

MCUXSDKMIMXRT1064GSUG

Getting Started with MCUXpresso SDK for EVK-MIMXRT1064

Rev. 2.15.000 — 10 January 2024

User guide

Document information

Information	Content
Keywords	MCUXSDKMIMXRT1064GSUG, EVK-MIMXRT1064, EVKMIMXRT1064, MIMXRT1064, Getting Started
Abstract	This document describes the steps to get started with MCUXpresso SDK for EVK-MIMXRT1064.



1 Overview

The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on general purpose, crossover, and Bluetooth-enabled MCUs from NXP. 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 optional RTOS integrations such as FreeRTOS and Azure RTOS, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for EVK-MIMXRT1064* (document MCUXSDKMIMXRT1064RN).

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

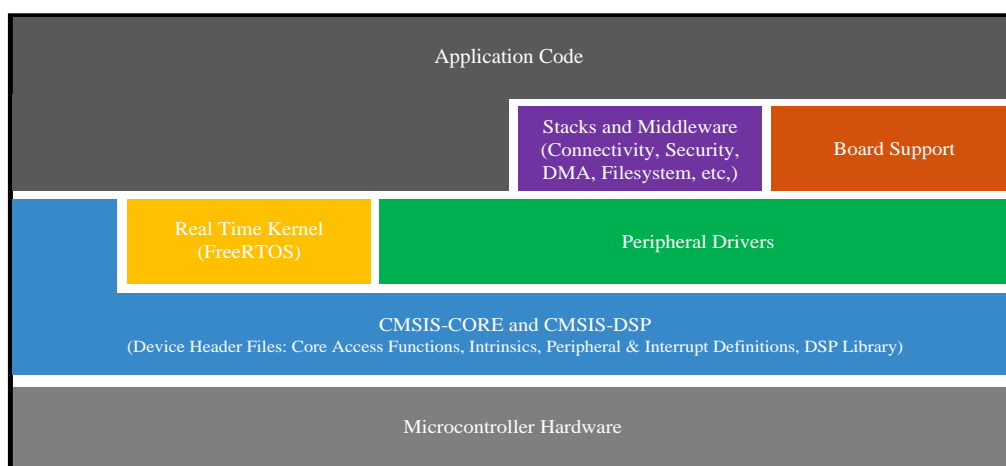


Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support package folders

MCUXpresso SDK board support package provides example applications for NXP development and evaluation boards for Arm Cortex-M cores including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside the top-level boards folder and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each `<board_name>` folder, there are various subfolders to classify the type of examples it contains. 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:

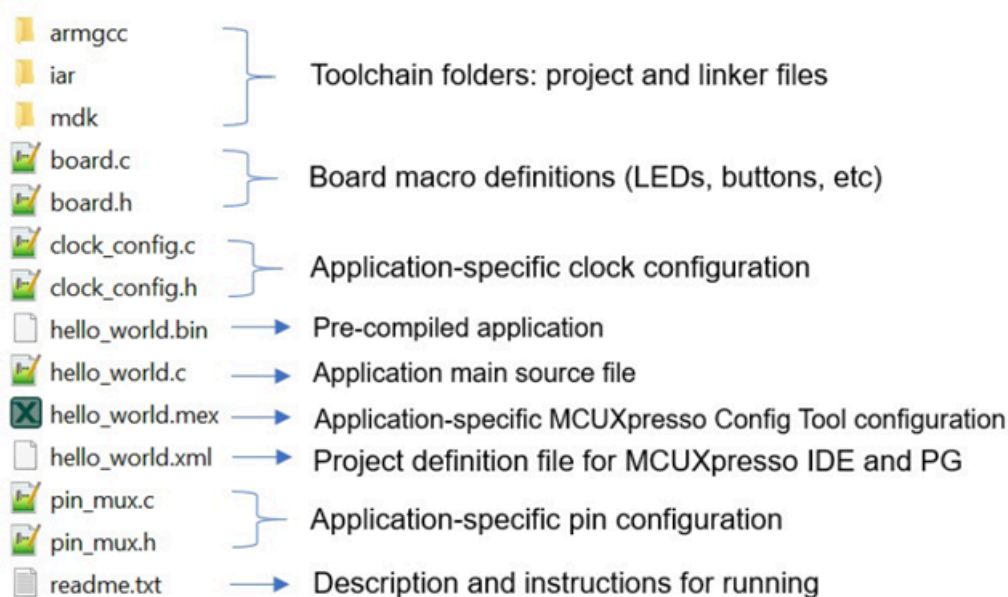


Figure 2. Application folder structure

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, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that 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

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

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the MIMXRT1064-EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

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 MIMXRT1064-EVK hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/evkmimxrt1064/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**.

