MCUXSDKMIMXRT118XKGSUG

Getting Started with MCUXpresso SDK for MIMXRT1180-EVK

Rev. 2.16.000 — 20 June 2024

User guide

Document information

Information	Content
Keywords	MCUXSDKMIMXRT118XKGSUG, MCUXpresso SDK, MIMXRT1180-EVK
Abstract	This document describes the steps to get started with MCUXpresso SDK for MIMXRT1180-EVK.

1 Overview

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for NXP 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 MIMXRT1180-EVK* (document MCUXSDKMIMXRT118XKRN).

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

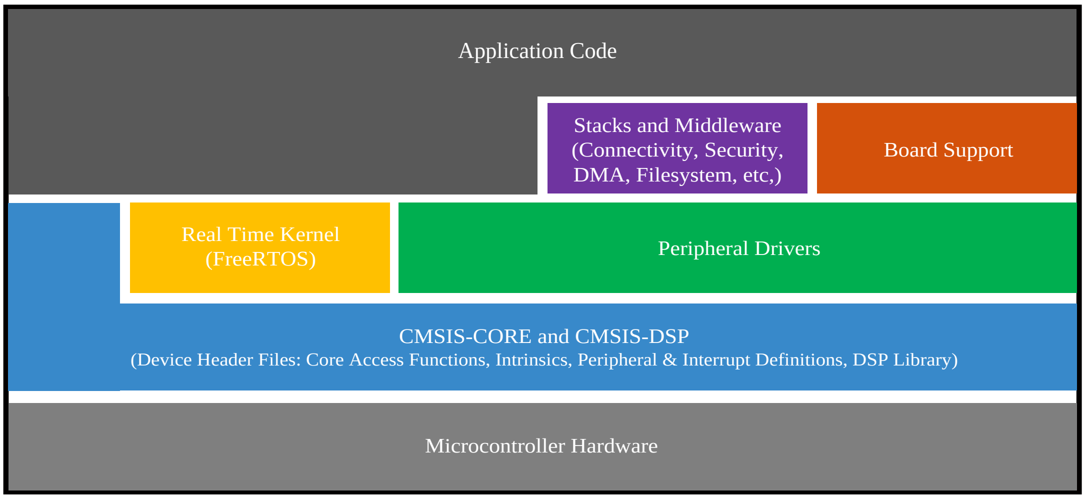


Figure 1. MCUXpresso SDK layers

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. 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 contains. These types 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 FreeRTOS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers.
- `Other_examples`: See detail in package *boards/evkmimxrt1180*.

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.

The *hello_world* application folder contains the following contents:

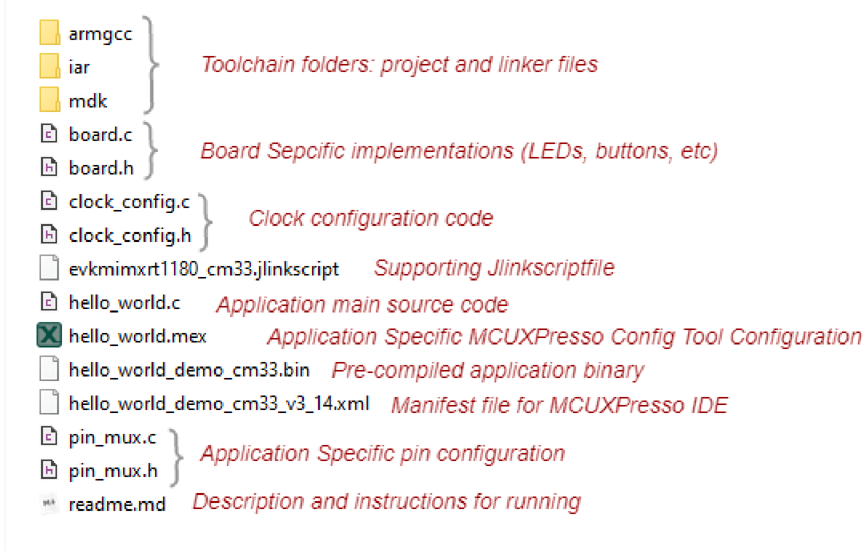


Figure 2. Application folder structure

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, various 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 CMSIS header file of the device, MCUXpresso SDK feature file and a few other files
- `devices/<device_name>/drivers`: All 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

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 MIMXRT1180-EVK hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

Both CMSIS-DAP and J-Link debugging interface is supported for MCUX IDE. When using CMSIS-DAP debugging interface, the `SW5[1..4]` should be put to `0100`. When using J-Link debugging interface, the `SW5[1..4]` should be put to `0001`. It is required to reset board for each download/debug.

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 the 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 locate outside 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.

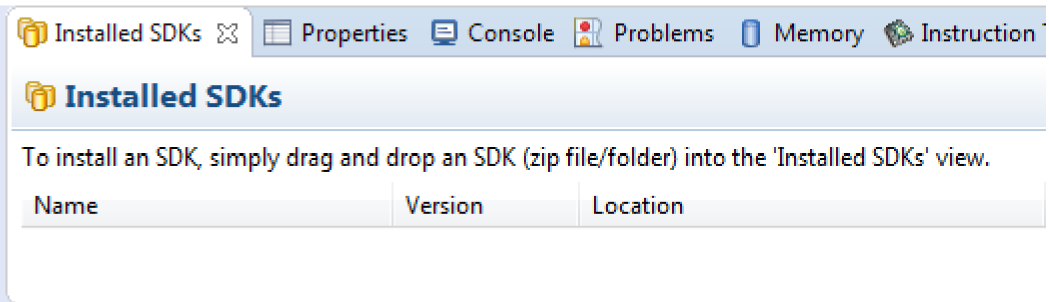


Figure 3. Install an SDK

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

Figure 4. Import an SDK example

3. In the window that appears, select **evkmimxrt1180** or **kits** and click **Next**.

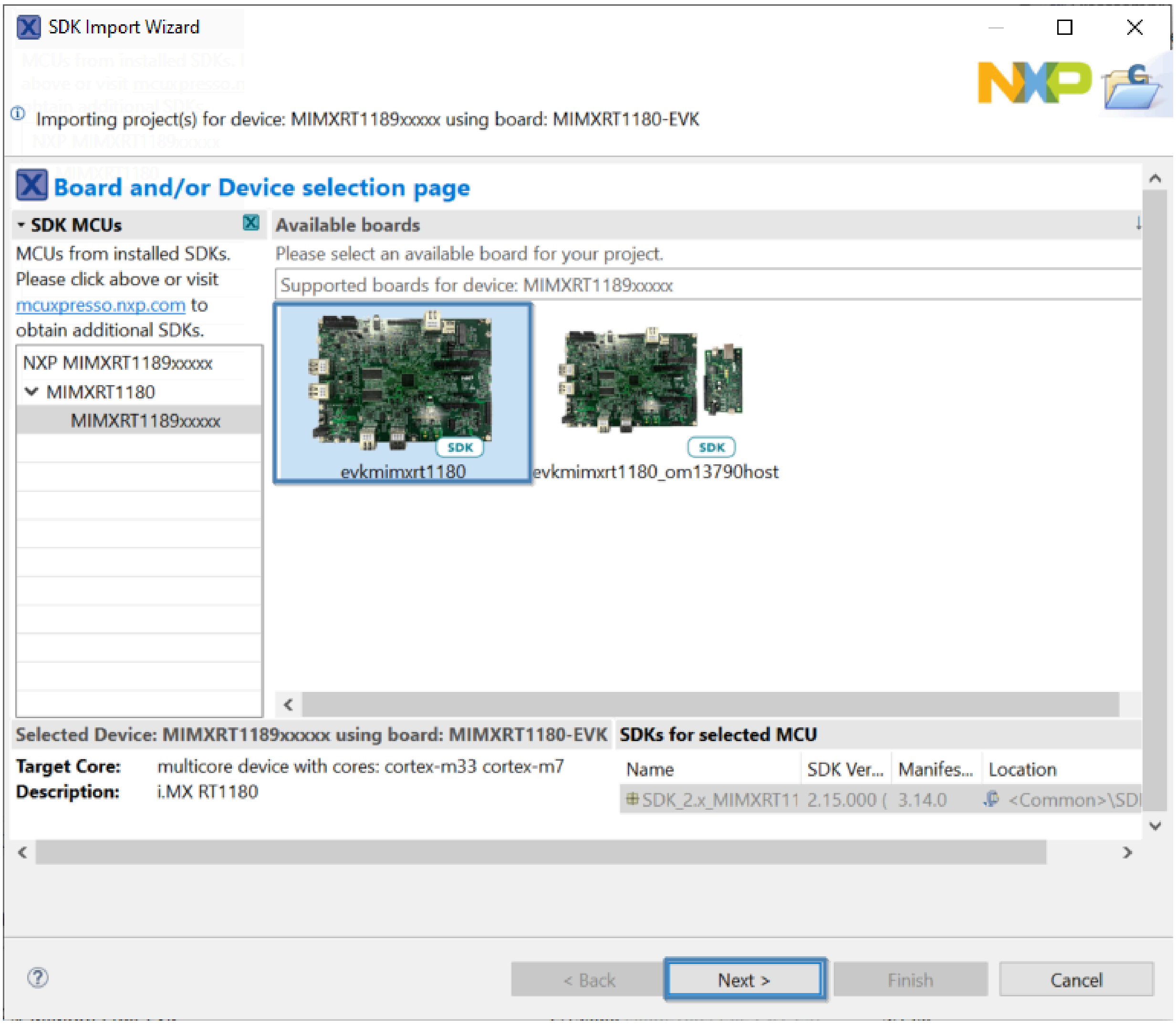


Figure 5. Select MIMXRT1180-EVK board

4. Expand the *demo_apps* folder and select `hello_world_demo_cm33`. Then, click **Next**.

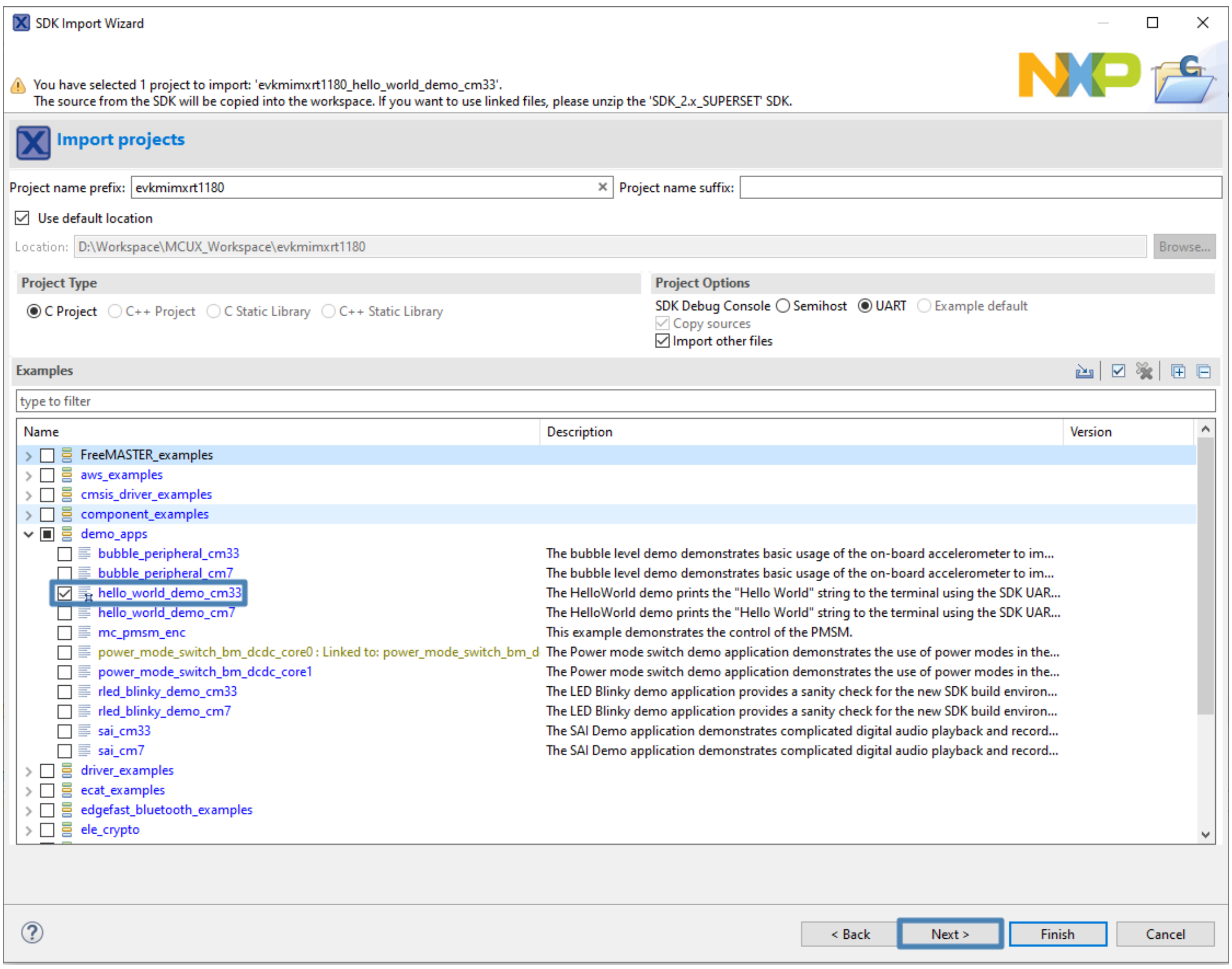


Figure 6. Select `hello_world`

5. Click **Finish**.

3.3 Run an example application

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

1. See Table 7 to determine the debug interface that comes loaded on your specific hardware platform.
   - For EVKMIMXRT1180, LPC55S69 is used.
   - J53 is used as the debugging port which provides both debugging functionality and console out.
2. Connect USB cable between J53 on EVK and PC USB port.
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 9. 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

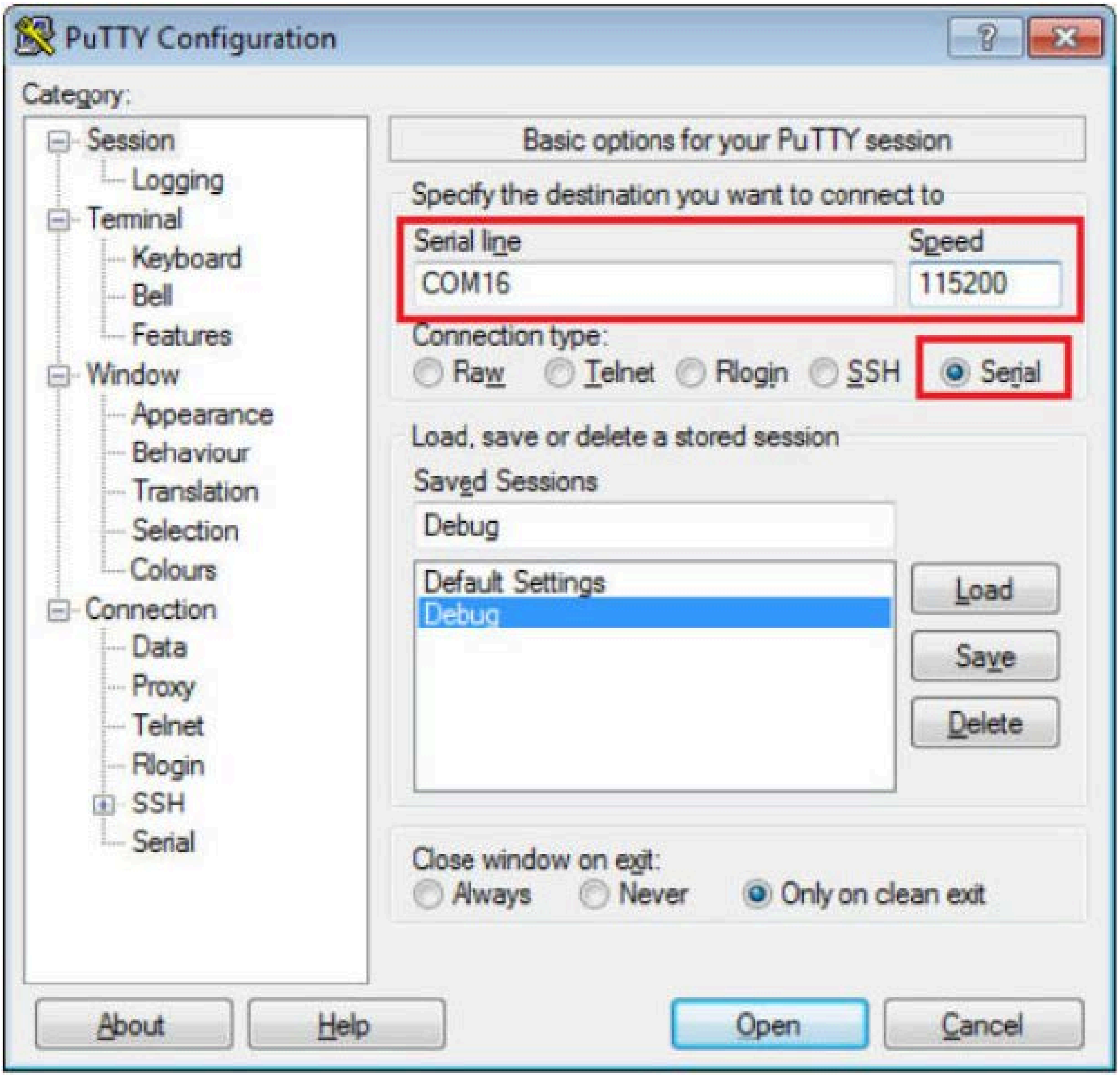


Figure 7. Terminal (PuTTY) configurations

4. On the **Quickstart Panel**, click **Debug**.

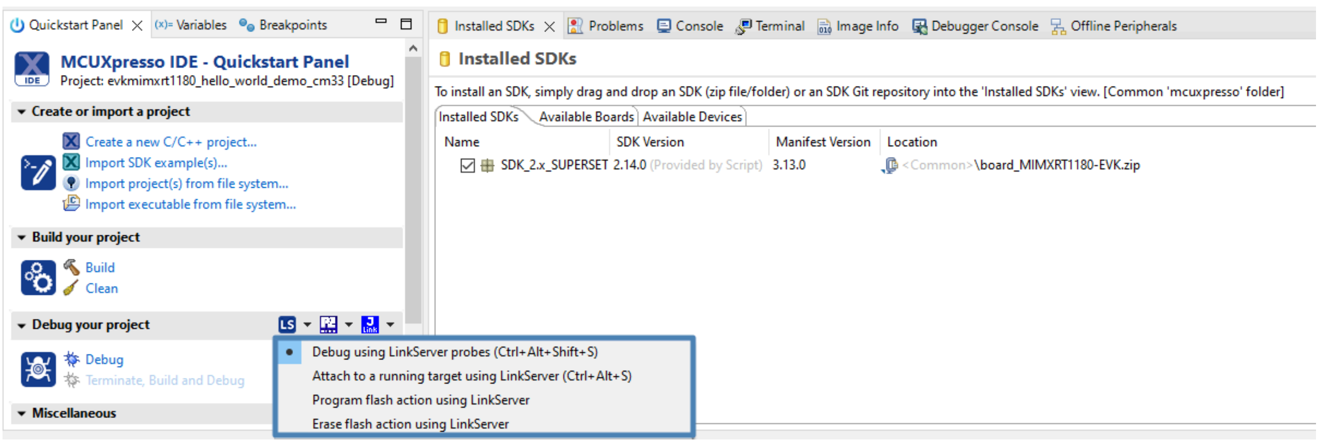


Figure 8. Debug `hello_world` case

> ***Note:*** *Erase the flash memory before running the MCUX IDE projects for the first time. To do so, use the* ***Erase Flash action using LinkServer*** *in [Figure 8](#) with `SW5[1..4]` on `0100`. In the worst case when MCUX IDE cannot erase the flash, use the secure provisioning tool. For more information, see [Section 7.4 "Use Secure Provisioning Tool to erase flash"](#).*

5. The first time you debug a project, the **Debug Emulator Selection** dialog is displayed, showing all supported probes that are attached to your computer. Select the probe through which you want to debug and click **OK**. Here we assume that you are using a CMSIS-DAP debug interface. (For any future debug sessions, the stored probe selection is automatically used, unless the probe cannot be found.)

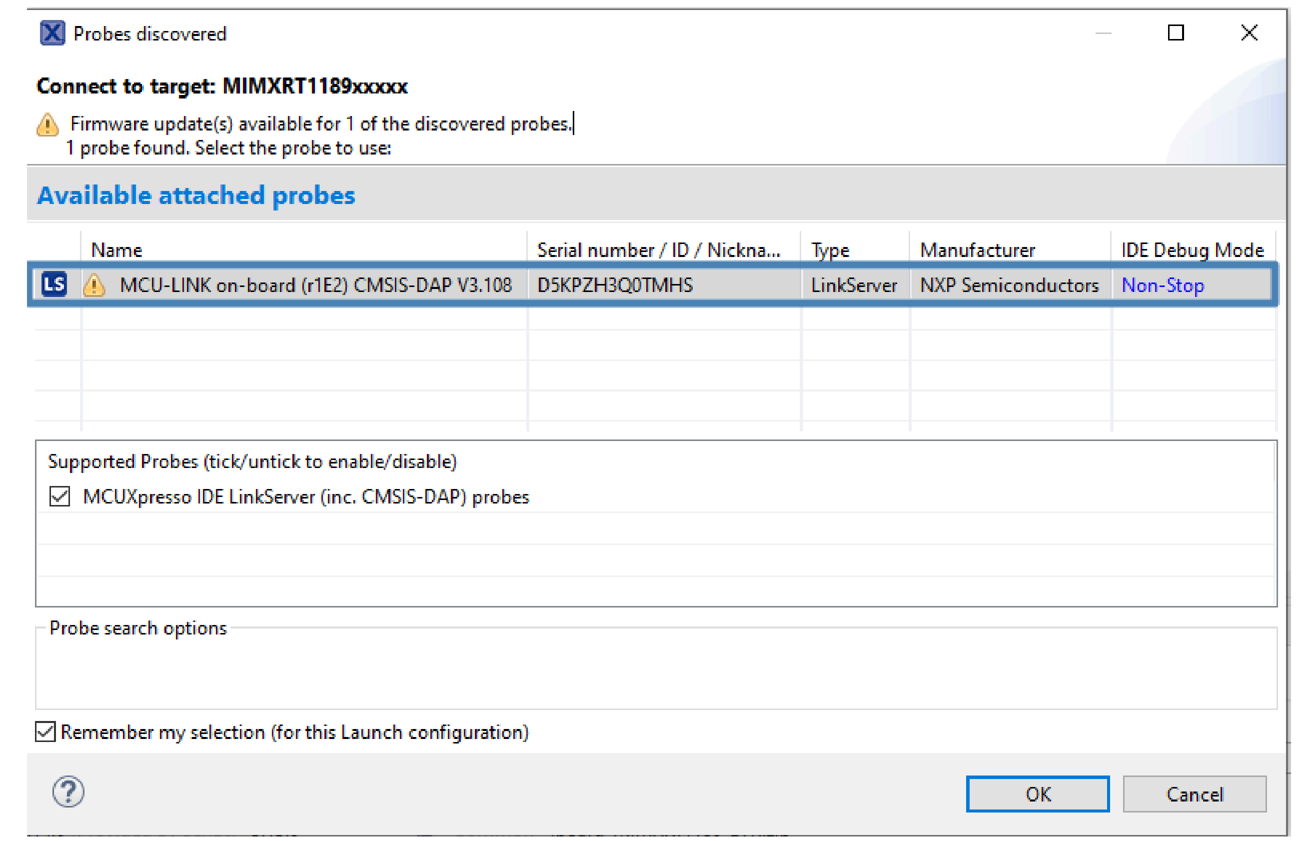


Figure 9. Attached Probes: debug emulator selection

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

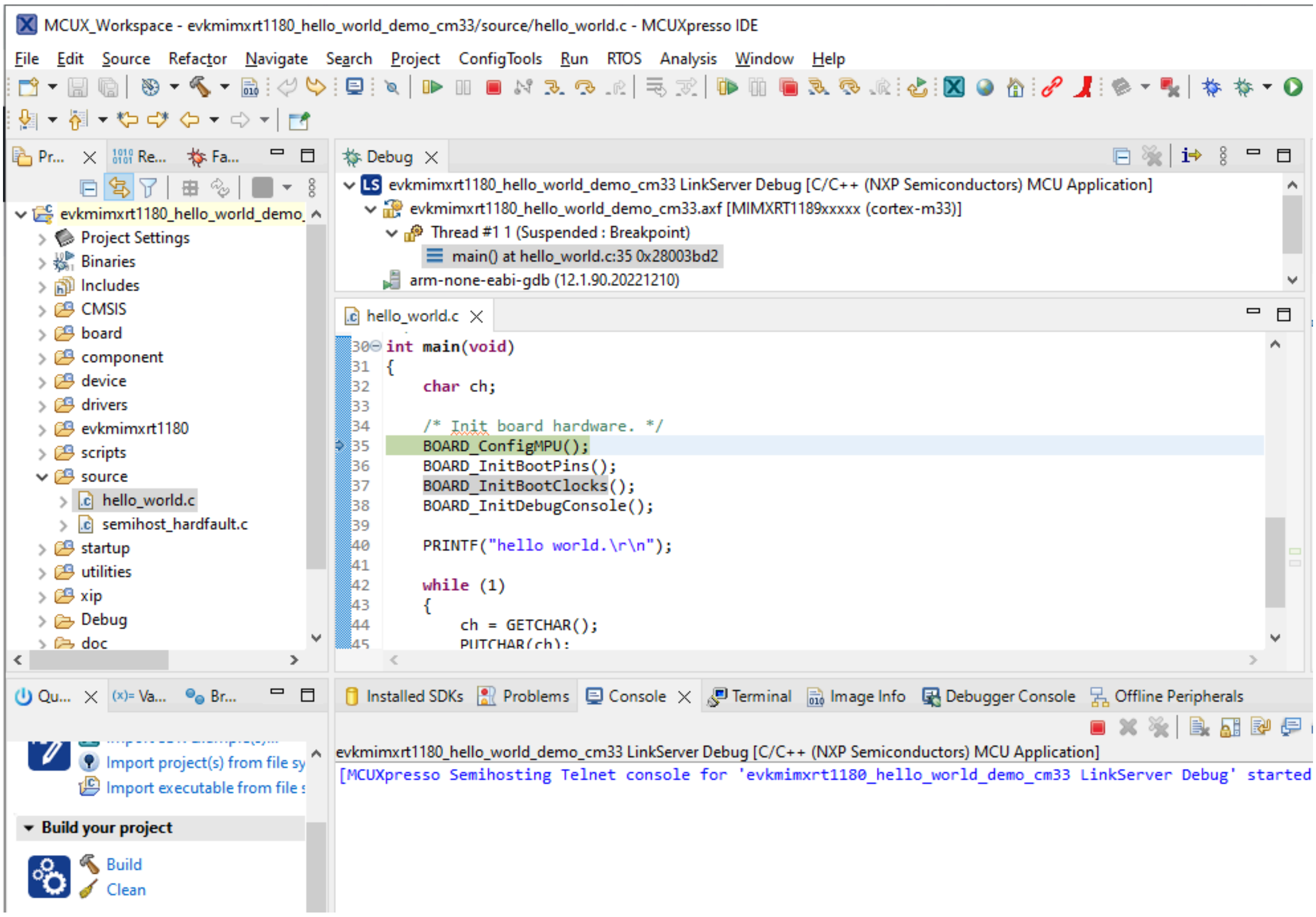


Figure 10. Stop at `main()` when running debugging

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

Figure 11. 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.

