NXP Semiconductors
User's Guide

Document Number: MCUXSDKMIMXRT1024GSUG
Rev. 0, July 2020

Getting Started with MCUXpresso SDK for MIMXRT1024-EVK

1 Overview

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

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MIMXRT1024-EVK* (document MCUXSDKMIMXRT1024RN).

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

Contents

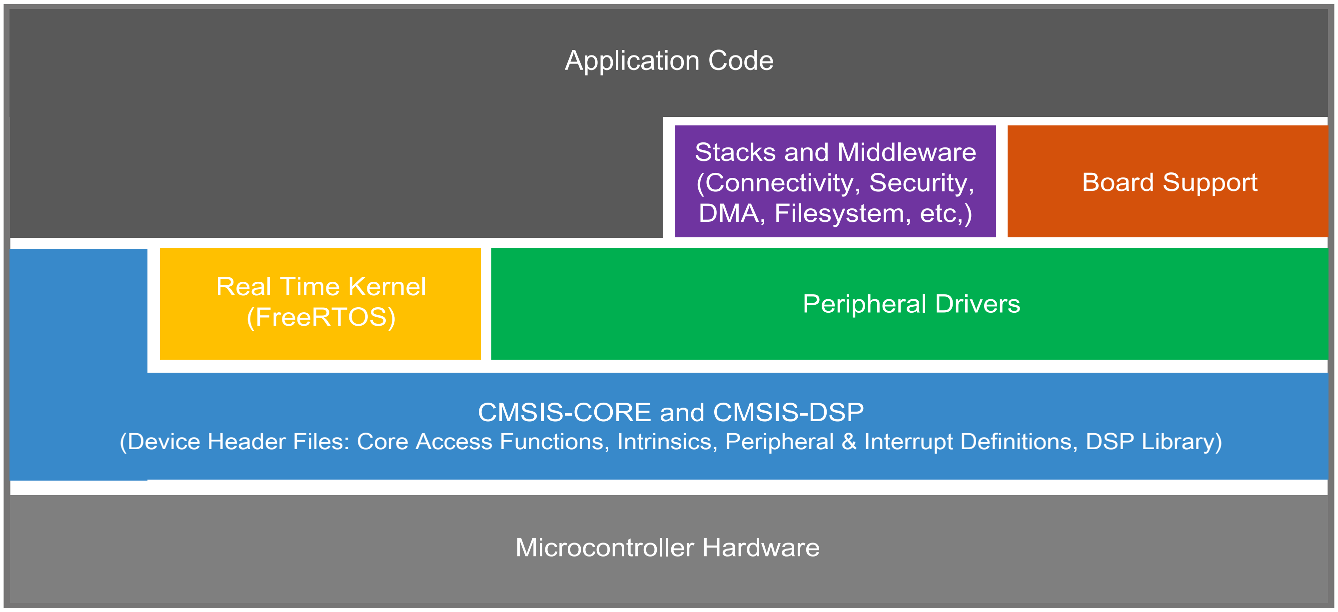


Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support package folders

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

- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- `rtos_examples`: Basic FreeRTOS™ OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers
- `wireless_examples`: Applications that use the Zigbee and OpenThread stacks.

2.1 Example application structure

This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

Each `<board_name>` folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the `hello_world` example (part of the `demo_apps` folder), the same general rules apply to any type of example in the `<board_name>` folder.

In the `hello_world` application folder you see the following contents:

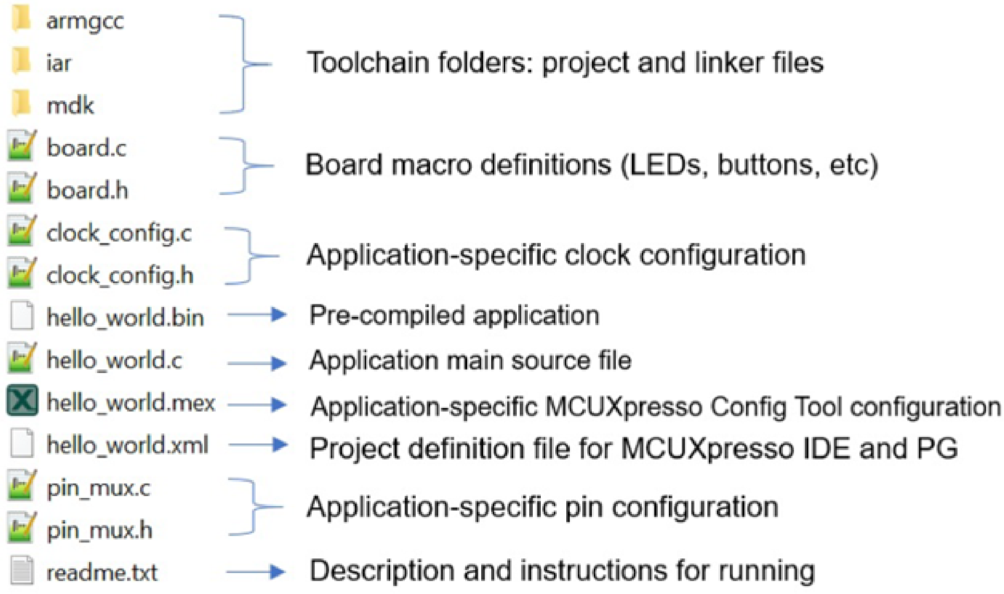


Figure 2. Application folder structure

All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

2.2 Locating example application source files

When opening an example application in any of the supported IDEs, a variety of source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- `devices/<device_name>`: The device's CMSIS header file, MCUXpresso SDK feature file and a few other files
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<devices_name>/project` Project template used in CMSIS PACK new project creation

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the `rtos` folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

3 Run a demo application using IAR

3.1 Build an example application

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

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

   Using the EVK-MIMXRT1024 hardware platform as an example, the `hello_world` workspace is located in:

   `<install_dir>/boards/evkmimxrt1024/demo_apps/hello_world/iar/hello_world.eww`

   Other example applications may have additional folders in their path.

2. Select the desired build target from the drop-down menu.

   There are twelve project configurations (build targets) supported for most MCUXpresso SDK projects:
   - `Debug` – Compiler optimization is set to low, and debug information is generated for the executable. The linker file is `RAM` linker, where text and data section is put in internal TCM.
   - `Release` – Compiler optimization is set to high, and debug information is not generated. The linker file is `RAM` linker, where text and data section is put in internal TCM.
   - `ram_0x1400_debug` – Project configuration is same as the debug target. The linker file is `RAM_0x1400` linker, where text is put in ITCM with offset 0x1400 and data put in DTCM.
   - `ram_0x1400_release` – Project configuration is same as the release target. The linker file is `RAM_0x1400` linker, where text is put in ITCM with offset 0x1400 and data put in DTCM.
   - `sdram_debug` – Project configuration is same as the debug target. The linker file is `SDRAM` linker, where text is put in internal TCM and data put in SDRAM.
   - `sdram_release` – Project configuration is same as the release target. The linker file is `SDRAM` linker, where text is put in internal TCM and data put in SDRAM.
   - `sdram_txt_debug` – Project configuration is same as the debug target. The linker file is `SDRAM_txt` linker, where text is put in SDRAM and data put in OCRAM.
   - `sdram_txt_release` – Project configuration is same as the release target. The linker file is `SDRAM_txt` linker, where text is put in SDRAM and data put in OCRAM.
   - `flexspi_nor_debug` – Project configuration is same as the debug target. The linker file is `flexspi_nor` linker, where text is put in flash and data put in TCM.
   - `flexspi_nor_release` – Project configuration is same as the release target. The linker file is `flexspi_nor` linker, where text is put in flash and data put in TCM.
   - `flexspi_nor_sdram_release` - Project configuration is same as the release target. The linker file is `flexspi_nor_sdram` linker, where text is put in flash and data put in SDRAM.
   - `flexspi_nor_sdram_debug` – Project configuration is same as the debug target. The linker file is `flexspi_nor_sdram` linker, where text is put in flash and data put in SDRAM.

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

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

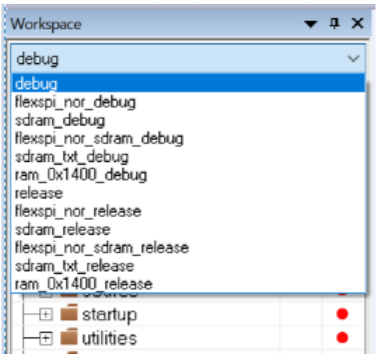


Figure 3. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 4.