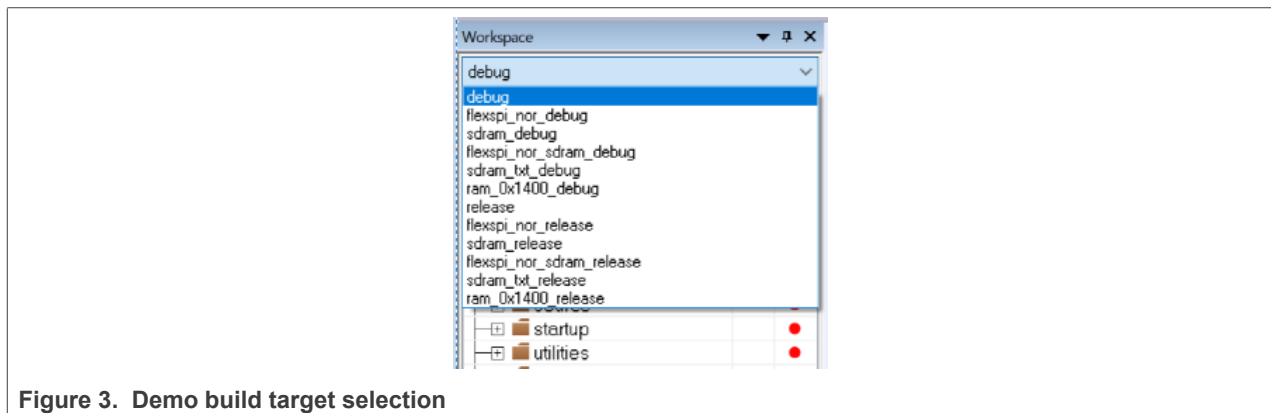


Figure 3. Demo build target selection

- To build the demo application, click **Make**, highlighted in red in [Figure 4](#).

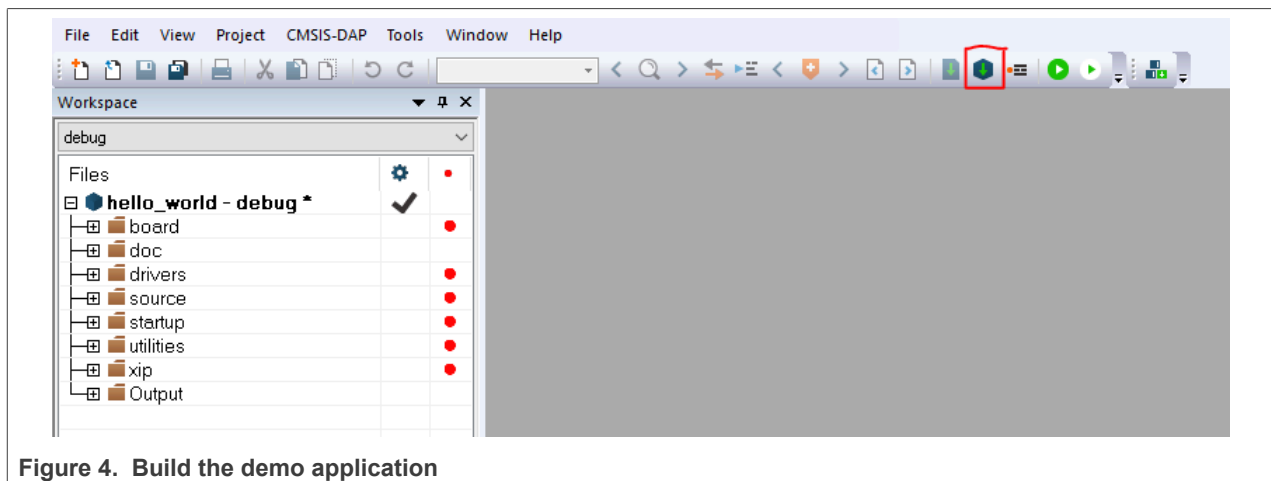


Figure 4. Build the demo application

- The build completes without errors.