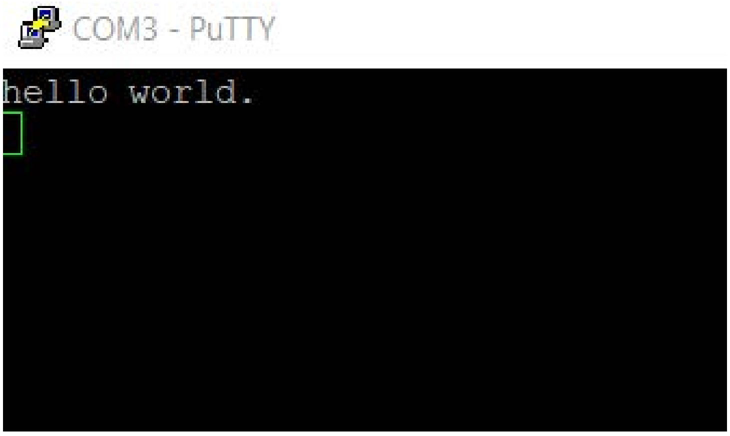


Figure 12. Text display of the `hello_world` demo

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 **evkmimxrt1180** 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 **evkmimxrt1180** is installed and available in the **Installed SDKs** view, click **Import SDK example(s)…** on the Quickstart Panel. In the window that appears, select **evkmimxrt1180** and click **Next**.

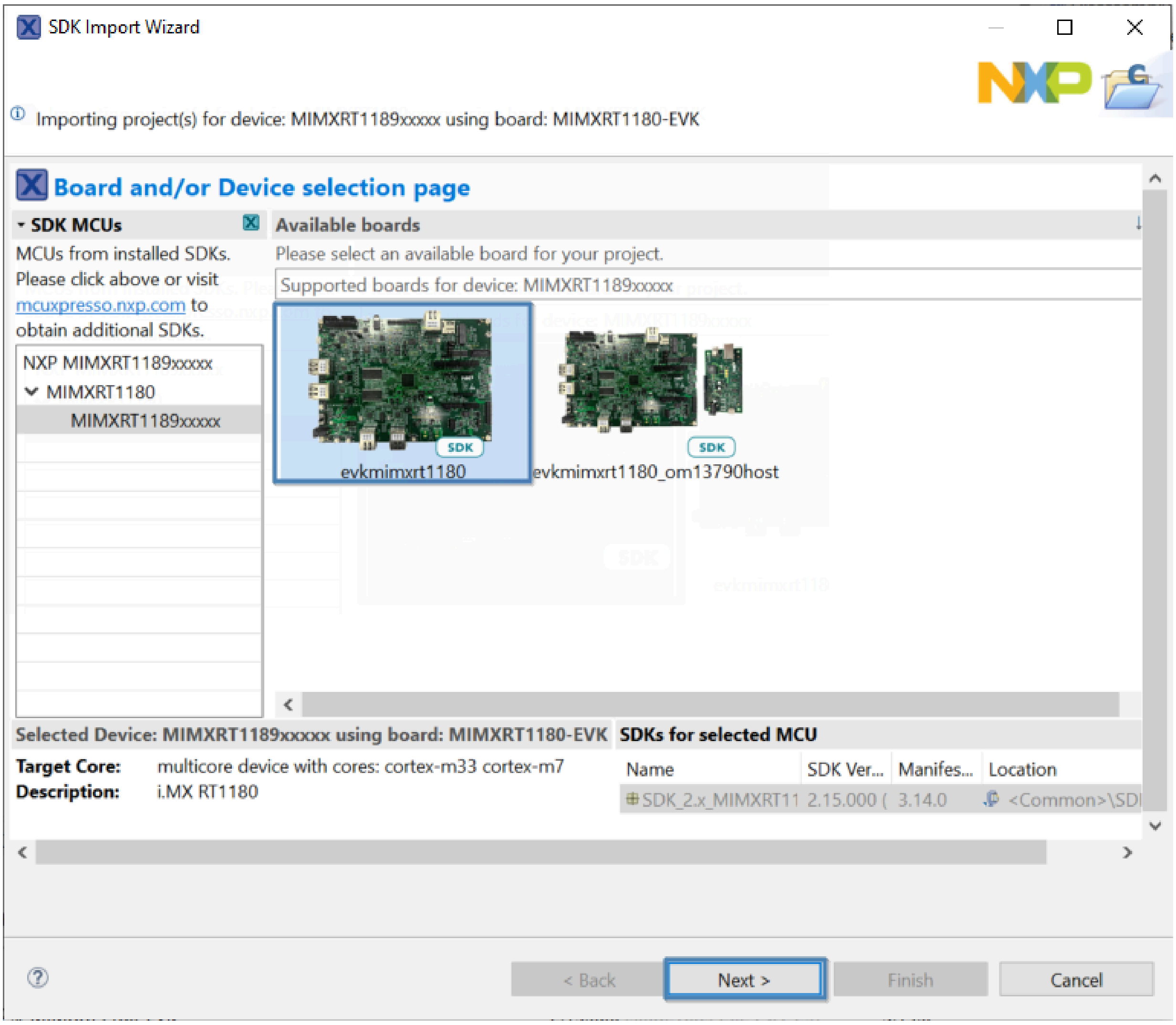


Figure 13. Select the evkmimxrt1180 board

2. Expand the `multicore_examples` folder and select **hello_world_cm33**. The `hello_world_cm7` counterpart project is automatically imported with the `cm33` project, because the multicore examples are linked together and there is no need to select it explicitly. Click **Finish**.

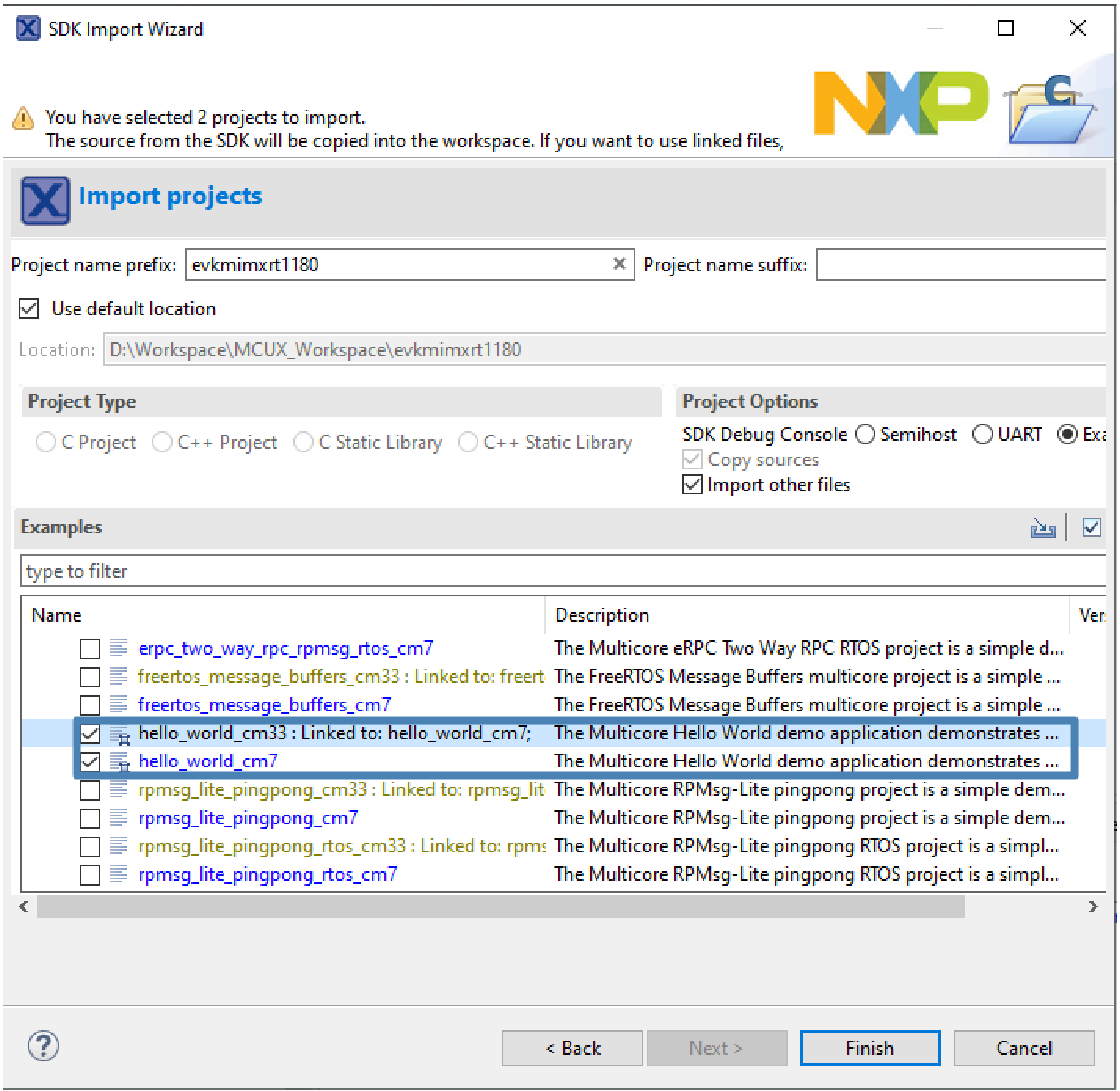


Figure 14. Select the hello_world multicore example

3. Now, two projects should be imported into the workspace. To start building the multicore application, highlight the `hello_world_cm33` 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 15. For this example, select **Debug**.

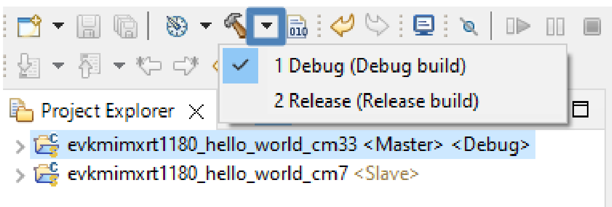


Figure 15. Selection of the build target in MCUXpresso IDE

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 (`cm33`) also makes the referenced auxiliary core application (`cm7`) 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 (`cm33`) build.*

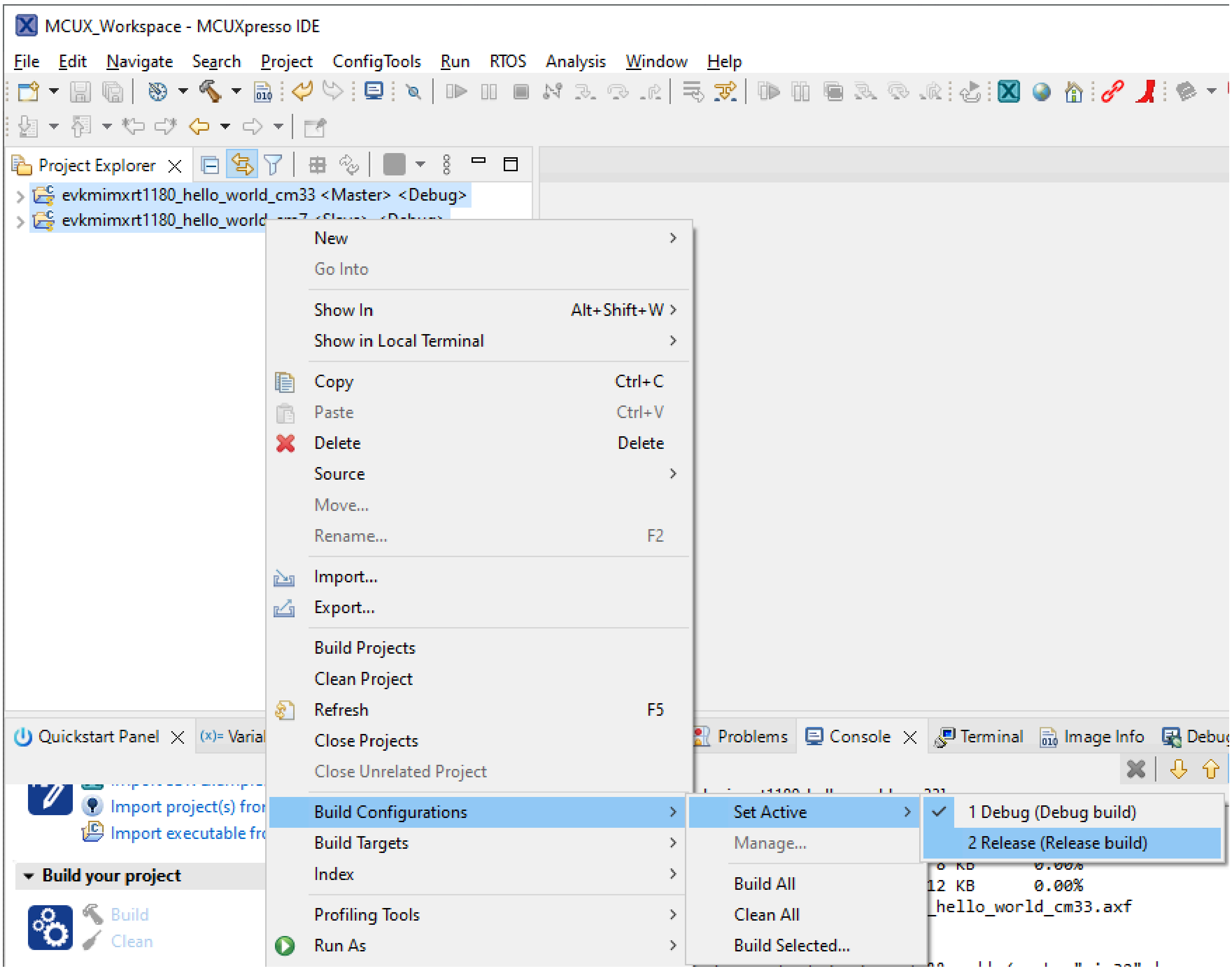


Figure 16. Switching multicore projects into the Release build configuration

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. Select the `cm7` project and start debugging the CM7 project. Then, select the `cm33` project and start debugging the CM33 project.

See Figure 17 to Figure 19 for reference.

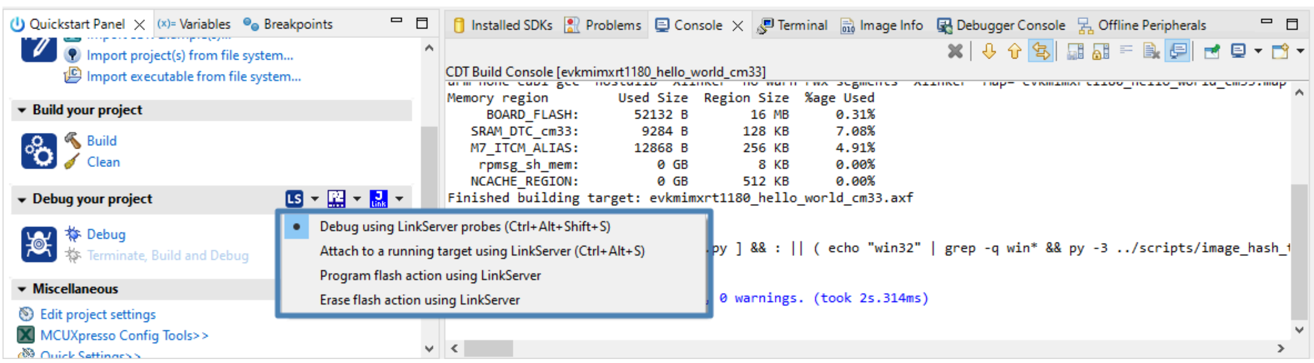


Figure 17. Debug hello_world_cm33 case

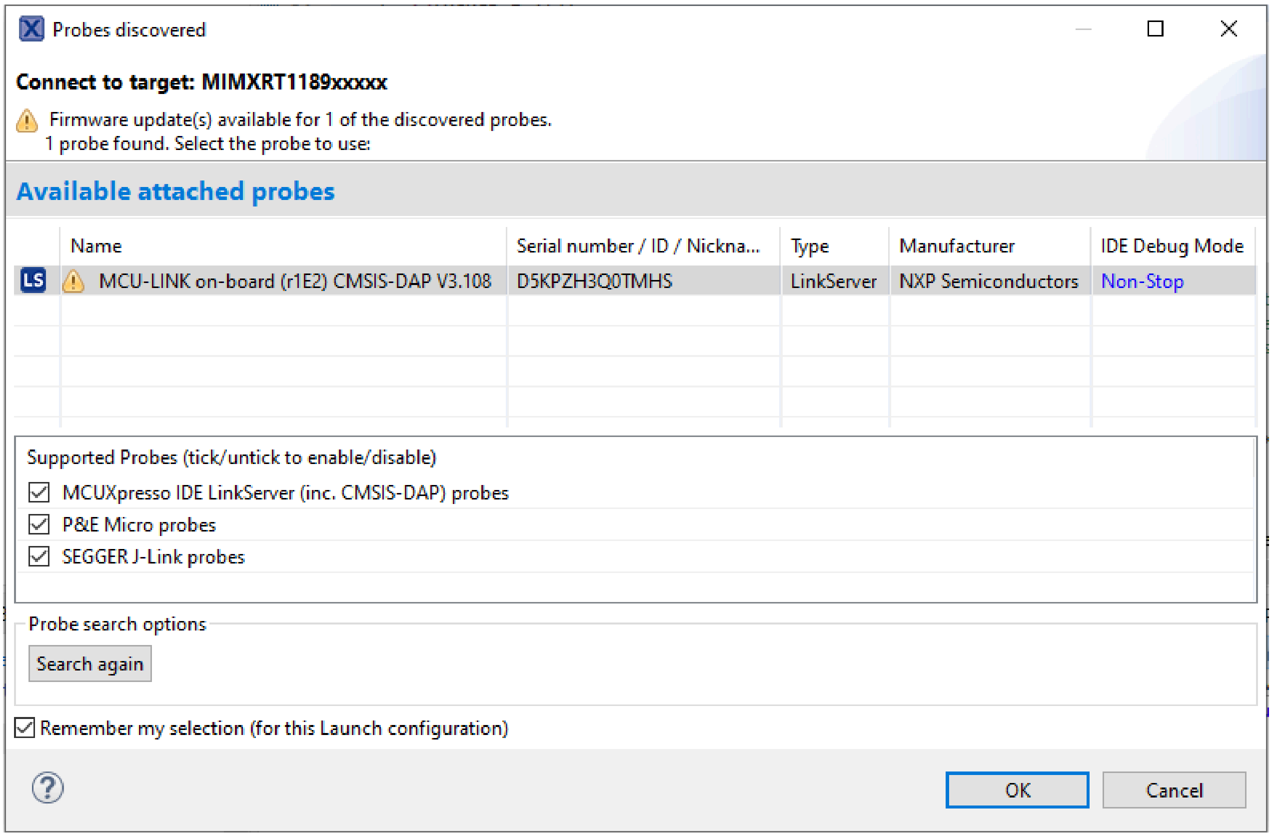


Figure 18. Attached Probes: debug emulator selection

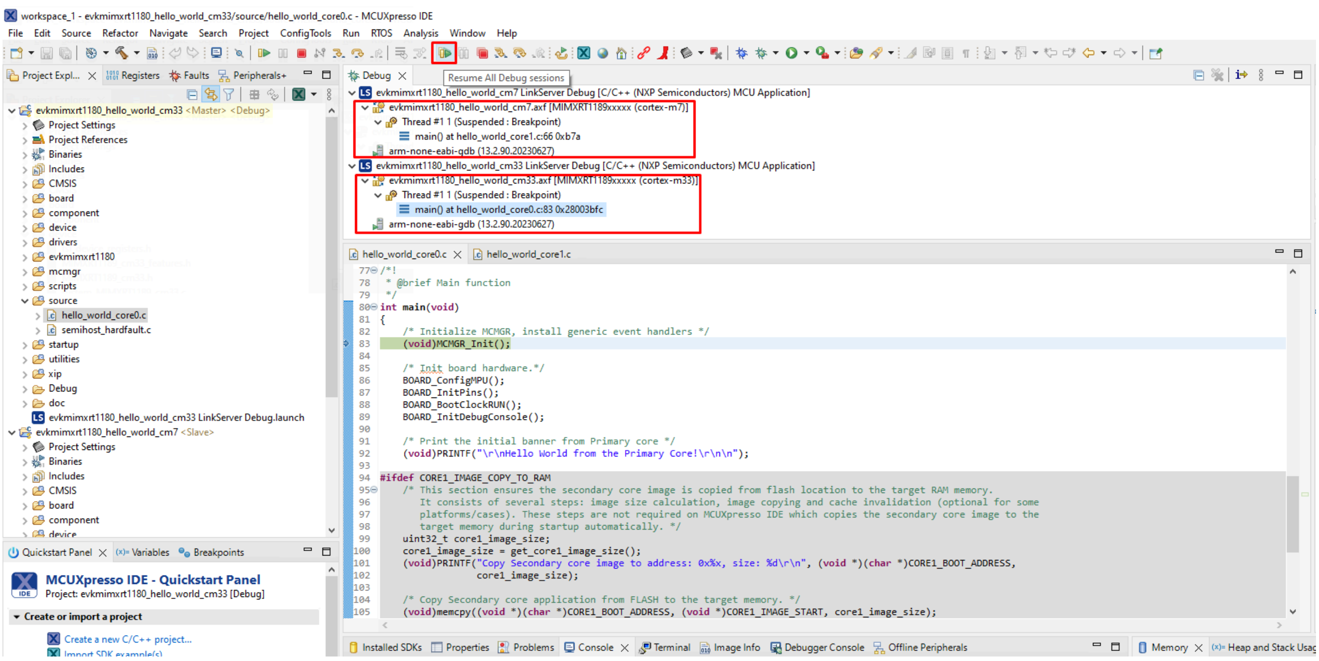


Figure 19. Stop the primary core application at main() when running debugging

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.