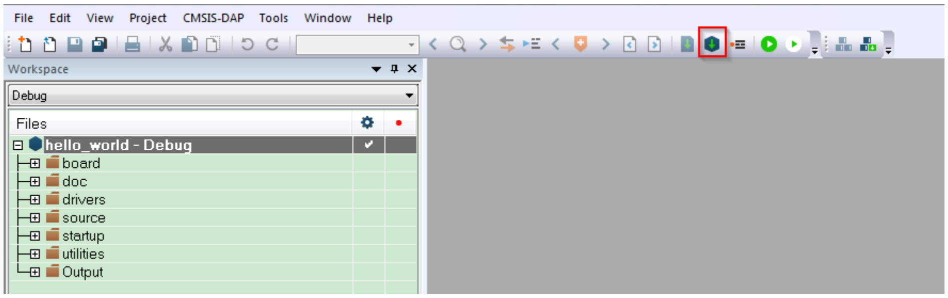


Figure 4. Build the demo application

4. The build completes without errors.

3.2 Run an example application

To download and run the application, perform these steps:

1. This board supports the CMSIS-DAP/mbed/DAPLink debug probe by default. Visit os.mbed.com/handbook/Windows-serial-configuration and follow the instructions to install the Windows® operating system serial driver. If running on Linux OS, this step is not required.
2. Connect the development platform to your PC via USB cable. Connect the USB cable to J41 and make sure SW7[1:4] is **0010b**.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see How to determine COM port). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

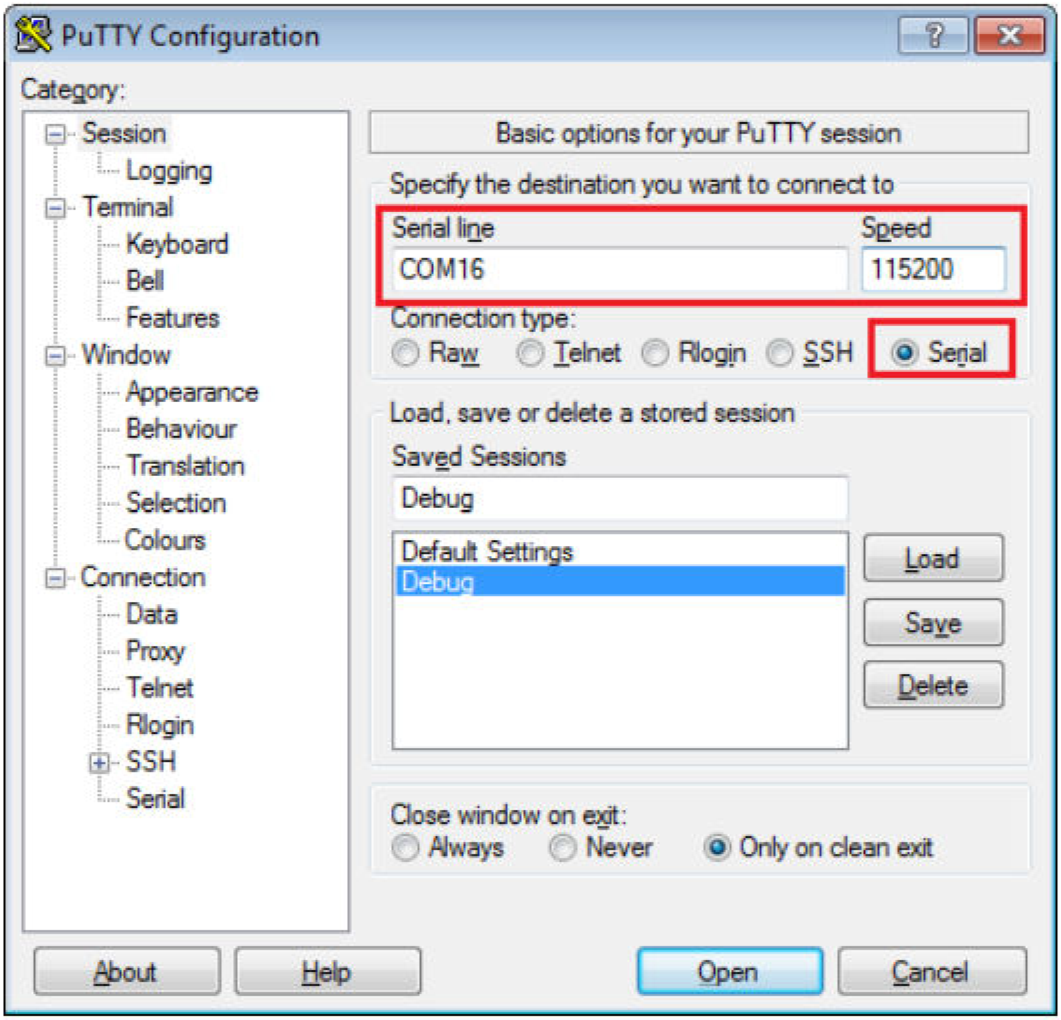


Figure 5. Terminal (PuTTY) configuration

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

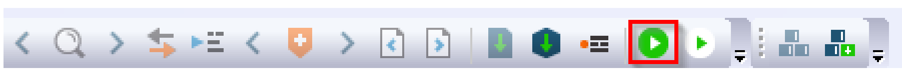


Figure 6. Download and Debug button

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

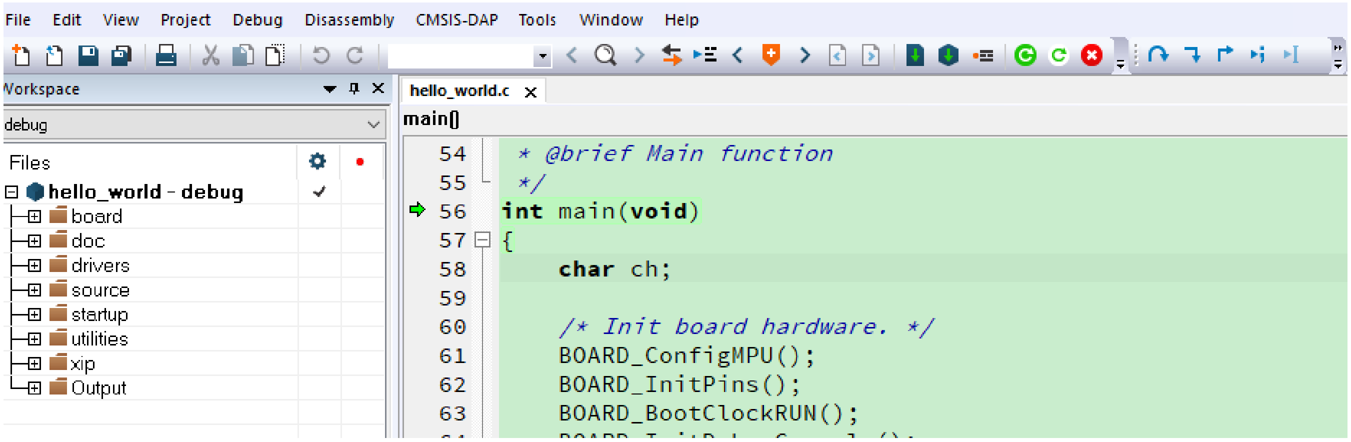


Figure 7. Stop at `main()` when running debugging

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

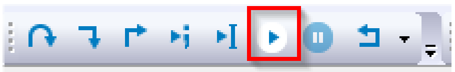


Figure 8. Go button

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.

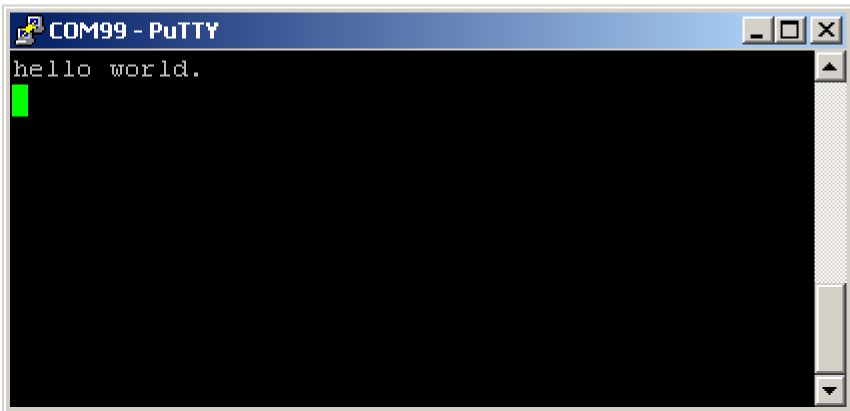


Figure 9. Text display of the `hello_world` demo

4 Run a demo using Keil® MDK/μVision

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

4.1 Install CMSIS device pack

After the MDK tools are installed, Cortex® Microcontroller Software Interface Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions and flash programming algorithms. Follow these steps to install the MIMXRT102x CMSIS pack.

1. Download the MIMXRT102x packs .

2. After downloading the DFP, double click to install it.

4.2 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/evkmimxrt1024/demo_apps/hello_world/mdk/hello_world.uvmpw
2. To build the demo project, select **Rebuild**, highlighted in red.

Figure 10. Build the demo

3. The build completes without errors.

4.3 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 os.mbed.com/handbook/Windows-serial-configuration and follow the instructions to install the Windows® operating system serial driver. If running on Linux OS, this step is not required.
2. Connect the development platform to your PC via USB cable.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see How to determine COM port). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