3.2 Run an example application

To download and run the application, perform these steps:

- 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.
- Connect the development platform to your PC via USB cable. Connect the USB cable to J41 and make sure SW7[1:4] is **0010b**.
- 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:
 - 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
 - No parity
 - 8 data bits
 - 1 stop bit

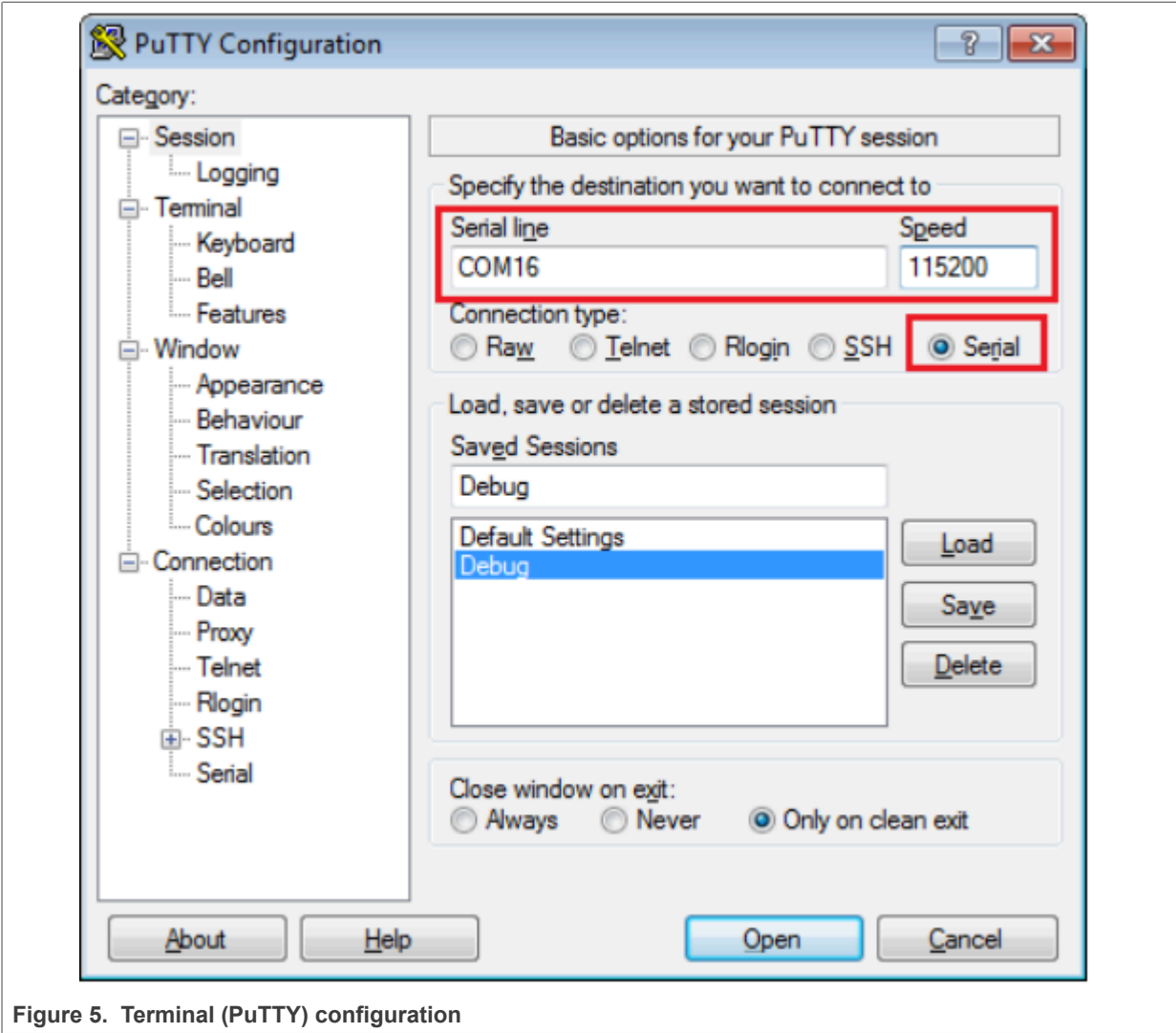


Figure 5. Terminal (PuTTY) configuration

