# MCUXSDKMIMXRT117XBGSUG

## Getting Started with MCUXpresso SDK for MIMXRT1170-EVKB

Rev. 0 — 31 December 2022 User guide

**Document information**

| Information | Content |
|---|---|
| Keywords | Getting Started, MCUXpresso SDK, MIMXRT1170-EVKB |
| Abstract | The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for Kinetis and LPC Microcontrollers. |

# 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-EVKB* (document MCUXSDKMIMXRT117XBRN).

For more details about MCUXpresso SDK, see MCUXpresso Software Development Kit (SDK).

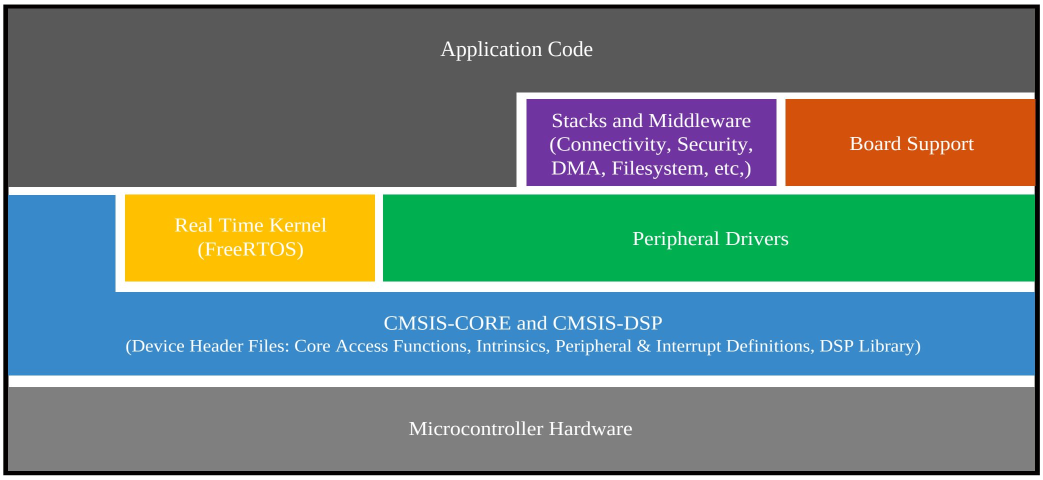


**Figure 1. MCUXpresso SDK layers**

# 2 MCUXpresso SDK board support package folders

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

- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).

- `rtos_examples`: Basic FreeRTOSTM OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers.
- `usb_examples`: Applications that use the USB host/device/OTG stack.
- `multicore_examples`: Applications for both cores showing the usage of multicore software components and the interaction between cores.
- `Other_examples`: See detail in package *boards/evkbmimxrt1170*.

## 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:

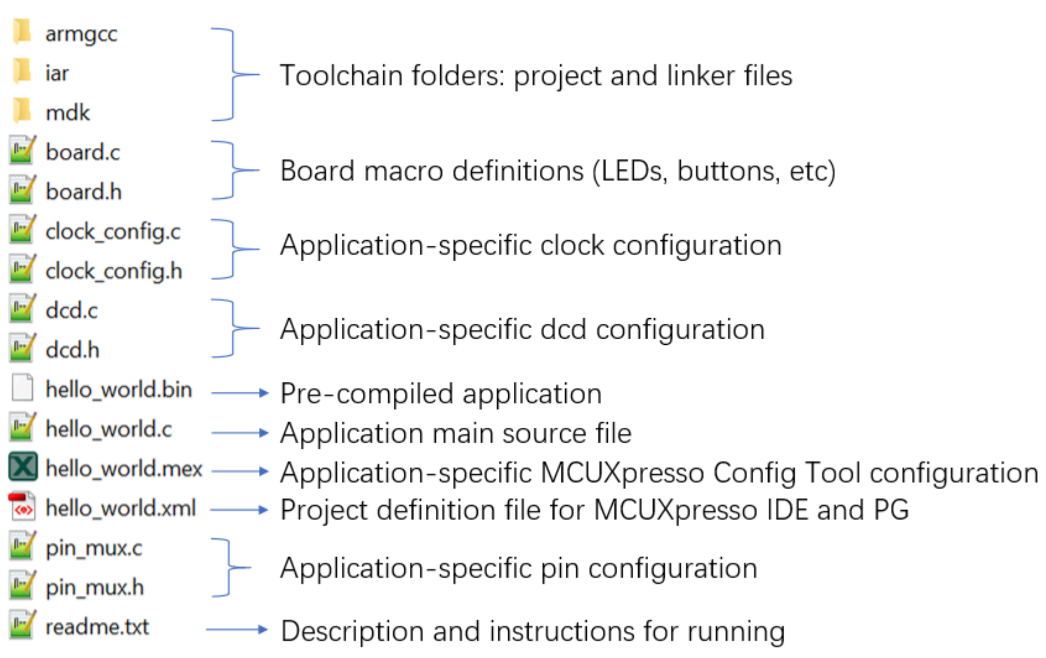


**Figure 2. Application folder structure**

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/<devices_name>/project_template` 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:***

*Most MCUXpresso projects provide two targets (debug and release). For CM7 projects, they are actually flash target. For CM4 projects, they are linked to RAM. To debug and run the CM7 examples, set **SW1[1:4]** to **0010** as internal flash boot mode. Currently, MCUXpresso IDE does not support CM4 download/debug.*

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-EVKB 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.

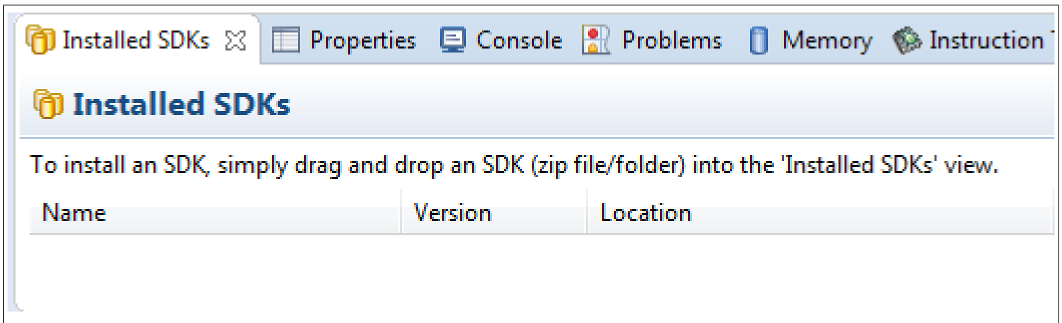


Figure 3. Install an SDK

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 **evkbmimxrt1170** and click **Next**.

 

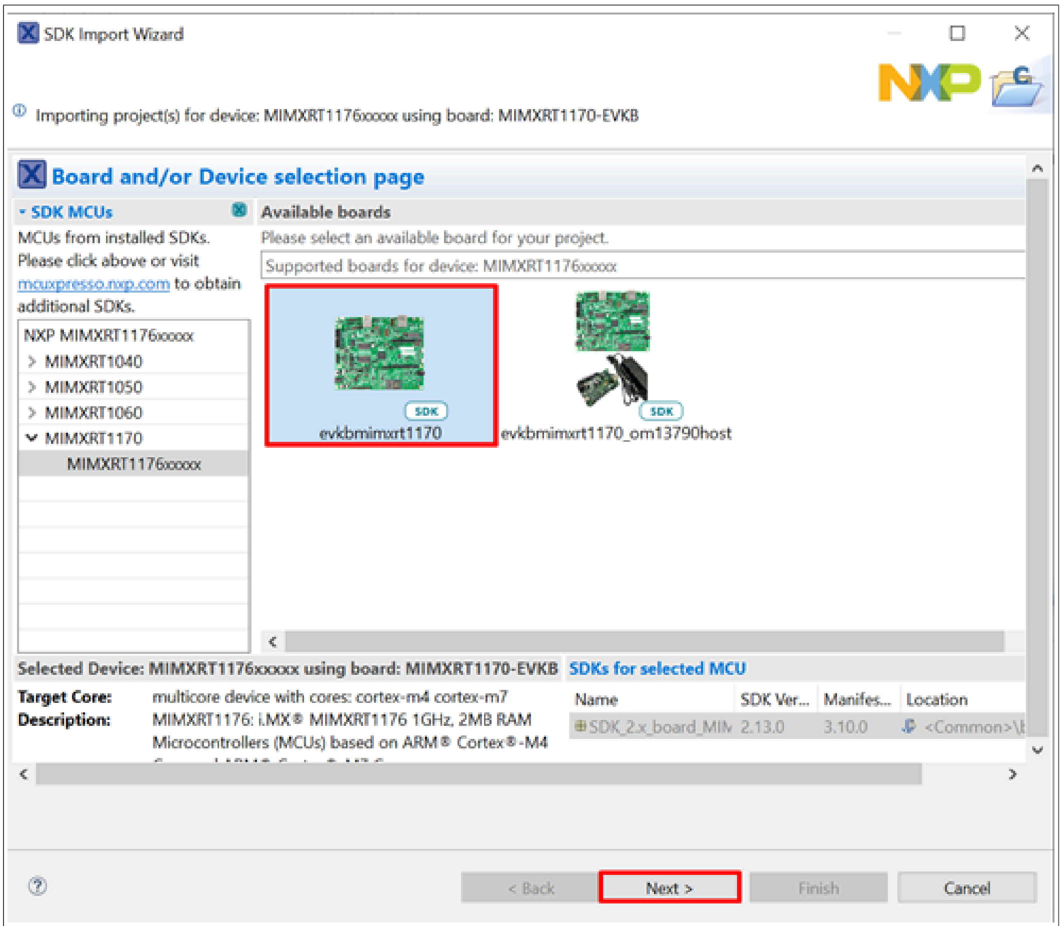


**Figure 5. Select MIMXRT1170-EVKB board**

4. Expand the *demo_apps* folder and select `hello_world`. Then, click **Next**.

 

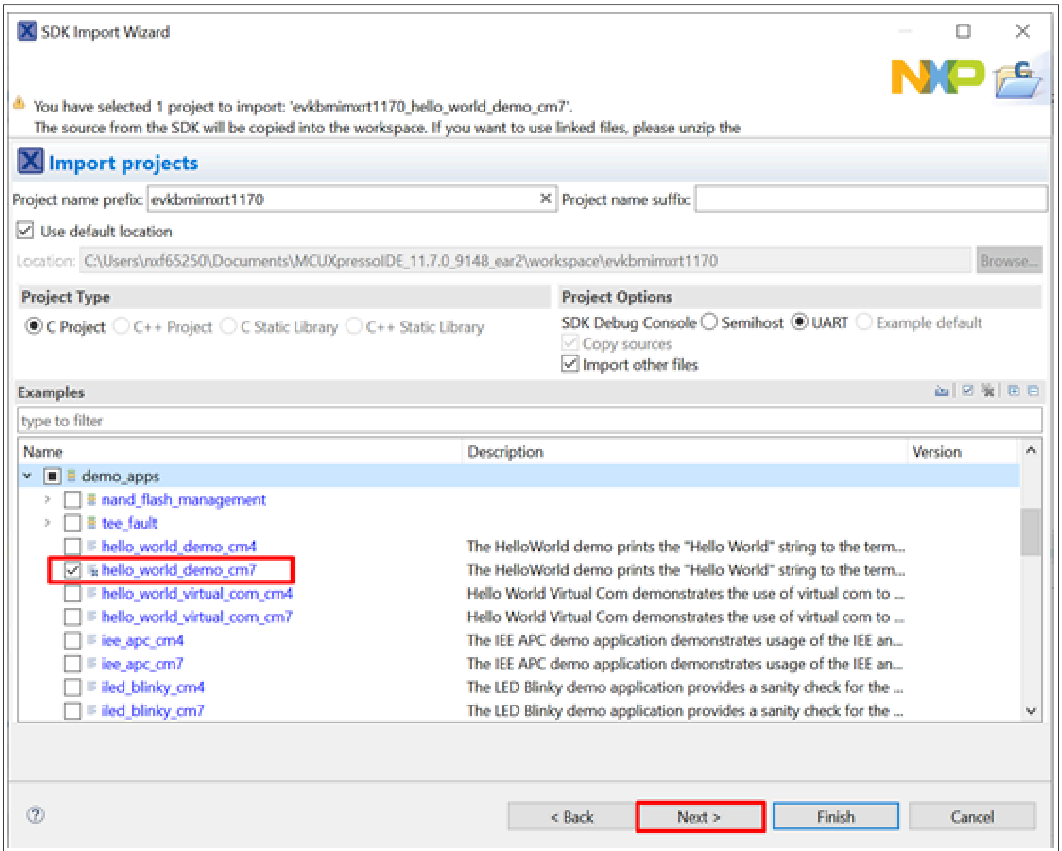


**Figure 6. Select `hello_world`**

5. 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**.

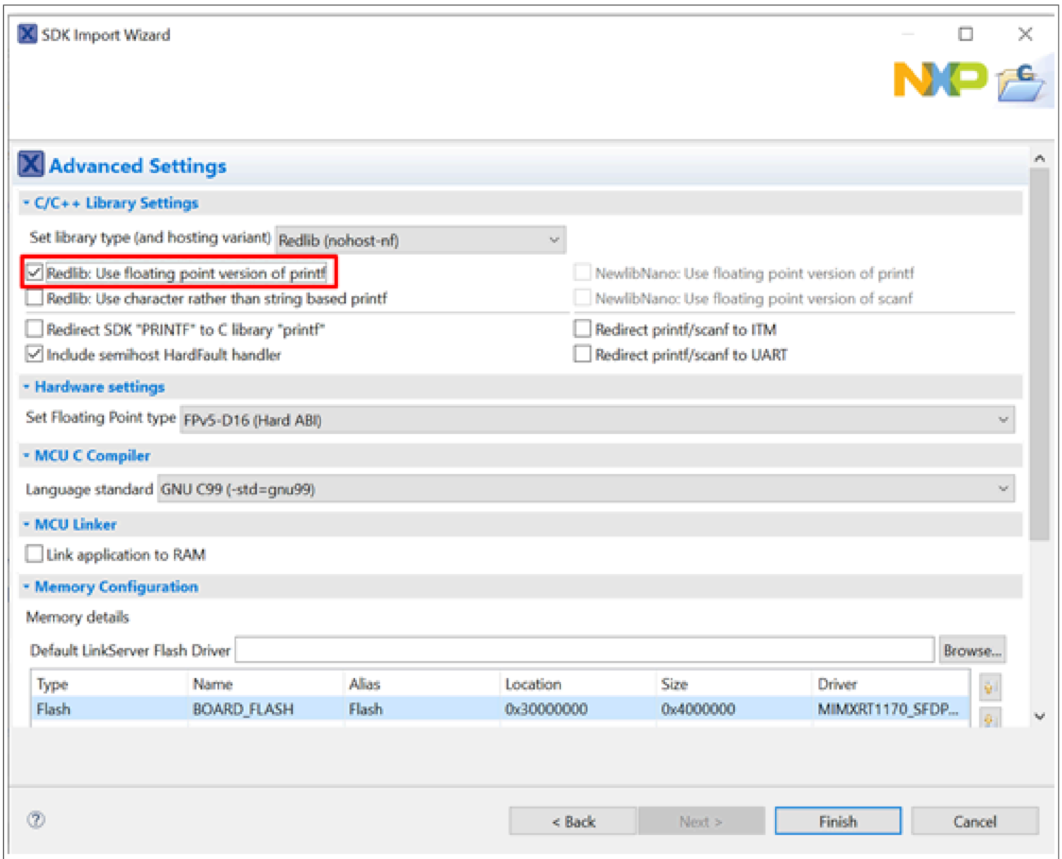


Figure 7. Select `Use floating point version of printf`

## 3.3 Run an example application

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

1. See Table 2 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 SEGGER.
   - To use J-Link for debug, remove J5, J6, J7, J8 jumpers and attach external J-Link on J1.
   - For boards with the OSJTAG interface, install the driver from KEIL.
2. Connect the development platform to your PC via a USB cable.
3. Connect USB cable between J11 on EVKB and PC USB port.
4. Open the terminal application on the PC, such as PuTTY or TeraTerm, J11 on EVKB, and connect to the debug serial port number. To determine the COM port number, see Section 8. Configure the terminal with these settings:
   a. 115200 baud rate 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