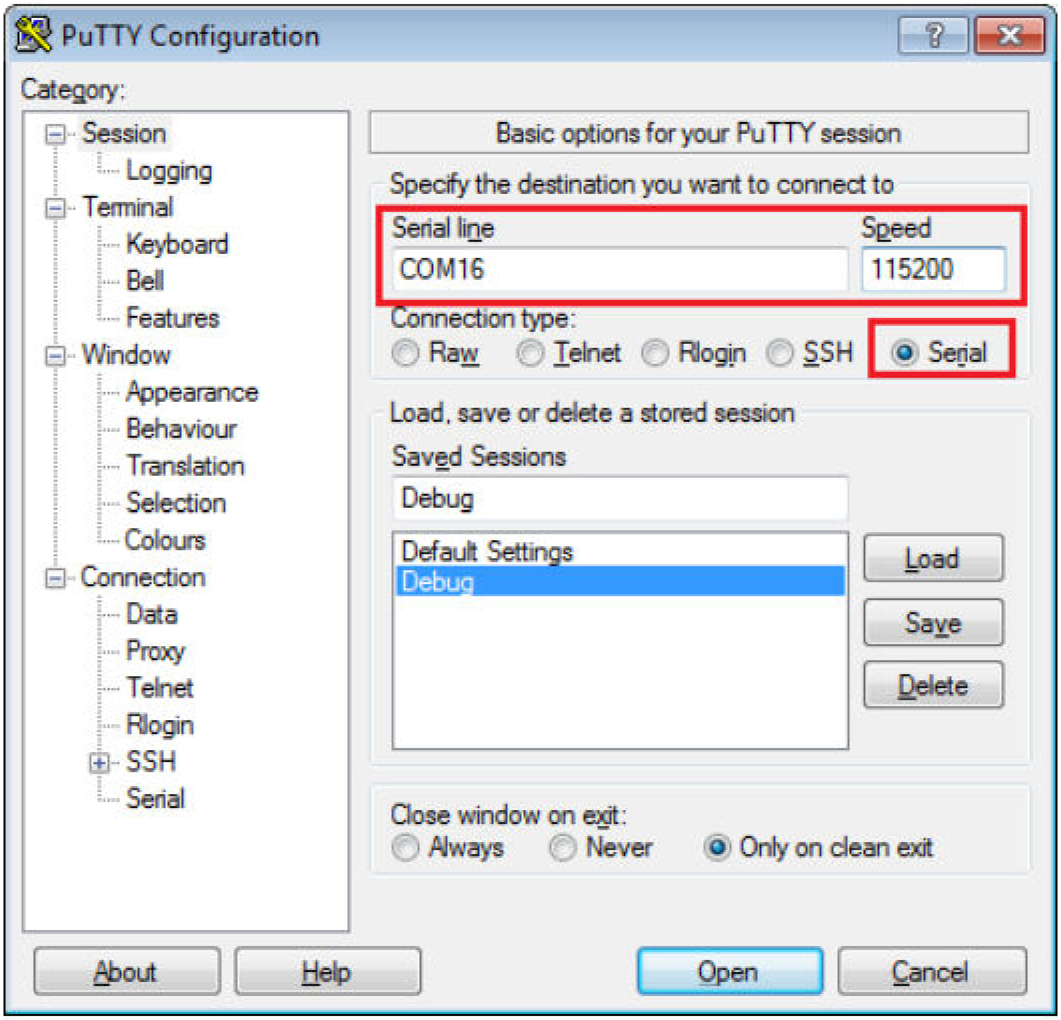


Figure 11. Terminal (PuTTY) configurations

4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button, highlighted in red in Figure 12. If using **J-Link** as the debugger, click **Project option >Debug >Settings >Debug >Port**, and select **SW**.

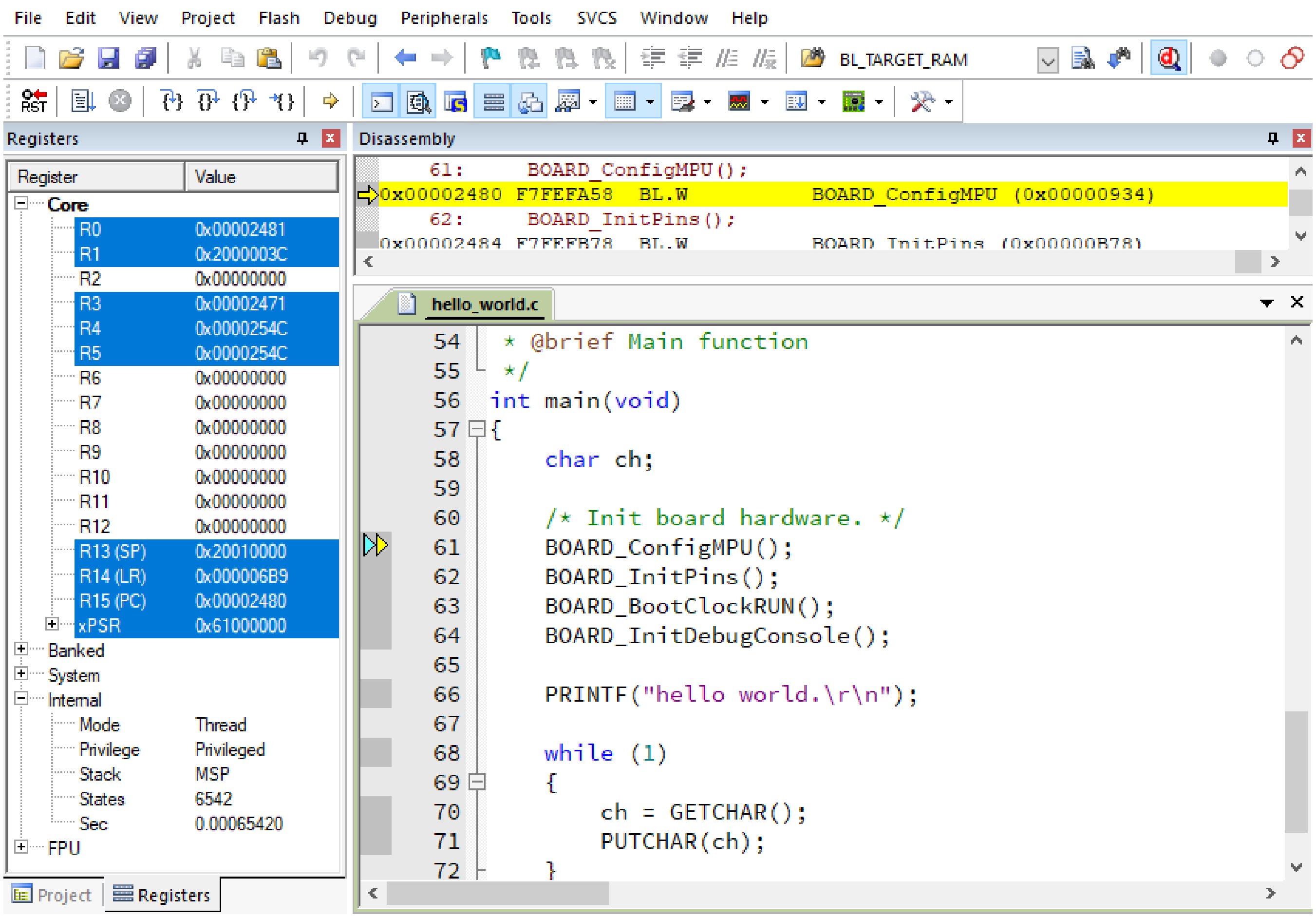


Figure 12. Stop at `main()` when run debugging

5. Run the code by clicking **Run** to start the application, as shown in Figure 13.

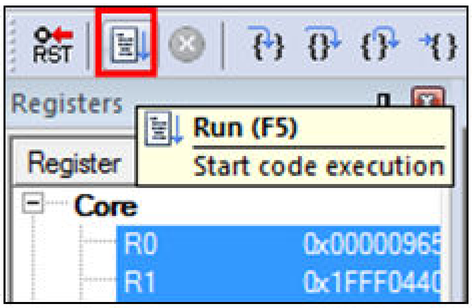


Figure 13. Run button

The `hello_world` application is now running and a banner is displayed on the terminal, as shown in Figure 14. If this is not true, check your terminal settings and connections.

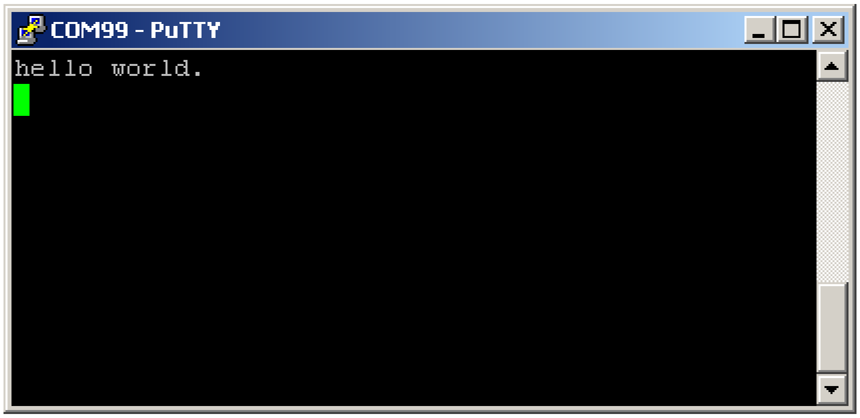


Figure 14. Text display of the `hello_world` demo

5 Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The `hello_world` demo application is targeted for the MIMXRT1024-EVK hardware platform which is used as an example.

NOTE

GCC ARM Embedded 8.2.1 is used as an example in this document. The latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes*.

5.1 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.

5.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset (in other words, compiler, linker, etc.). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes Supporting MIMXRT1024-EVK* (document MCUXSDKMIMXRT1024RN) .

5.1.2 Install MinGW (only required on Windows OS)

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

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

NOTE

The installation path cannot contain any spaces.

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

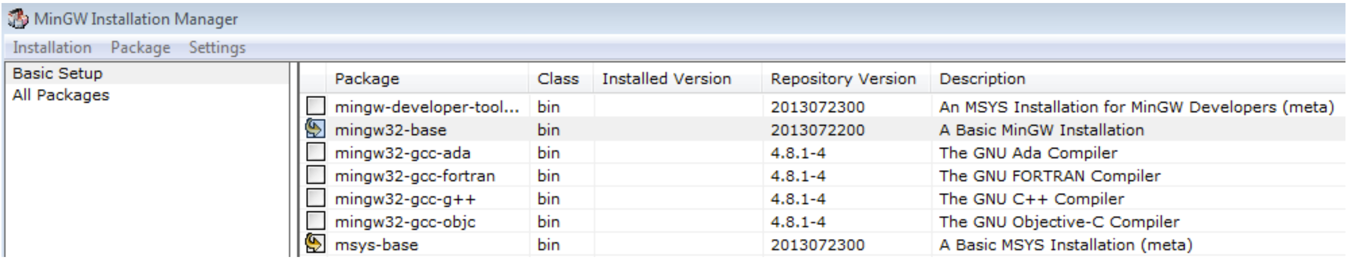


Figure 15. Set up MinGW and MSYS

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

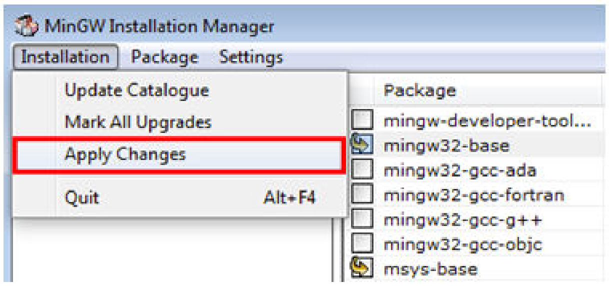


Figure 16. Complete MinGW and MSYS installation

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

```
<mingw_install_dir>\bin
```

