER` to the image by default. Because of this, ROM can boot and run this image directly on external flash.

Macros for the boot leader:

- The following three macros are added in `flexspi_nor` targets to support XIP, as described in [Table C-1](#).

Table C-1. Macros added in `flexspi_nor`

XIP_EXTERNAL_FLASH	1: Exclude the code which changes the clock of FLEXSPI.
	0: Make no changes.

Table continues on the next page...

**Table C-1. Macros added in flexspi_nor
(continued)**

XIP_BOOT_HEADER_ENABLE	1: Add FLEXSPI configuration block, image vector table, boot data, and device configuration data (optional) to the image by default.
	0: Add nothing to the image by default.
XIP_BOOT_HEADER_DCD_ENABLE	1: Add device configuration data to the image.
	0: Do NOT add device configuration data to the image.

- [Table C-2](#) shows the different effect on the built image with a different combination of these macros.

Table C-2. Effects on built image with different macros

		XIP_BOOT_HEADER_DCD_ENABLE=1	XIP_BOOT_HEADER_DCD_ENABLE=0
XIP_EXTERNAL_FLASH=1	XIP_BOOT_HEADER_ENABLE=1	<ul style="list-style-type: none"> • Can be programmed to qspi flash by IDE and can run after POR reset if qspi flash is the boot source. • SDRAM will be initialized. 	<ul style="list-style-type: none"> • Can be programmed to qspi flash by IDE, and can run after POR reset if qspi flash is the boot source. • SDRAM will NOT be initialized.
	XIP_BOOT_HEADER_ENABLE=0	<ul style="list-style-type: none"> • CANNOT run after POR reset if it is programmed by IDE, even if qspi flash is the boot source. 	
XIP_EXTERNAL_FLASH=0		<ul style="list-style-type: none"> • This image CANNOT complete XIP because when this macro is set to 1, it excludes the code, which changes the clock for FLEXSPI. 	

Where to change the macros for each toolchain in MCUXpresso SDK?

Take hello_world as an example:

- **IAR**

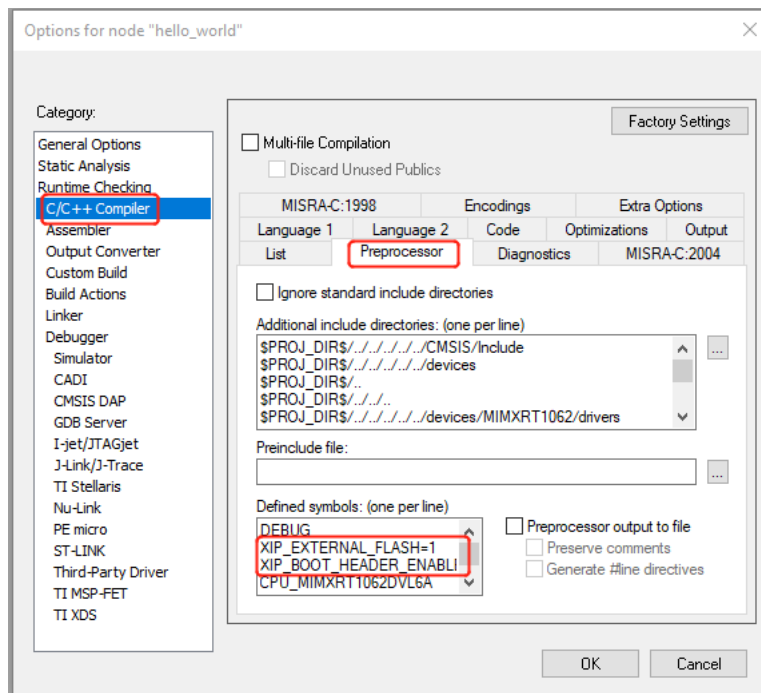


Figure C-1. Options node IAR

- MDK

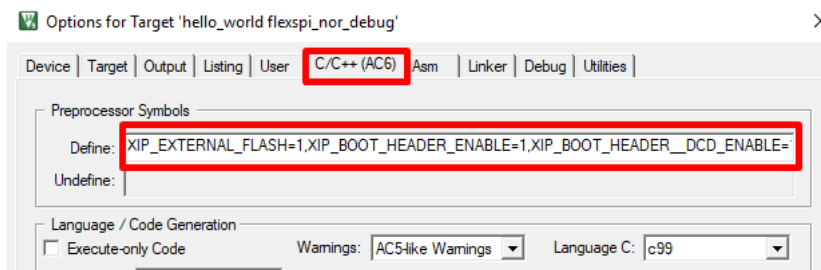


Figure C-2. Options for target

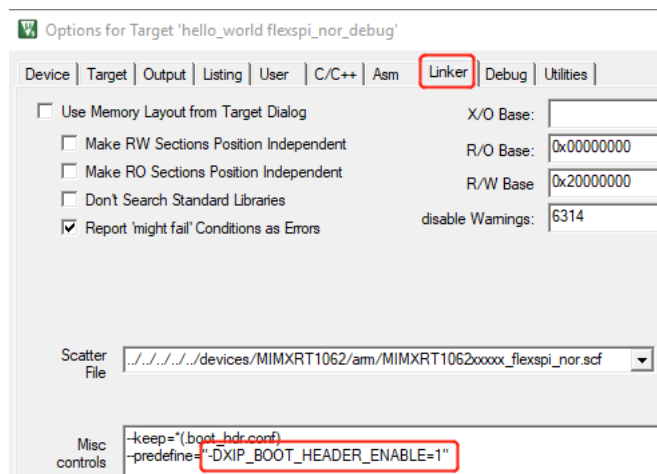


Figure C-3. Change configuration Misc controls

- ARMGCC

Change the configuration in CMakeLists.txt.

```

SET(CMAKE_C_FLAGS_SDRAM_RELEASE "${CMAKE_C_FLAGS_SDRAM_RELEASE} -std=gnu99")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DXIP_EXTERNAL_FLASH=1")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DXIP_BOOT_HEADER_ENABLE=1")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DXIP_BOOT_HEADER_DCD_ENABLE=1")

SET(CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG "${CMAKE_C_FLAGS_FLEXSPI_NOR_DEBUG} -DCPU_MIMXRT1052DVL6A")

```

Figure C-4. Change configuration CMakeLists.txt

- MCUX

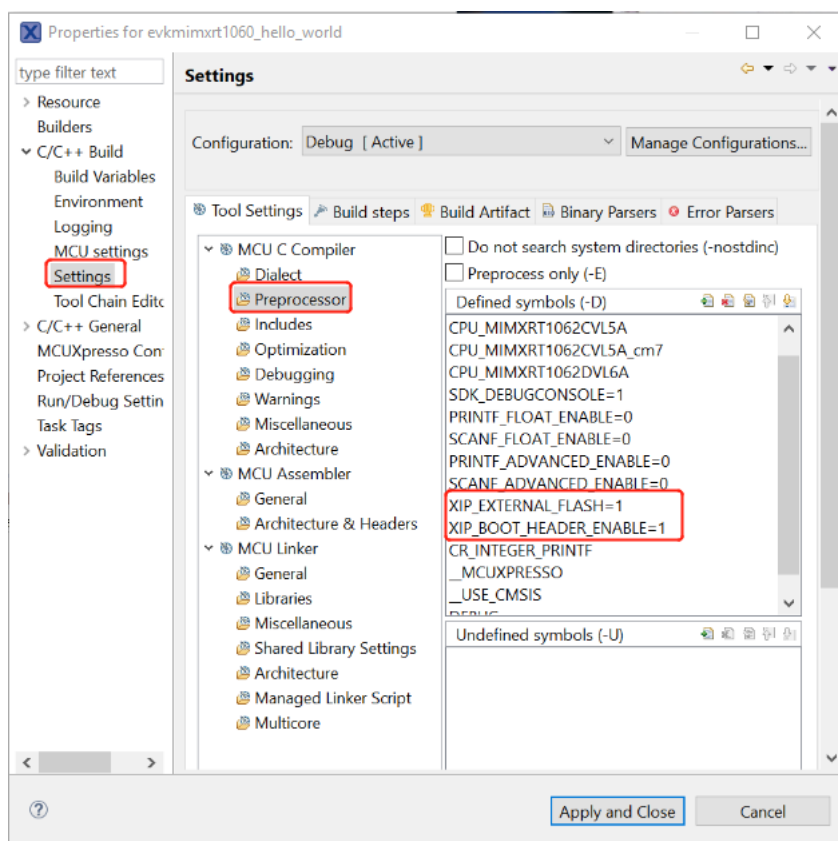


Figure C-5. Properties for evkmimxrt1170

How to Reach Us:**Home Page:**nxp.com**Web Support:**nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μ Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2020 NXP B.V.

Document Number MCUXSDKMIMXRT117XGSUG
Revision 0, 08/2020