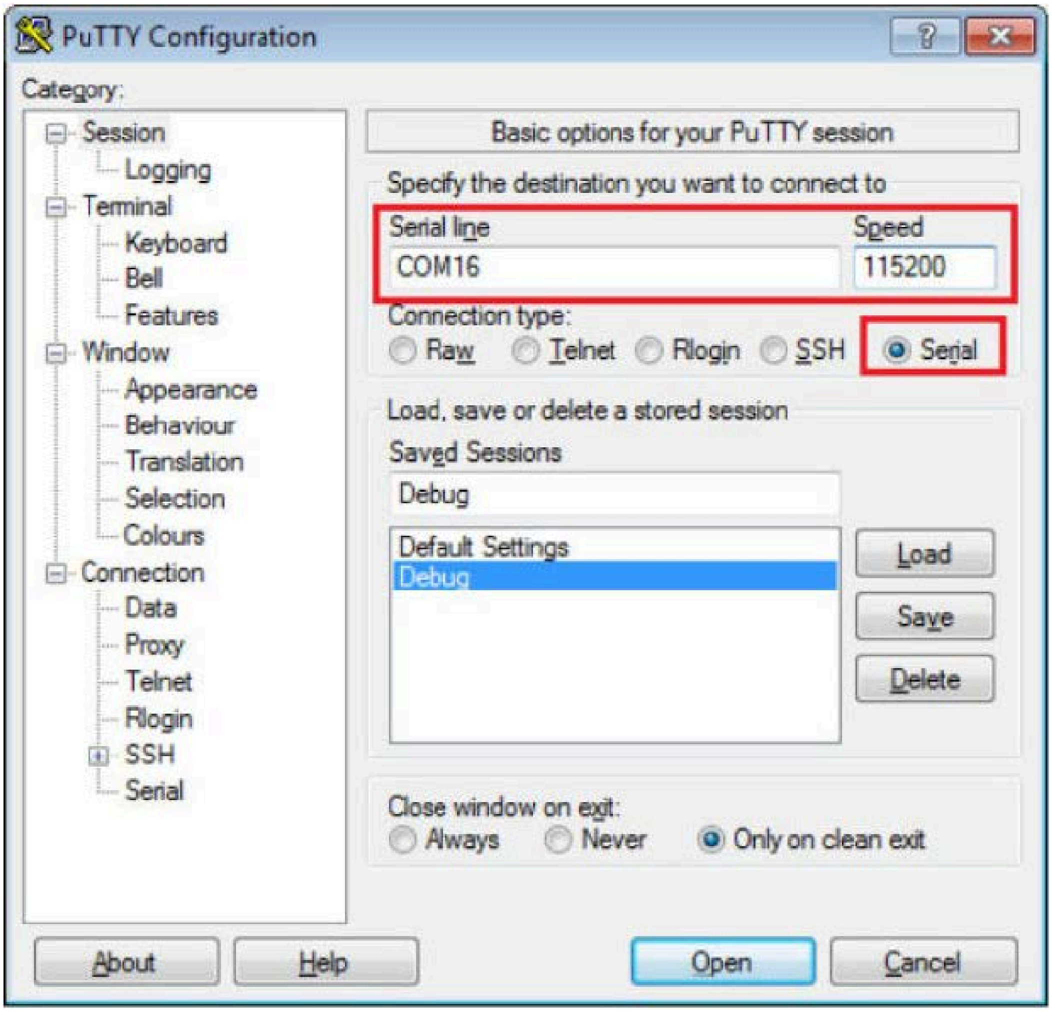


**Figure 8. Terminal (PuTTY) configurations**

5. On the **Quickstart Panel**, click **Debug 'evkbmimxrt1170_demo_apps_hello_world' [Debug]**.

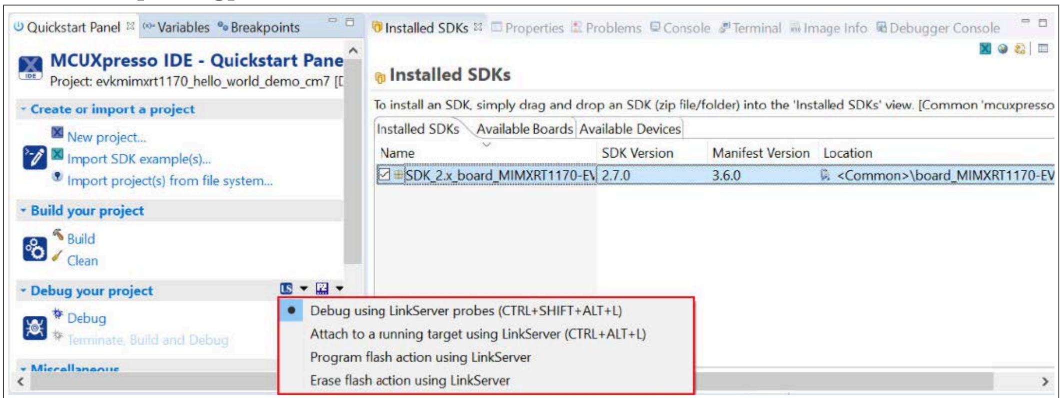


**Figure 9. Debug `hello_world` case**

6. 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.)

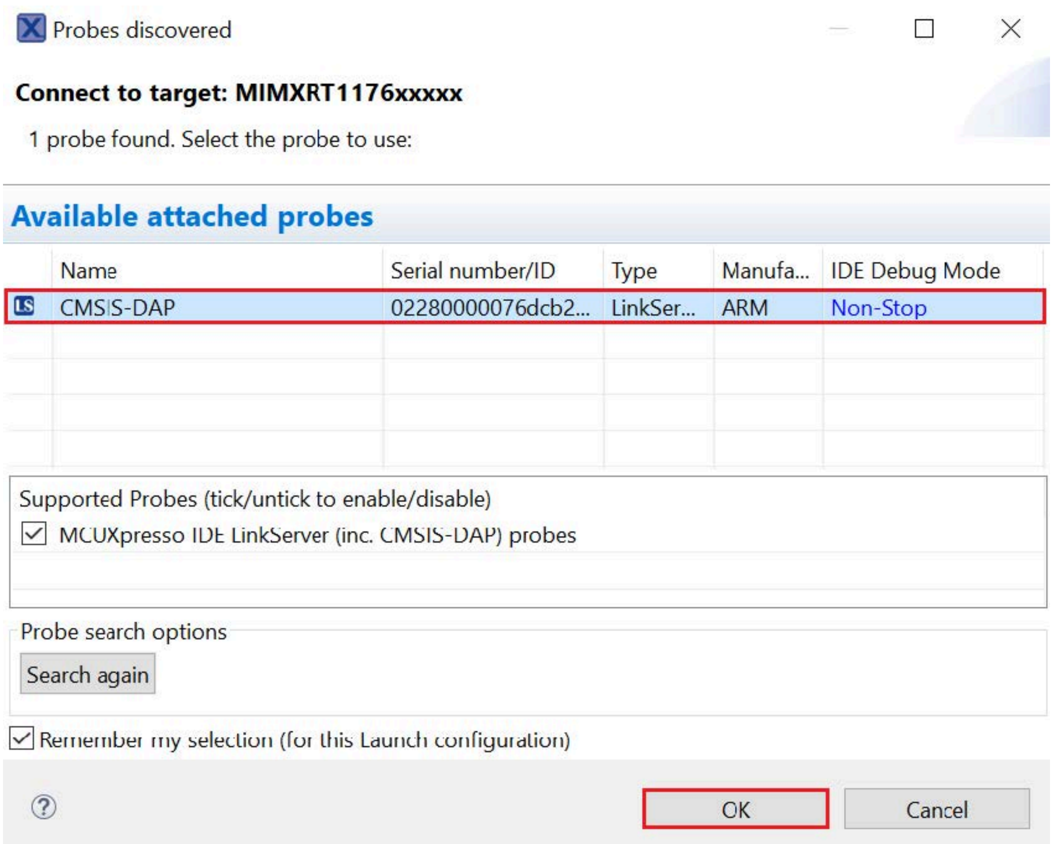


**Figure 10. Attached Probes: debug emulator selection**

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

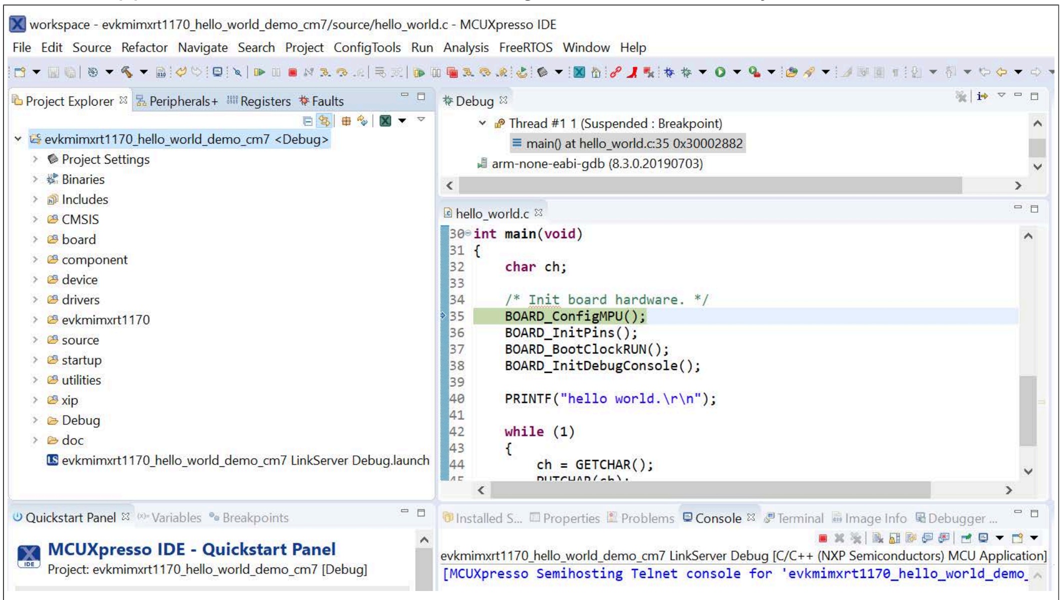


**Figure 11. Stop at `main()` when running debugging**

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


**Figure 12. Resume button**

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

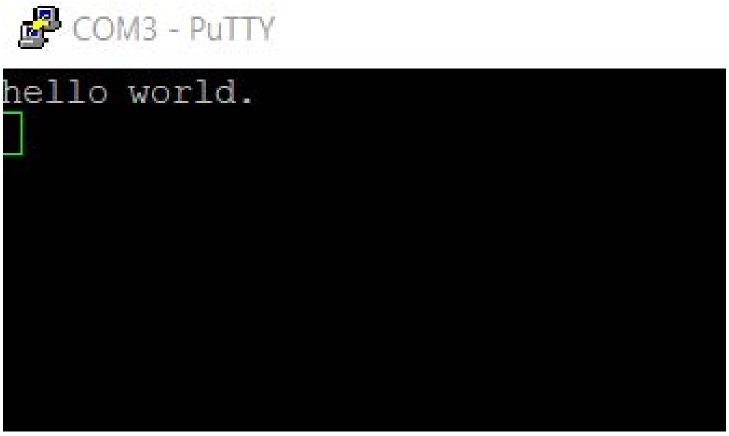


**Figure 13. Text display of the `hello_world` demo**

## 3.4 Build a multicore example application

This section describes the steps required to configure MCUXpresso IDE 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 **evkbmimxrt1170** 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 Section 3.2. When the SDK zip package for **evkbmimxrt1170** 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 **evkbmimxrt1170** and click **Next**.

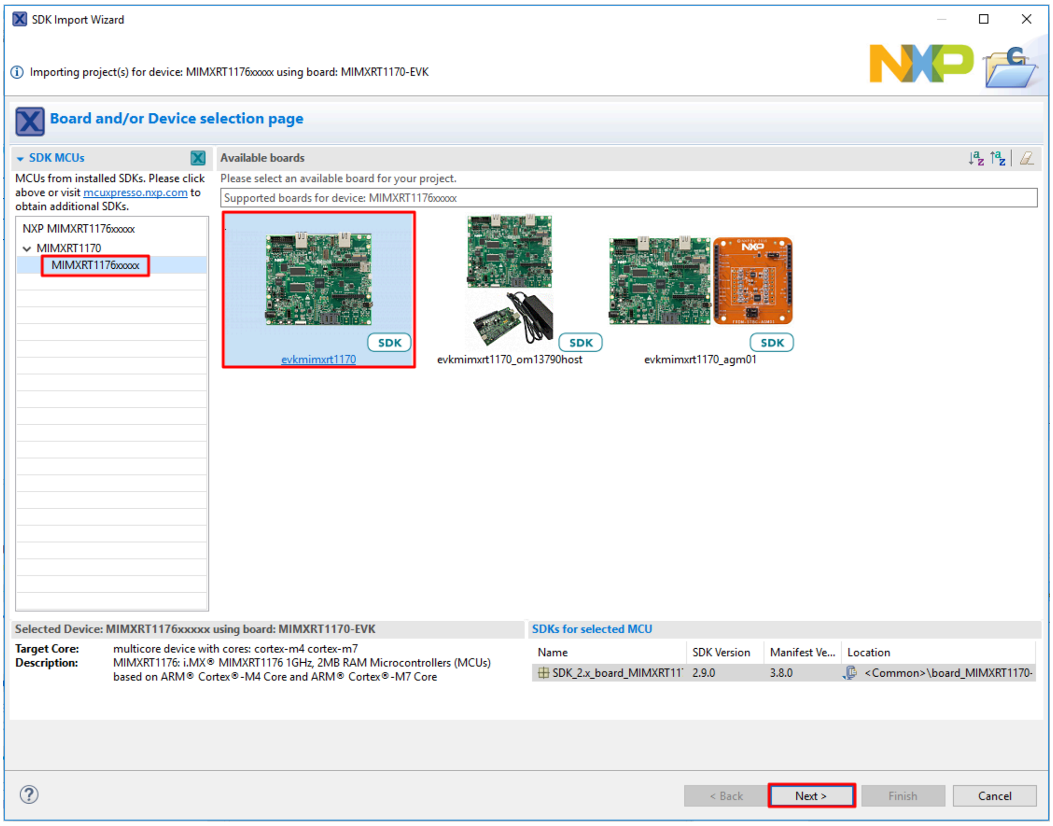


**Figure 14. Select the evkbmimxrt1170 board**

2. 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**.