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

NOTE

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

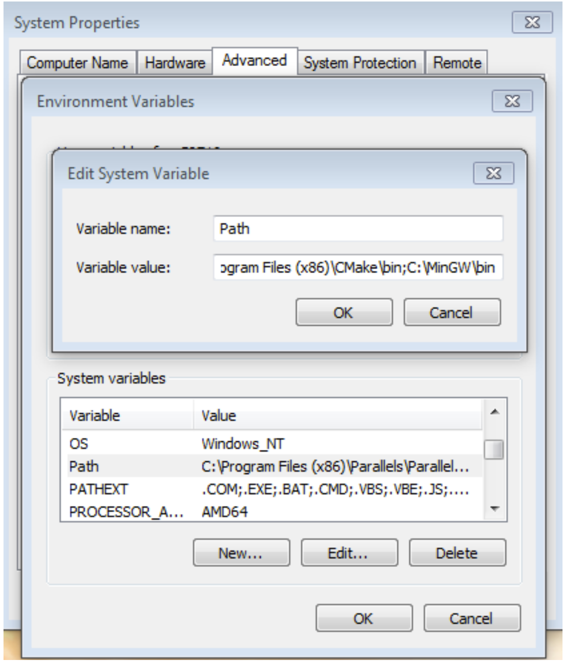


Figure 17. Add Path to systems environment

5.1.3 Add a new system environment variable for ARMGCC_DIR

Create a new *system* environment variable and name it 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 2019-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 command `for %I in (.) do echo %~sI` in above path.

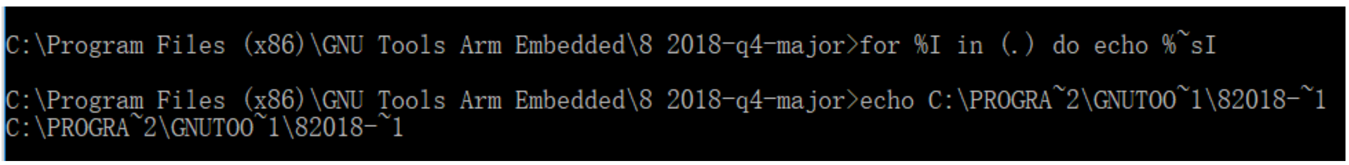


Figure 18. Convert path to short path

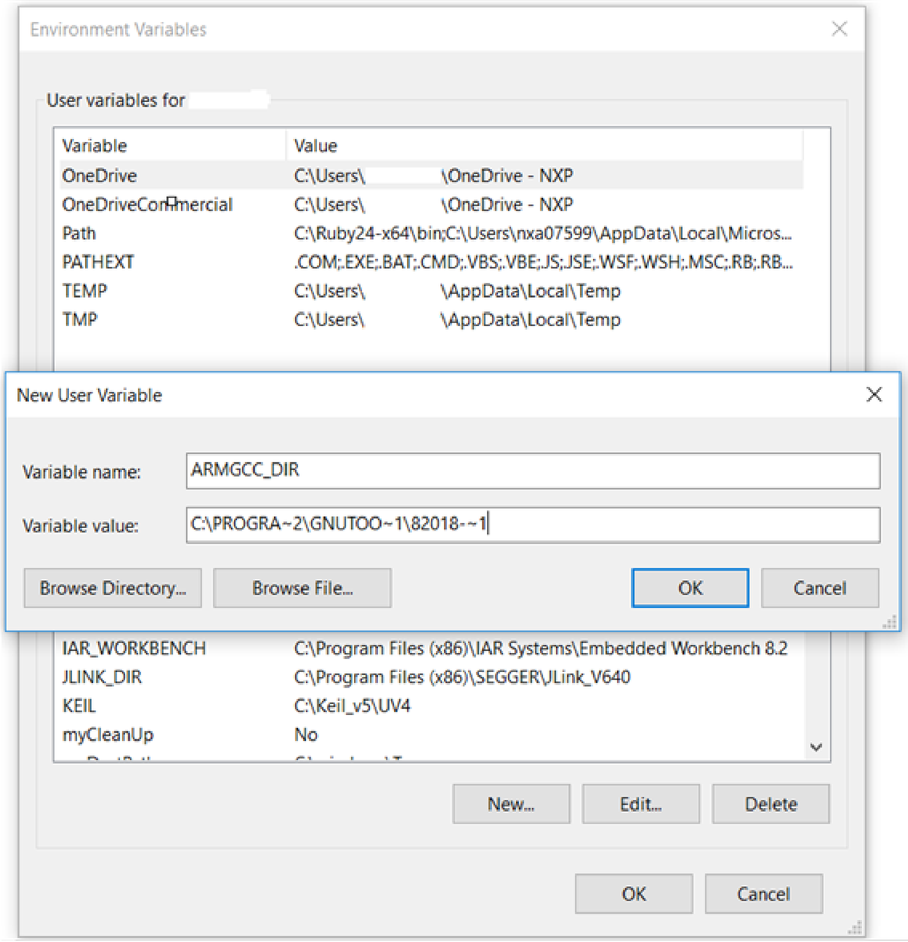


Figure 19. Add ARMGCC_DIR system variable

5.1.4 Install CMake

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

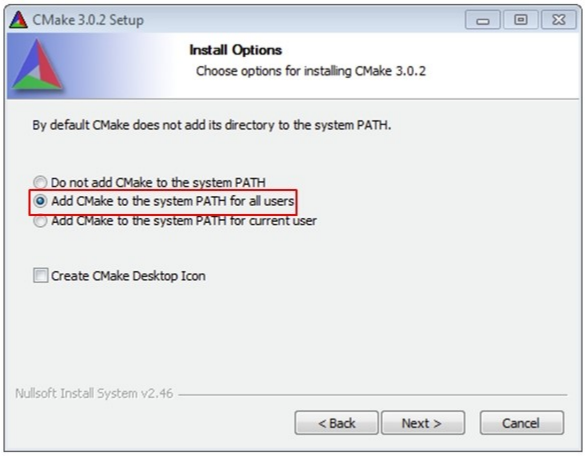


Figure 20. Install CMake

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

5.2 Build an 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**.

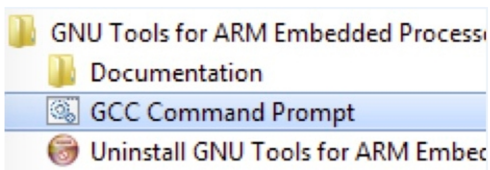


Figure 21. Launch command prompt

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

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

For this example, the exact path is:

```
<install_dir>/examples/evkmimxrt1024/demo_apps/hello_world/armgcc
```

NOTE

To change directories, use the `cd` command.

3. Type **build_debug.bat** on the command line or double click on **build_debug.bat** file in Windows Explorer to build it. The output is as shown in Figure 22.

Figure 22. `hello_world` **demo build successful**

5.3 Run an example application

This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, make sure that either:

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

NOTE

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

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

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

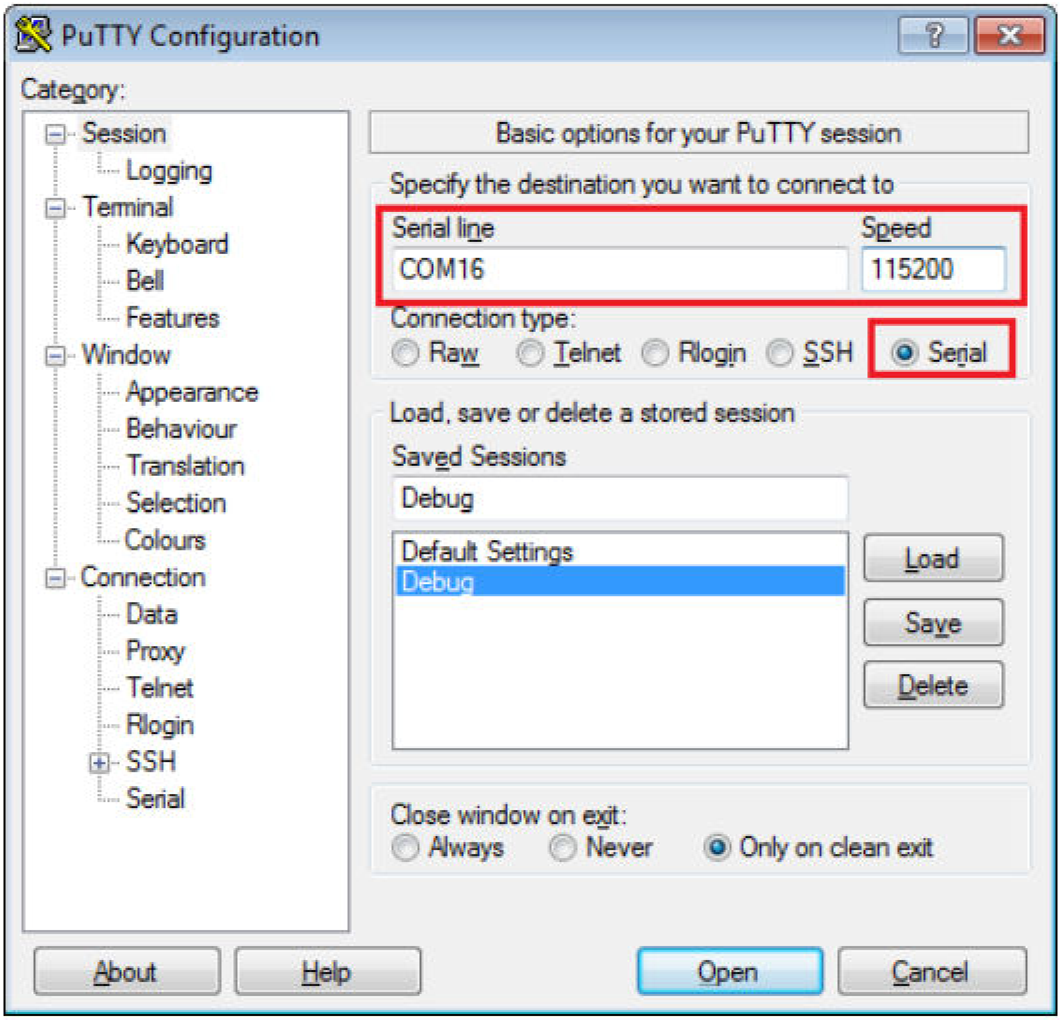


Figure 23. Terminal (PuTTY) configurations

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-scriptfile <install_dir>/boards/evkmimxrt1024/demo_apps/hello_world/evkmimxrt1024_sdram_init.jlinkscript` command
5. The target device selection chosen for this example is **MIMXRT1024DAG5A**.
6. After it is connected, the screen should resemble Figure 24.