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

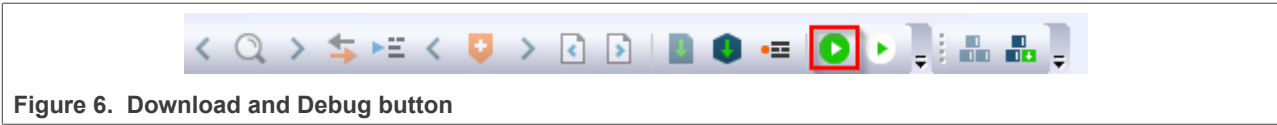


Figure 6. Download and Debug button

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

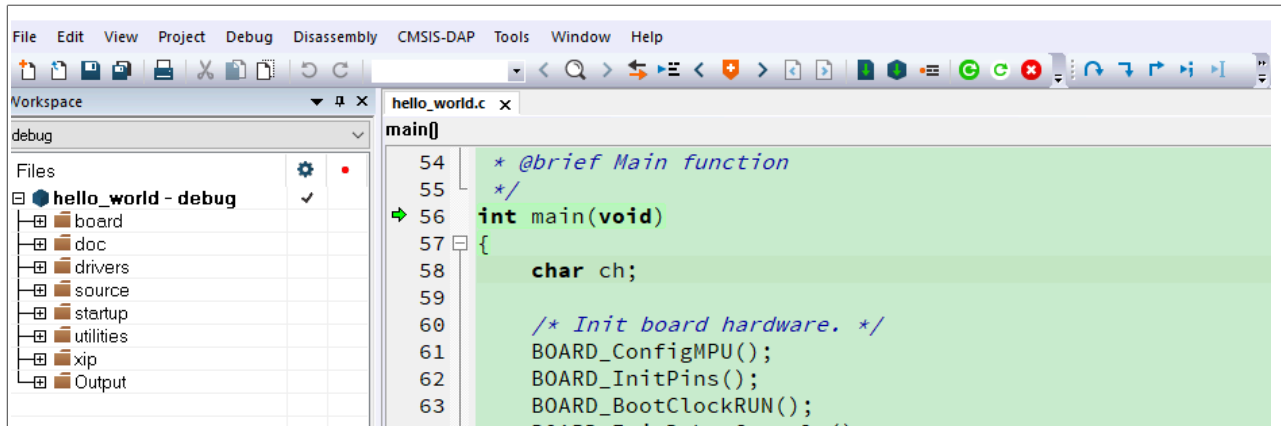


Figure 7. Stop at `main()` when running debugging

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.

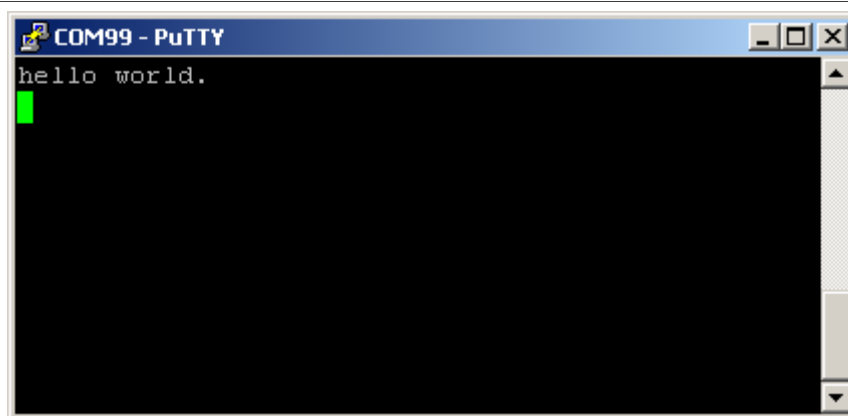


Figure 9. Text display of the `hello_world` demo

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 CMSIS pack.