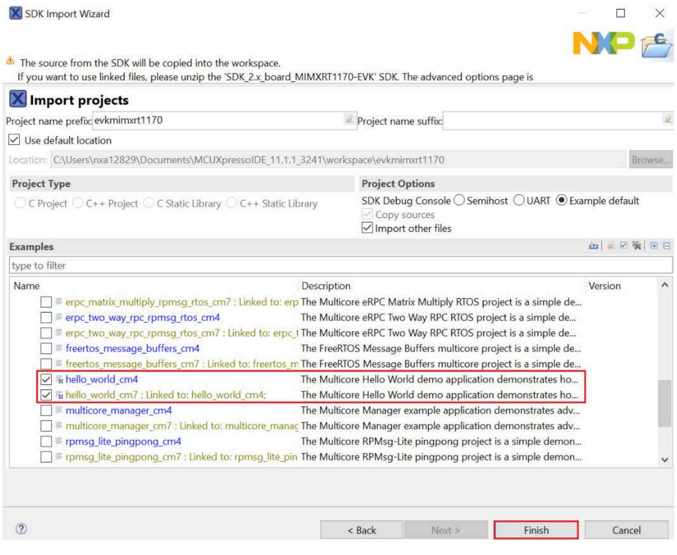


**Figure 15. Select the hello_world multicore example**

3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `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 16. For this example, select **Debug**.

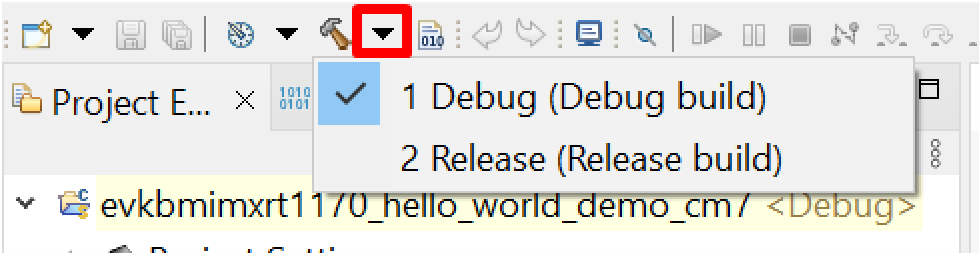


Figure 16.  Selection of the build target in MCUXpresso IDE

Press the **Build** button to start the multi-core project build.. Because of the project reference settings in multicore projects, triggering the build of the primary core application (`cm7`) also makes 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.*

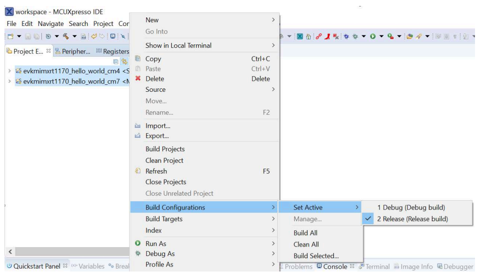

*Figure 17.  Switching multicore projects into the Release build configuration*

## 3.5 Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform all steps as described in Section 3.3. 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 Figure 18 to Figure 20 as reference.

***Note:*** *On MCUXpresso IDE, the feature to simultaneously debug two cores is only supported by CMSIS-DAP debugger.*

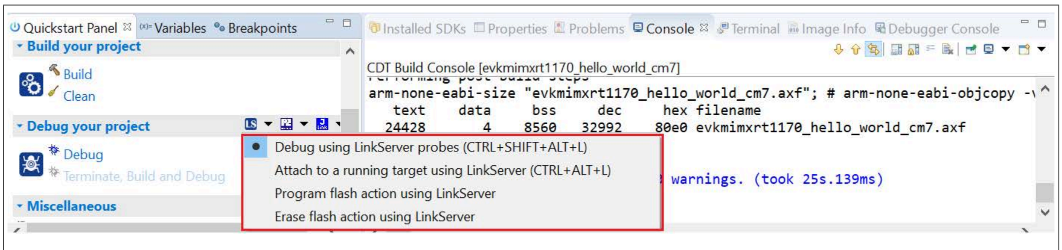


**Figure 18. Debug hello_world_cm7 case**

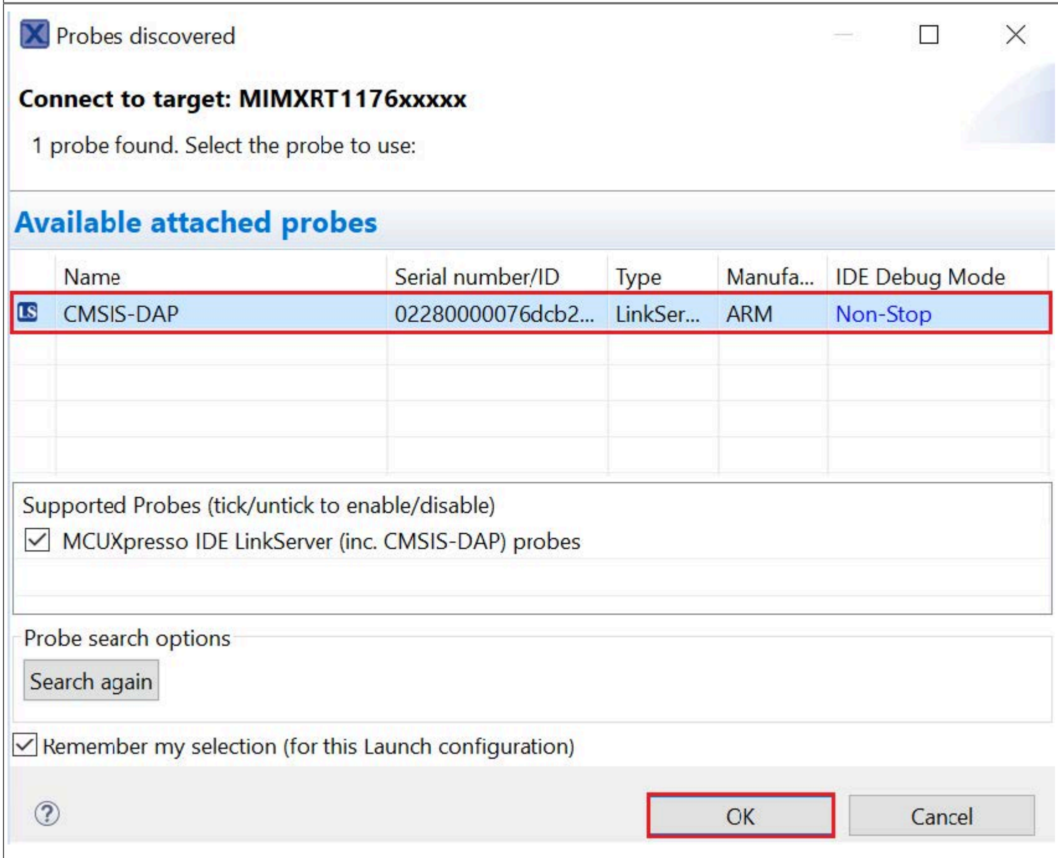


**Figure 19. Attached Probes: debug emulator selection**

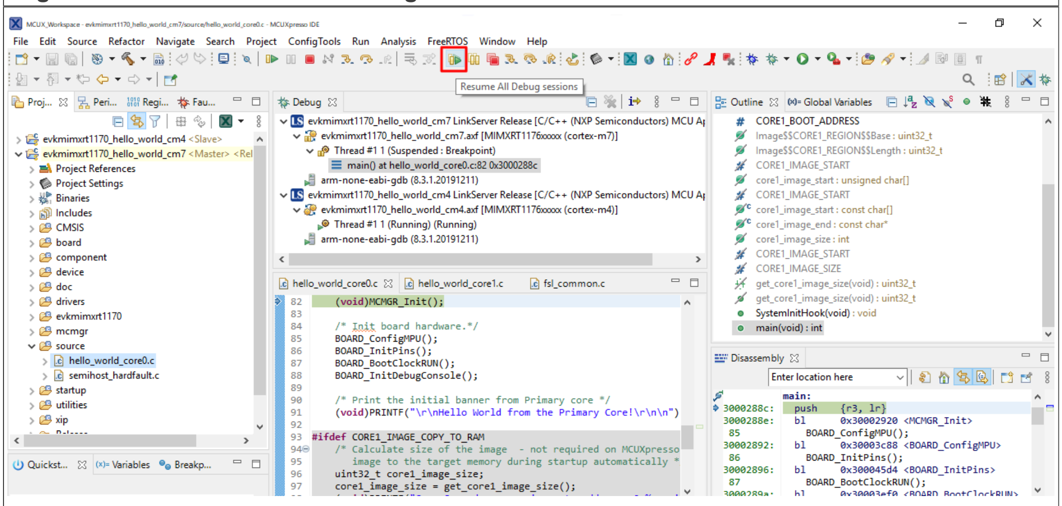

**Figure 20. Stop the primary core application at main() when running debugging**

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.

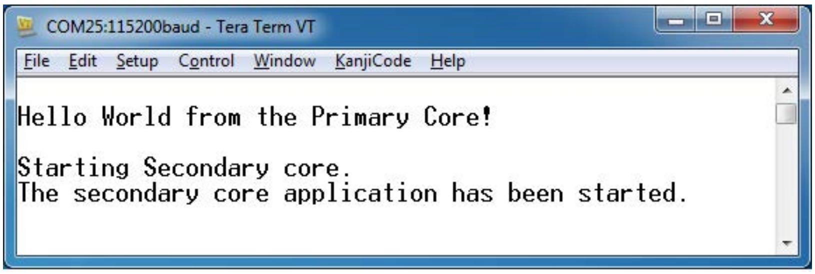


**Figure 21. Hello World from the primary core message**

***Note:*** *There are some limitations on MCUXpresso IDE debugging. For details, see* ***Section 8.5 MCUXPresso IDE limitation*** *in MCUXpresso SDK Release Notes for MIMXRT1170-EVKB (document MCUXSDKMIMXRT117XBRN).*

# 4 Run a demo application using IAR

***Note:***

*When erasing flash on IAR, IAR will show all range that can connect to flash. Please only check address flash connect to practically. Take the evkbmimxrt1170 for example:*

- *M7: 0x30000000-0x3fffffff*
- *M4: 0x8000000-0x17ffffff*

*When using IAR download/debug `flexspi_nor` related targets, make sure the boot switch is put to internal flash boot mode* ***SW1[1:4]:0010***

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-EVKB 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` demo application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:
   *<install_dir>/boards/<board_name>/<example_type>/<application_name>/<core_type>/iar*
   Using the MIMXRT1170-EVKB hardware platform as an example, the `hello_world` workspace is located in:
   *<install_dir>/boards/evkbmimxrt1170/demo_apps/hello_world/cm7/iar/hello_world_demo_cm7.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 across 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**.

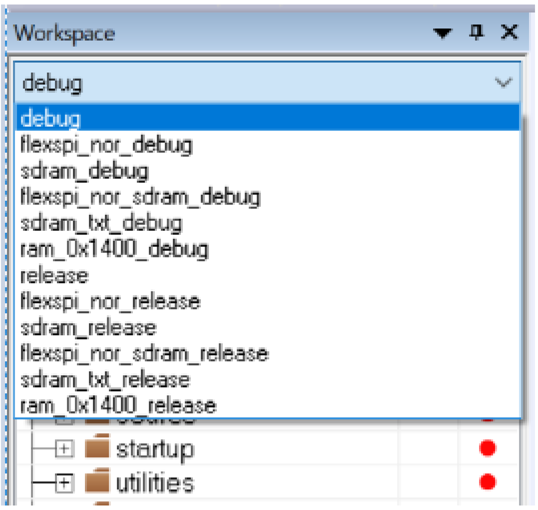


**Figure 22. Demo build target selection**

3. To build the demo application, click **Make**, highlighted in red in [Figure 23](#).

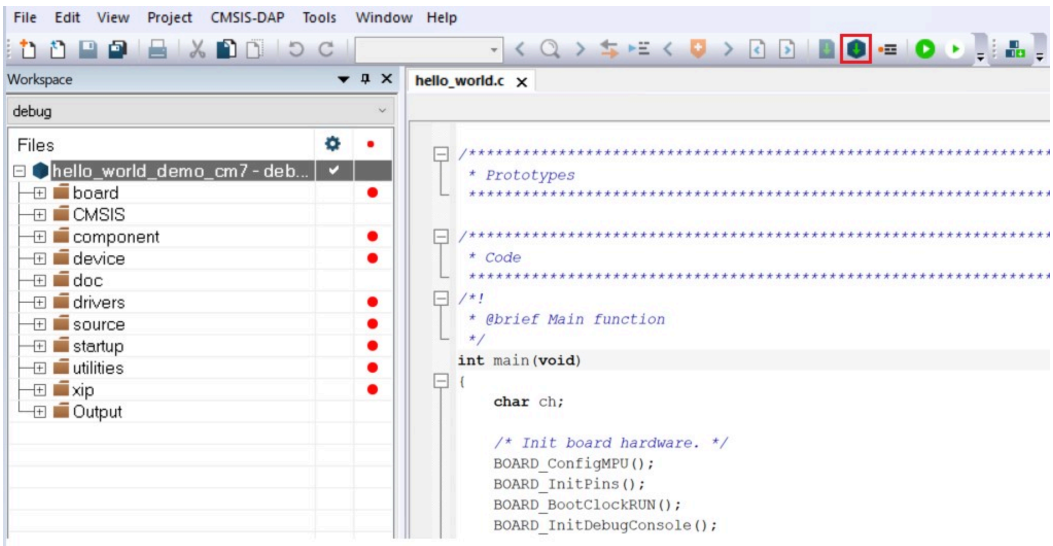


**Figure 23. Build the demo application**

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 [MBED](#) 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 [Section 8](#)). 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