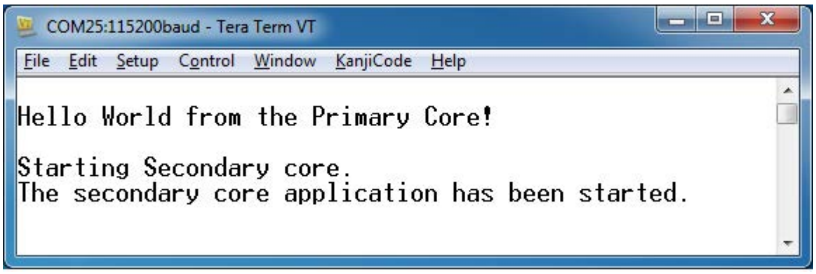


Figure 20. Hello World from the primary core message

3.6 Run applications via JLink Debug Interface

The previous sections assume that you are using the CMSIS-DAP debugging interface for MCUXpresso IDE to do the debugging. If you are using J-Link probe as a debugging interface, the configurations are slightly different.

1. After you click the **Debug** button, the IDE shows the J-Link probe identified. Click **OK** to continue the debugging.

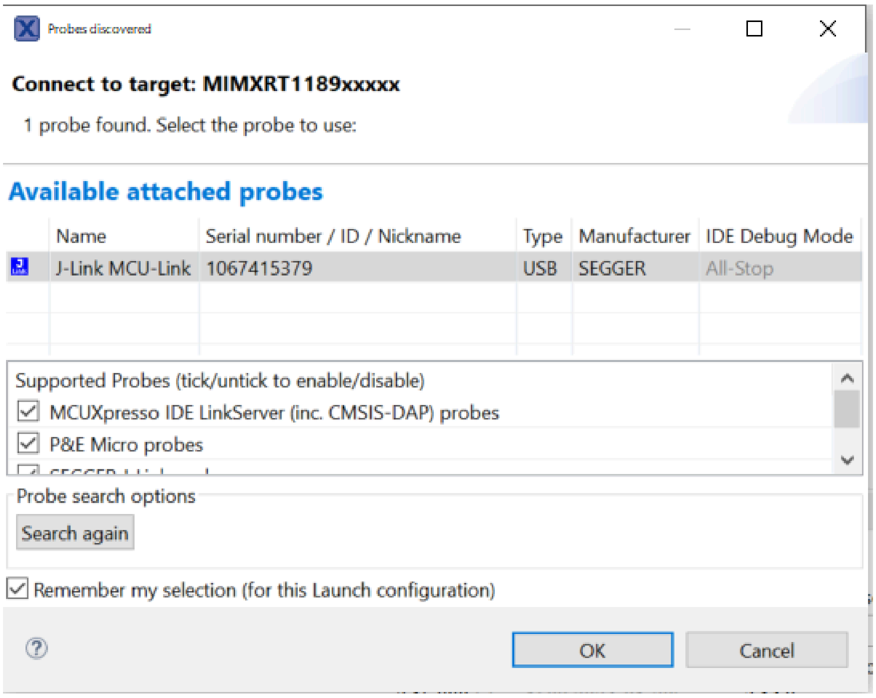


Figure 21. Identify JLink Probe in MCUX IDE

2. If you investigate on debug configurations, you will notice two settings, which are different from CMSIS-DAP debugging. The **Reset before running** is unchecked, and a jlinkscript is specified. This is because of the difference in CMSIS-DAP and JLink debugging implementation. The differences are pre-configured by MCUX default project settings.

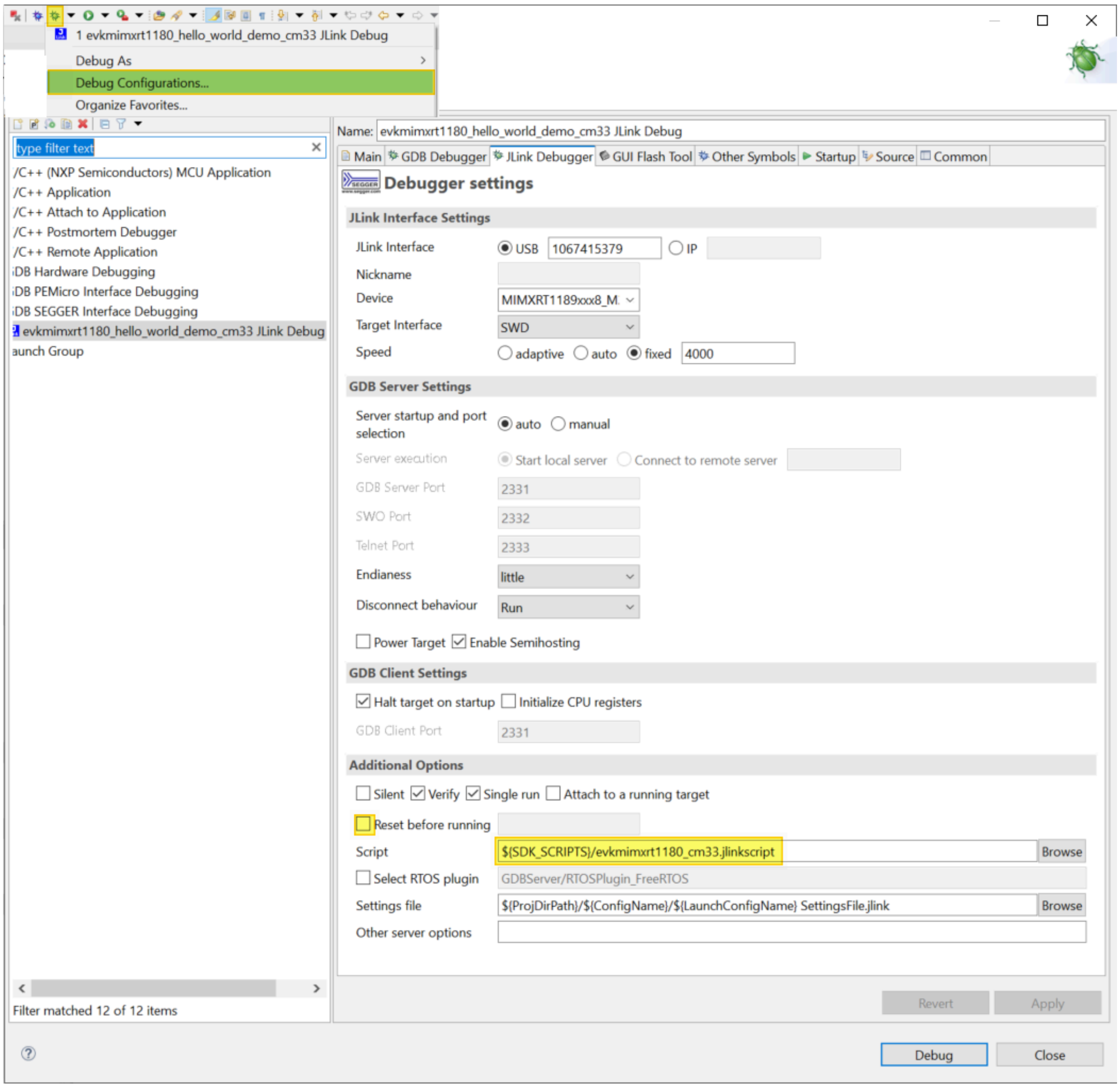


Figure 22. JLink Specific Setting in MCUX IDE

3. The screenshot here shows a CM33 demo. For CM7 demo, situations are similar.

4 Run a demo using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. This document uses `hello_world` demo application targeted for the MIMXRT1180-EVK as an example. These steps can be applied to any example application in the MCUXpresso SDK.

Both CMSIS-DAP and J-Link debugging interfaces are supported for IAR IDE. It is recommended to set `SW5[1..4]` to `0001` for both debugging interfaces. It is required to reset board for each download/debug.

4.1 Build an example application

To build the `hello_world` demo application, perform the following steps:

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 MIMXRT1180-EVK hardware platform as an example, the `hello_world` workspace is located in: *<install_dir>/boards/evkmimxrt1180/demo_apps/hello_world/cm33/iar/hello_world_demo_cm33.eww*
Other example applications may have additional folders in their path.

2. Select the desired build target from the drop-down menu.
The below lists 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.
   - `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.
   - `hyperram_debug` – Project configuration is same as the debug target. The linker file is HYPERRAM linker, where text is put in internal TCM and data put in HYPERRAM.
   - `hyperram_release` – Project configuration is same as the release target. The linker file is HYPERRAM linker, where text is put in internal TCM and data put in HYPERRAM.
   - `hyperram_txt_debug` – Project configuration is same as the debug target. The linker file is `HYPERRAM_txt` linker, where text and data section is put in HYPERRAM.
   - `hyperram_txt_release` – Project configuration is same as the release target. The linker file is `HYPERRAM_txt` linker, where text and data section is put in HYPERRAM.
   - `flexspi_nor_hyperram_release` - Project configuration is same as the release target. The linker file is `flexspi_nor_hyperram` linker, where text is put in flash and data put in HYPERRAM.
   - `flexspi_nor_hyperram_debug` – Project configuration is same as the debug target. The linker file is `flexspi_nor_hyperram` linker, where text is put in flash and data put in HYPERRAM.

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

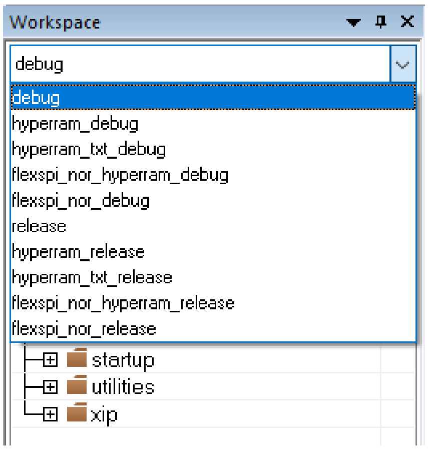


Figure 23. Demo build target selection

3. To build the demo application, click Make, highlighted in red in Figure 24.

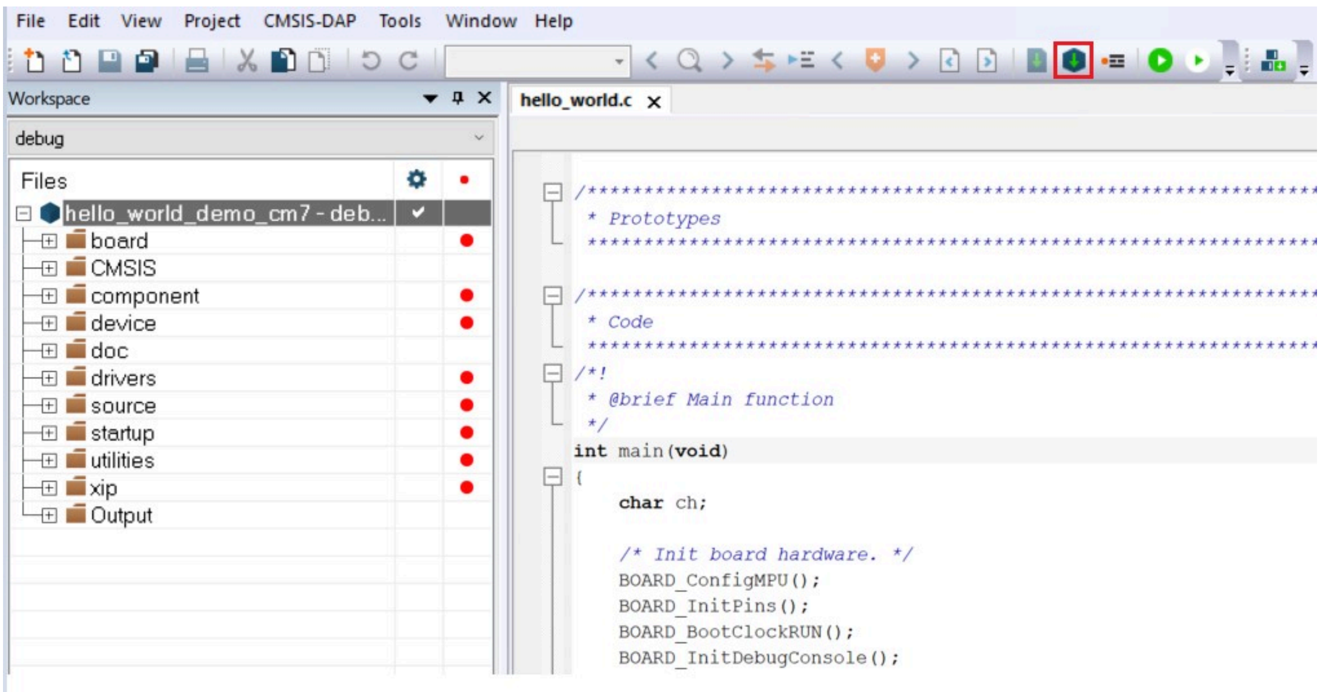


Figure 24. Build the demo application

4. The build completes without errors.

4.2 Run an example application

The default IAR project setting assumes that CMSIS-DAP debug interface is used. When such a debug interface is used, perform the following steps to do the debugging:

1. Connect the development platform to your PC via USB cable. Connect the USB cable to **J53** for serial output.
2. 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 9). Configure the terminal with these settings:
   a. 115200 baud 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

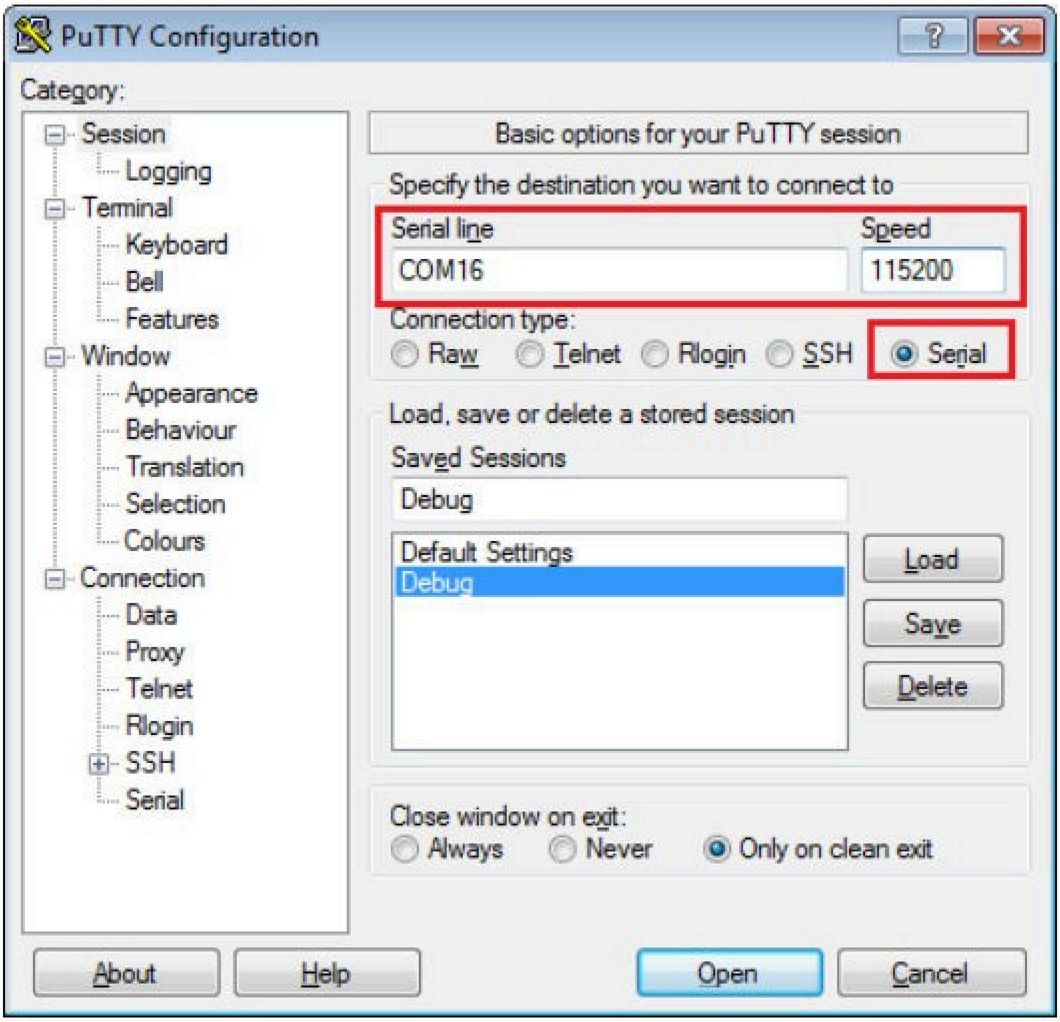


Figure 25. Terminal (PuTTY) configuration

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

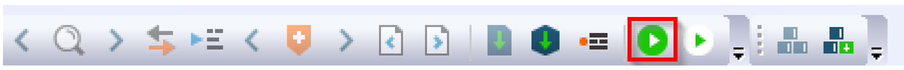


Figure 26. Download and Debug button

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

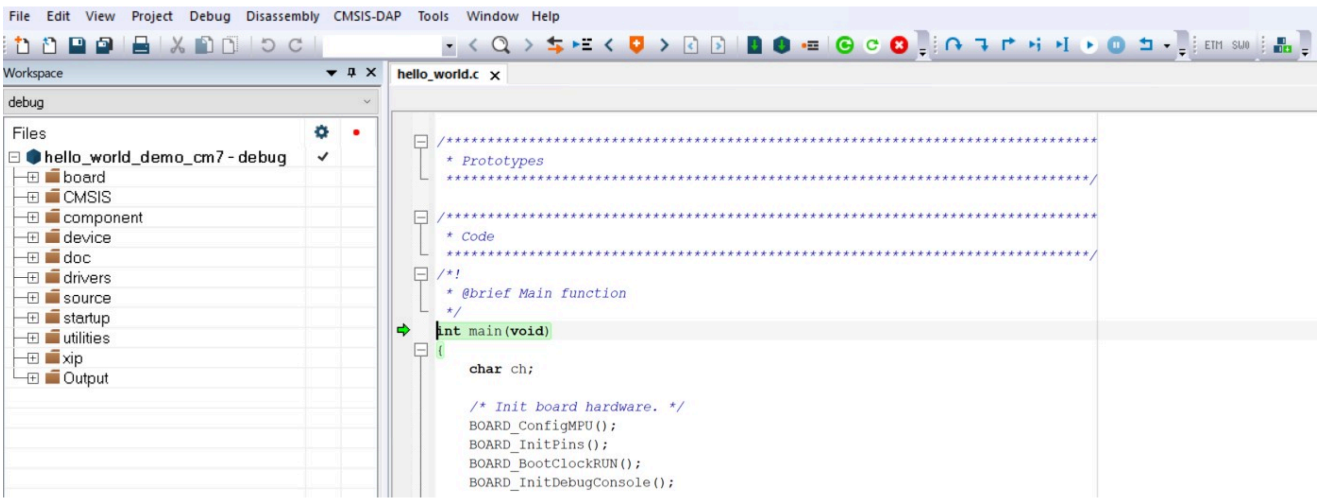


Figure 27. Stop at main() when running debugging

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

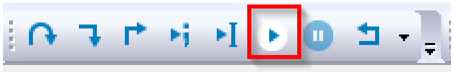


Figure 28. Go button

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

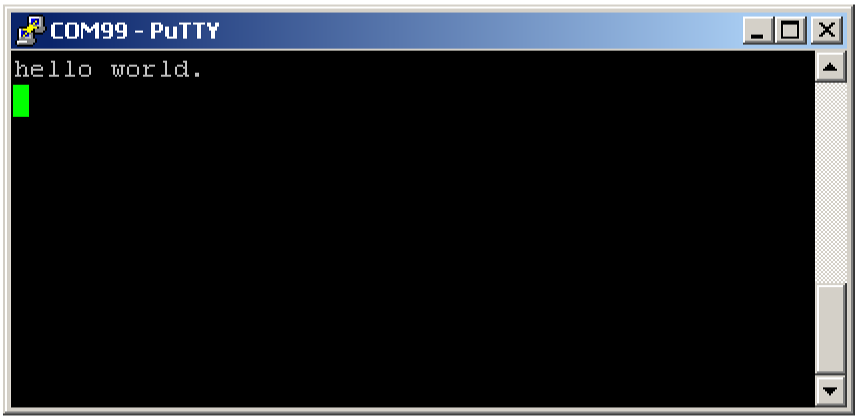


Figure 29. Text display of the hello_world demo

4.3 Build and run a multicore example application

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

`<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm7/iar/hello_world_cm7.eww`

`<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm33/iar/hello_world_cm33.eww`

Build both applications separately by clicking the **Make** button. Build the application for the auxiliary core (cm7) first, because the primary core application project (cm33) must know the auxiliary core application binary when running the linker. When the auxiliary core application binary is not ready, it is impossible to finish the primary core linker.

By default, the primary core flexspi_nor_debug target links the auxiliary core debug target, and the primary core flexspi_nor_release target links the auxiliary core release target. During the primary core execution, the auxiliary core image is copied from flash into the CM7 RAM and executed.

1. Build the CM7 and CM33 projects respectively.
2. Only click the `Download and Debug` button on the CM33 project, IAR could help start to debug a multicore project. It is user-friendly to debug multicore examples with CMSIS-DAP on IAR (Multicore Project is set on Debugger->Multicore window).