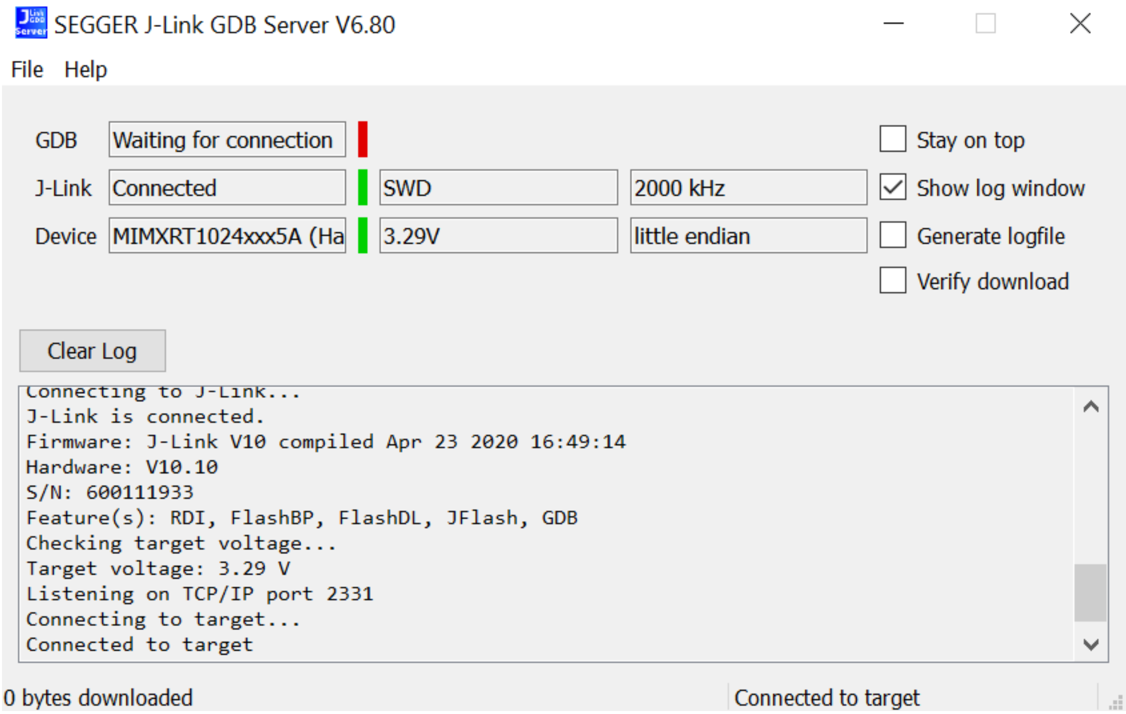


Figure 24. SEGGER J-Link GDB Server screen after successful connection

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**.

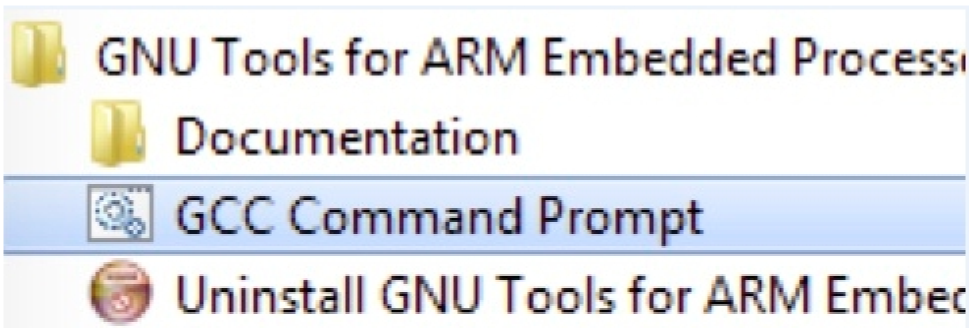


Figure 25. Launch command prompt

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>/armgcc/debug
```

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

For this example, the path is:

```
<install_dir>/boards/evkmimxrt1024/demo_apps/hello_world/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 26. Run `arm-none-eabi-gdb`

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

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

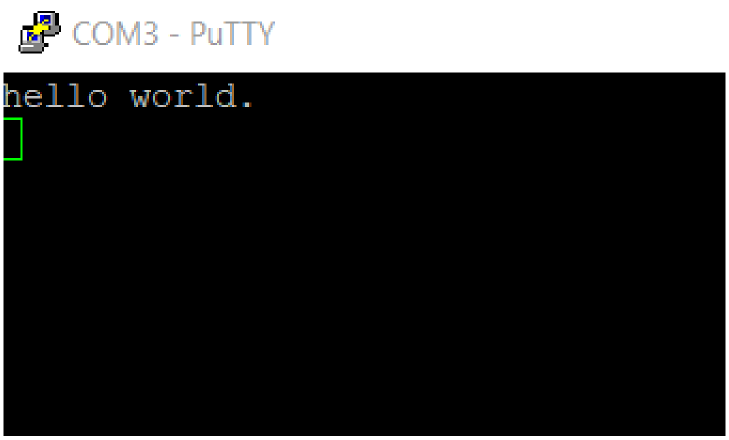


Figure 27. Text display of the `hello_world` demo

6 Run a demo using MCUXpresso IDE

NOTE

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

MCUXPresso IDE is not supported in this release.

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 MIMXRT1024-EVK platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

NOTE

By default, three macros, `XIP_EXTERNAL_FLASH=1`, `XIP_BOOT_HEADER_ENABLE=1`, and `XIP_BOOT_HEADER_DCD_ENABLE=1`, are set in the project. If you do not use `Board_Flash` in the project, these macros should be removed or set value to **0** in project settings.

6.1 Select the workspace location

Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects 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.

6.2 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 the MCUXpresso SDK. In the window that appears, click **OK** and wait until the import has finished.

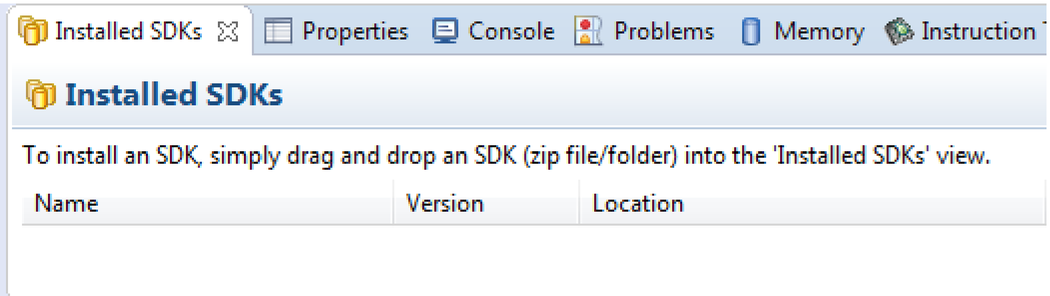


Figure 28. Install an SDK

2. On the **Quickstart Panel**, click **Import SDK example(s)…**, as shwon in Figure 29.

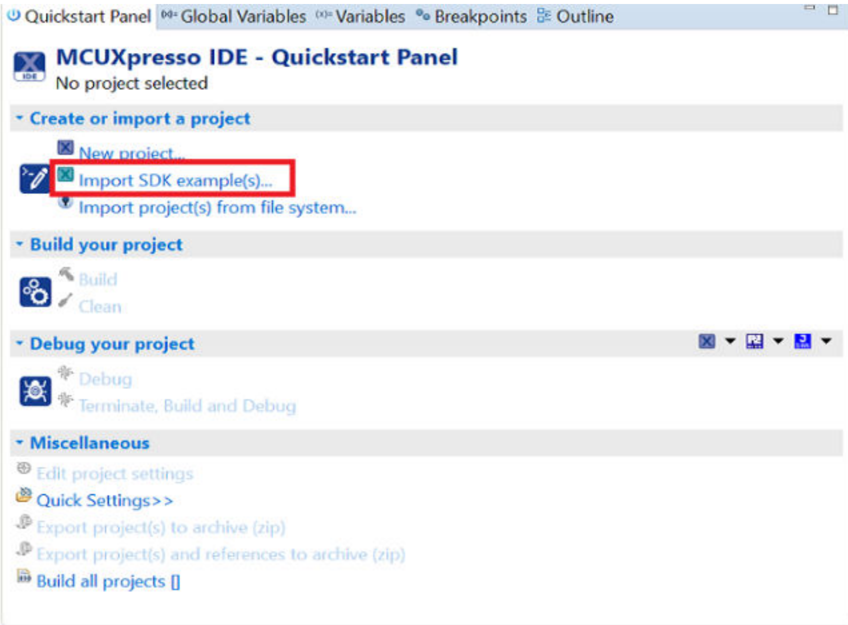


Figure 29. Import an SDK example

3. In the window that appears, expand the **MIMXRT1024** folder and select **MIMXRT1024xxxxx**. Then, select **evkmimxrt1024** and click **Next**, as shown in Figure 30.