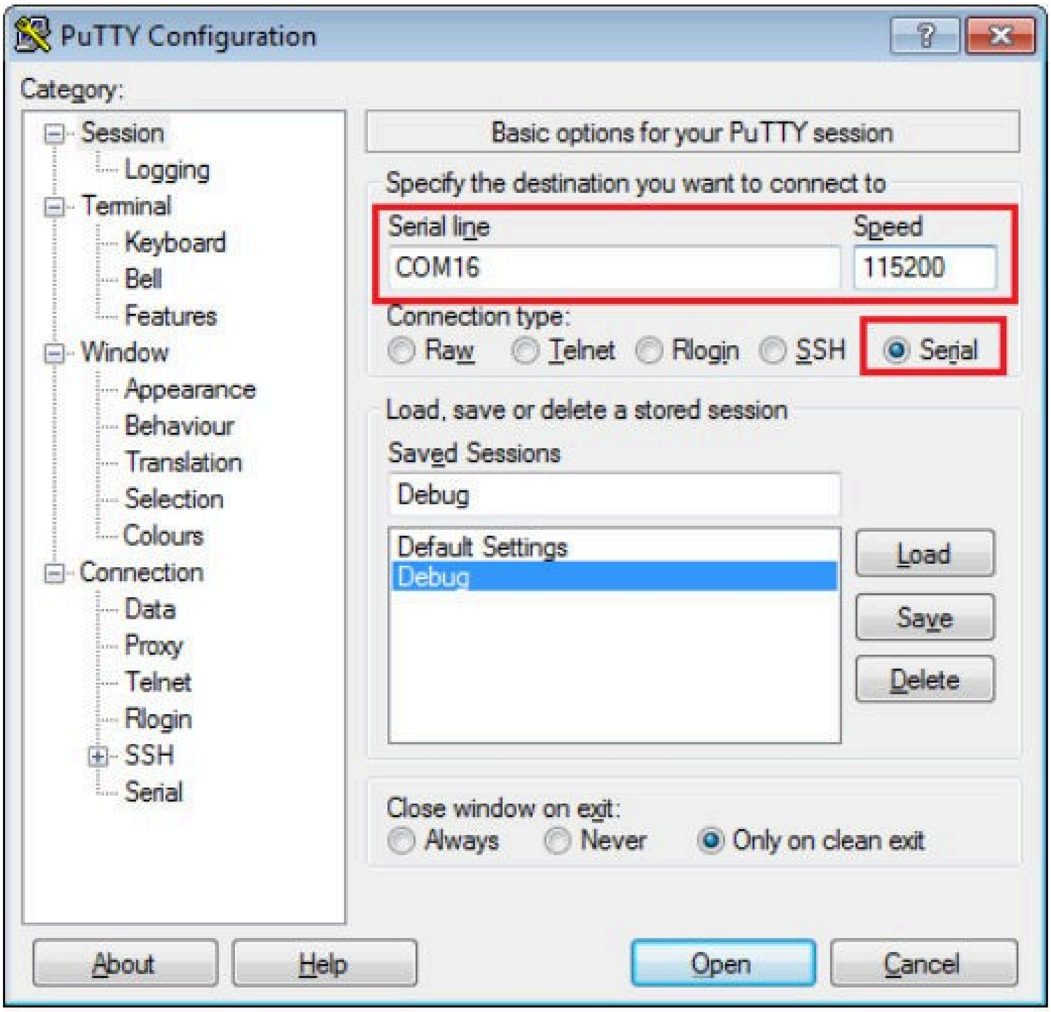


**Figure 24. Terminal (PuTTY) configuration**

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

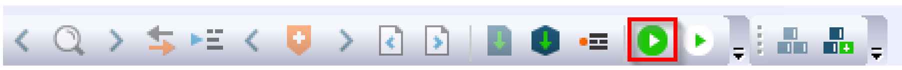

**Figure 25. Download and Debug button**

   - When using CMSIS-DAP to debug cm4 project on IAR, an extra option need to be specified in debugger settings. Check and fill in **--macro_param enable_core=1** in **Debugger** -> **Extra Options** -> **Command line options**, as shown in Figure 26.

 

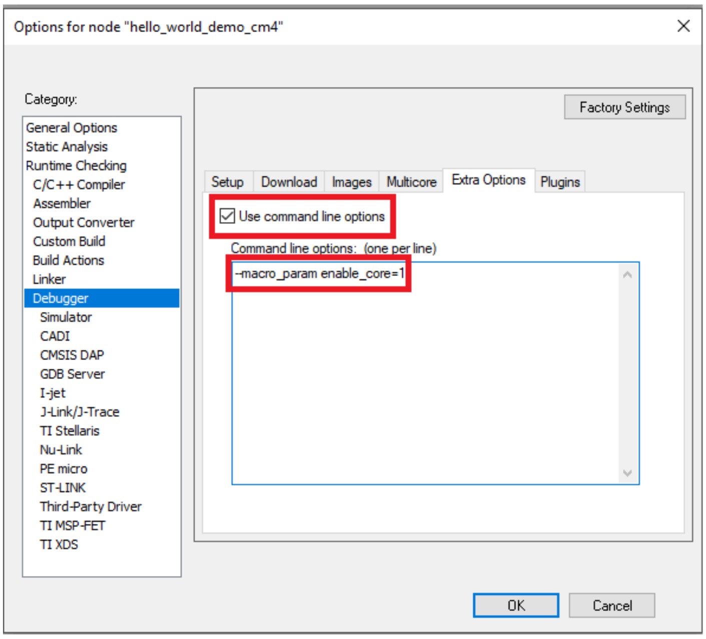


Figure 26. Selecting Command line options

- If debug with JLINK as probe, `jlinkscript` file is needed.
  - When downloading the `cm7` project, check **Use command line options**, as shown in Figure 27.

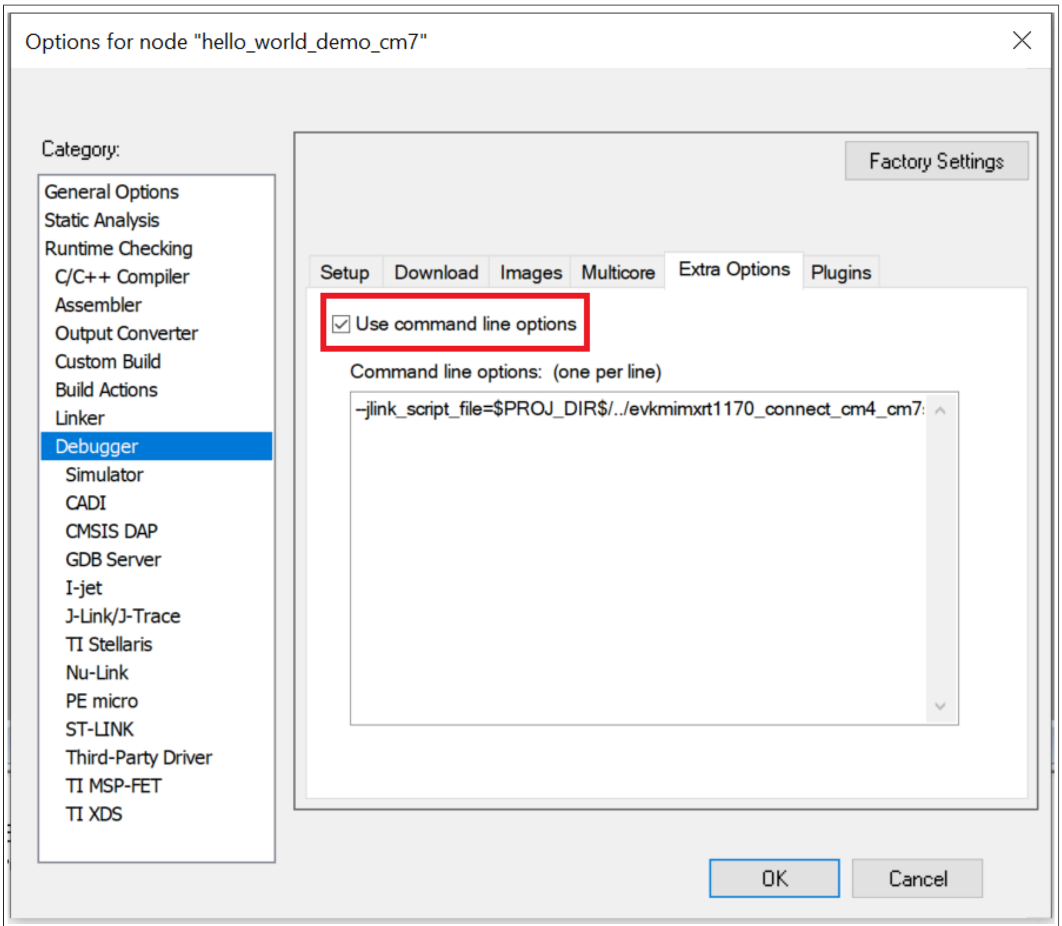


**Figure 27. Check Use command line options**

– When downloading the `cm4` project, uncheck **Use flash loaders**, as shown in Figure 28, and change the contents of command line options as below:
  – Target with SDRAM

```
_ --jlink_script_file=$PROJ_DIR$/../
evkbmimxrt1170_connect_cm4_cm4side_sdram.jlinkscript
```

  – Other target

```
--jlink_script_file=$PROJ_DIR$/../
evkbmimxrt1170_connect_cm4_cm4side.jlinkscript
```

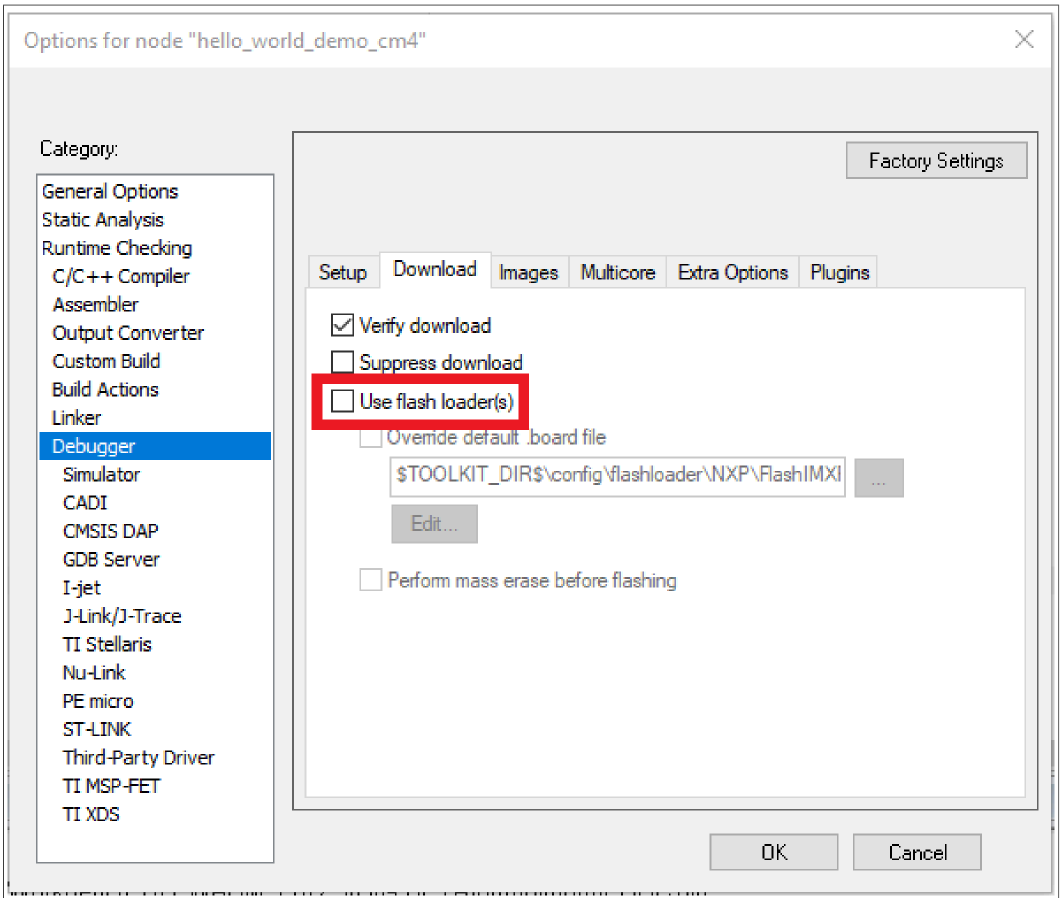


Figure 28. Uncheck Use flash loader(s)

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

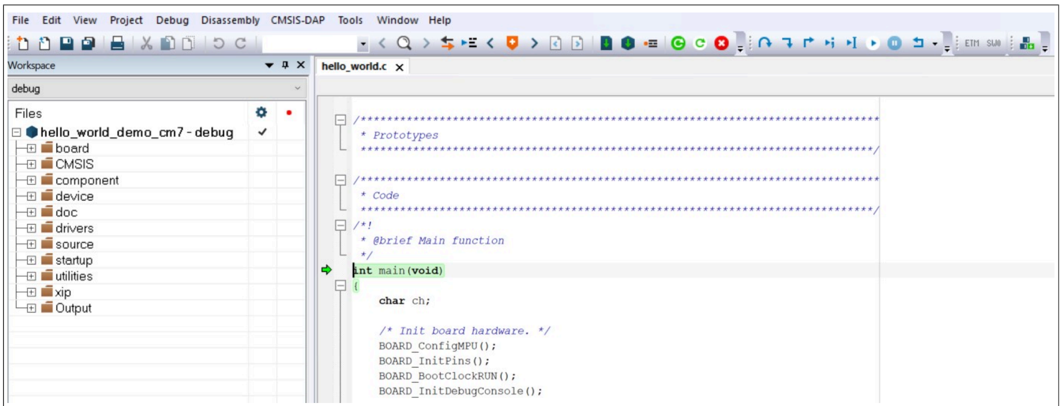


Figure 29. Stop at main() when running debugging

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

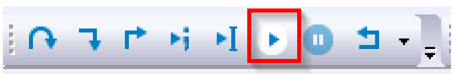

Figure 30. Go button

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.

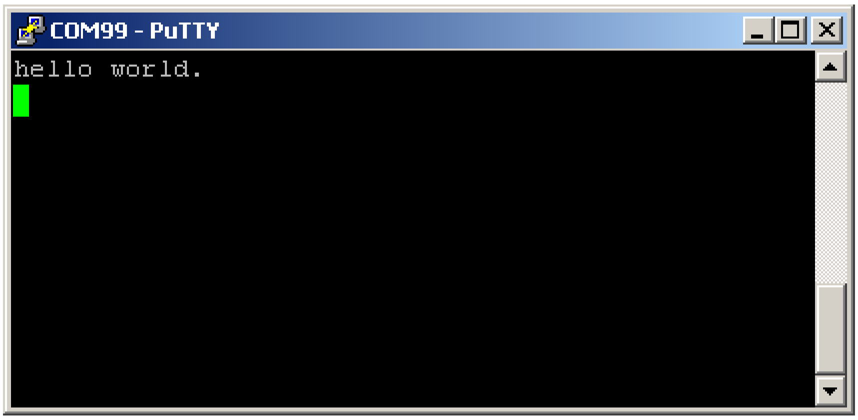


**Figure 31. Text display of the hello_world demo**

***Note:*** *There are some limitations on MCUXpresso IDE debugging. For details, see* ***Section 8.6 IAR debug limitation*** *in MCUXpresso SDK Release Notes for MIMXRT1170-EVKB (document MCUXSDKMIMXRT117XBRN).*

## 4.3 Build a multicore example application

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

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

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

*<install_dir>/boards/evkbmimxrt1170/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww*

*<install_dir>/boards//evkbmimxrt1170/multicore_examples/hello_world/cm7/iar/hello_world_cm7.eww*

Build both applications separately by clicking the **Make** 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, only debug and release RAM targets are present in the project. When building the primary core project, it is possible to select either *debug/release* RAM targets or `flexspi_nor_debug`/`flexspi_nor_release` Flash targets. When choosing Flash targets (preferred) 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 both primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform Steps 1 – 4 as described in

Figure 28. These steps are common for both single core and dual-core applications in IAR.