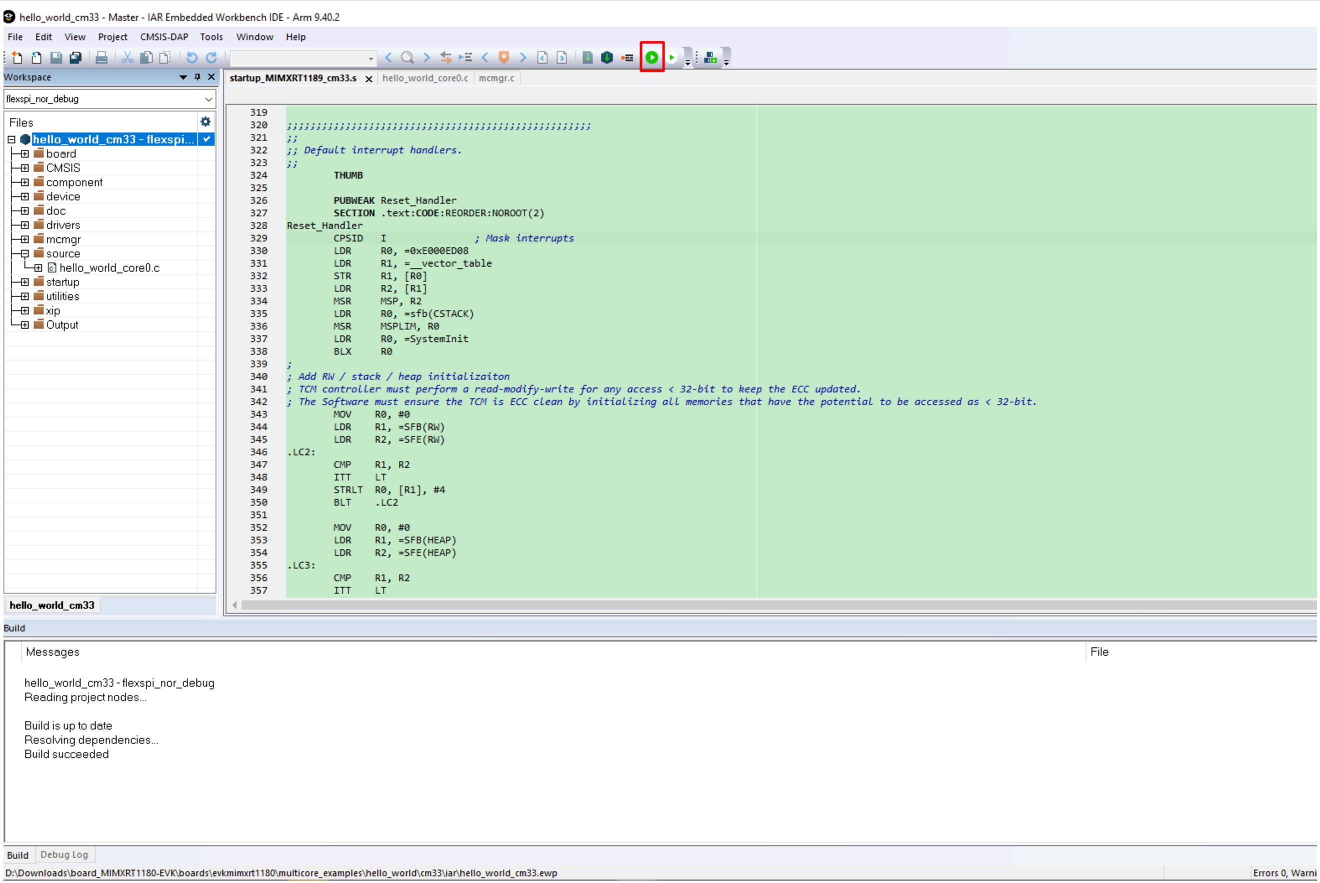


Figure 30. Debug multicore project

3. Start `core0` and then start `core1` on the CM33 project.

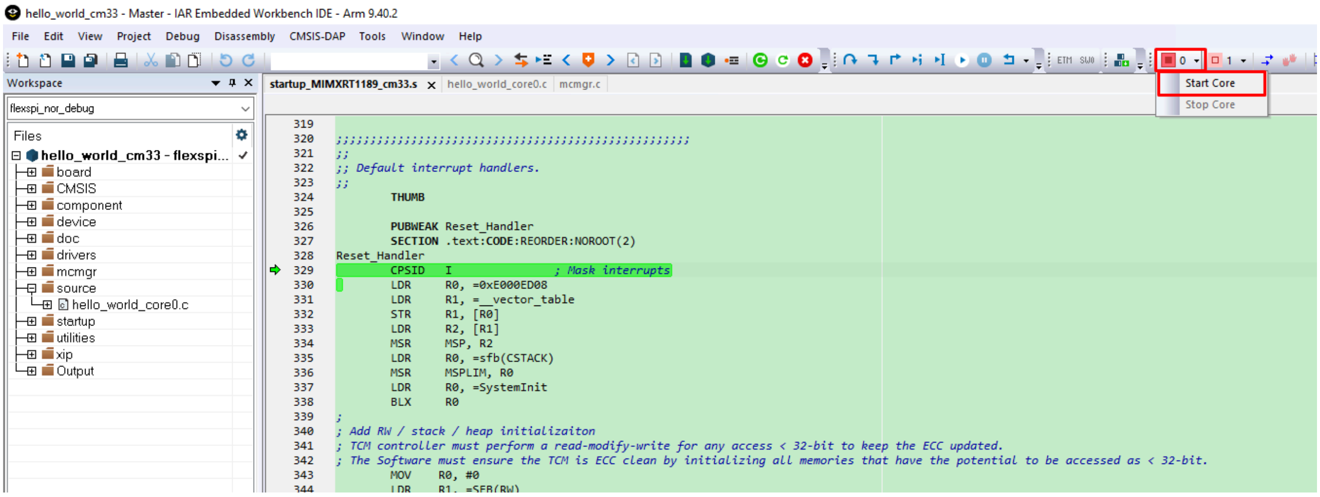


Figure 31. Start core0

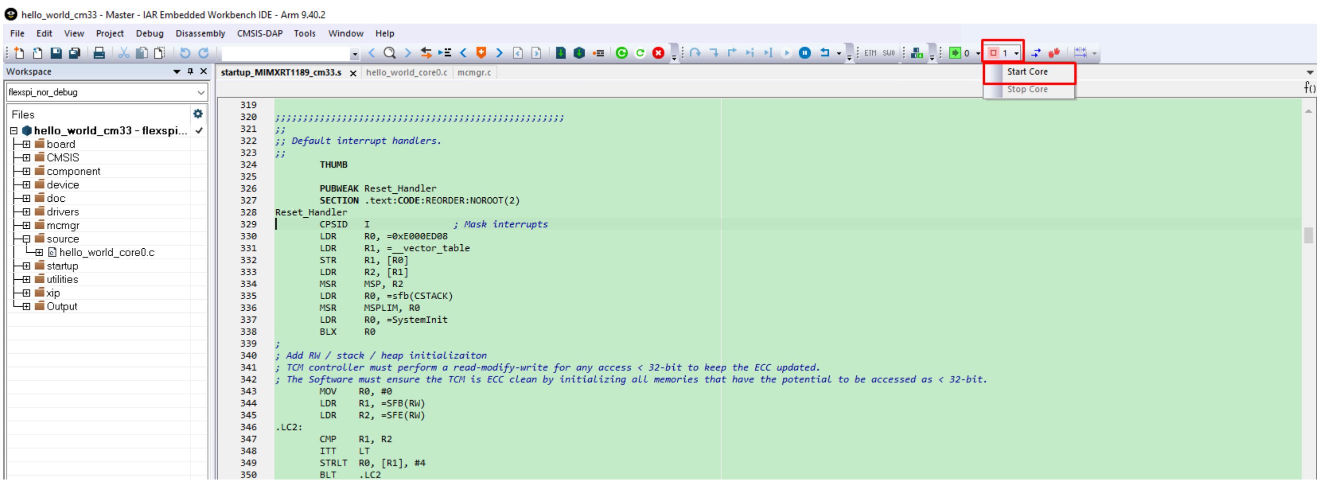


Figure 32. Start core1

4. `Hello_World` multicore demos are now running. A banner appears on the terminal and the LED D6 blinks. If this is not true, check your terminal settings and connections.

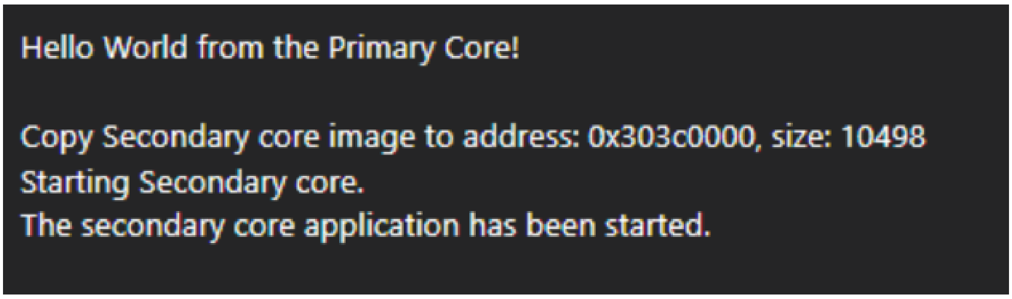


Figure 33. The banner appears when multicore demos run successfully

4.4 Run applications via JLink debug interface

When JLink Debug Interface is used, manually revise **Debug Configuration** under **Project** > **Options** > **Debugging**.

1. Change the debug driver to `J-Link/J-Trace`.

2. Uncheck **Use macro file(s)**.
3. In the **Extra Options**, specify a `jlinkscript`. The `jlinkscript` files can be found in SDK package *boards/evkmimxrt1180/jlinkscript*.
   `evkmimxrt1180_cm33.jlinkscript` is for CM33 projects and `evkmimxrt1180_cm7.jlinkscript` is for CM7 projects. Use `--jlink_script_file=<full patch>` to specify it.

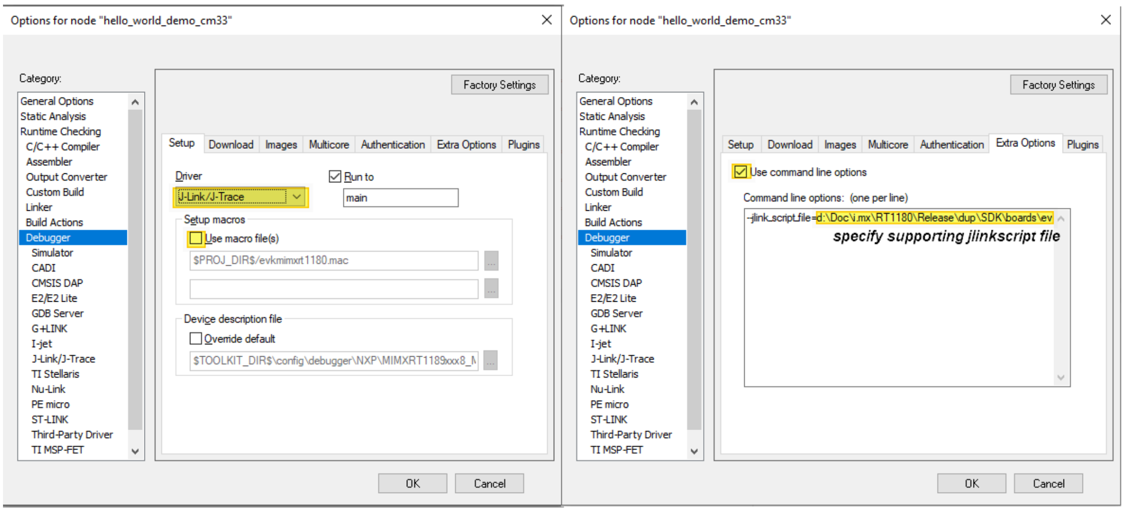


Figure 34. IAR Debugging via JLink

After these settings, debugging can be performed via the J-Link debugging interface.

Note: *If IAR version is higher than or equal to 9.60.1, perform just [Step 1](#).*

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.

Both CMSIS-DAP and J-Link debugging interfaces are supported for Keil IDE. It is recommended to set `SW5[1..4]` to `0001` for both debugging interfaces. It is required to reset board for each download/debug.

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 MIMXRT118x CMSIS pack.

1. Download the MIMXRT1181, MIMXRT1182, MIMXRT1187, and MIMXRT1189 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/evkmimxrt1180/demo_apps/hello_world/cm33/mdk/hello_world_demo_cm33.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

The default MDK project settings assumes CMSIS-DAP debug interface is used. If you are using a CMSIS-DAP debugging interface, perform the following steps:

1. Connect the development platform to your PC via USB cable.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number. To determine the COM port number, see Section 9. 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

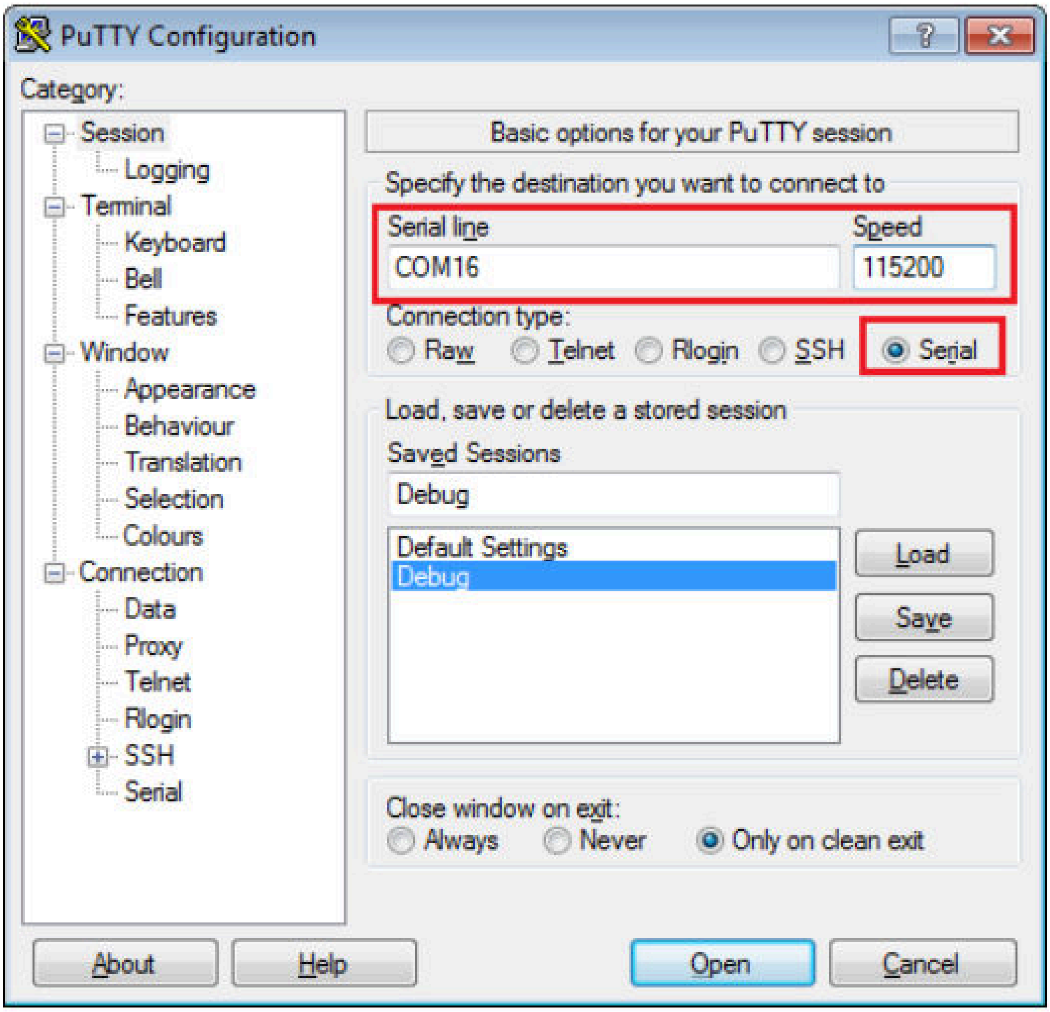


Figure 36. Terminal (PuTTY) configurations

3. To debug the application, click the **Start/Stop Debug Session** button or **Ctrl** + **F5**, highlighted in Figure 37.

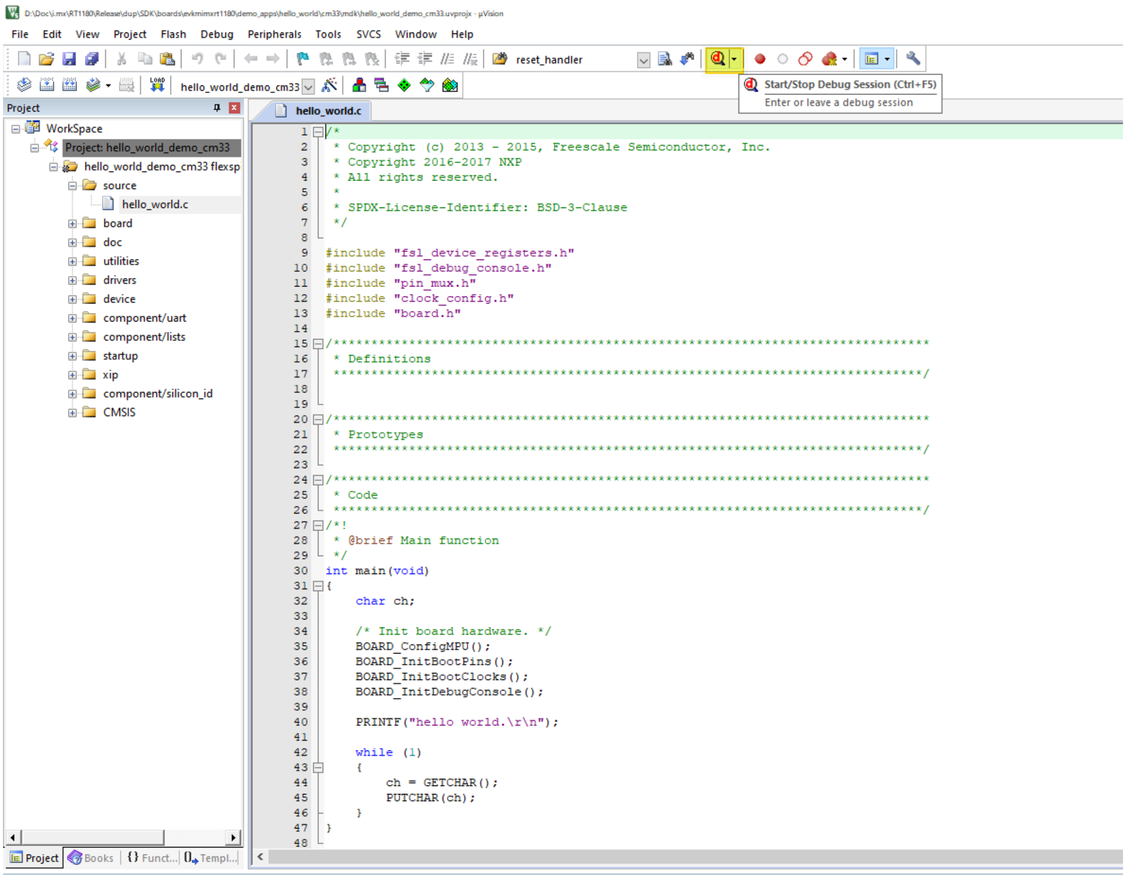


Figure 37. Stop at `main()` when run debugging

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

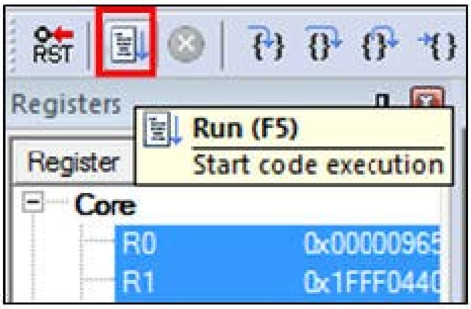


Figure 38. Run button

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/evkmimxrt1180/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/evkmimxrt1180/multicore_examples/hello_world/cm7/mdk/hello_world_cm7.uvmpw

<install_dir>/boards//evkmimxrt1180/multicore_examples/hello_world/cm33/mdk/hello_world_cm33.uvmpw

Build both applications separately by clicking the **Rebuild** button. Build the application for the auxiliary core (cm7) first, because the primary core application project (cm33) 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 CM7 RAM and executed.

5.5 Run a multicore example application

This section describes the steps to run a multicore example application. The primary core debugger flashes both the primary and the auxiliary core applications into the SoC flash memory.

1. To download and run the multicore application, switch to the primary core application project and perform Step 1 – Step 3 as described in Section 5.4.
2. For the secondary core project, select the `Load Application at Startup` button as shown in Figure 40. These steps are common for both single-core and dual-core applications in µVision.

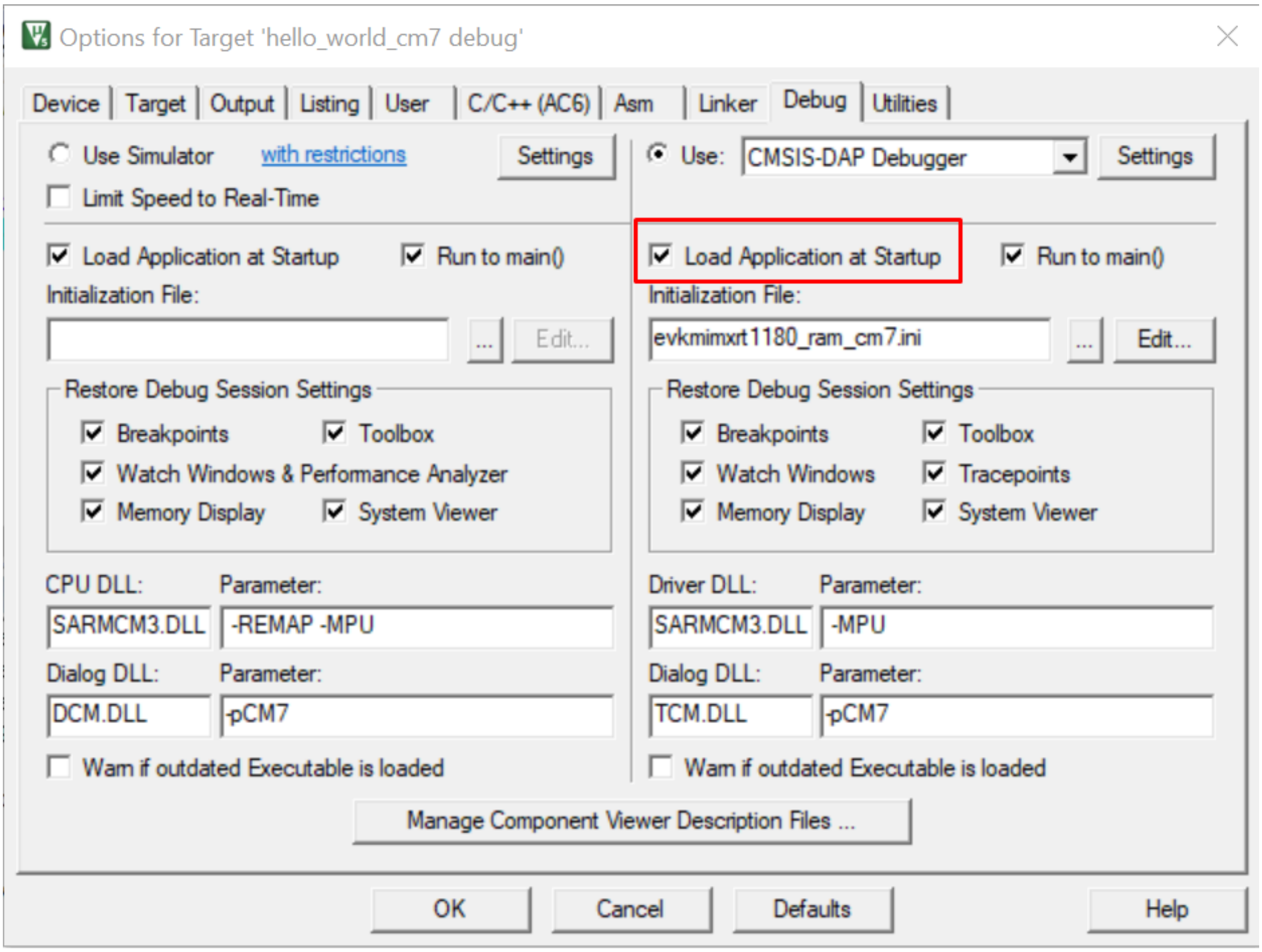


Figure 40. Load Application at Startup

3. Run the primary core project and then run the secondary core project.

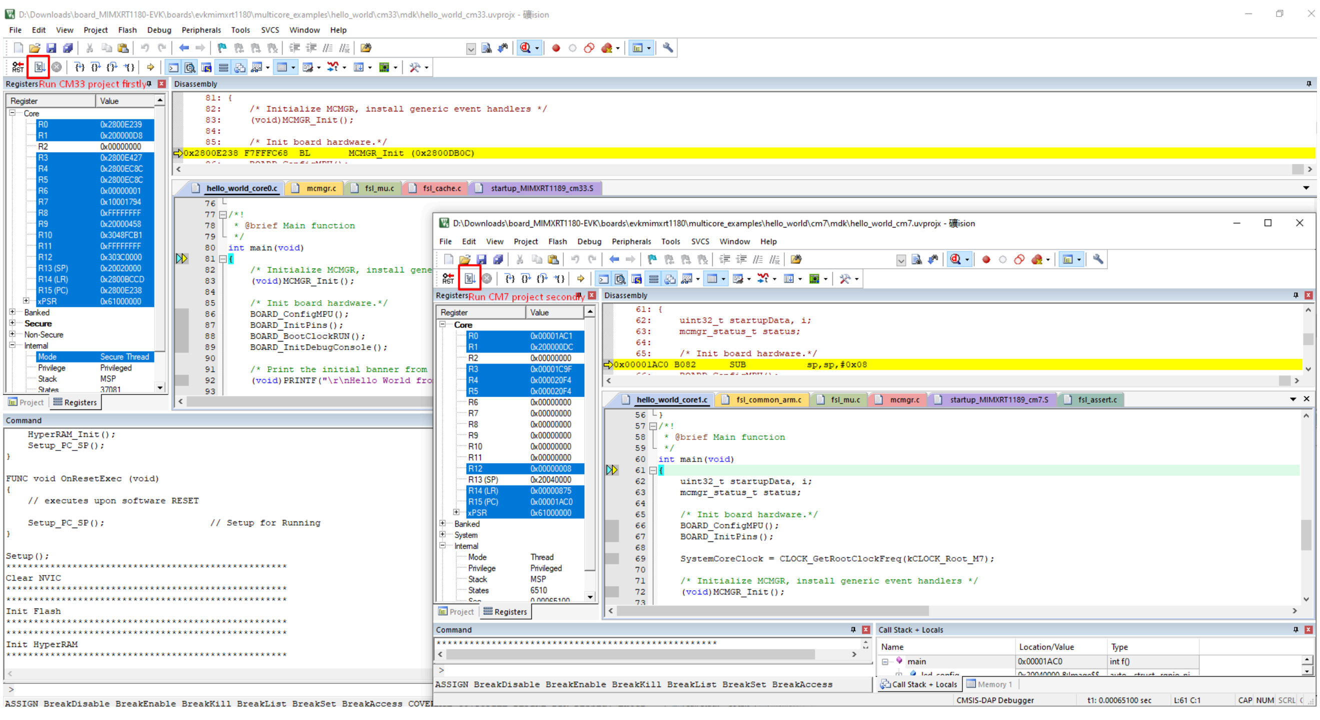


Figure 41. Stop the primary/auxiliary core application at main() when debugging

4. `Hello_World` multicore demos are now running. A banner appears on the terminal and the LED D6 blinks. If this is not true, check your terminal settings and connections.