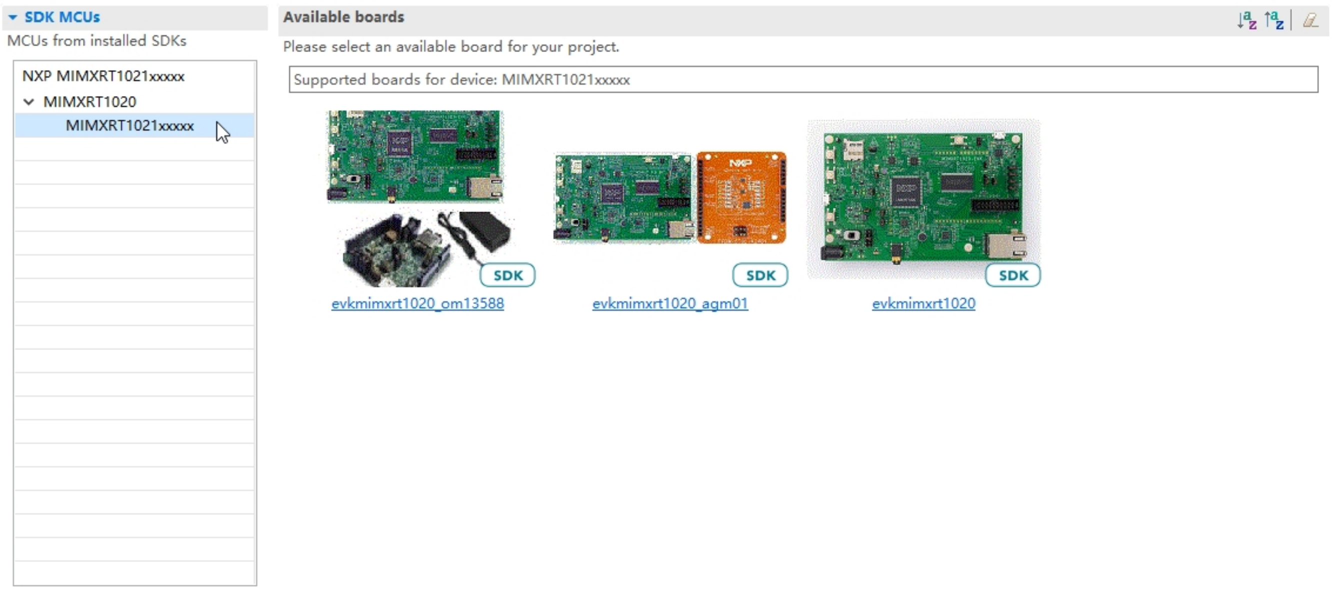


Figure 30. Selecting MIMXRT1024-EVK board

4. Expand the `demo_apps` folder, select `hello_world`, and then click **Next**, as shown in .
5. Ensure the option **Redlib: Use floating point version of printf** is selected if the cases print floating point numbers on the terminal (for demo applications such as `dac32_adc12`, `dac_adc`, `dac_cadc`, `ecompass`, `sai`, `coremark`, `mbedtls_benchmark`, `wolfssl_benchmark`, and for `mmcau_examples` such as `mmcau_api`). Otherwise, there is no need to select it. Click **Finish**.

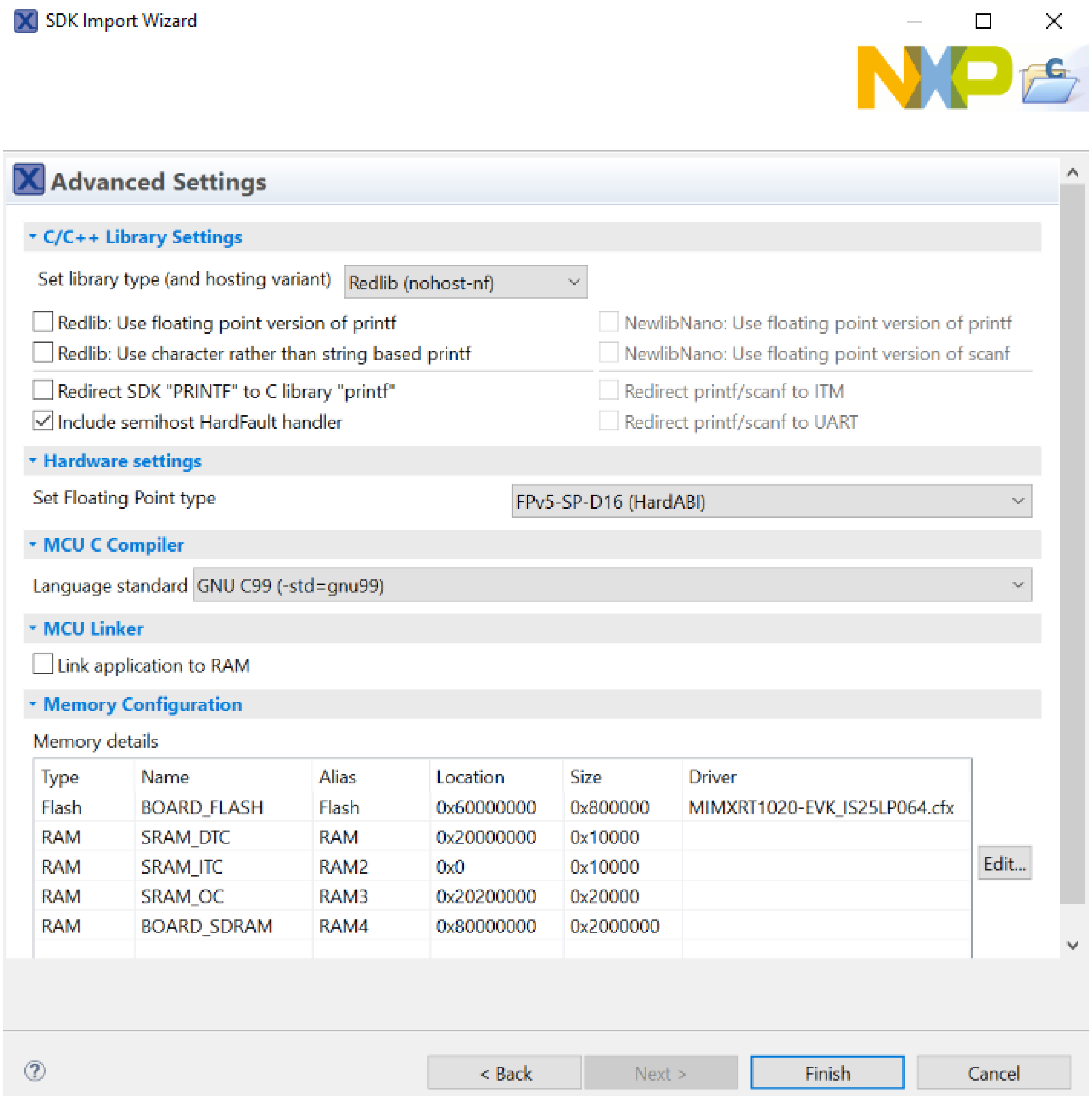


Figure 31. Selecting User floating point version of printf

NOTE

If you want to use semihost to print log, first select the **Semihost** button when importing projects, as shown in Figure 32.

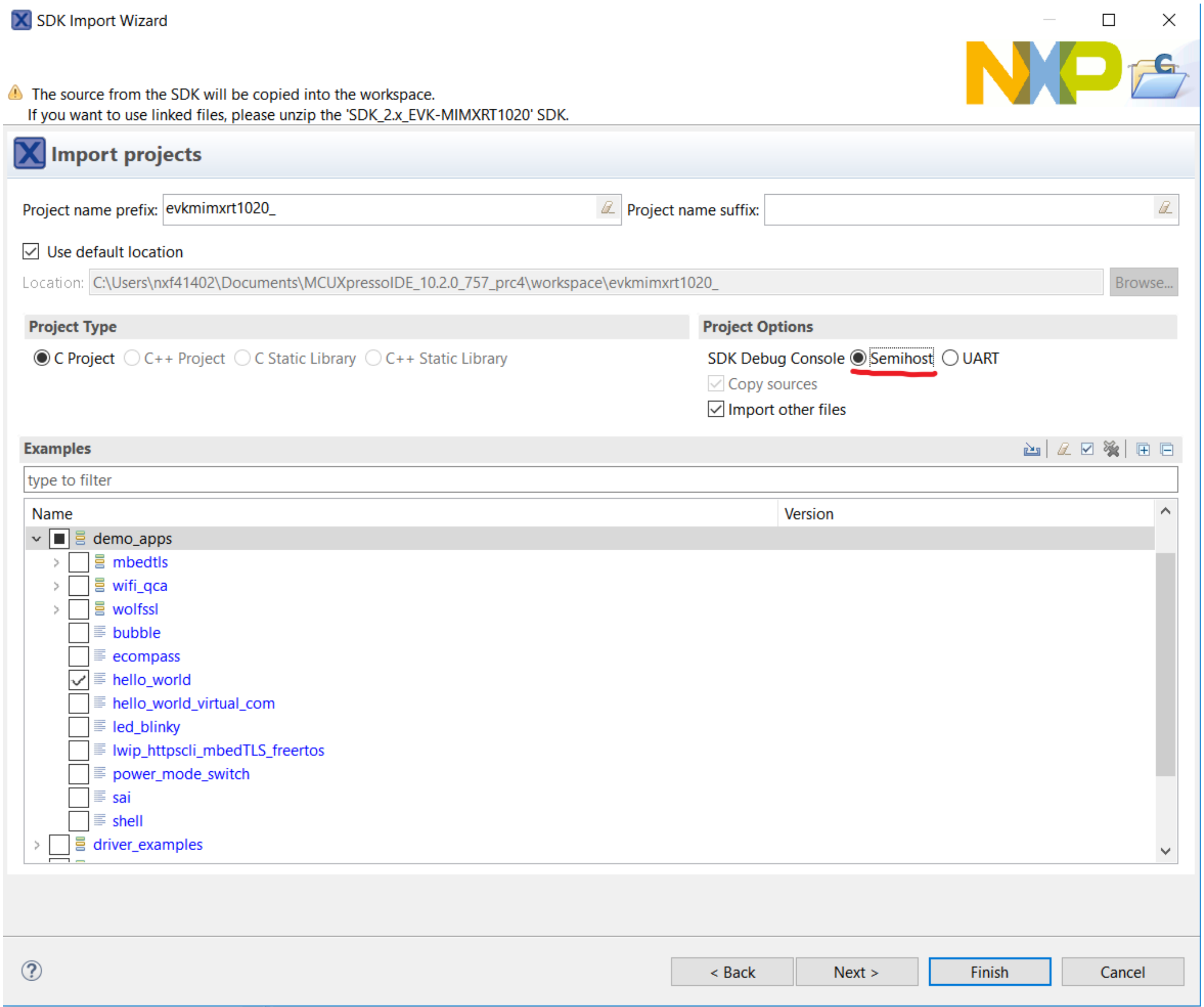


Figure 32. Selecting Semihost

6. On the **Quickstart** panel, click **build** **`evkmimxrt1024_demo_apps_hello_world [Debug]`**, as shown in Figure 33.