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

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


**Figure 32. Start all cores button**

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

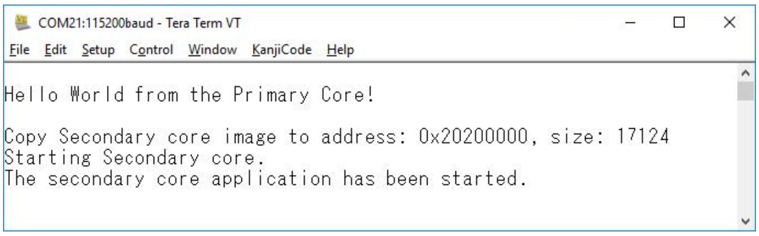


**Figure 33. Hello World from primary core message**

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** buttons to stop or run both cores simultaneously.


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

***Note:*** *On IAR, the feature to simultaneously debug two cores is only supported by CMSIS-DAP debugger.*

# 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. Be patient when the DFP is installed. It will take approximate 15 minutes for the installation to complete.

### 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/evkbmimxrt1170/demo_apps/hello_world/cm7/mdk/hello_world_demo_cm7.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/mbed/DAPLink` debug probe by default. Visit MBED 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, J11, and connect to the debug serial port number. To determine the COM port number, see Section 8. 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

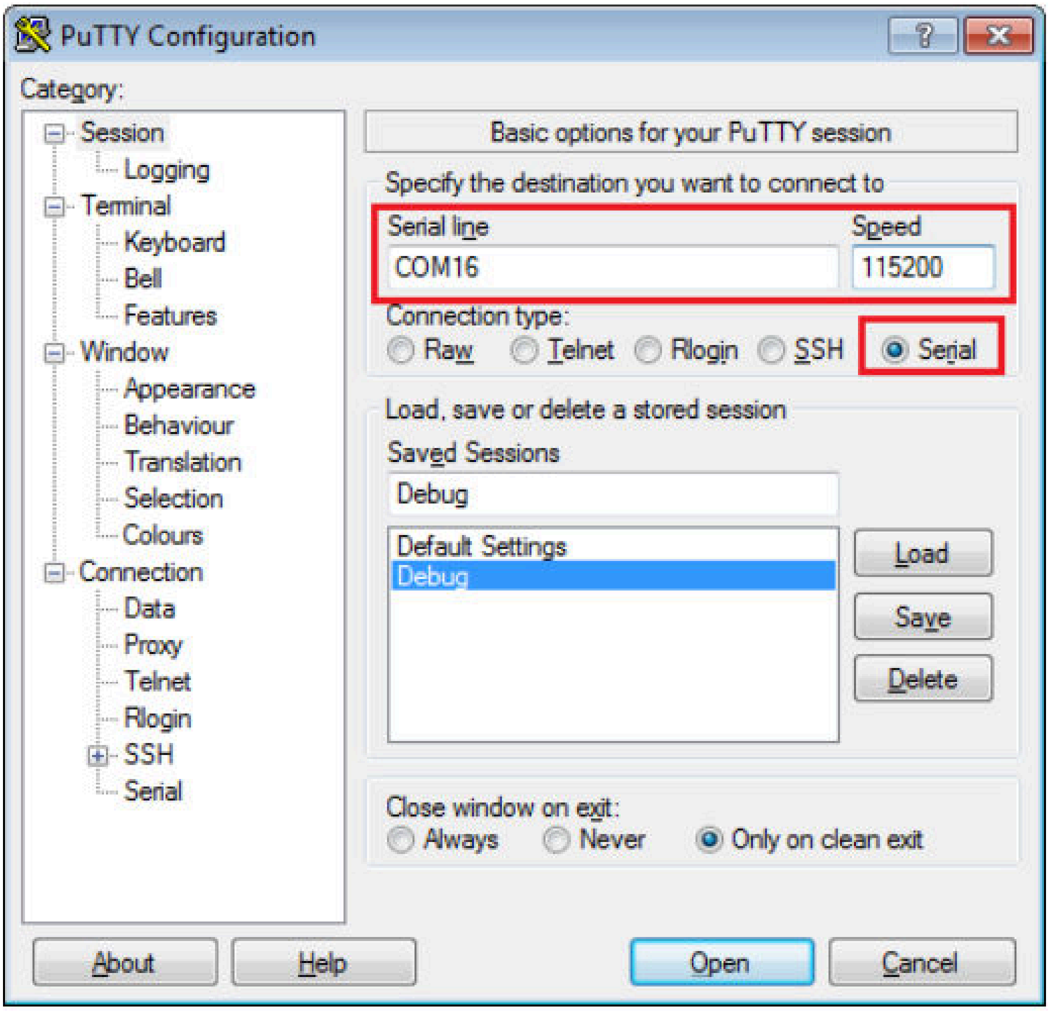


Figure 36. Terminal (PuTTY) configurations

4. To debug the application, click **load** or press the F8 key for flexspi target which needs to download the program to flash memory. 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:***
   *When using jlink in MDK for cm4 projects, it expects one jlinkscript file named JLinkSettings.JLinkScript in the folder where the uVision project files are located. Please refer to Segger Wiki for more information.*
   *For the contents in this `JlinkSettings.JLinkScript`, use contents in `evkbmimxrt1170_connect_cm4_cm4side.jlinkscript` (non-sdram targets) and `evkbmimxrt1170_connect_cm4_cm4side_sdram.jlinkscript` (sdram targets).*

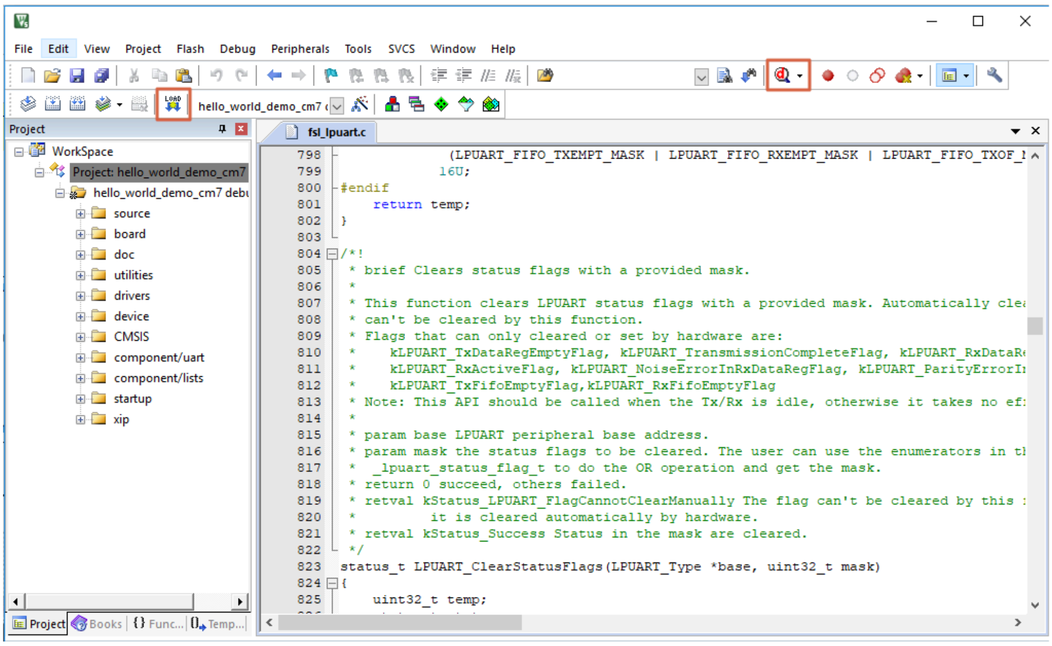

Figure 37. Stop at main() when run debugging

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

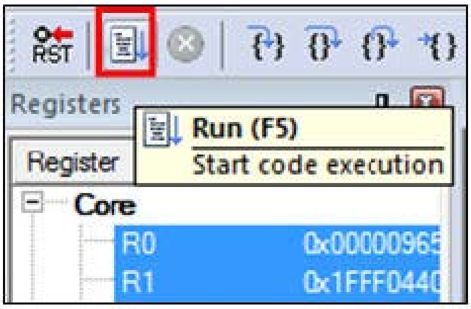


Figure 38. Run button

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.

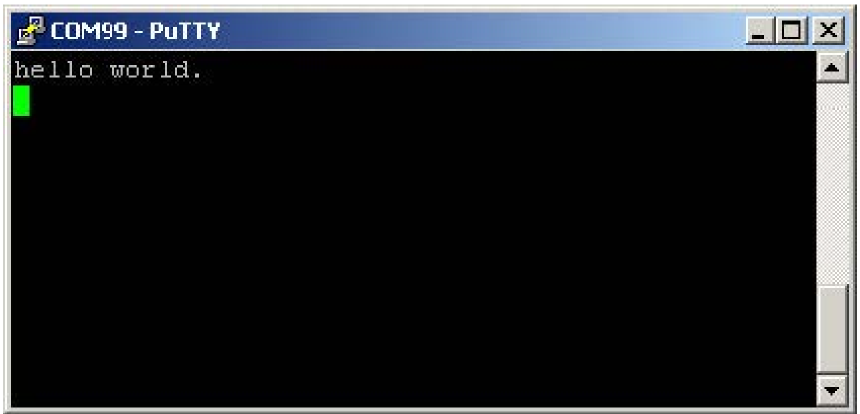


Figure 39. Text display of the hello_world demo

## 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/evkbmimxrt1170/multicore_examples/<application_name>/<core_type>/mdk*

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

*<install_dir>/boards/evkbmimxrt1170/multicore_examples/hello_world/cm4/mdk/hello_world_cm4.uvmpw*

*<install_dir>/boards//evkbmimxrt1170/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 Section 5.3. 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.

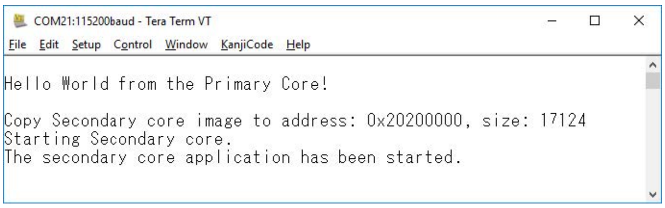


**Figure 40. Hello World from primary core message**

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 the **Start/Stop Debug Session** button. After this, the second debug session is opened and the auxiliary core

application can be debugged.

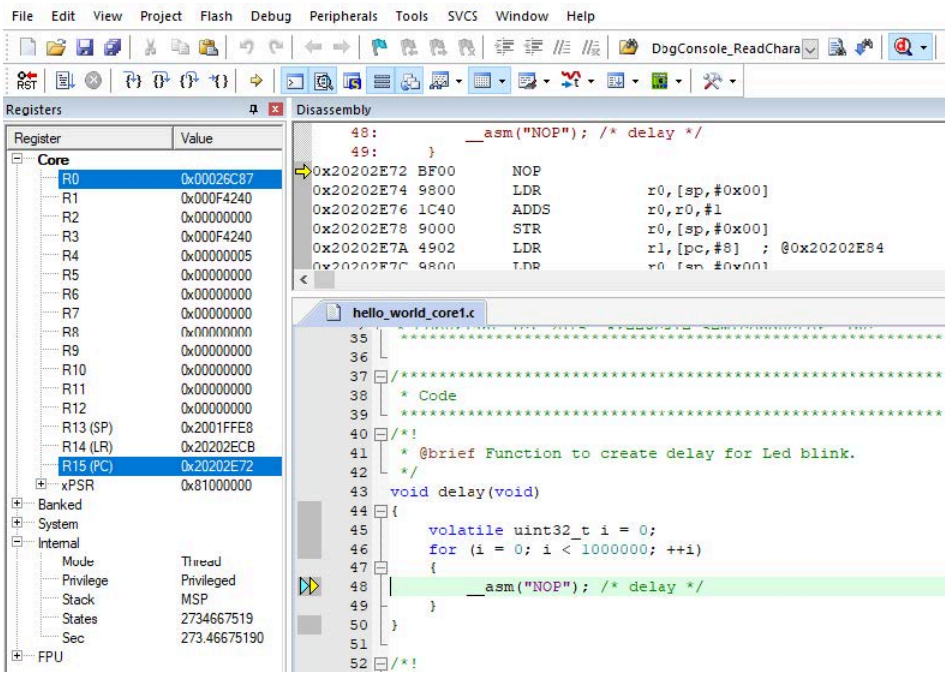


**Figure 41. Second debugging auxiliary core application**

# 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 GNU Arm Embedded Toolchain. 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 for MIMXRT1170-EVKB* (document MCUXSDKMIMXRT117XBRN).

### 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.
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**.

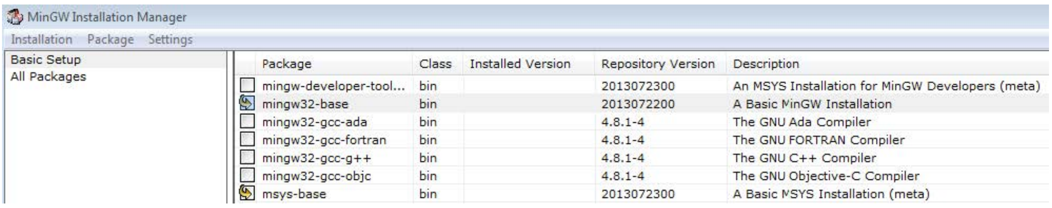


**Figure 42. Set up MinGW and MSYS**

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

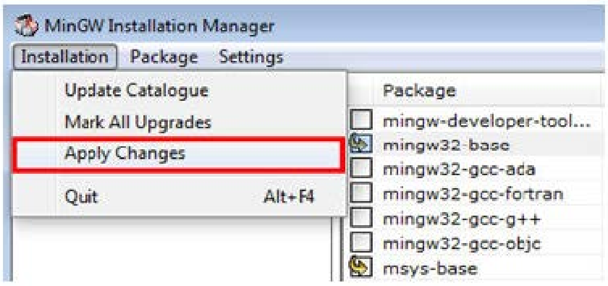


**Figure 43. Complete MinGW and MSYS installation**

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 is *C:\MinGW*, an example is as shown in Figure 44. If the path is not set correctly, the toolchain will 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.*

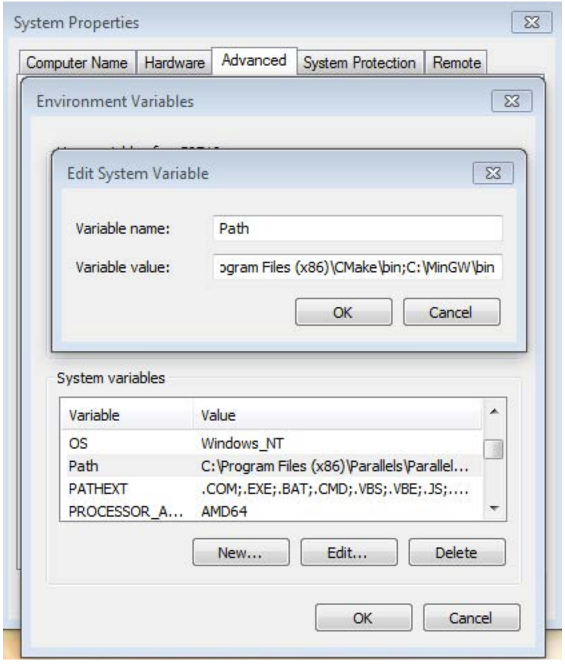


**Figure 44. Add Path to systems environment**

### 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*

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 the `for %I in (.) do echo %~sI` command in above path.