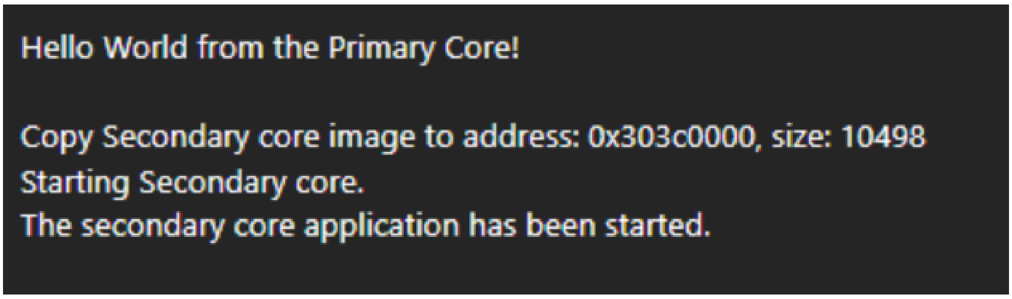


Figure 42. The banner appears when multicore demos run successfully

5.6 Run applications via JLink debug interface

When JLink Debug Interface is used, manually revise **Debug Configuration**.

1. Under **Project** > **Options** > **Debug**, change the debug interface to `J-LINK / J-TRACE Cortex`.

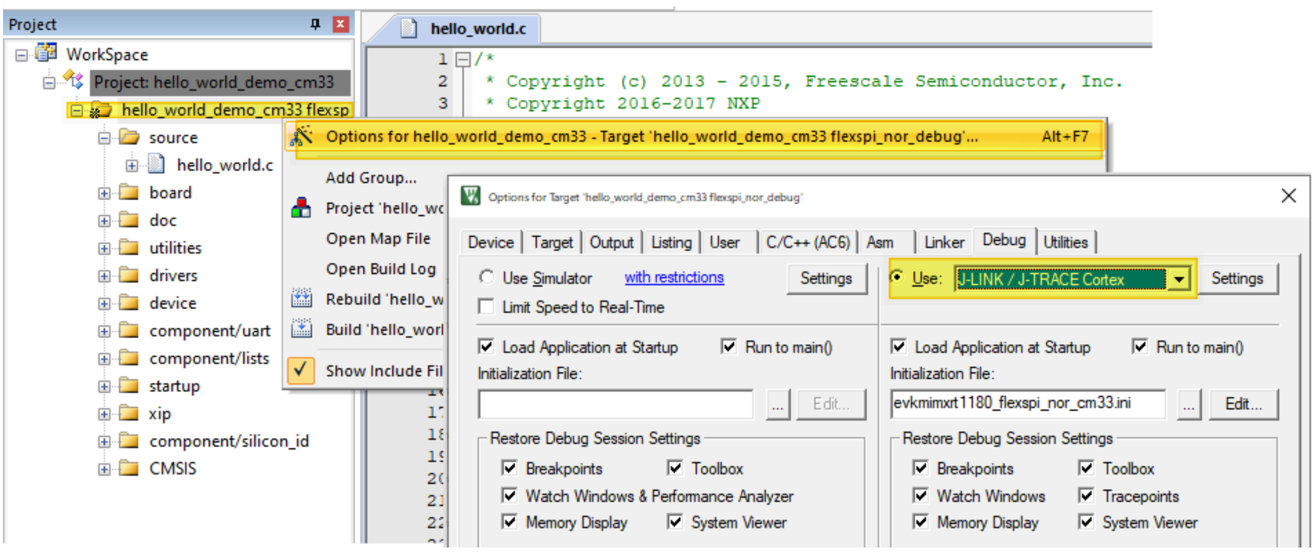


Figure 43. Change KEIL debugging interface

2. Copy the supporting `JLinkscript` file from *boards/evkmimxrt1180/jlinkscript* to project root `dir`. For CM33 projects, use `evkmimxrt1180_cm33.jlinkscript` and for CM7 projects, use `evkmimxrt1180_cm7.jlinkscript`. Rename them to `JLinkSettings.JLinkScript`.

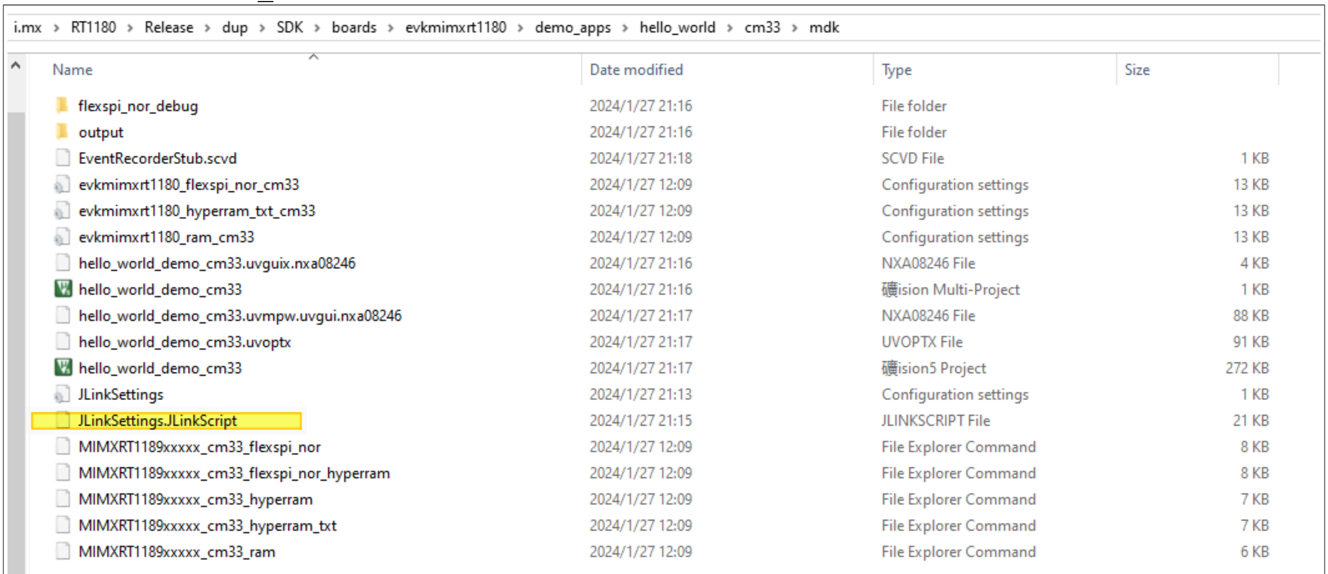


Figure 44. Rename supporting JLinkscript

After these settings, debugging can be performed via the J-Link debugging interface.

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.

Only J-Link debugging interface is supported for JLinkGDBServer and GDB. It is recommended to set `SW5[1..4]` to `0001` when using these tools to debug. It is required to reset board for each download/debug.

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 MIMXRT1180-EVK* (document MCUXSDKMIMXRT118XKRN).

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**.

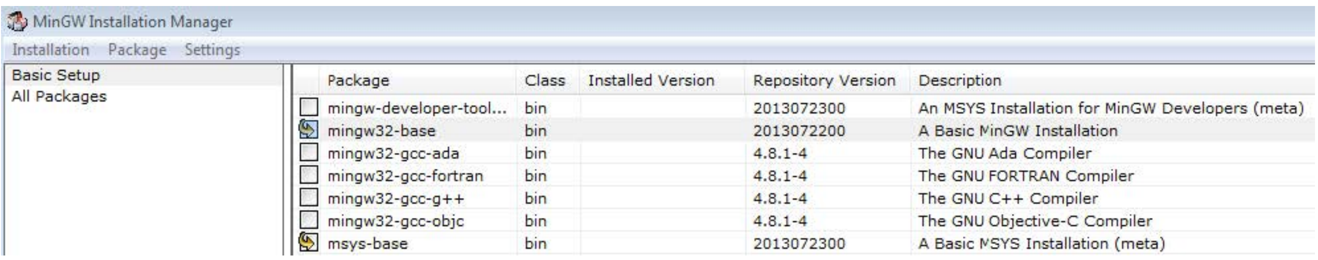


Figure 45. Set up MinGW and MSYS

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

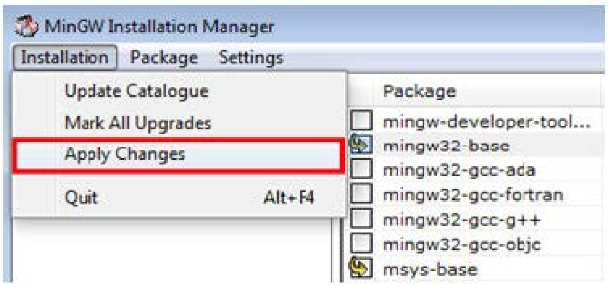


Figure 46. Complete MinGW and MSYS installation

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 47. 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.*

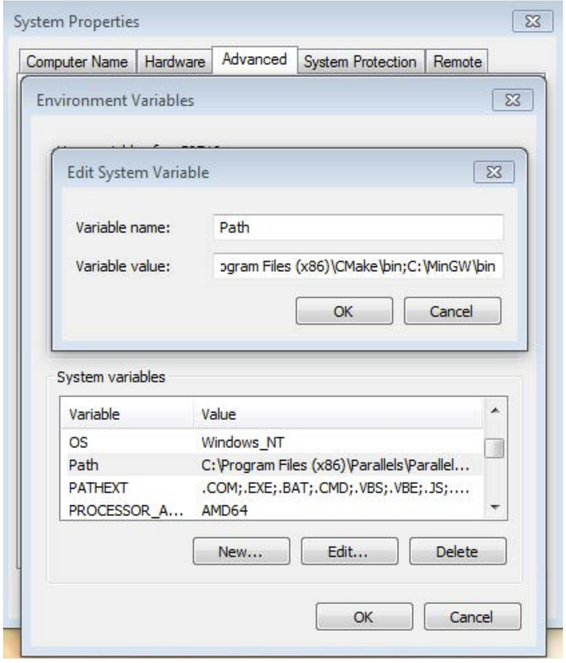


Figure 47. Add Path to systems environment

6.1.3 Add a system environment variable for ARMGCC_DIR

Create a `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.

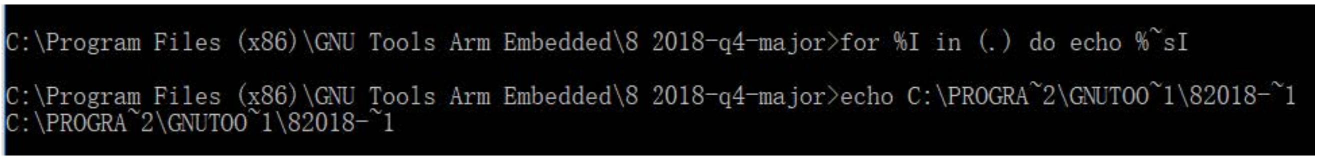


Figure 48. Convert path to short path

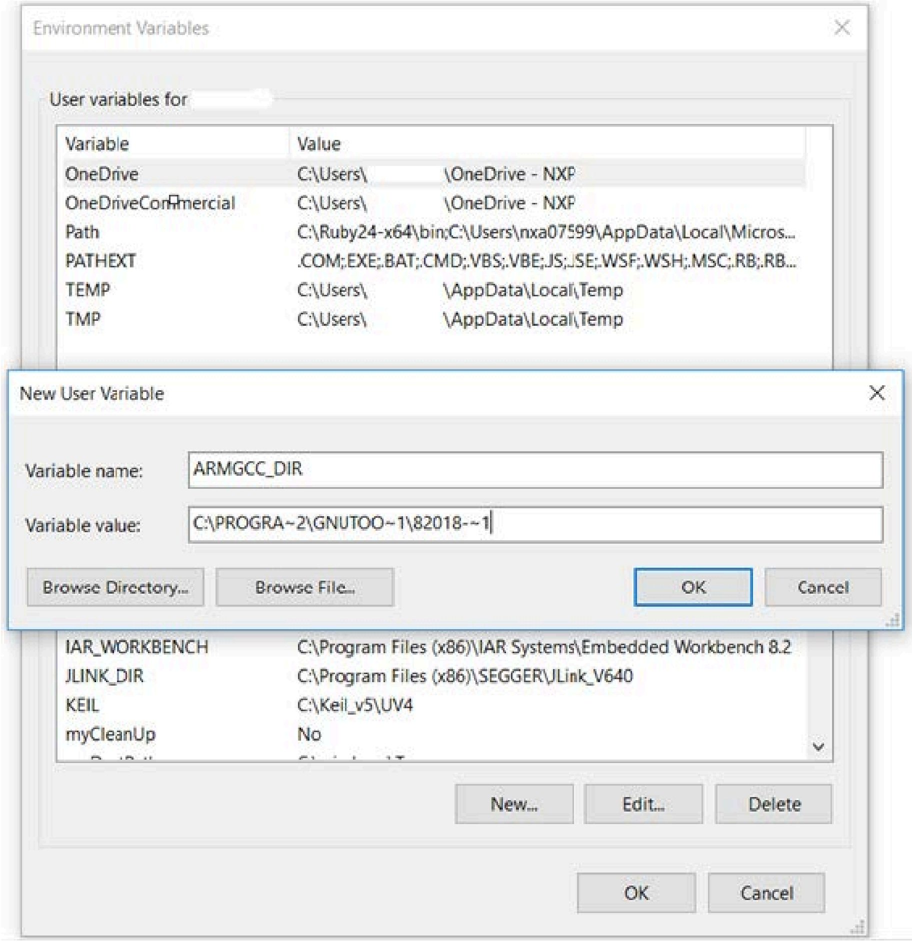


Figure 49. Add ARMGCC_DIR system variable

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.

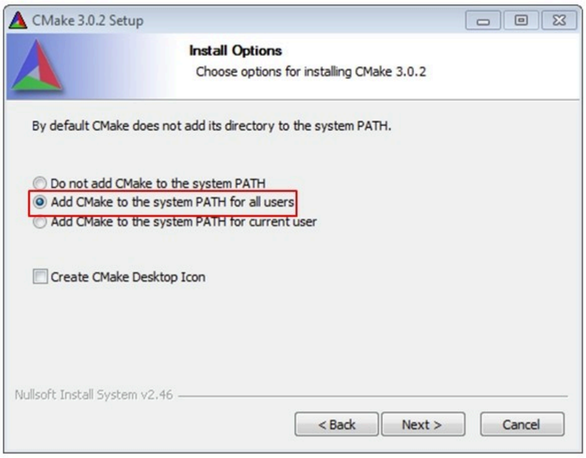


Figure 50. Install CMake

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, perform the following 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**.

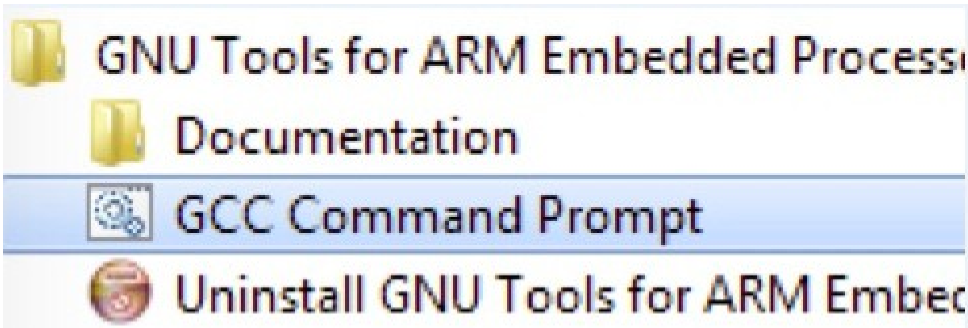


Figure 51. Launch command prompt

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/evkmimxrt1180/demo_apps/hello_world/cm33/armgcc
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 52.

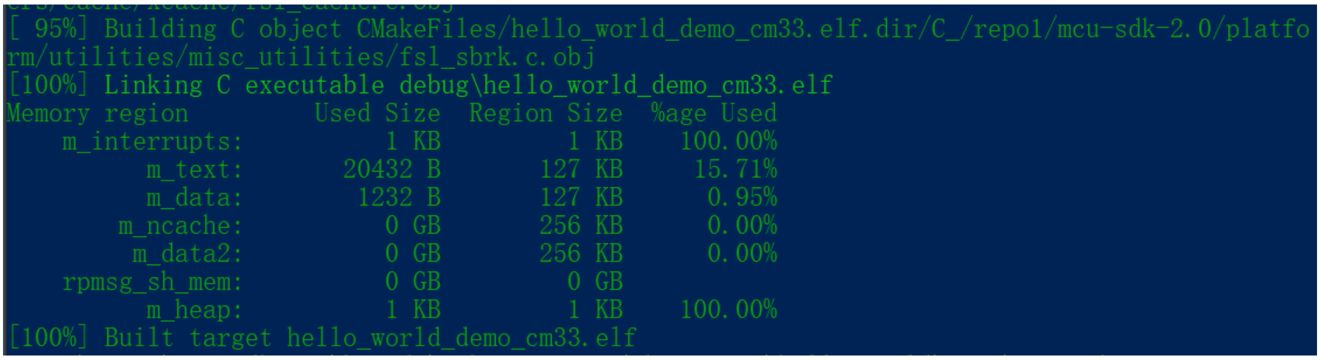


Figure 52. hello_world demo build successful

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 a standalone J-Link pod is connected to the debug interface of your board.

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.
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 9). 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

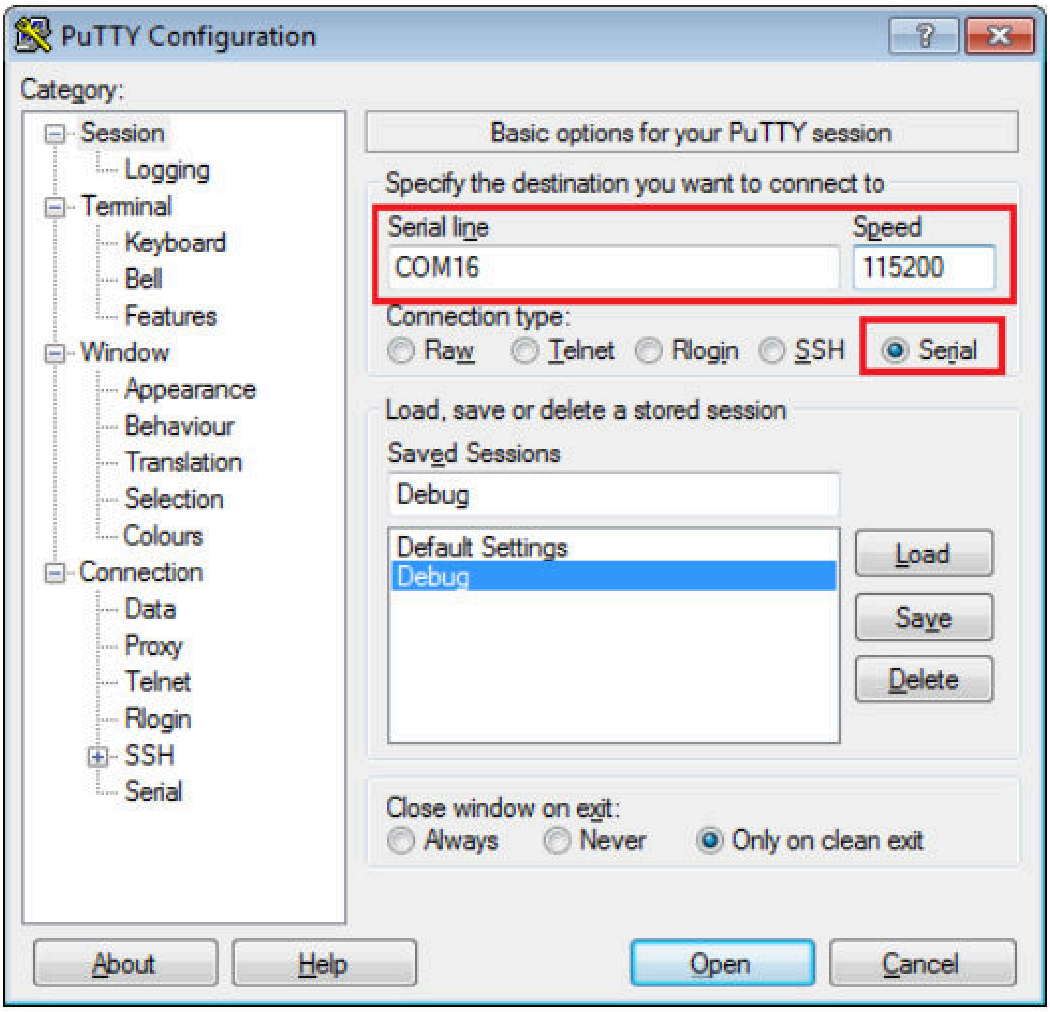


Figure 53. Terminal (PuTTY) configurations

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. To debug CM33, run the following command:

```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M33 -if SWD -speed 4000 -
jlinkscriptfile <evkmimxrt1180_cm33.jlinkscript> -stayontop -ir
```

   To debug CM7, run the following command:

```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M7 -if SWD -speed 4000 -
jlinkscriptfile <evkmimxrt1180_cm7.jlinkscript> -stayontop -ir
```

   Note: *The supporting* `jlinkscript` *file can be found in boards/evkmimxrt1180/jlinkscript.*