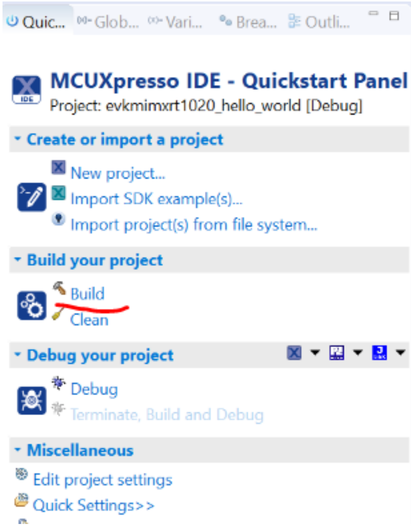


Figure 33. Building `hello world` case

6.3 Run an example application

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

To download and run the application, perform these steps:

NOTE

Make sure that the board is on the QSPI_Flash mode before download (set SW8: 0010).

1. On the **Quickstart Panel**, click **Debug evkmimxrt1024_demo_apps_hello_world [Debug]**.

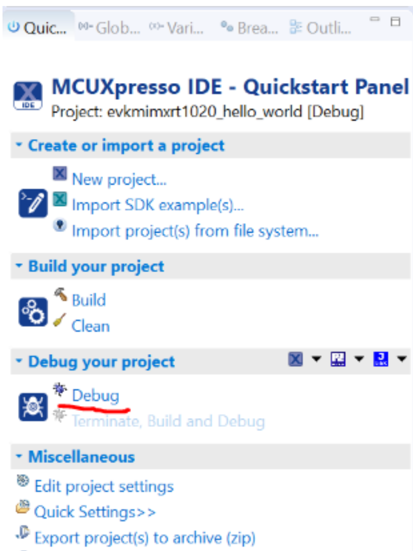


Figure 34. Debugging `hello_world` case

2. 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.)

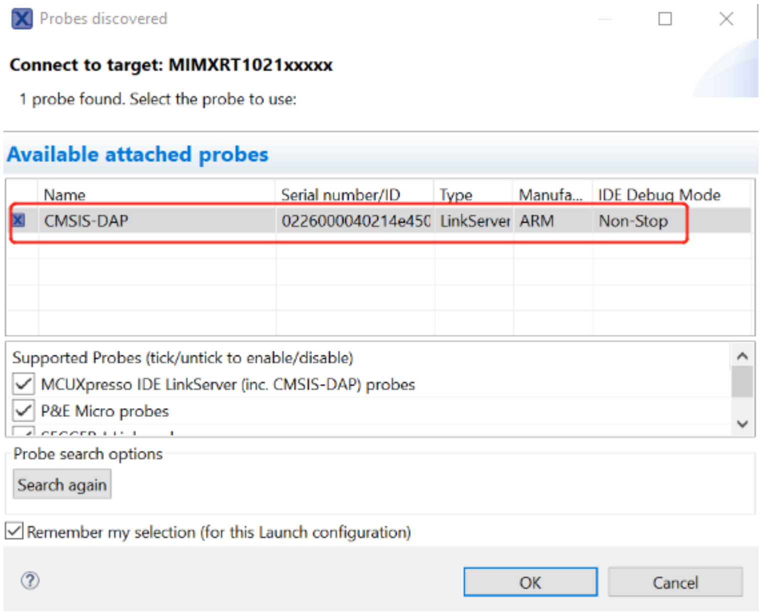


Figure 35. Attached Probes: debug emulator selection

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

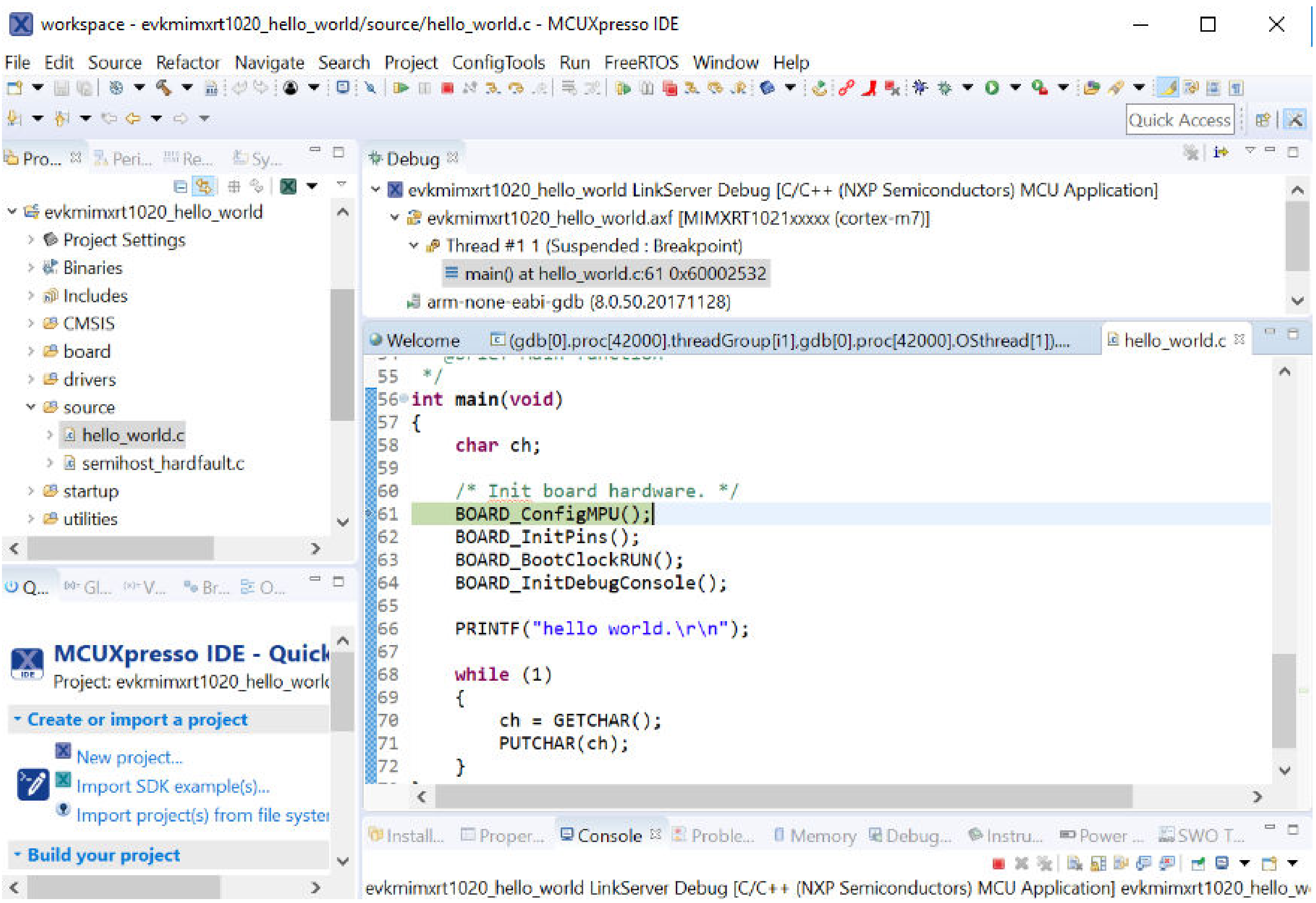


Figure 36. Stop at `main()` when running debugging

4. Start the application by clicking the **Resume** button.

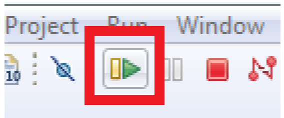


Figure 37. Resume button

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

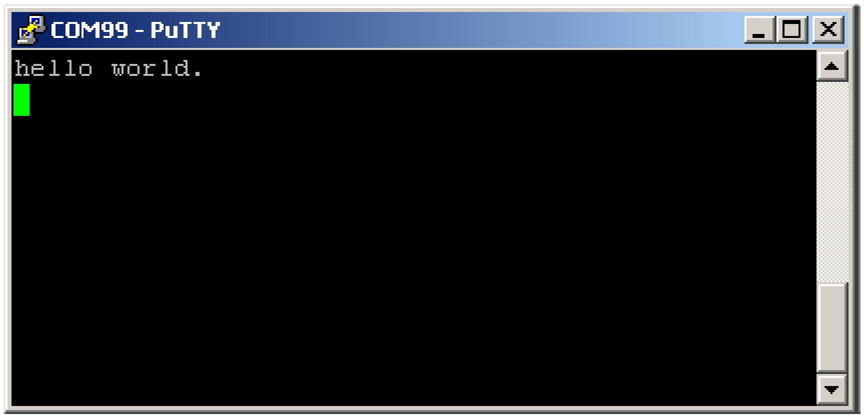


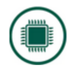
Figure 38. Text display of the `hello_world` demo

7 MCUXpresso Config Tools

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

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

Table 1. MCUXpresso Config Tools

Config Tool	Description	Image
Pins tool	For configuration of pin routing and pin electrical properties.	
Clock tool	For system clock configuration	
Peripherals tools	For configuration of other peripherals	
TEE tool	Configures access policies for memory area and peripherals helping to protect and isolate sensitive parts of the application.	
Device Configuration tool	Configures Device Configuration Data (DCD) contained in the program image that the Boot ROM code interprets to setup various on-chip peripherals prior the program launch.	