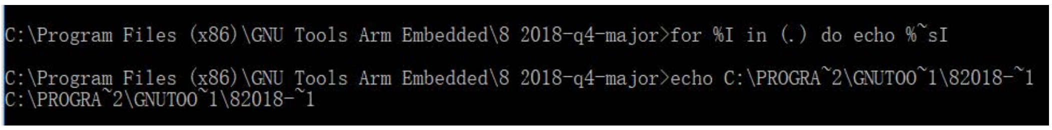


**Figure 45. Convert path to short path**

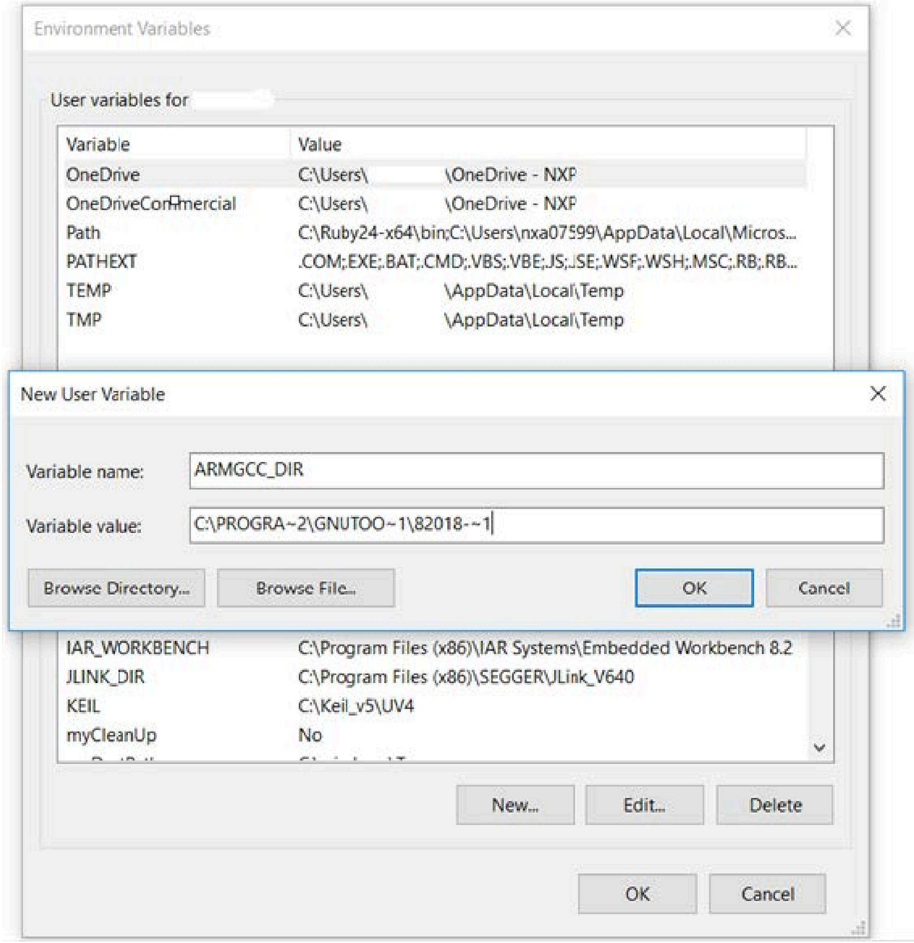


Figure 46. Add ARMGCC_DIR system variable

### 6.1.4 Install CMake

1. Download CMake 3.0.x from CMAKE.
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.

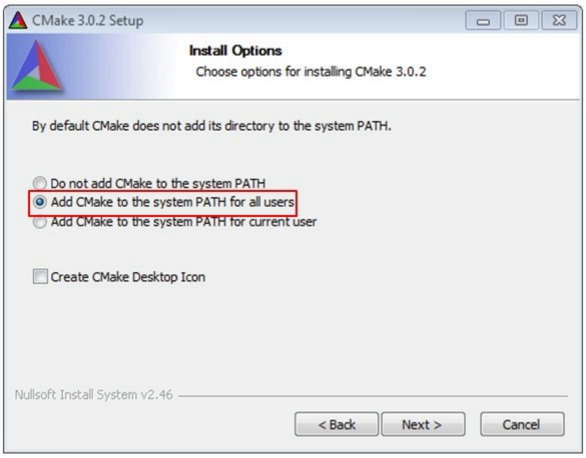


**Figure 47. Install CMake**

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**.

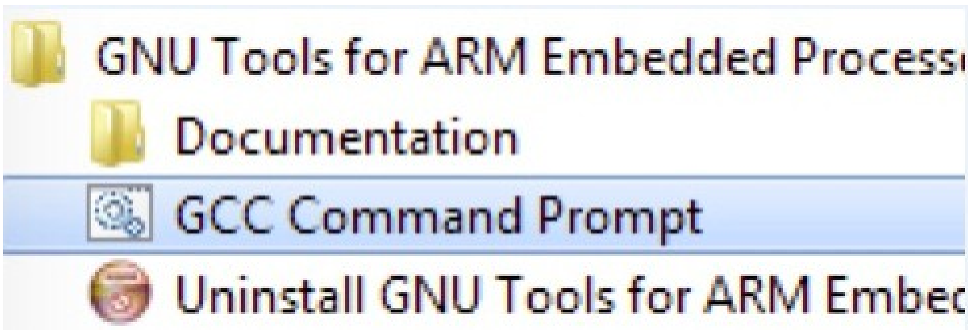


**Figure 48. Launch command prompt**

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/evkbmimxrt1170/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.


**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 SEGGER.
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 Section 8). 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

 

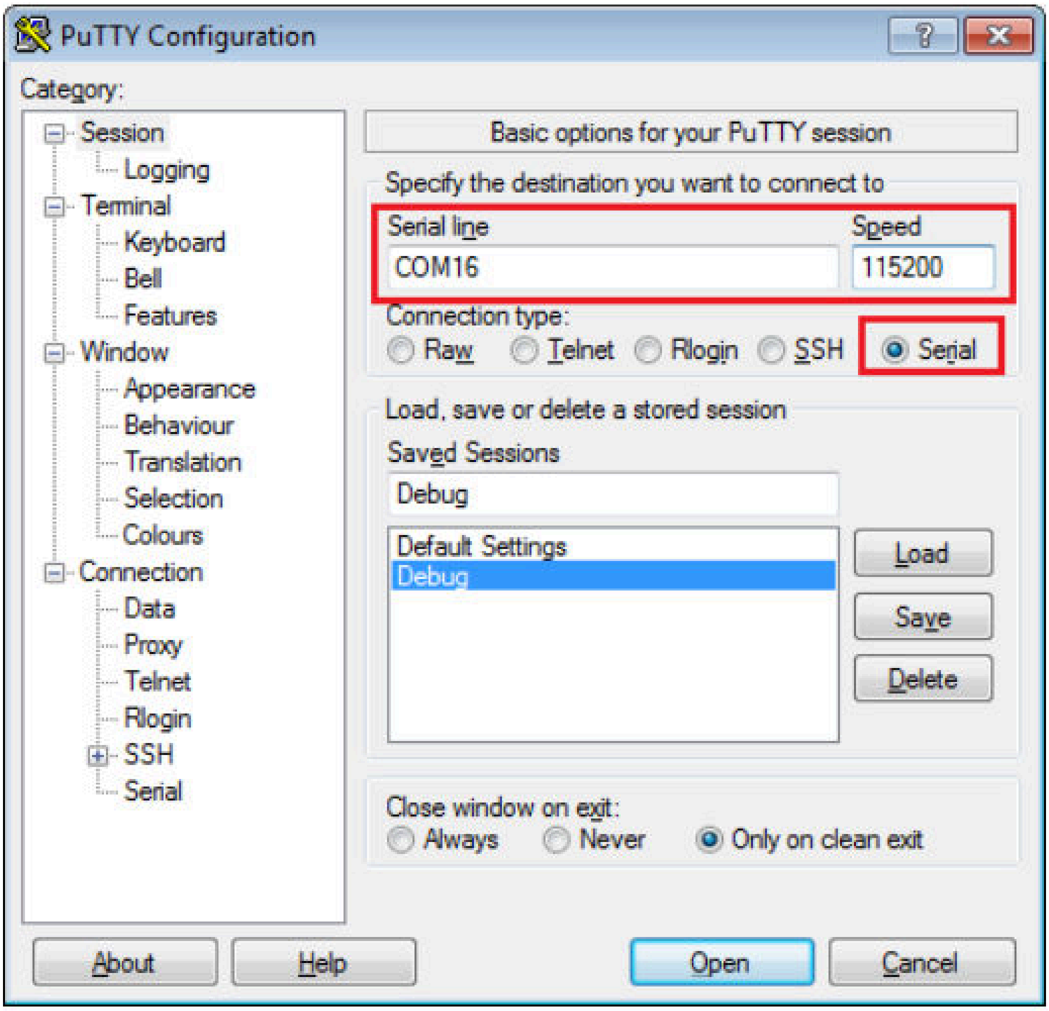


**Figure 50. Terminal (PuTTY) configurations**

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/evkbmimxrt1170/demo_apps/hello_world/cm7/evkbmimxrt1170_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.
   ***Note:***
   *When download the cm4 project, use **MIMXRT1176VDMAA_cm4** as the target device. Use the evkbmimxrt1170_connect_cm4_cm4side.jlinkscript file for non-sdram targets and evkbmimxrt1170_connect_cm4_cm4side_sdram.jlinkscript for sdram targets.*

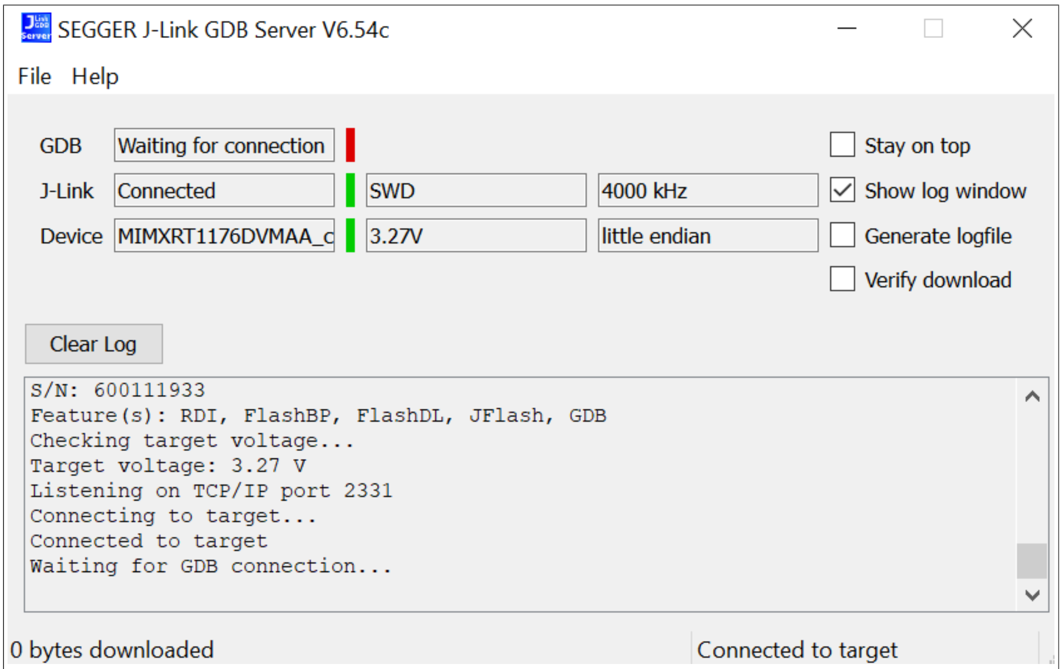


**Figure 51. SEGGER J-Link GDB Server screen after successful connection**

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**.

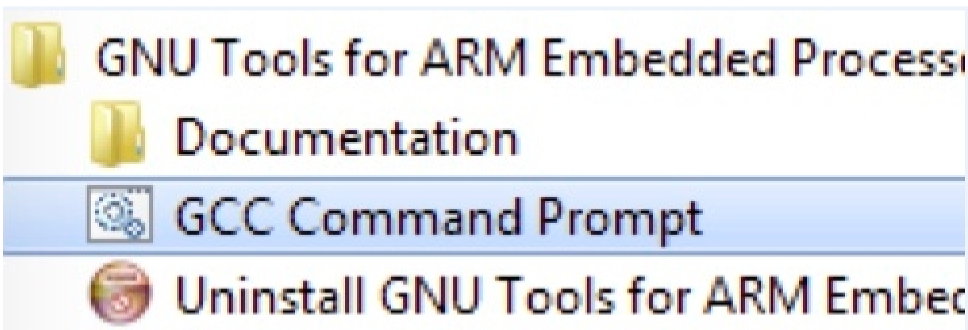


**Figure 52. Launch command prompt**

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/evkbmimxrt1170/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`.