5. After it is connected, the screen should resemble Figure 54.

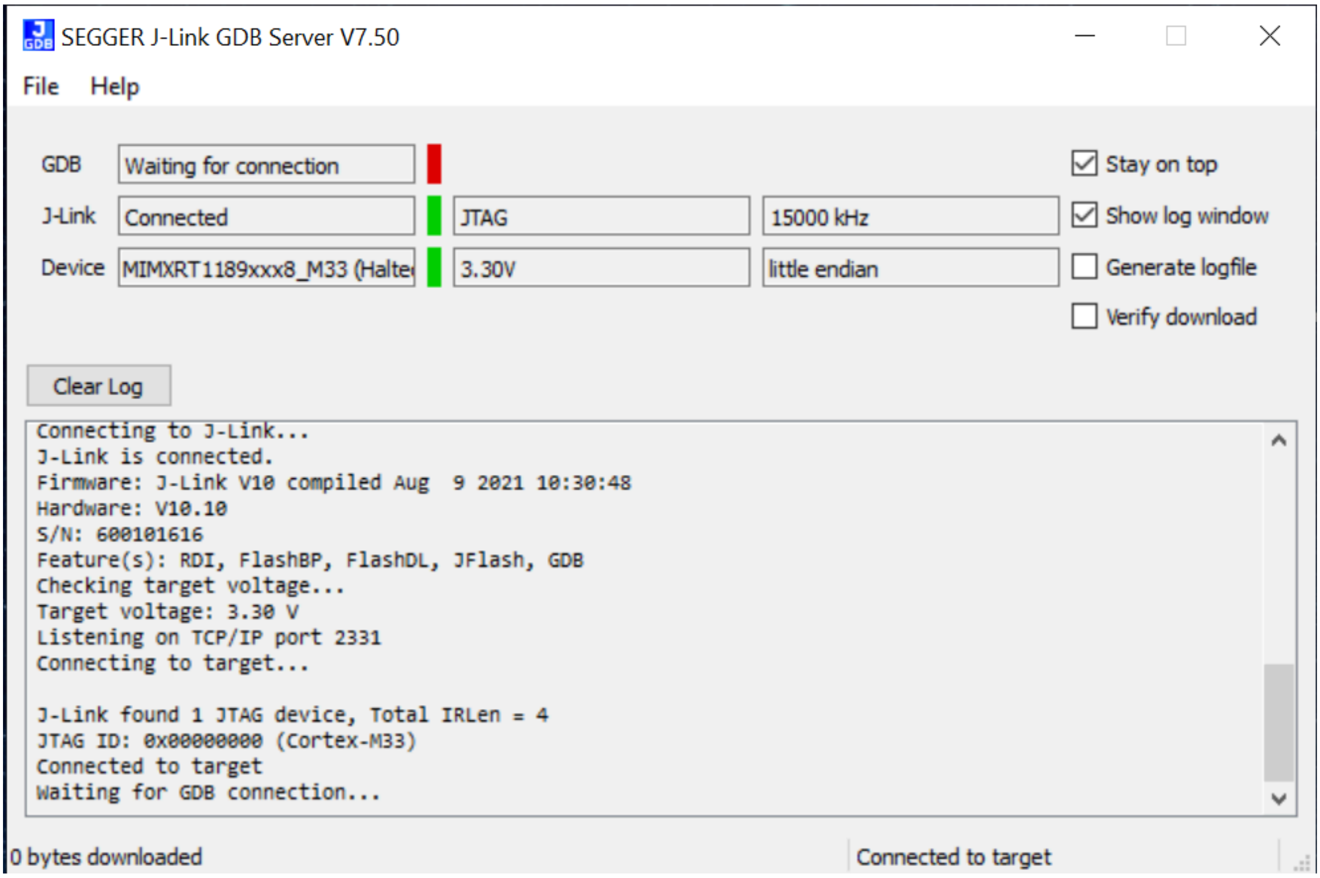


Figure 54. SEGGER J-Link GDB Server screen after successful connection

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

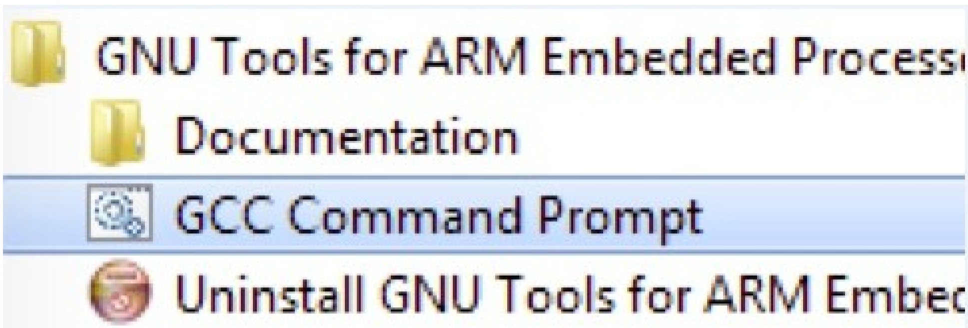
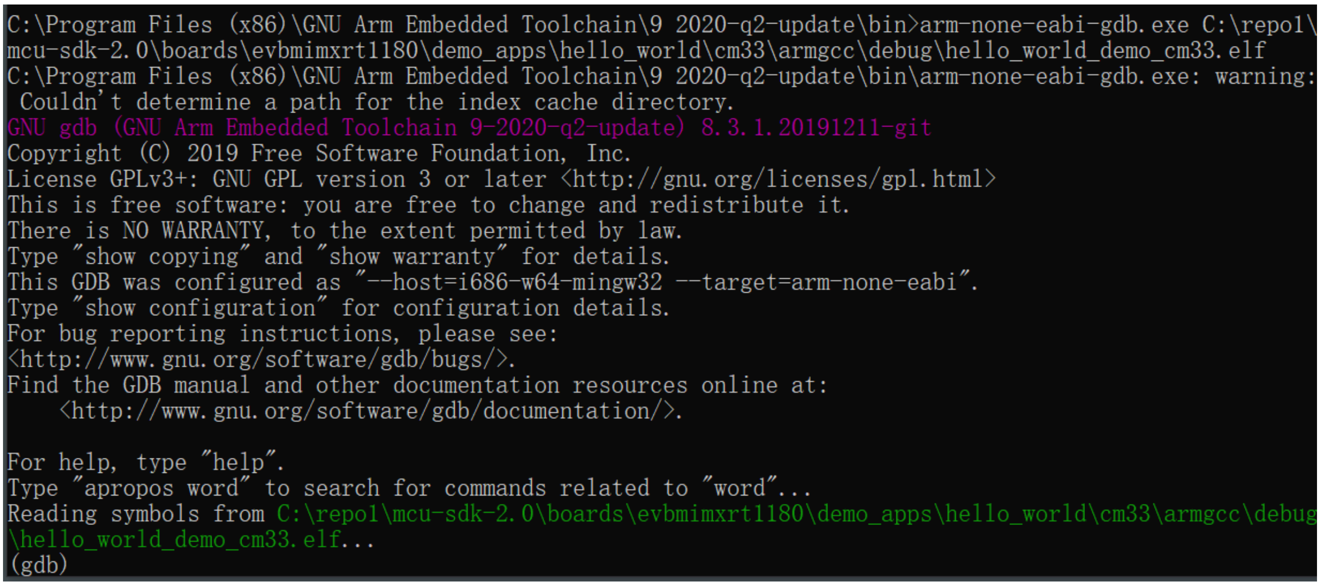


Figure 55. Launch command prompt

7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:
   <install_dir>/boards/<board_name>/<example_type>/<application_name>/<core>/armgcc/<target>
   For this example, the path is:
   <install_dir>/boards/evkmimxrt1180/demo_apps/hello_world/<core>/armgcc/debug
8. 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 Arm Embedded Toolchain\9 2020-q2-update\bin>arm-none-eabi-gdb.exe C:\repo1\
mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug\hello_world_demo_cm33.elf
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin\arm-none-eabi-gdb.exe: warning:
 Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-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 C:\repo1\mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug
\hello_world_demo_cm33.elf...
(gdb)
```

Figure 56. Run arm-none-eabi-gdb

9. Run these commands:

```
target remote localhost:2331
monitor reset
monitor halt
load
```

10. The application is now downloaded and halted at the reset vector. Execute the `continue` or `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.

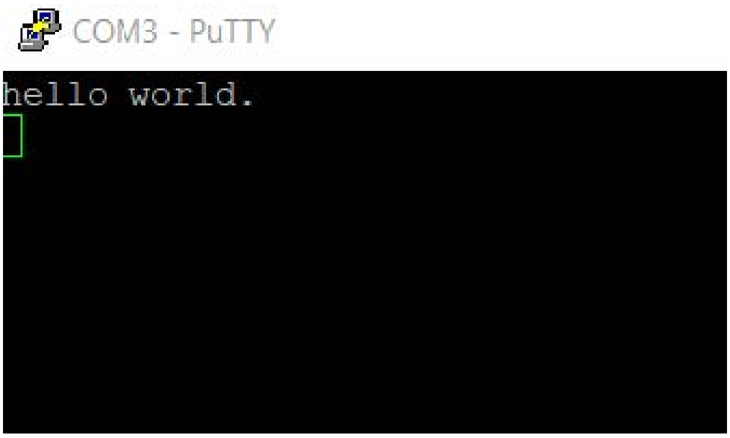


Figure 57. Text display of the hello_world demo

6.4 Build and run a multicore example application

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

<install_dir>/boards/evkmimxrt1180/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/evkmimxrt1180/multicore_examples/hello_world/cm7/armgcc/build_debug.bat

<install_dir>/boards/evkmimxrt1180/multicore_examples/hello_world/cm33/armgcc/build_flexspi_nor_debug.bat

Build the application for the auxiliary core (cm7) first, because the primary core application project (cm33) must 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.

By default, the primary core **flexspi_nor_debug** target links the auxiliary core **debug** target, and the primary core **flexspi_nor_release** target links the auxiliary core **release** target. During the primary core execution, the auxiliary core image is copied from flash into CM7 RAM and executed.

7 Bootable image generation

RT1180 can only boot from CM33. For CM33 application image to boot, a sophisticated boot header structure is needed. To ease the customer from complex settings, the **Secure Provisioning Tool** is provided. Among other rich features, it provides an easy way to use GUI for customer to generate bootable image from raw application image. In addition, in order for users to run the CM7 image conveniently, SDK provides the `multicore_trigger` demo for users to kick off CM7 image, combined with the SPT tool, we can run CM7 image by from POR.

In this section, two kinds of images are defined

Table 1. Image definition

Terminology	Description
RAW image	Demo image without boot header
POR image	Demo image with boot header

Most examples that we provided in SDK package create RAW images from project settings. Only some OOBE examples can create POR image. POR image can provide good out-of-box experience, but the cost is losing some flexibility. While RAW images, with the usage of `Secure Provisioning Tool`, have the most widely usage like image signning, multiple image combination, and so on. For demos that by default generate a POR boot image, see Section 7.3.

7.1 Use SPT tool to boot cm33 image

Run the Secure Provisioning Tool, and the following GUI shows.

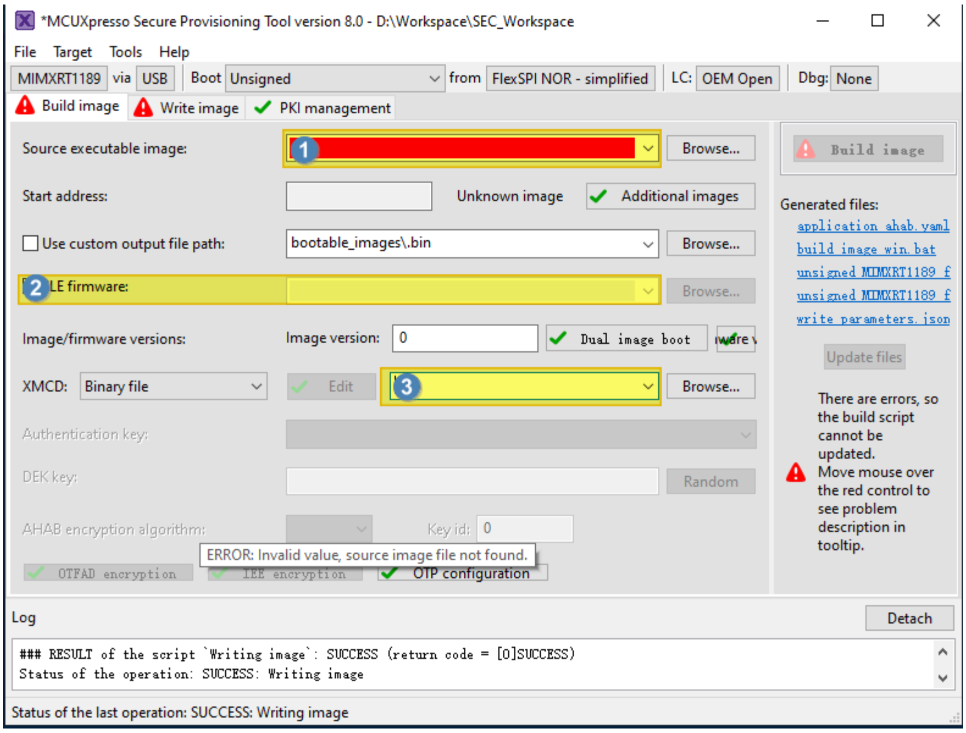


Figure 58. Secure Provisiong Tool start page

Select the following items:

1. Toolchain generated images
   All SDK generated images are supported no matter they boot from TCM, FlexSPI Nor, or HyperRAM.

   Table 2. Toolchains

Toolchain	Suffix
ARMGCC	.elf
IAR	.out
MDK	.out/.hex
MCUX	.axf

   Note:
   - *Use the RAW image for CM33. Some SDK project targets are POR image by default, it must to be switched to RAW image for SPT usage. For more information, see Section 7.3.*
   - *It is recommended to use elf/out files, thus SPT can parse the **Start address** automatically.*
2. XMCD file
   This is on-board SDRAM/HyperRAM initialization file. For the RT1180 EVK board, select *boards/evkmimxrt1180/xmcd/hyperram_xmc_auto_detect.bin*.
   Note:
   - *If the application image uses `hyperram` (target hyperram_debug/release, flexspi_hyperram_debug/release, hyperram_txt_debug/release), the XMCD is necessary.*
   - *hyperram_xmc_auto_detect.bin is board specific.*
3. (Optional) ELE firmware
   This is needed for certain application images which need special ELE FW service. For most SDK demos, simply leave it empty.

After the selection, click `Build Image` to generate the bootable image (POR image).

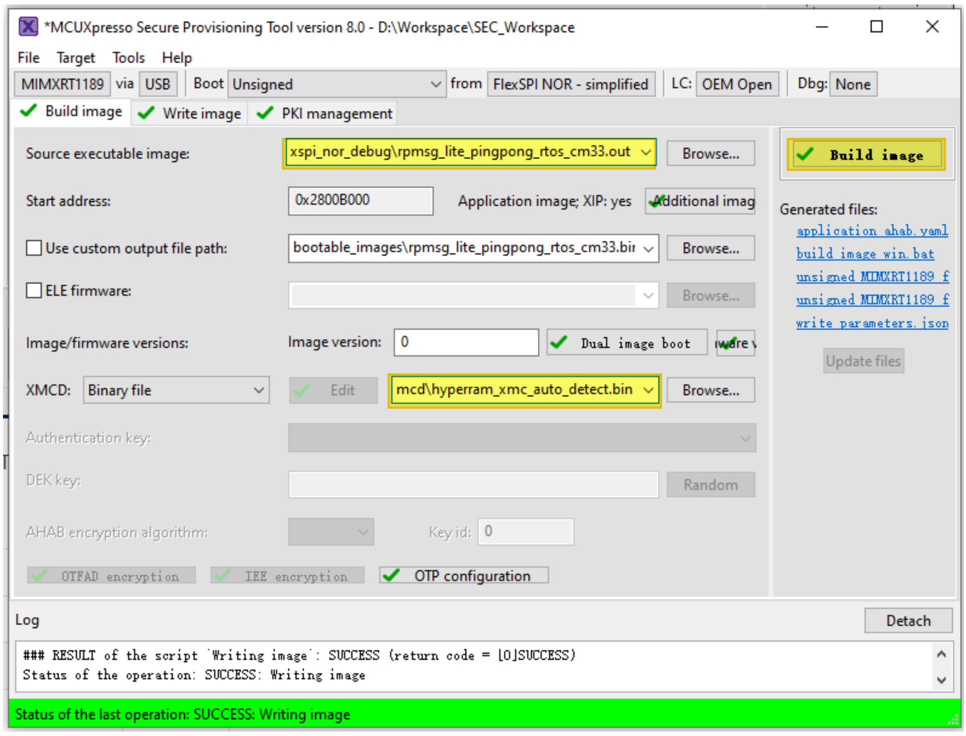


Figure 59. Build single CM33 POR image

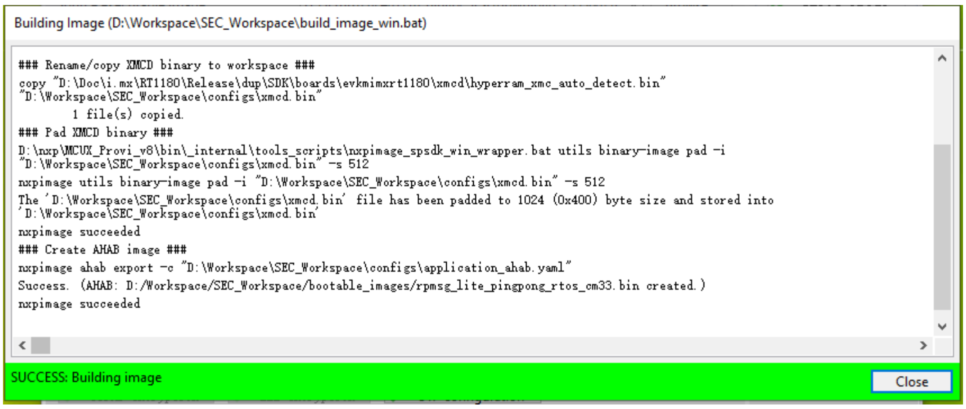


Figure 60. Build single CM33 POR image success

Now switch to the **Write Image** tab. All information is automatically generated from the previous page. Connect both USB OTG1 port (J33) and MCU-Link port (J53) to the computer and power up the board (or reset via SW3). Make sure that the EVK board boot configuration switch `SW5[1..4]` are on `0001` SDP mode and click **Write Image**.