MCUXpresso Config Tools can be accessed in the following products:

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

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

8 MCUXpresso IDE New Project Wizard

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

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in Figure 39.

Figure 39. MCUXpresso IDE Quickstart Panel

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

Appendix A How to determine COM port

This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

1. **Linux**: The serial port can be determined by running the following command after the USB Serial is connected to the host:

```
$ dmesg | grep "ttyUSB"
                    [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
                    [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
```

There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click on the **Start** menu and type **Device Manager** in the search bar.

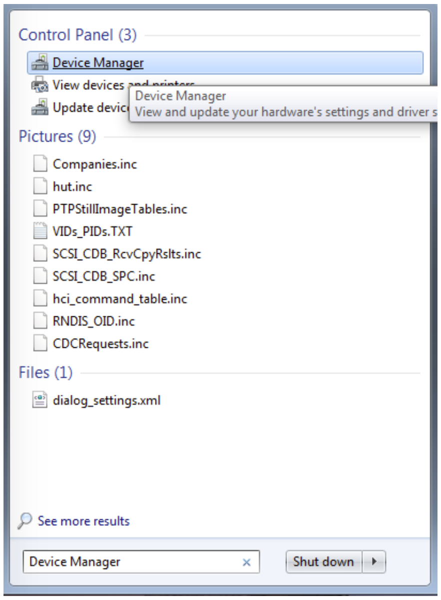


Figure A-1. Device Manager

3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names will be different for all the NXP boards.

Appendix B How to add or remove boot header for XIP targets

The MCUXpresso SDK for i.MX RT1024 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 B-1.

Table B-1. 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 B-2](#) shows the different effect on the built image with a different combination of these macros.

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

		XIP_BOOT_HEADER_DCD_ENABLE=1	XIP_BOOT_HEADER_DCD_ENABLE=0
XIP_EXTERNAL_FLASH=1	**XIP_BOOT_HEADER_ENABLE=1**	• Can be programmed to `qspiflash` by IDE and can run after POR reset if `qspiflash` is the boot source. • SDRAM will be initialized.	• Can be programmed to `qspiflash` by IDE, and can run after POR reset if `qspiflash` is the boot source. • SDRAM will **NOT** be initialized.
	XIP_BOOT_HEADER_ENABLE=0	• **CANNOT** run after POR reset if it is programmed by IDE, even if `qspiflash` is the boot source.	
XIP_EXTERNAL_FLASH=0		• This image **CANNOT** complete XIP because when this macro is set to 1, it excludes the code, which changes the clock for FLEXSPI.	

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

Take `hello_world` as an example:

- **IAR**

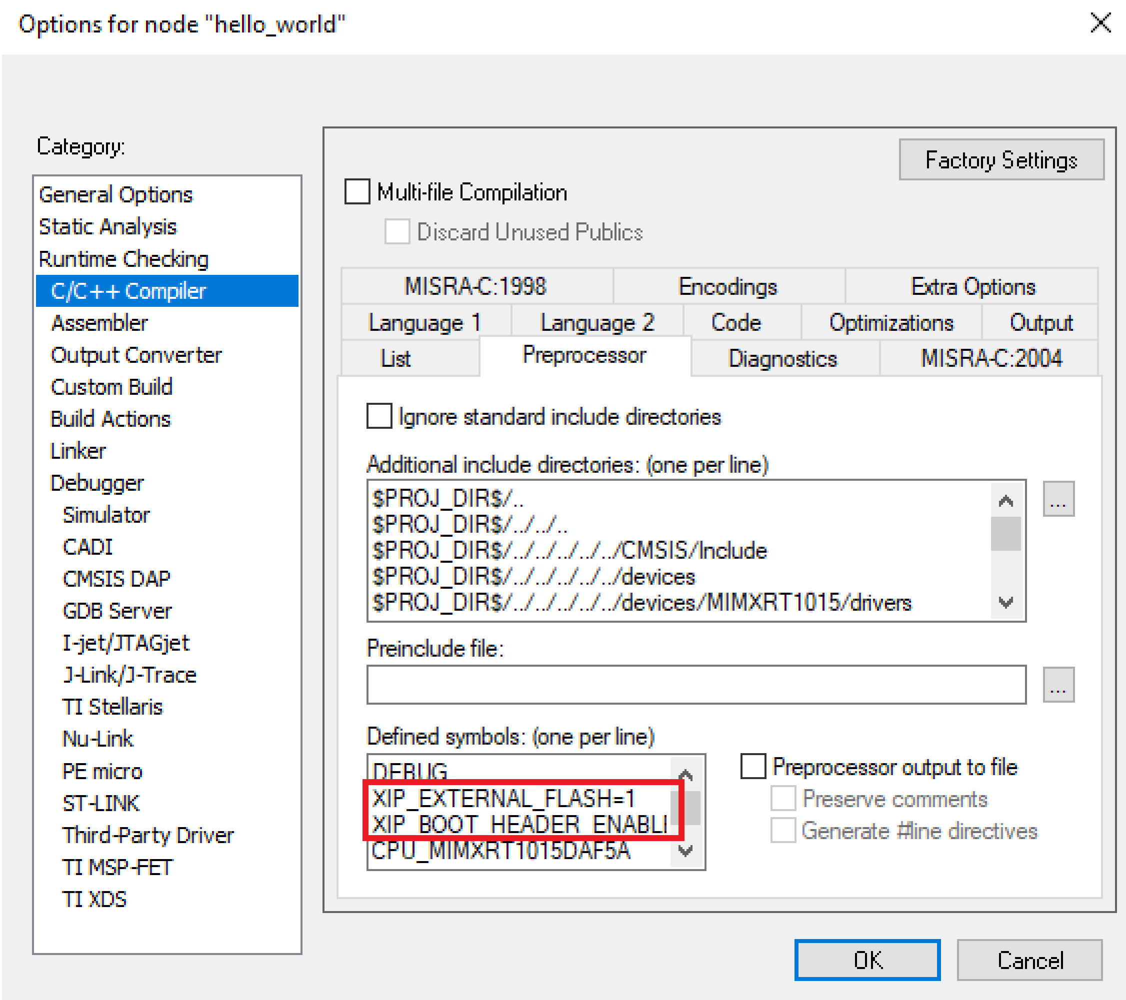


Figure B-1. Options node IAR

- **MDK**

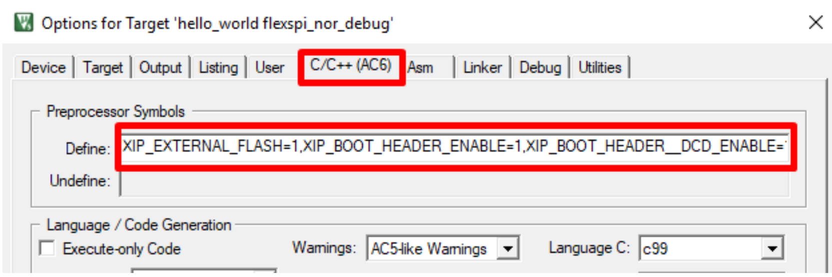


Figure B-2. Options for target

- **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 B-3. Change configuration CMakeLists.txt

- **MCUX**

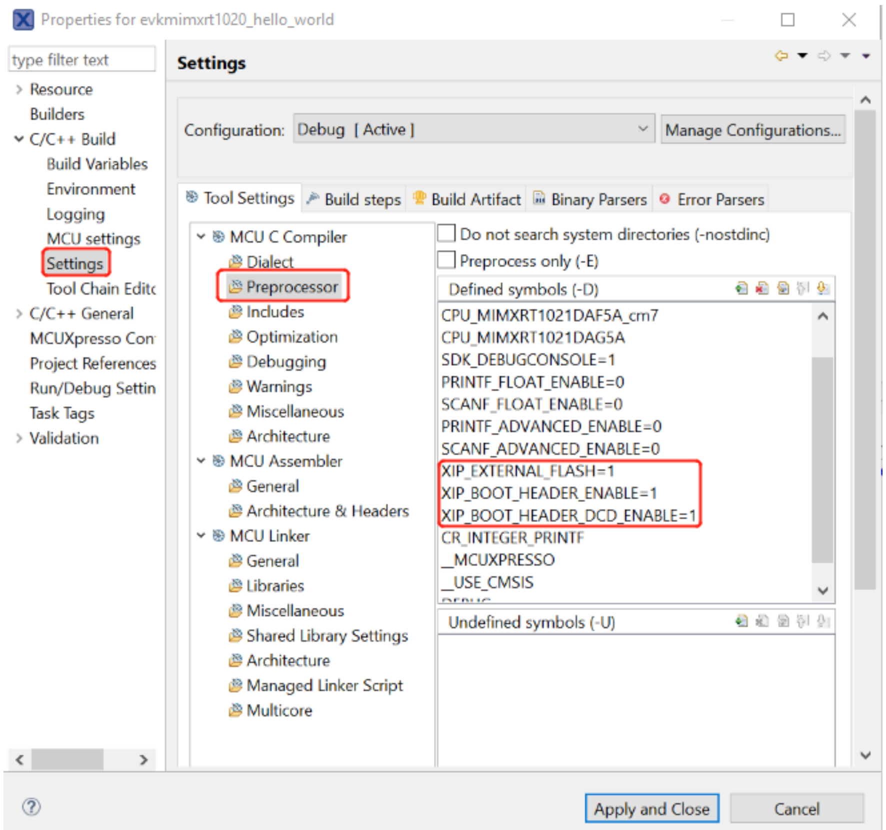


Figure B-4. Properties for evkbimxrt1024

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MCUXSDKMIMXRT1024GSUG
Revision 0, July 2020