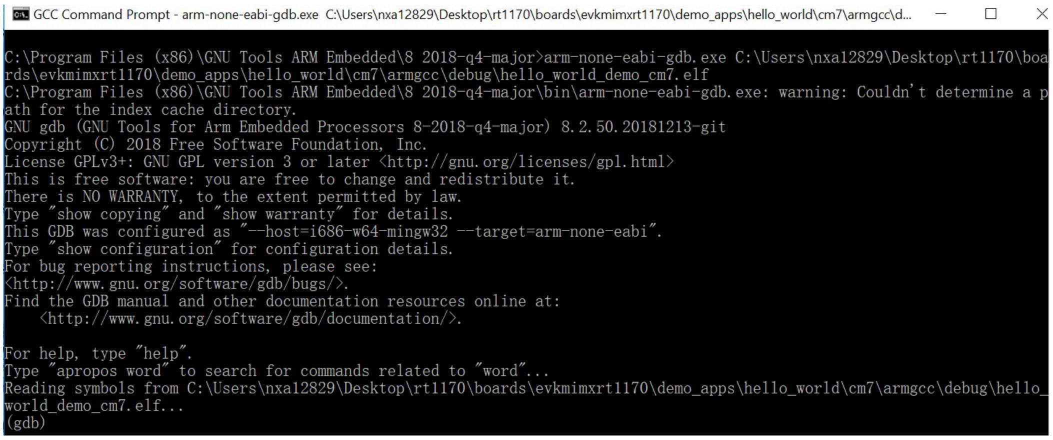


**Figure 53. Run arm-none-eabi-gdb**

10. Run these commands:

```
target remote localhost:2331
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.

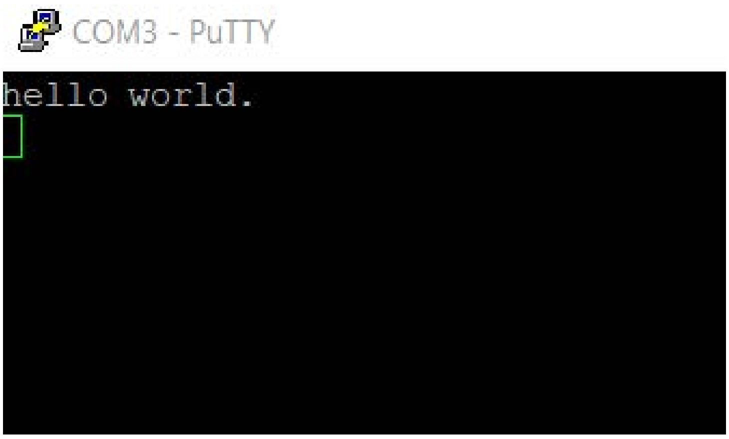


**Figure 54. Text display of the hello_world demo**

## 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/evkbmimxrt1170/multicore_examples/<application_name>/<core_type>/armgcc*

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

*<install_dir>/boards/evkbmimxrt1170/multicore_examples/hello_world/cm4/armgcc/build_debug.bat*

*<install_dir>/boards/evkbmimxrt1170/multicore_examples/hello_world/cm7/armgcc/build_flexspi_nor_debug.bat*

Build both applications separately following steps for single core examples as described in Figure 55 and Figure 56.

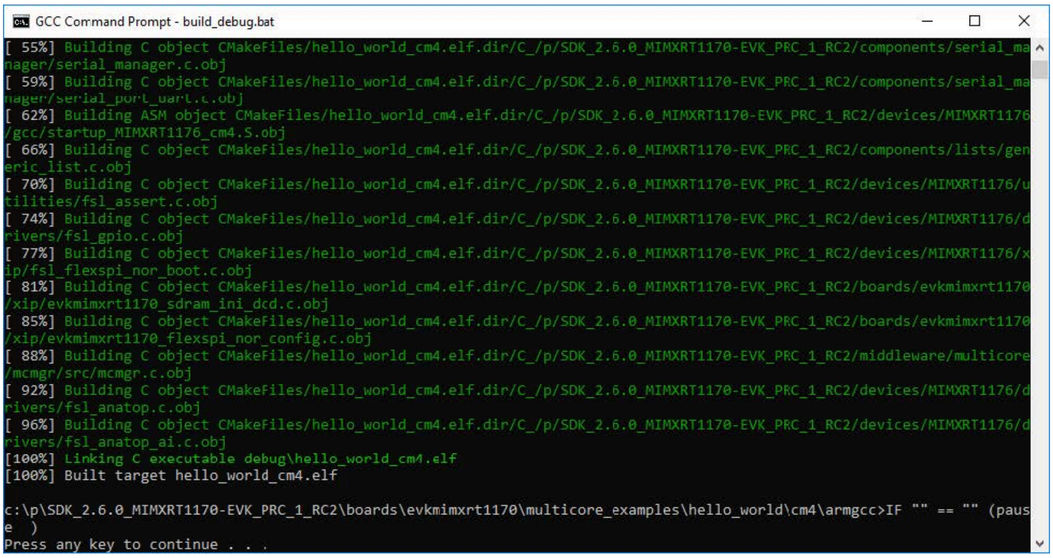


Figure 55. hello_world_cm4 example build successful

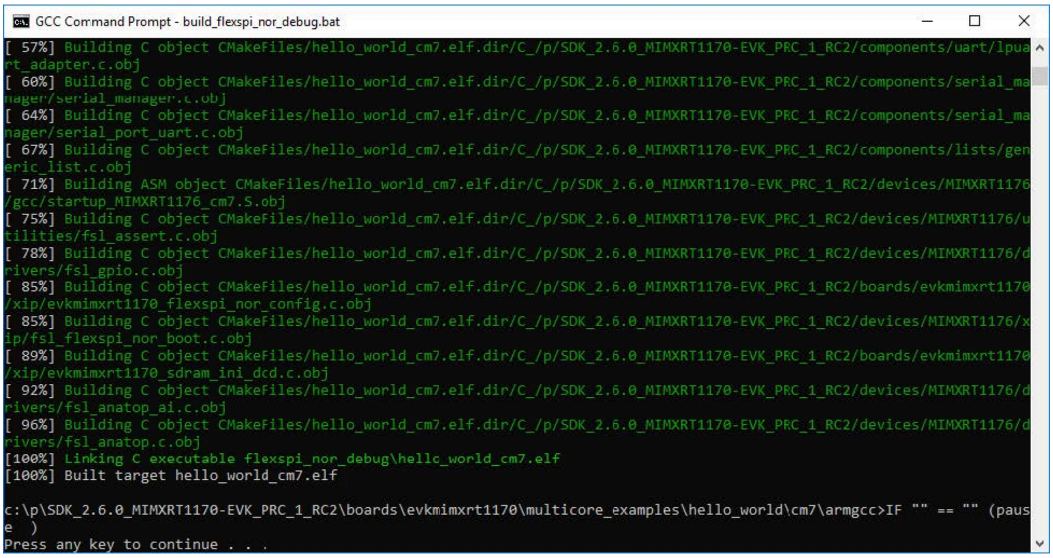


Figure 56. hello_world_cm7 example build successful

## 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 Section 6.3.

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

Section 6.3. 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.

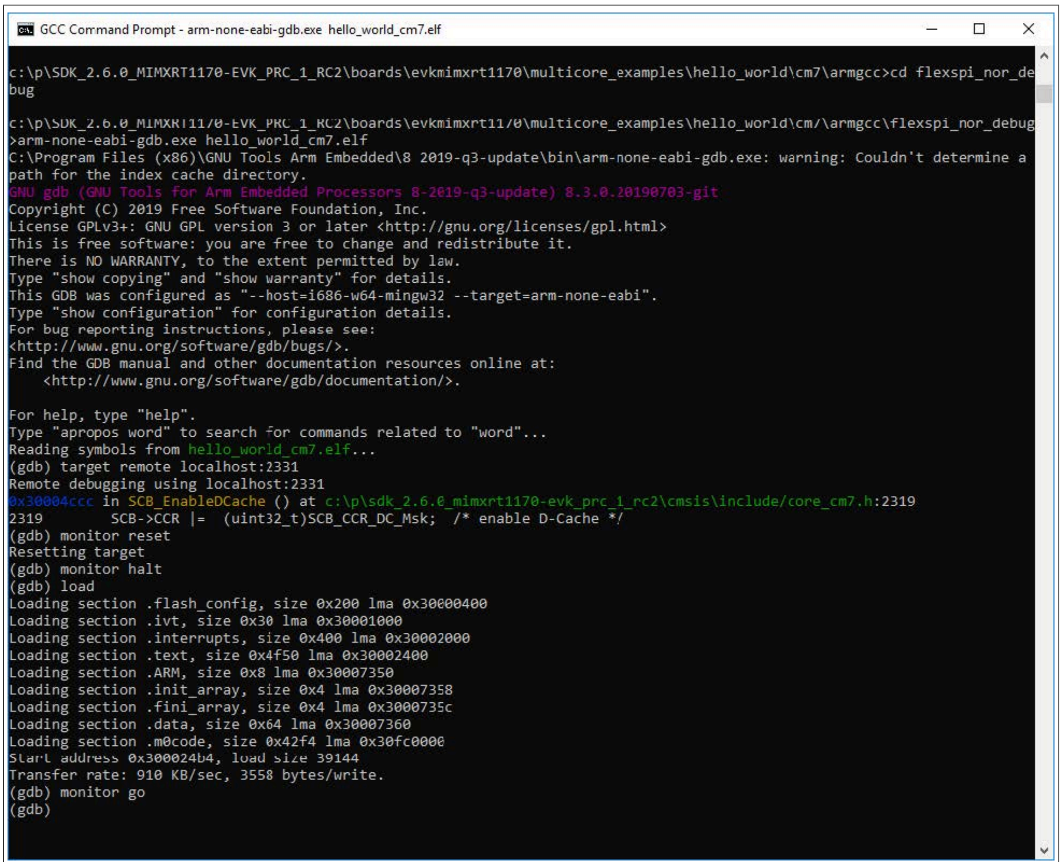


Figure 57. Loading and running the multicore example

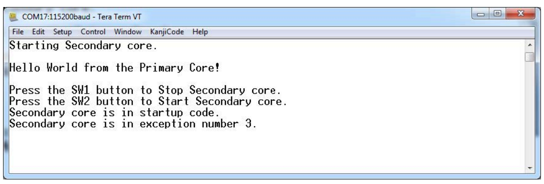


Figure 58. Hello World from primary core message

# 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 MCUXPRESSO. Recommended for customers using IAR Embedded Workbench, Keil MDK µVision, or Arm GCC.
- **Online version** available on MCUXPRESSO. Recommended to do a quick evaluation of the processor or use the tool without installation.

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

## 8 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 in Figure 59.

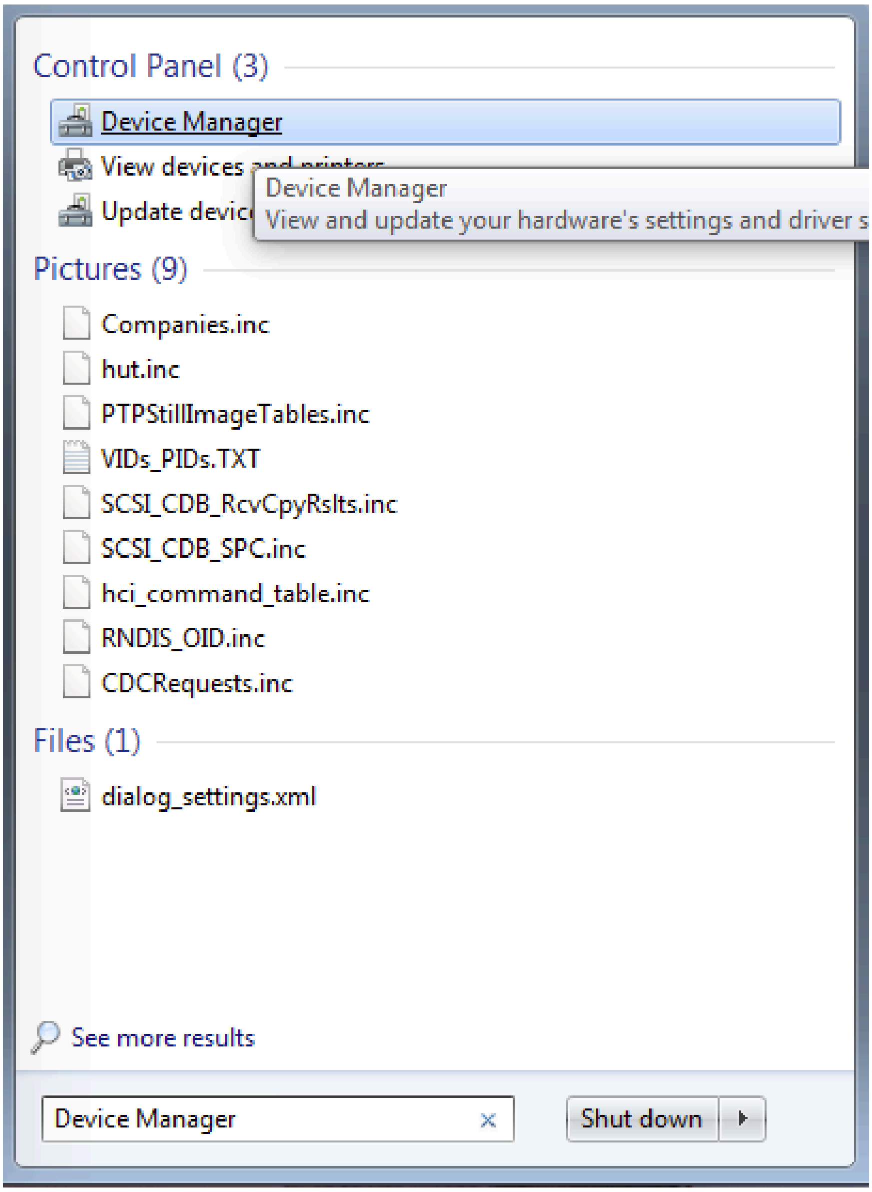


**Figure 59. Device Manager**

2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. Depending on the NXP board you are using, the COM port can be named differently.
    a. OpenSDA – CMSIS-DAP/mbed/DAPLink interface:

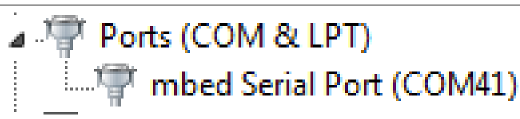


**Figure 60. OpenSDA – CMSIS-DAP/mbed/DAPLink interface**

    b. OpenSDA – P&E Micro:

 

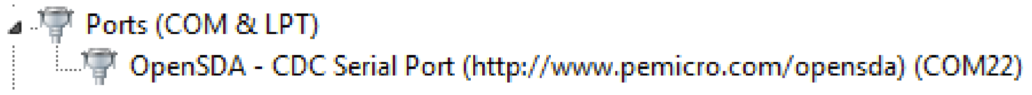


Figure 61. OpenSDA – P&E Micro

c. OpenSDA – J-Link:

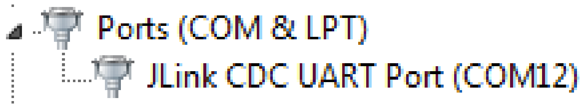


Figure 62. OpenSDA – J-Link

d. P&E Micro OSJTAG:

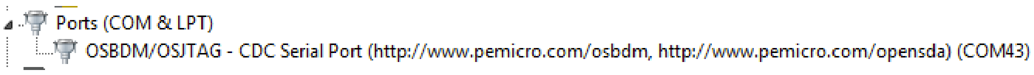


Figure 63. P&E Micro OSJTAG

e. LPC-Link2:

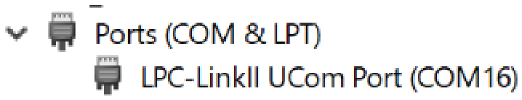


Figure 64. LPC-Link2

f. FTDI UART:

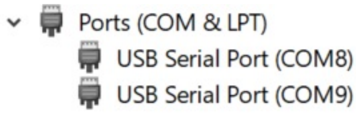


Figure 65. FTDI UART

# 9 Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with a variety of factory programmed debug interface configurations. Table 2 lists the hardware platforms supported by the MCUXpresso SDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

Table 2. Hardware platforms supported by SDK

| Hardware platform | Default interface | OpenSDA details[1] |
|---|---|---|
| EVK-MC56F83000 | P&E Micro OSJTAG | N/A |
| EVK-MIMXRT595 | CMSIS-DAP | N/A |
| EVK-MIMXRT685 | CMSIS-DAP | N/A |
| FRDM-K22F | CMSIS-DAP/mbed/DAPLink | OpenSDA v2.1 |
| FRDM-K28F | DAPLink | OpenSDA v2.1 |
| FRDM-K32L2A4S | CMSIS-DAP | OpenSDA v2.1 |
| FRDM-K32L2B | CMSIS-DAP | OpenSDA v2.1 |
| FRDM-K32W042 | CMSIS-DAP | N/A |
| FRDM-K64F | CMSIS-DAP/mbed/DAPLink | OpenSDA v2.0 |
| FRDM-K66F | J-Link OpenSDA | OpenSDA v2.1 |

Table 2. Hardware platforms supported by SDK*...continued*

| Hardware platform | Default interface | OpenSDA details[1] |
|---|---|---|
| FRDM-K82F | CMSIS-DAP | OpenSDA v2.1 |
| FRDM-KE15Z | DAPLink | OpenSDA v2.1 |
| FRDM-KE16Z | CMSIS-DAP/mbed/DAPLink | OpenSDA v2.2 |
| FRDM-KL02Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL03Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL25Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL26Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL27Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL28Z | P&E Micro OpenSDA | OpenSDA v2.1 |
| FRDM-KL43Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL46Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KL81Z | CMSIS-DAP | OpenSDA v2.0 |
| FRDM-KL82Z | CMSIS-DAP | OpenSDA v2.0 |
| FRDM-KV10Z | CMSIS-DAP | OpenSDA v2.1 |
| FRDM-KV11Z | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KV31F | P&E Micro OpenSDA | OpenSDA v1.0 |
| FRDM-KW24 | CMSIS-DAP/mbed/DAPLink | OpenSDA v2.1 |
| FRDM-KW36 | DAPLink | OpenSDA v2.2 |