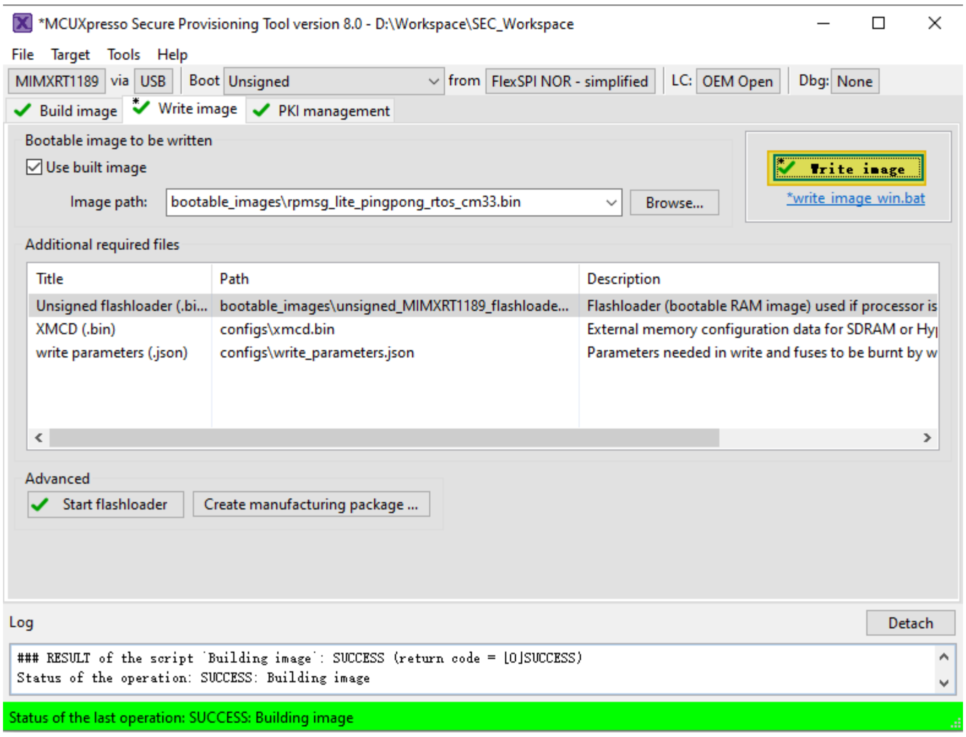


Figure 61. Write single CM33 POR image

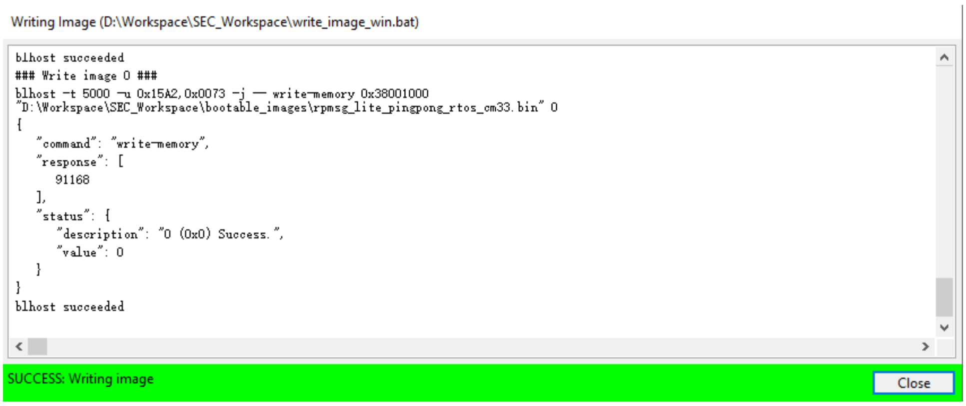


Figure 62. Write single CM33 POR image success

Change the `SW5[1..4]` to QSPI boot mode `0100` and reset the board, and then you can observe that the image is running from POR boot.

7.2 Use SPT tool and `multicore_trigger` image to kick off `cm7` binary image

This section introduces how to create a CM7 POR boot image with the aid of a CM33 kicker application. The CM33 kicker application is located in `demo_apps/multicore_trigger`.

For a secure provisioning tool to generate such a bootable CM7 image, specify the CM33 `multicore_trigger` image in `Source executable image`. The CM7 application image to be kicked off must be specified with `additional images`.

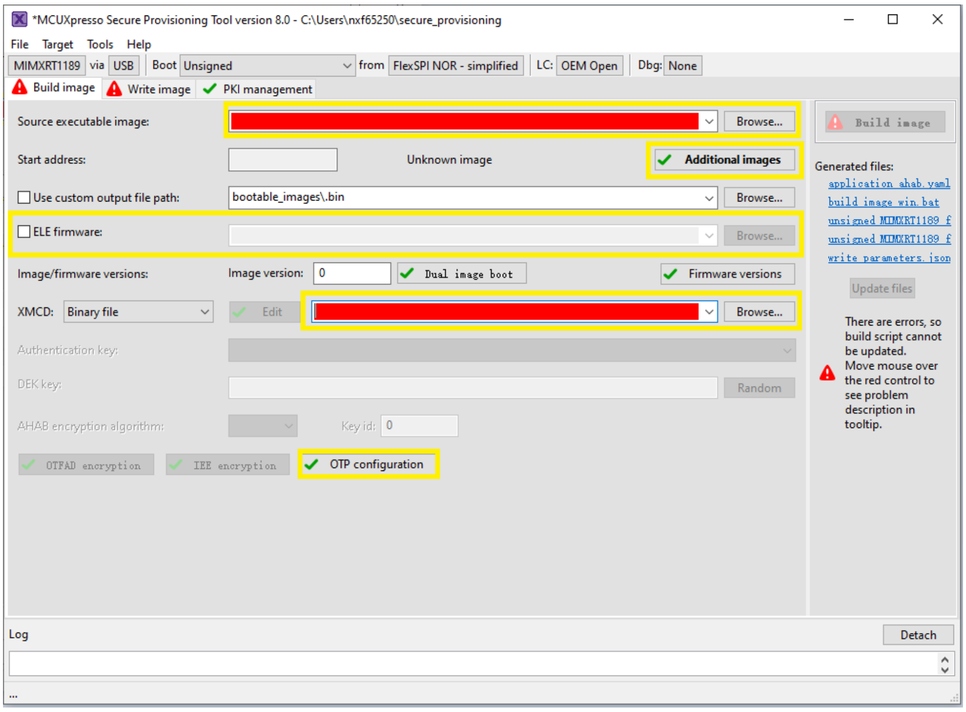


Figure 63. Build CM7 POR image

7.2.1 CM7 XIP image runs from external FLASH, and `multicore_trigger` CM33 image runs from internal RAM

Perform the following steps:

1. Prepare the `multicore_trigger_demo_cm33` ram image (`debug` or `release` targets) and specify it in `Source executable image`.
2. Specify the XMCD file if necessary (see Section 7.2).
3. Use `Additional images` to specify CM7 application image running from flash memory. See Table 3 for how to fill in the necessary information for this additional image.

Table 3. Image parameters

Parameter	Explanation
Image offset	`0xA000` The offset in bytes from start of the current container header to beginning of the image. In our case, the CM7 image is flashed to `0x2800_B000`. The container header is put to `0x2800_1000` and the yield offset = `0xA000`.
Load address	`0x2800B000` The destination address of the CM7 image. ROM copies the image to load address, if it is not within flash address space. In our case, copy does not happen.
Entry point	`0x2800B000` The start address of CM7 image vector table from CM7 core address space.
Core ID	cortex-m7
Image type	Executable
Encrypted	No

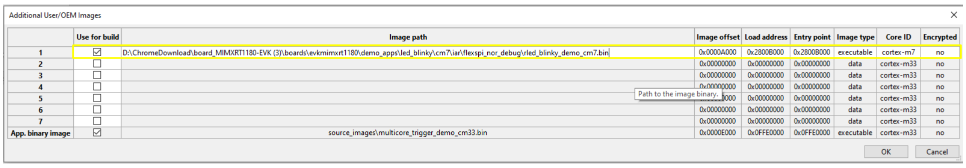


Figure 64. Select an extra CM7 XIP image

4. (Optional) ELE firmware.
   It is needed for a certain application image, which need special ELE FW service. For most SDK demos, simply leaves it empty.

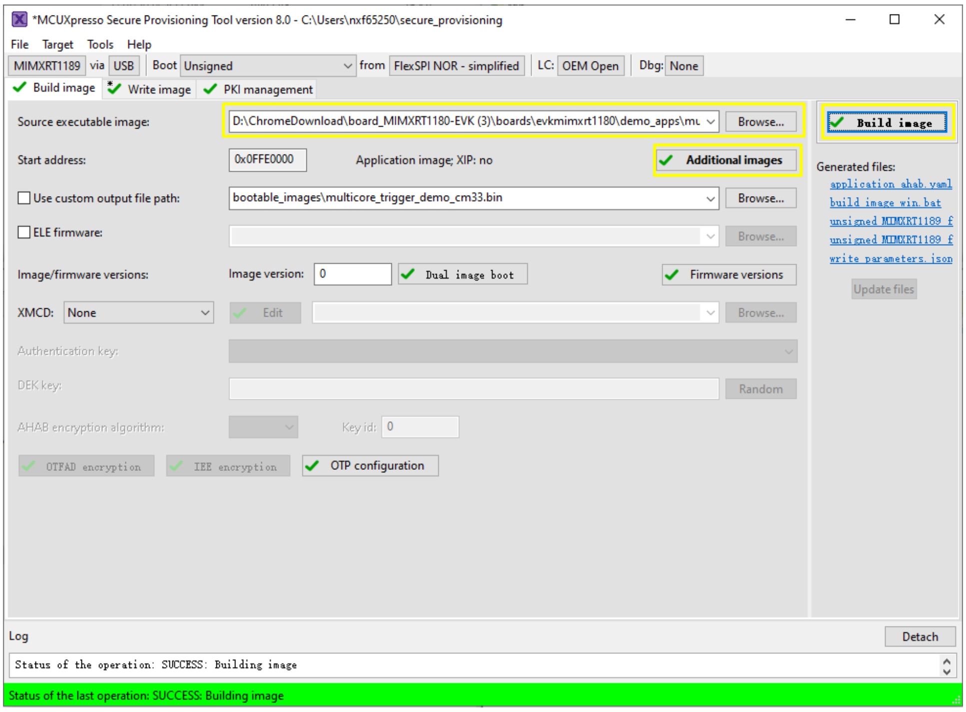


Figure 65. Build CM33 and CM7 combined image Type 1

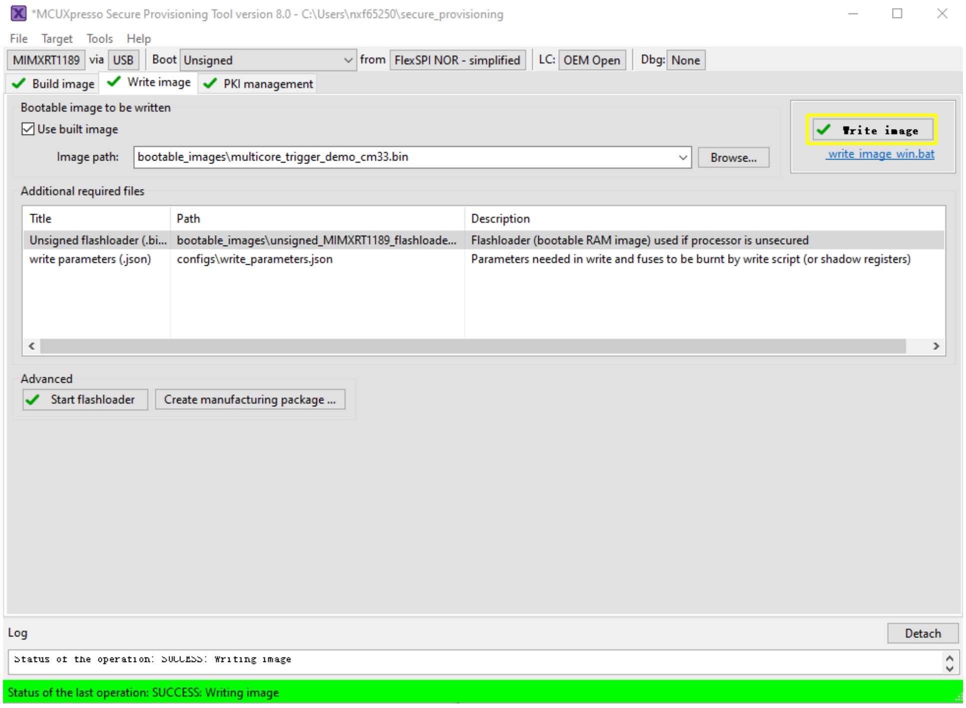


Figure 66. Write CM33 and CM7 combined image Type 1

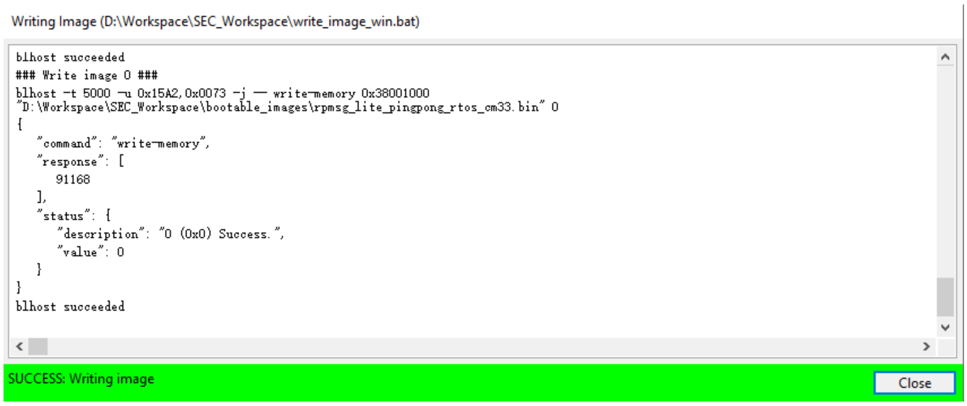


Figure 67. Write CM33 and CM7 combined image Type 1 success

After a successful write, change the `SW5[1..4]` to QSPI boot mode `0100` and reset the board. You can observe that the image is running from POR boot.

7.2.2 CM7 binary image runs from internal TCM, and `multicore_trigger` CM33 image runs from external FLASH

Note: *Be very careful that this scenario requires fuse manipulation, which is not revertible.*

1. Prepare the `multicore_trigger_demo_cm33` flash target image (`flexspi_nor_debug` or `flexspi_nor_release`) and specify it in `Source executable image`.
2. Use `Additional images` to specify CM7 application image running from ITCM memory. See Table 4 for how to fill in necessary information for this additional image.

Table 4. Image parameters

Parameter	Explanation
Image offset	`0xA000` The flash allocation is the same as previous scenario.
Load address	`0x303C0000` This is the secured alias of CM7 ITCM in CM33/ROM memory space. ROM will copy image to this address.
Entry point	`0x0` The start address of vector table from CM7 core address space.
Core ID	`cortex-m7`
Image type	Executable
Encrypted	No

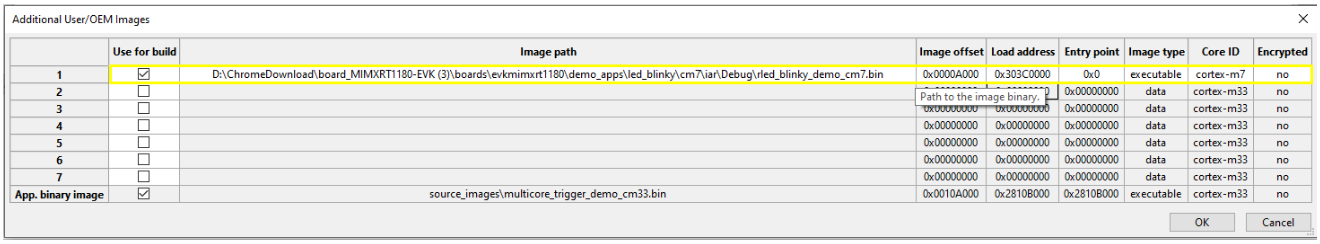


Figure 68. Select Additional CM7 ITCM Image

3. For ROM to successful copy image from flash to CM7 TCM. CM7 TCM ECC should be pre-configured. To archive this, a Fuse need to be set. Click OTP configuration, set `POR_PRELOAD_CM7_TCM_ECC` and `RELEASE_M7_RST_STAT` fuses in `BOOT_CFG7` to 1.
__Note:__ Be very careful that any setting on OTP won't be reverted once set. There is possibility that the chip get bricked if you are fusing the wrong fuse word. Only do this if you really need to try this boot scenario.

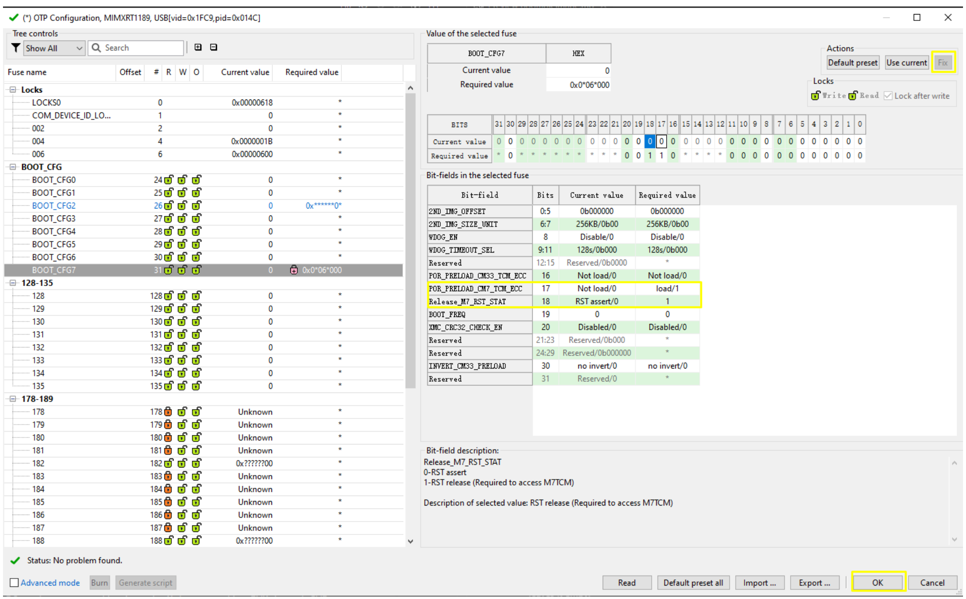


Figure 69. Configure Fuse

4. (Optional) ELE firmware.

It is needed for a certain application image which needs a special ELE FW service. For most SDK demos, simply leave it empty.

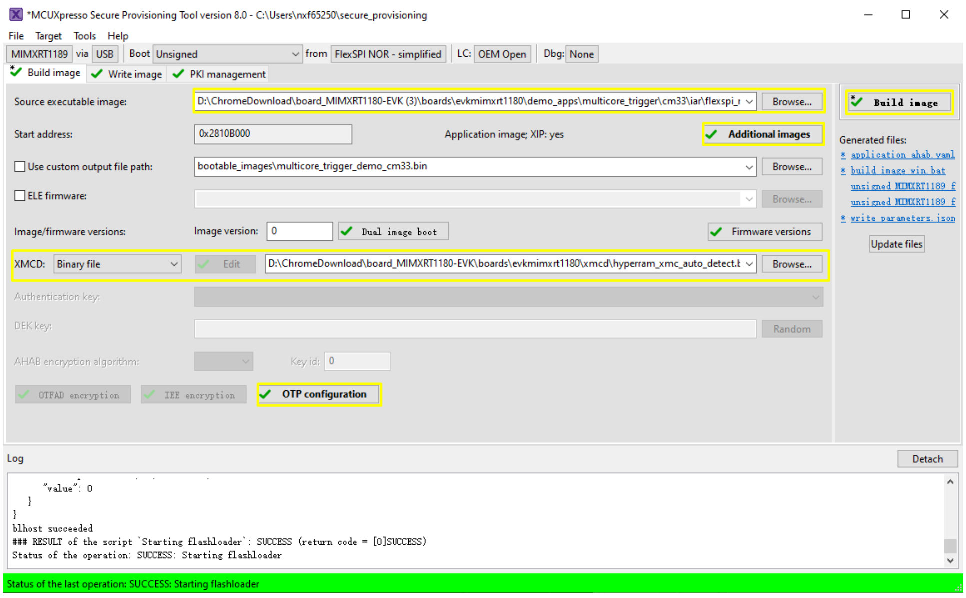


Figure 70. Build CM33 and CM7 combined image Type 2

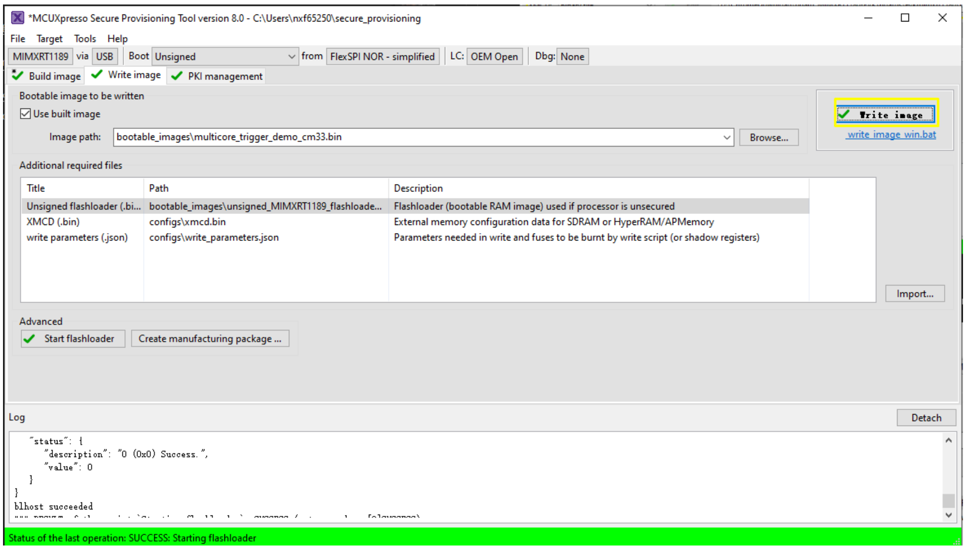


Figure 71. Write CM33 and CM7 combined image Type 2

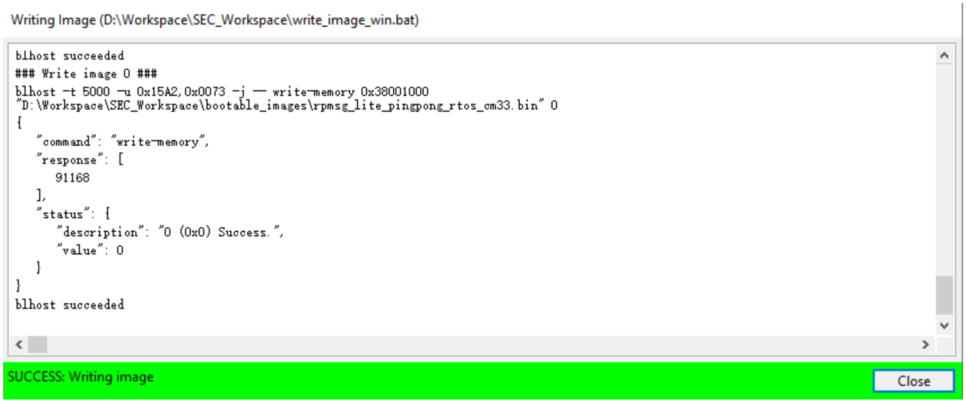


Figure 72. Write CM33 and CM7 combined image Type 2 success

After a successful write, change the `SW5[1..4]` to QSPI boot mode `0100` and reset the board. You can observe that the image is running from the POR boot.

Note:

1. *The new image constructed under **Build images** is still not a complete image, and the Memory Configuration Block is missing. When writing image, SPT downloads the contents of the Memory Configuration Block to FLASH.*
2. *When CM33 + CM7 image runs with the following target combination, it is required to set `CM33_SET_TRDC` to 1U in `multicore_trigger` demo, also requires to set `CM33_SET_TRDC` to 1U in CM7 demo if it exists.*