1. Download the MIMXRT1064 pack.
2. After downloading the DFP, double click to install it.

4.2 Build an example application

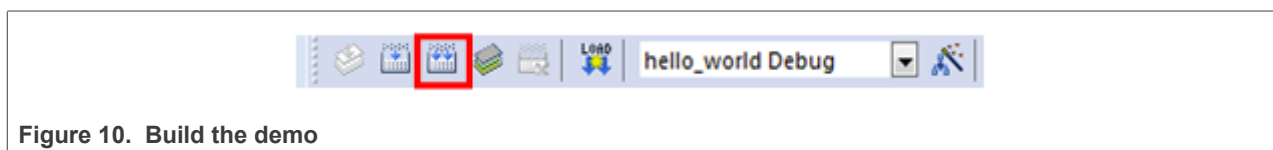
1. Open the desired example application workspace in:

```
<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/evkmimxrt1064/demo_apps/hello_world/mdk/hello_world.uvmpw
```

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



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 [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

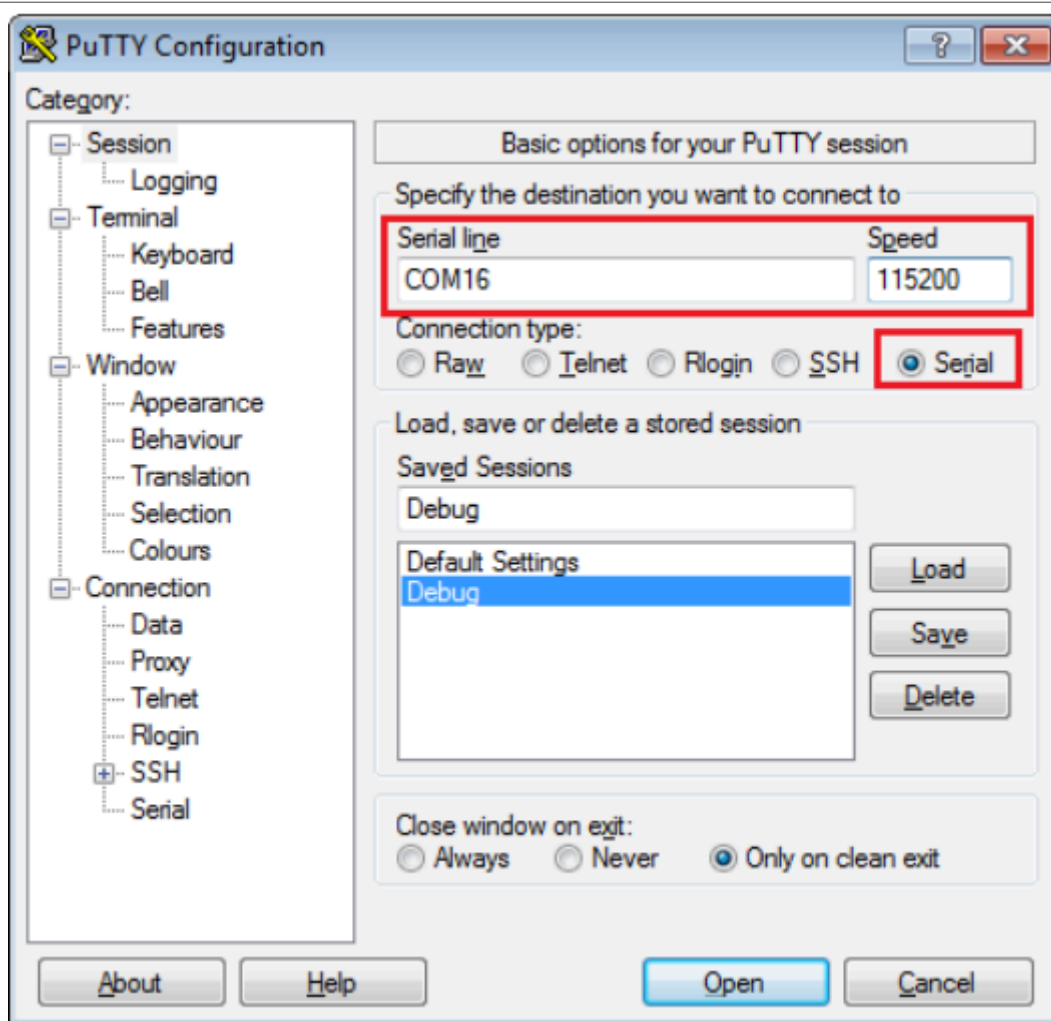


Figure 11. Terminal (PuTTY) configurations

- 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**.