| FRDM-KW41Z | CMSIS-DAP/DAPLink | OpenSDA v2.1 or greater |
| Hexiwear | CMSIS-DAP/mbed/DAPLink | OpenSDA v2.0 |
| HVP-KE18F | DAPLink | OpenSDA v2.2 |
| HVP-KV46F150M | P&E Micro OpenSDA | OpenSDA v1 |
| HVP-KV11Z75M | CMSIS-DAP | OpenSDA v2.1 |
| HVP-KV58F | CMSIS-DAP | OpenSDA v2.1 |
| HVP-KV31F120M | P&E Micro OpenSDA | OpenSDA v1 |
| JN5189DK6 | CMSIS-DAP | N/A |
| LPC54018 IoT Module | N/A | N/A |
| LPCXpresso54018 | CMSIS-DAP | N/A |
| LPCXpresso54102 | CMSIS-DAP | N/A |
| LPCXpresso54114 | CMSIS-DAP | N/A |
| LPCXpresso51U68 | CMSIS-DAP | N/A |
| LPCXpresso54608 | CMSIS-DAP | N/A |
| LPCXpresso54618 | CMSIS-DAP | N/A |
| LPCXpresso54628 | CMSIS-DAP | N/A |
| LPCXpresso54S018M | CMSIS-DAP | N/A |
| LPCXpresso55s16 | CMSIS-DAP | N/A |

**Table 2. Hardware platforms supported by SDK***...continued*

| Hardware platform | Default interface | OpenSDA details[1] |
|---|---|---|
| LPCXpresso55s28 | CMSIS-DAP | N/A |
| LPCXpresso55s69 | CMSIS-DAP | N/A |
| MAPS-KS22 | J-Link OpenSDA | OpenSDA v2.0 |
| MIMXRT1170-EVK | CMSIS-DAP | N/A |
| MIMXRT1170-EVKB | CMSIS-DAP | N/A |
| TWR-K21D50M | P&E Micro OSJTAG | N/AOpenSDA v2.0 |
| TWR-K21F120M | P&E Micro OSJTAG | N/A |
| TWR-K22F120M | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-K24F120M | CMSIS-DAP/mbed | OpenSDA v2.1 |
| TWR-K60D100M | P&E Micro OSJTAG | N/A |
| TWR-K64D120M | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-K64F120M | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-K65D180M | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-K65D180M | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-KV10Z32 | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-K80F150M | CMSIS-DAP | OpenSDA v2.1 |
| TWR-K81F150M | CMSIS-DAP | OpenSDA v2.1 |
| TWR-KE18F | DAPLink | OpenSDA v2.1 |
| TWR-KL28Z72M | P&E Micro OpenSDA | OpenSDA v2.1 |
| TWR-KL43Z48M | P&E Micro OpenSDA | OpenSDA v1.0 |
| TWR-KL81Z72M | CMSIS-DAP | OpenSDA v2.0 |
| TWR-KL82Z72M | CMSIS-DAP | OpenSDA v2.0 |
| TWR-KM34Z75M | P&E Micro OpenSDA | OpenSDA v1.0 |
| 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 |

[1] The OpenSDA details is not applicable to LPC.

# 10 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 3.

Table 3. Macros added in flexspi_nor

| | |
|---|---|
| **XIP_EXTERNAL_FLASH** | 1: Exclude the code which changes the clock of FLEXSPI. |
| | 0: Make no changes. |
| **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 4 shows the different effect on the built image with a different combination of these macros.

Table 4. Effects on built image with different macros

<table>
<tr><th colspan="2"></th><th>XIP_BOOT_HEADER_DCD_ENABLE=1</th><th>XIP_BOOT_HEADER_DCD_ENABLE=0</th></tr>
<tr><td rowspan="2">XIP_EXTERNAL_FLASH=1</td><td>XIP_BOOT_HEADER_ENABLE=1</td><td>– Can be programmed to <code>qspiflash</code> by IDE and can run after POR reset if <code>qspiflash</code> is the boot source.<br>– SDRAM will be initialized.</td><td>– Can be programmed to <code>qspiflash</code> by IDE, and can run after POR reset if <code>qspiflash</code> is the boot source.<br>– SDRAM will NOT be initialized.</td></tr>
<tr><td>XIP_BOOT_HEADER_ENABLE=0</td><td>– CANNOT run after POR reset if it is programmed by IDE, even if <code>qspiflash</code> is the boot source.</td><td rowspan="2">—</td></tr>
<tr><td colspan="2">XIP_EXTERNAL_FLASH=0</td><td>– This image CANNOT complete XIP because when this macro is set to 1, it excludes the code, which changes the clock for FLEXSPI.</td></tr>
</table>

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

Take `hello_world` as an example:

- **IAR**

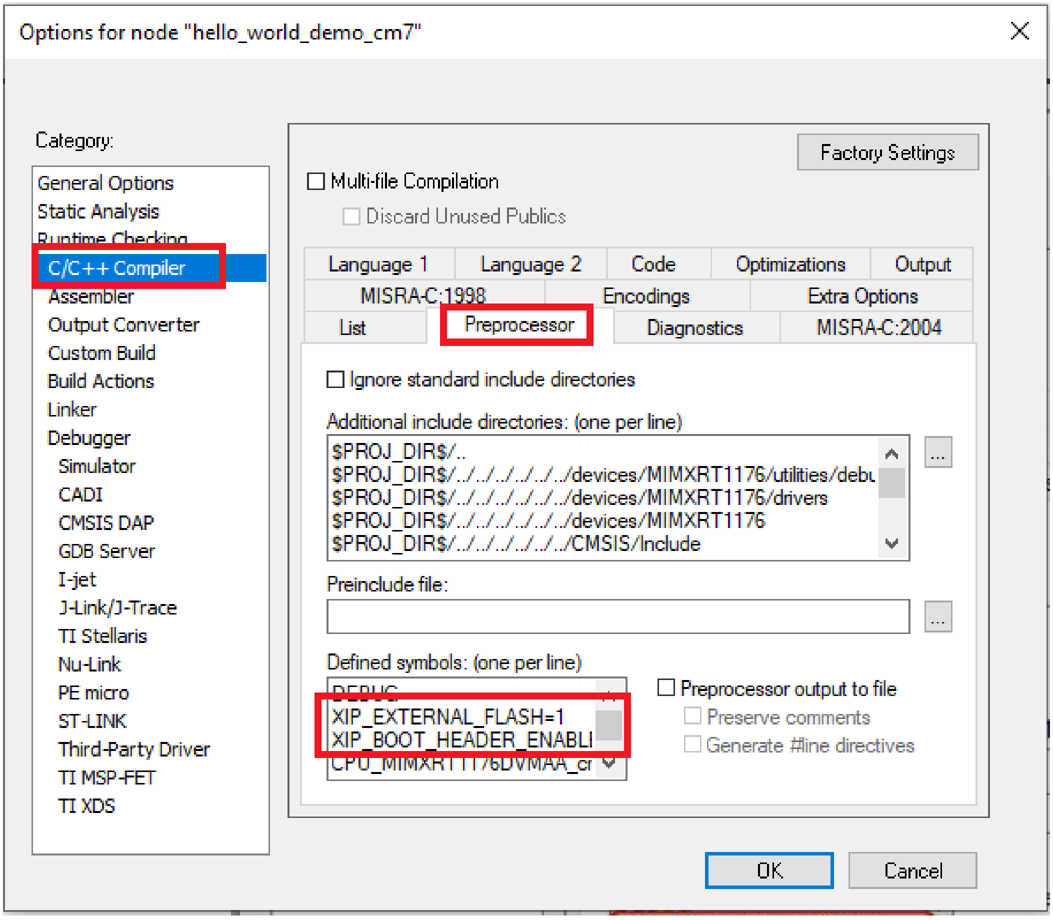


Figure 66. Options node IAR

- **MDK**

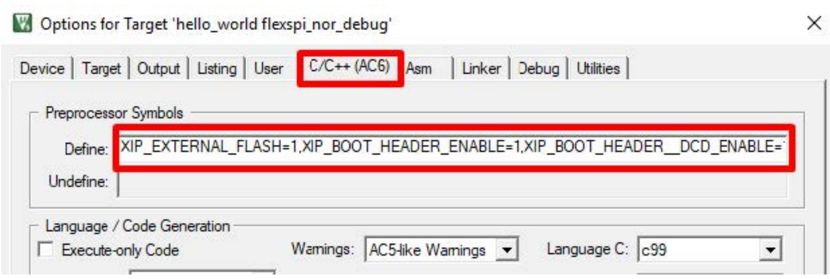


Figure 67. Options for target

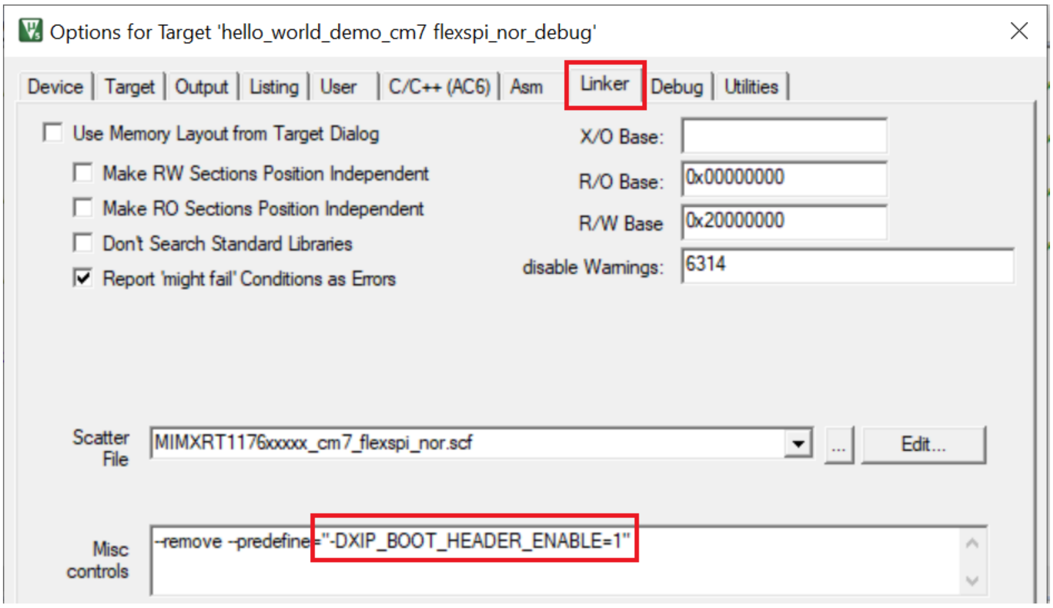


Figure 68. Change configuration Misc controls

- **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 69. Change configuration CMakeLists.txt

- **MCUX**

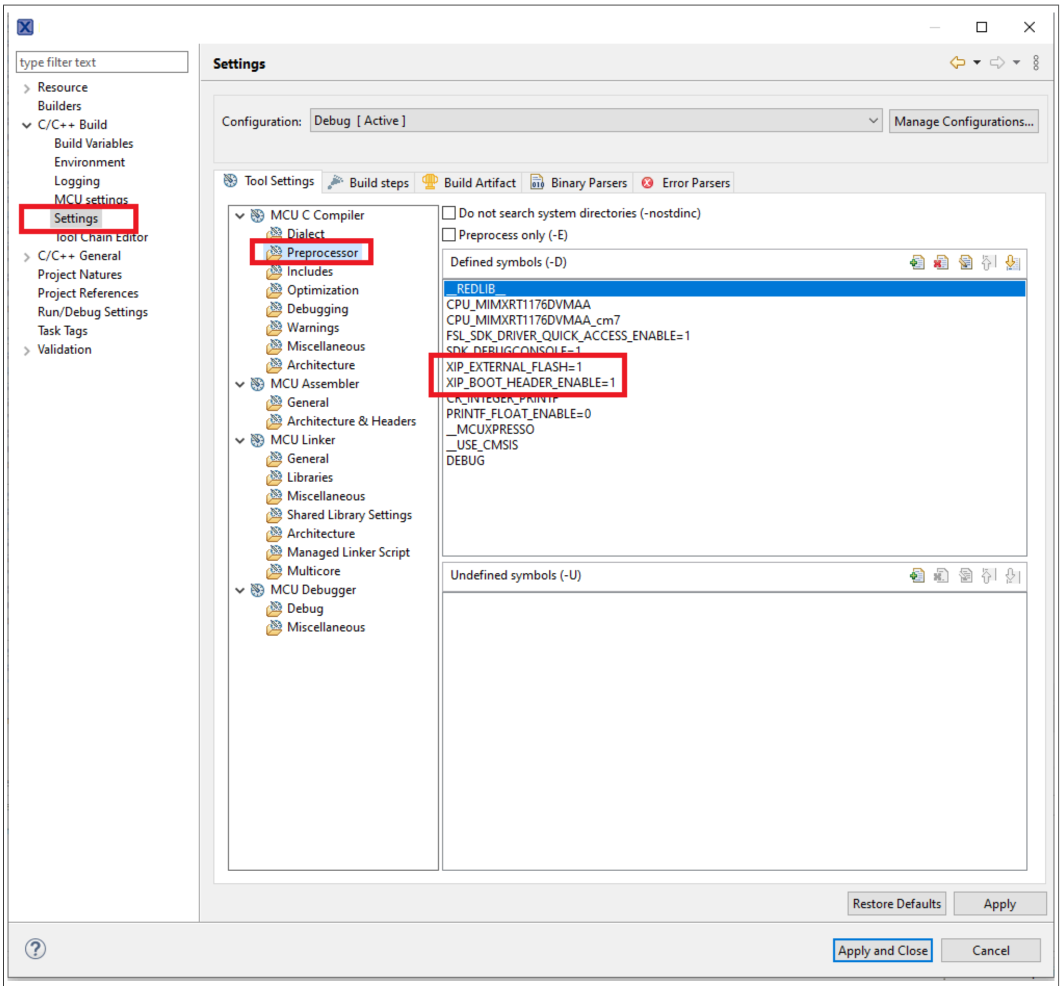

Figure 70.  Properties for evkbmimxrt1170

# 11 Revision history

| Rev. | Date | Description |
|---|---|---|
| 0 | 28 December 2022 | Initial release for MCUXpresso SDK 2.13.0 MIMXRT1170-EVKB RFP |

# 12 Legal information

## 12.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## 12.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

## 12.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

## Contents





**Date of release: 31 December 2022**
**Document identifier: MCUXSDKMIMXRT117XBGSUG**