```
- CM33 flexspi + CM7 flexspi, flexspi means flexspi_nor_debug/release,
  flexspi_nor_sdram_debug/release, flexspi_nor_hyperram_debug/release
- CM33 HYPERRAM + CM7 HYPERRAM, HYPERRAM means hyperram_txt_debug/release.
```

3. *Only those `cm7` projects whose linkages are similar with `hello_world_demo_cm7`, support POR run via `multicore_trigger_cm33`.*

7.3 RAW/POR image switch

In current delivered SDK package, most demos generate RAW images. To enhance customer OOBE experience, the following RT1180 SDK CM33 project/targets generate POR image, which means that they can POR run after you debug/download it in IDE.

- IAR/GCC/MDK, *hello_world_demo_cm33 flexspi_nor_debug/release*, *flexspi_nor_hyperram_debug/release*
- IAR/GCC/MDK, *multicore_hello_world_cm33 flexspi_nor_debug/release*, *flexspi_nor_hyperram_debug/release*
- MCUX CM33 image are all POR images except the `multicore_trigger` demo.

Note:

- *These POR images are not signed, just for develop convenience, not recommended for product usage.*
- *It is highly recommended to use SPT for image download.*

For the SDK CM33 IAR/MDK/ARMGCC project, `flexspi_nor` targets, as well as for MCUX project, debug/release targets. It is easy to switch RAW/POR image, via the project macro `XIP_BOOT_HEADER_ENABLE` setting. Table 5describes the rules.

Table 5. Rules

XIP_BOOT_HEADER_ENABLE setting	Image type
1	POR

Table 5. Rules*...continued*

XIP_BOOT_HEADER_ENABLE setting	Image type
0	RAW

- For IAR/ARMGCC, change the macro `XIP_BOOT_HEADER_ENABLE` in the compiler setting.
- For MDK, change the macro `XIP_BOOT_HEADER_ENABLE` in compiler and link setting simultaneously.

For detailed settings, see the following sections.

7.3.1 IAR settings for image type switch

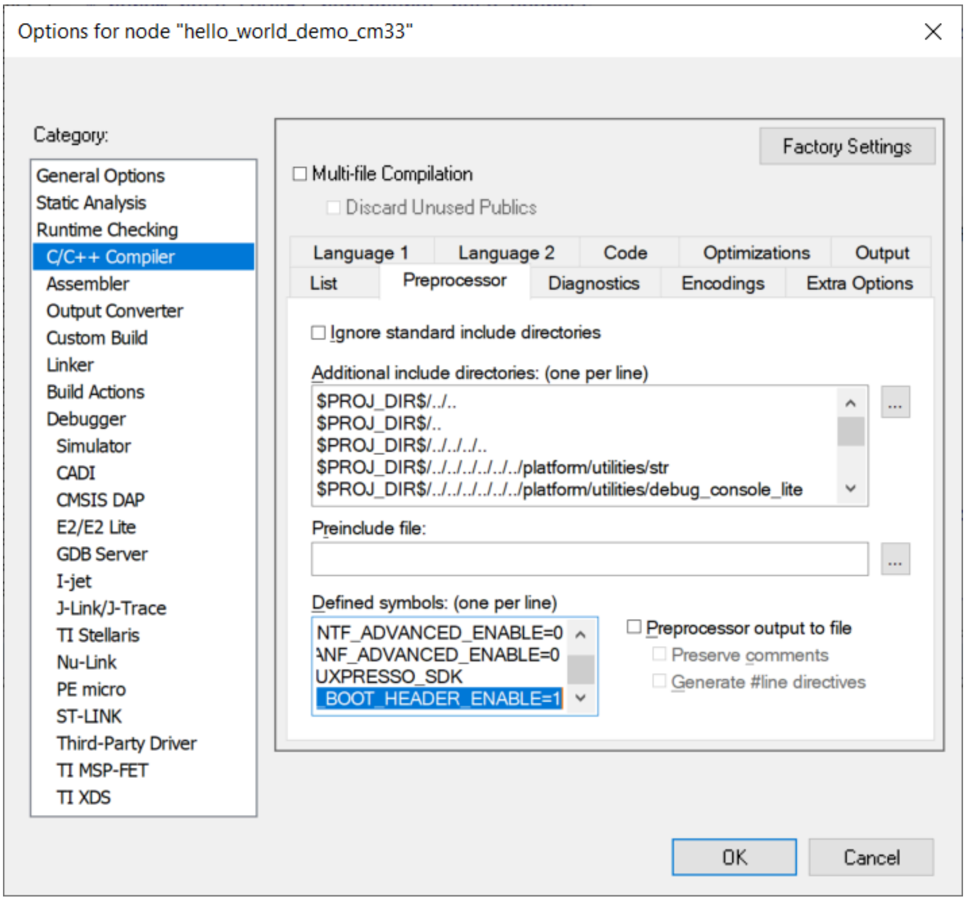


Figure 73. IAR settings for image type switch

7.3.2 ARMGCC settings for image type switch

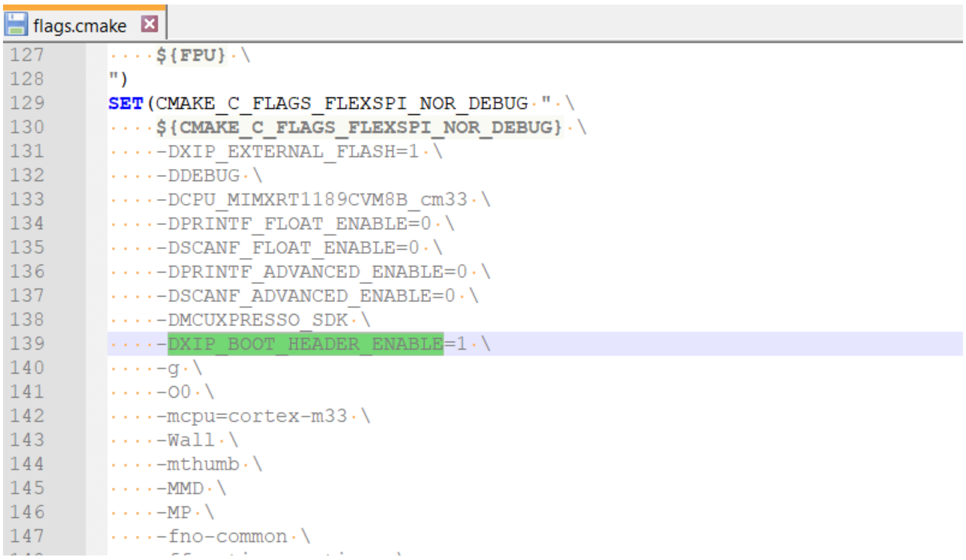


Figure 74. ARMGCC setting for Image Type Switch

7.3.3 MDK setting for image type switch

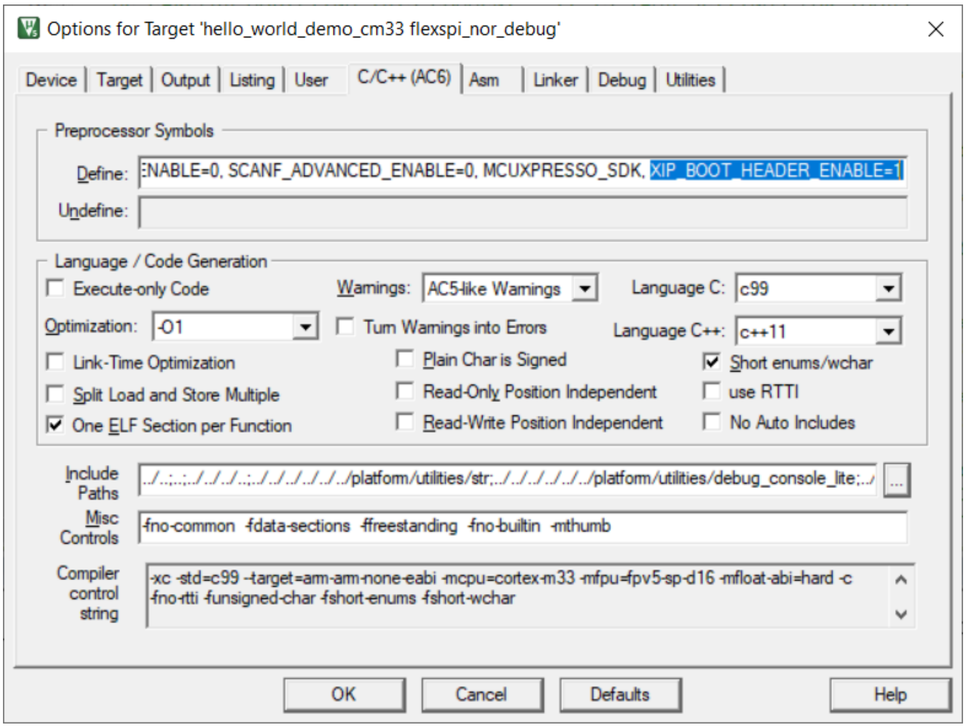


Figure 75. MDK compiler setting for image type switch

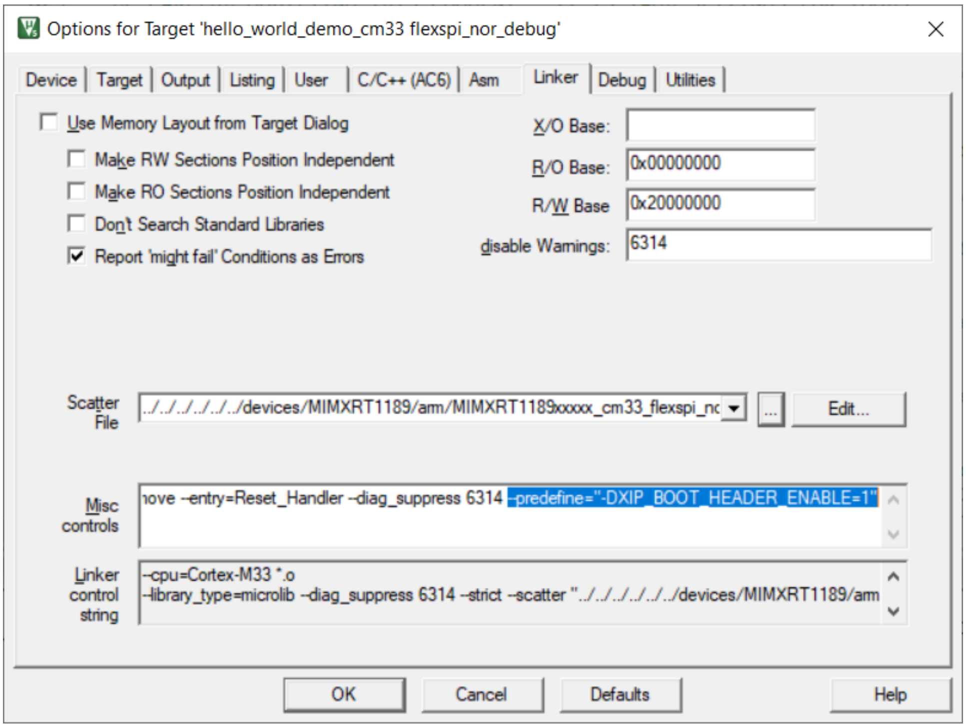


Figure 76. MDK linker setting for image type switch

7.3.4 MCUxpresso settings for image type switch

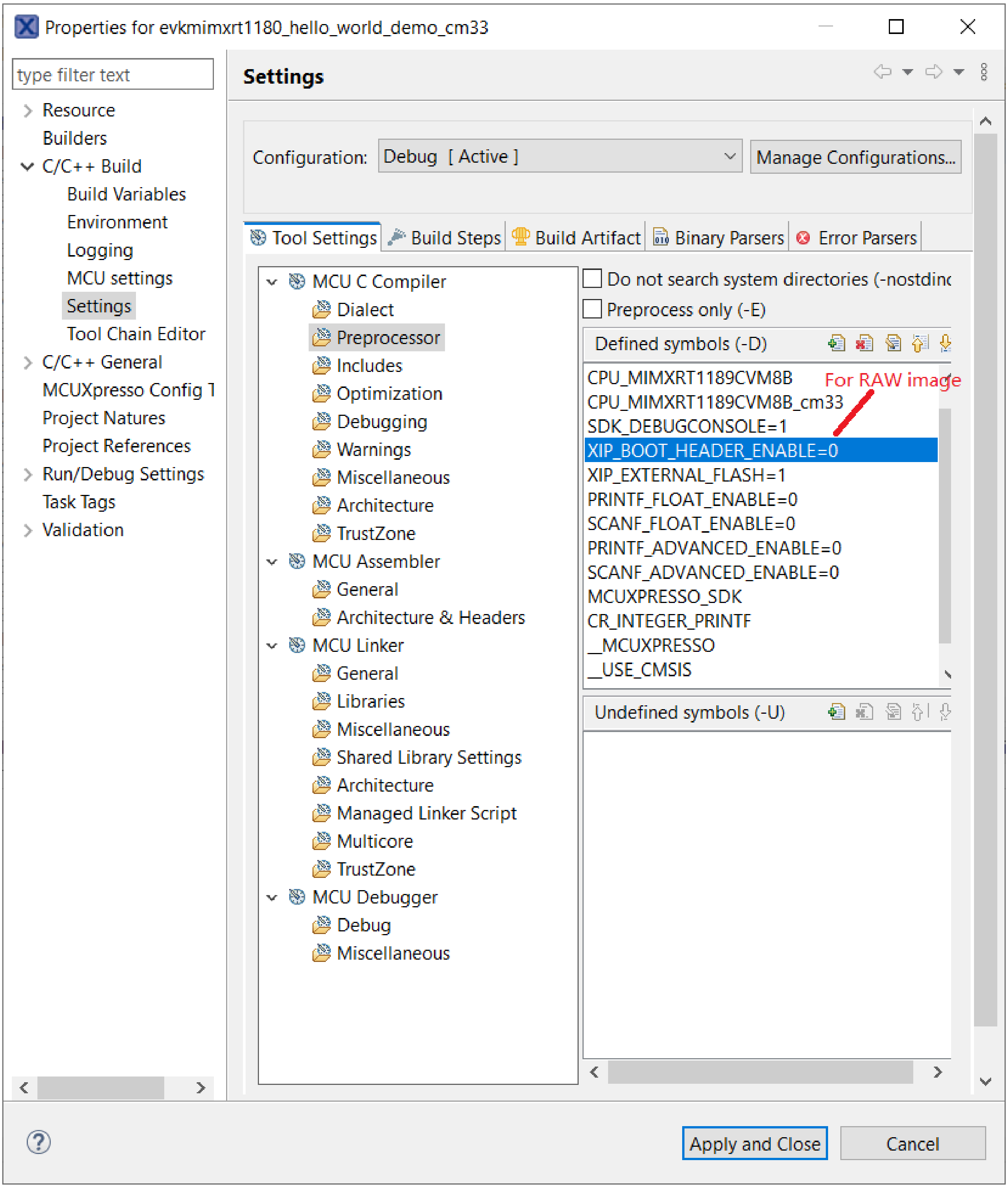


Figure 77. MCUxpresso settings for image type switch

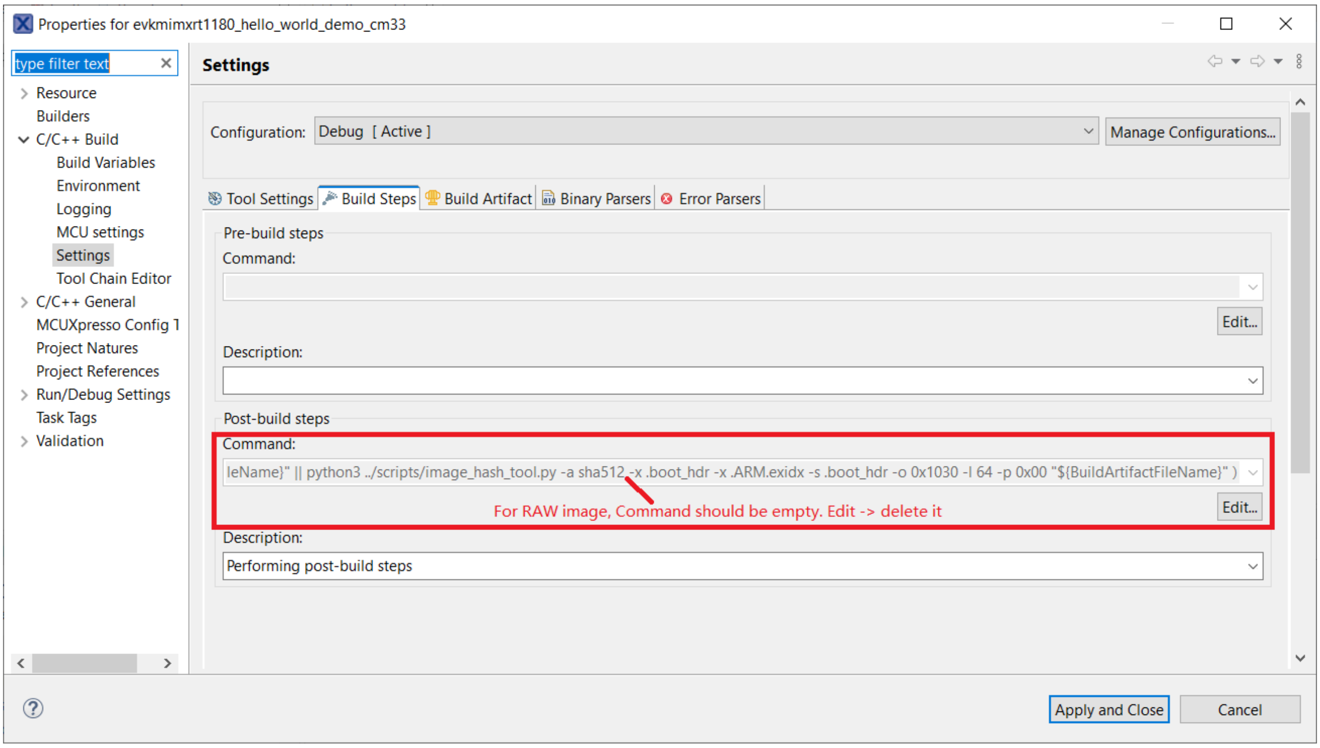


Figure 78. MCUxpresso settings for image type switch for raw image

7.4 Use Secure Provisiong Tool to erase flash

In worst cases a debugger cannot successfully connect to the board. A major reason for this is there are POR boot image inside the on board flash memory which causes the whole system in a trouble state. Secure Provisioning Tool provide a reliable way to erase the flash in this worst case. To achieve this

1. Set SW5[1:4] to `0001`.
2. Connect the board to your PC from both `J53` (Debugging) and `J33` (USB OTG 1), then power up the board.
3. Start `Secure Provisioning Tool`, select `Tool` > `Flash Programmer` or `Ctrl + R`.
4. Click `Yes` to prompt up questions, wait memory initialization finish.

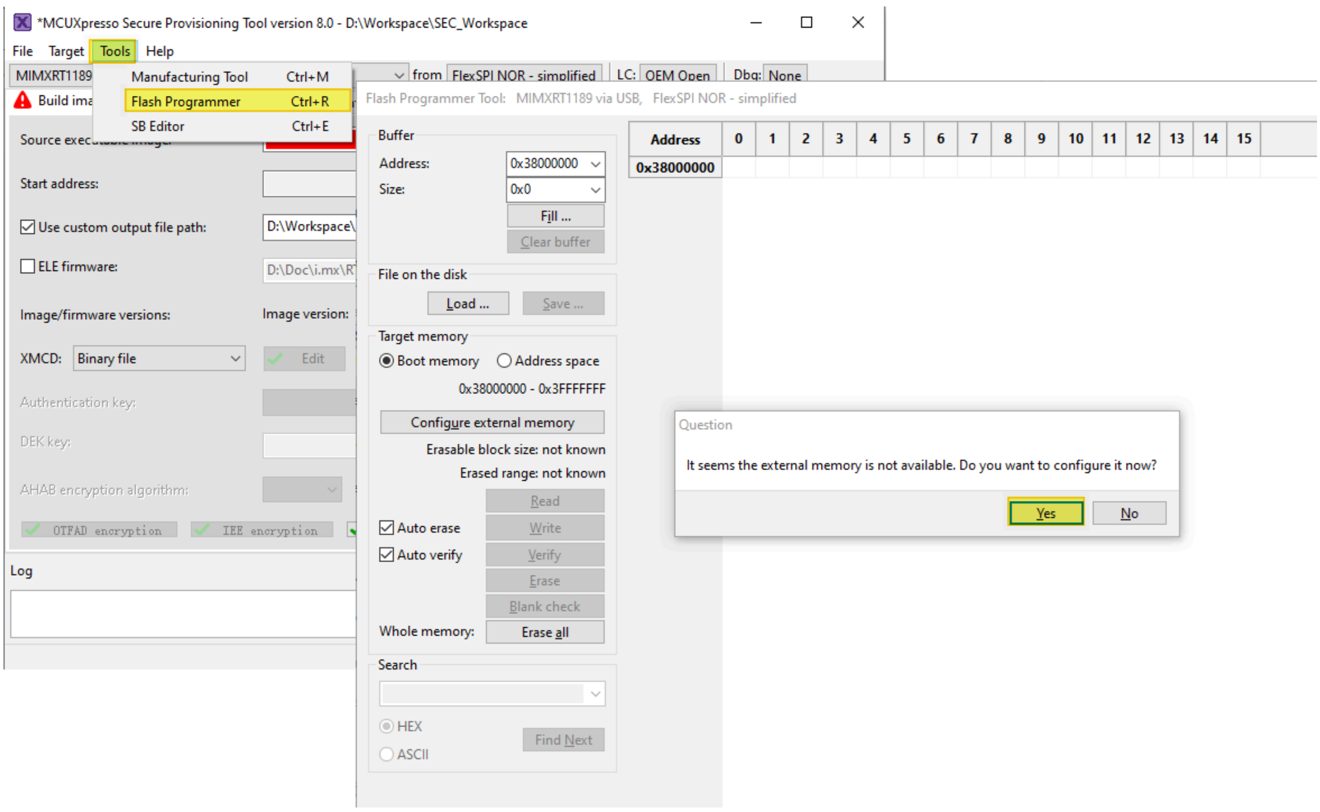


Figure 79. Secure Provisiong Tool Initialize Flash Programming Tool

5. Click `Erase all` button and wait a minute for the erase process to finish.

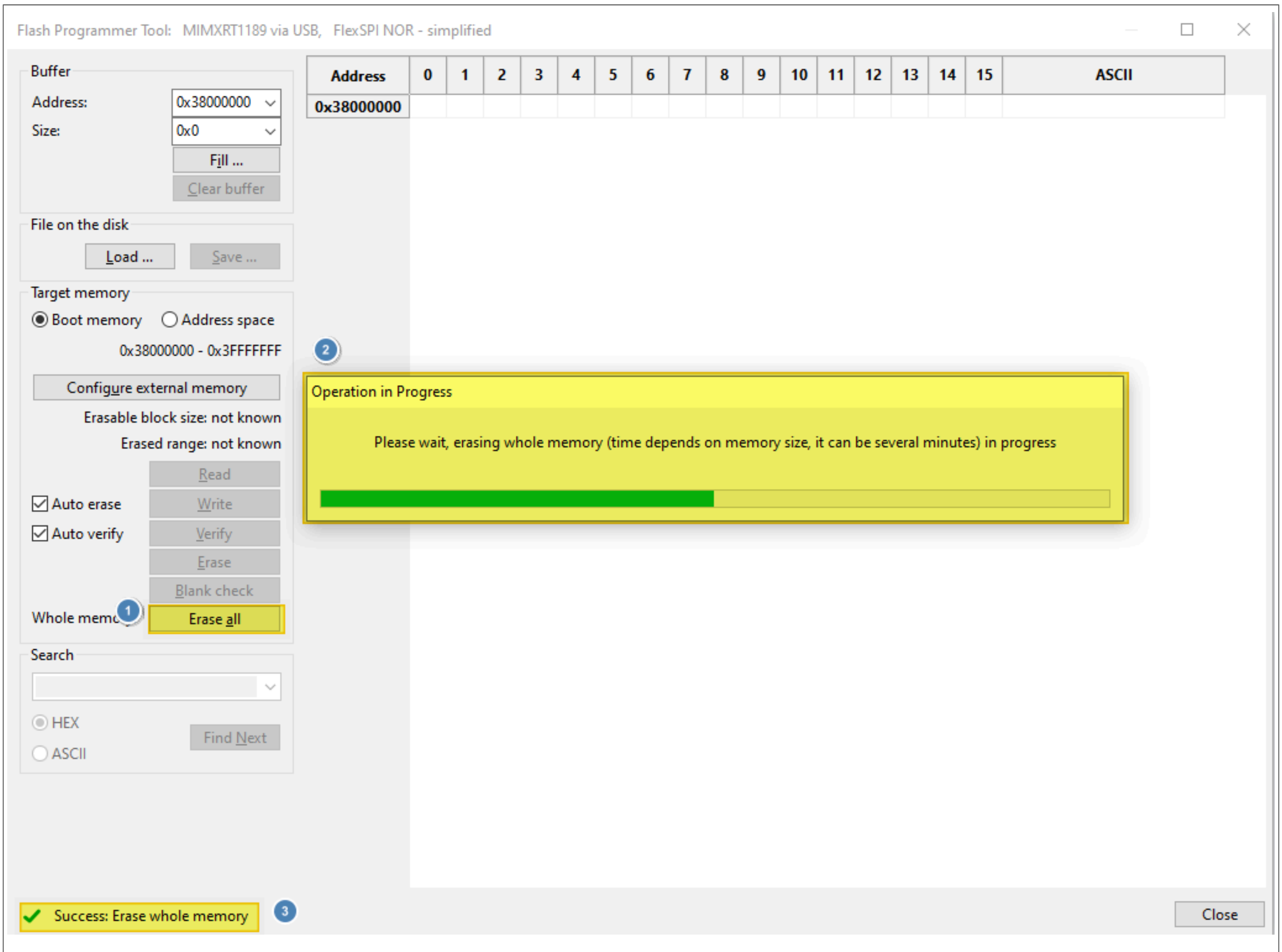


Figure 80. Secure Provisiong Tool Initialize Flash Programming Tool

6. After `Success: Erase whole memory` shows up, the board should have restored to a good state to work in various mode.

8 MCUXpresso configuration 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 configuration for the selected board or processor. The generated code is used with MCUXpresso SDK version 2.x.

Table 6 describes the tools included in the MCUXpresso config tools.

Table 6. MCUXpresso config tools

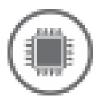
Config tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration.	

Table 6. MCUXpresso config tools*...continued*

Config tool	Description	Image
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.	

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 ding 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.

9 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 81.

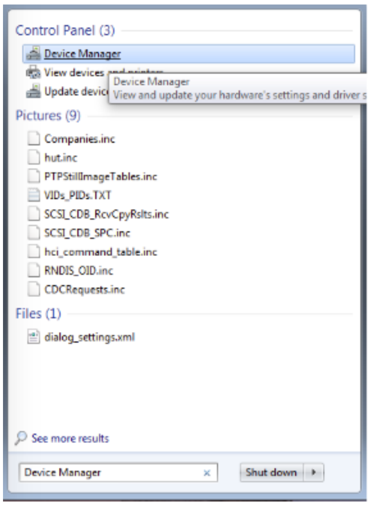


Figure 81. Device Manager

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

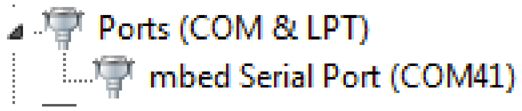


Figure 82. OpenSDA – CMSIS-DAP/mbed/DAPLink interface

b. OpenSDA – P&E Micro:

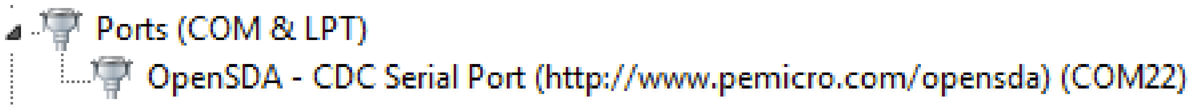


Figure 83. OpenSDA – P&E Micro

c. OpenSDA – J-Link:

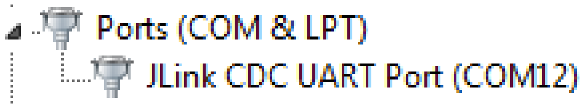


Figure 84. OpenSDA – J-Link

d. P&E Micro OSJTAG:

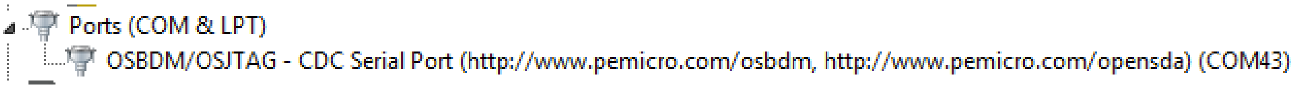


Figure 85. P&E Micro OSJTAG

e. LPC-Link2:

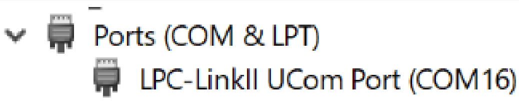


Figure 86. LPC-Link2

f. FTDI UART:

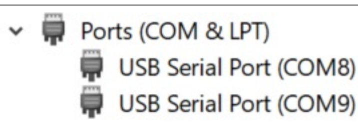


Figure 87. FTDI UART

g. MCU-Link::

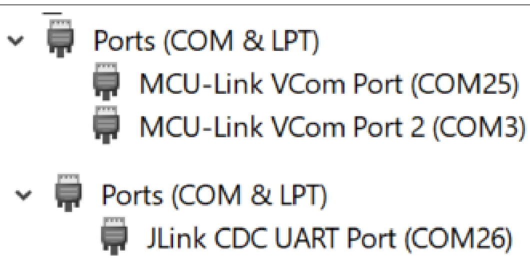


Figure 88. MCU-Link

10 Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with a variety of factory programmed debug interface configurations. Table 7 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 7. 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
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

Table 7. Hardware platforms supported by SDK*...continued*

Hardware platform	Default interface	OpenSDA details[1]
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
MIMXRT1180-EVK	CMSIS-DAP	OpenSDA v2.2
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

Table 7. Hardware platforms supported by SDK*...continued*

Hardware platform	Default interface	OpenSDA details[1]
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.

11 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

12 Revision history

Table 8 summarizes the revisions to this document.

Table 8. Revision history

Revision number	Release date	Description
MCUXSDKMIMXRT118XKGSUG v2.16.000	20 June 2024	Initial release for MCUXpresso SDK 2.16.000

Legal information

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.

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 https://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 document 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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

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.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.

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, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) 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.

EdgeLock — is a trademark of NXP B.V.

IAR — is a trademark of IAR Systems AB.

J-Link — is a trademark of SEGGER Microcontroller GmbH.

Kinetis — is a trademark of NXP B.V.

Tower — is a trademark of NXP B.V.

Tables

Figures

Contents

Date of release: 20 June 2024
Document identifier: MCUXSDKMIMXRT118XKGSUG