Note: When debugging with *jlink*, it expects one *jlinkscript* file named *JLinkSettings.JLinkScript* in the folder where the *uVision* project files are located. For details, see *Segger Wiki*. For the contents in this *JlinkSettings.JLinkScript*, use contents in *evkmimxrt1020_sdram_init.jlinkscript*.

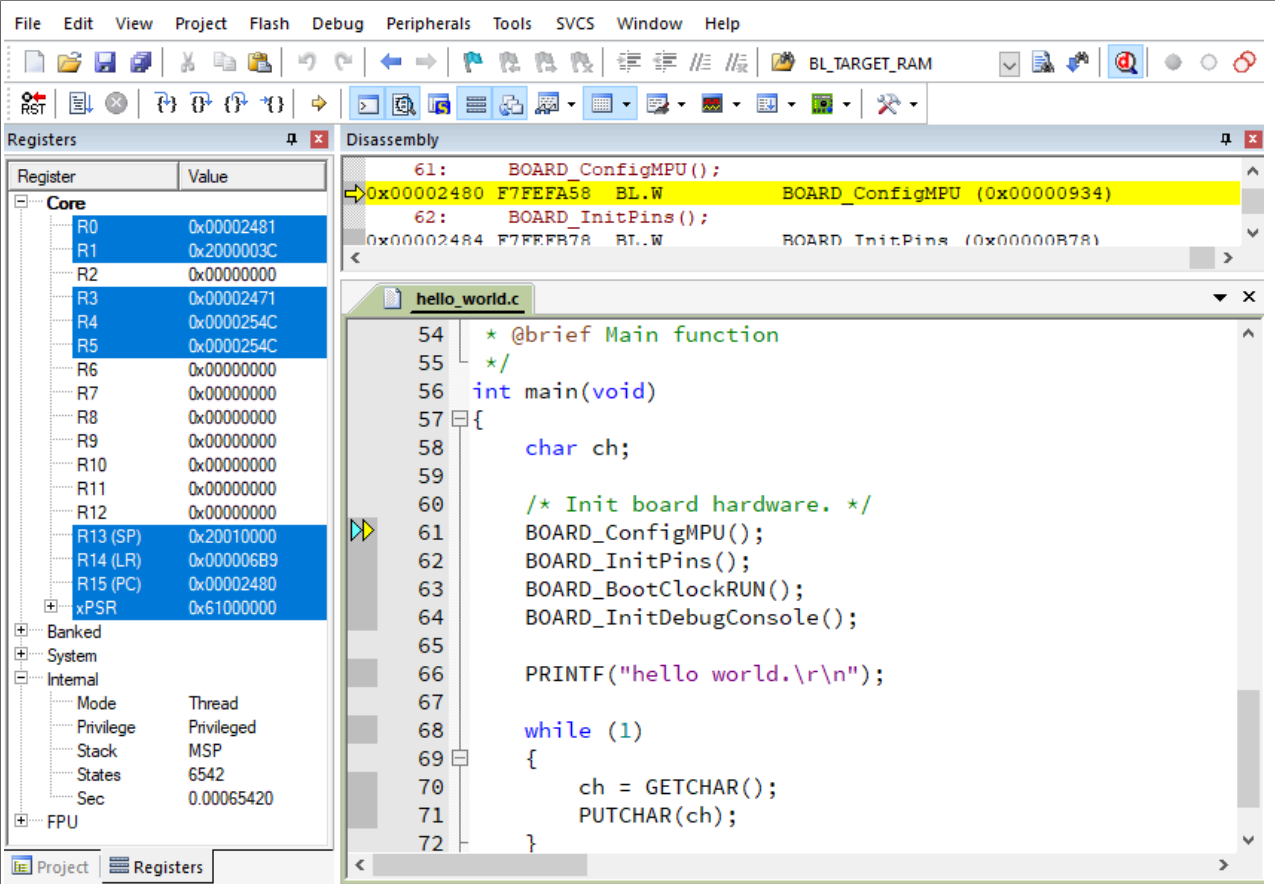


Figure 12. Stop at main() when run debugging

5. Run the code by clicking **Run** to start the application, as shown in [Figure 13](#).

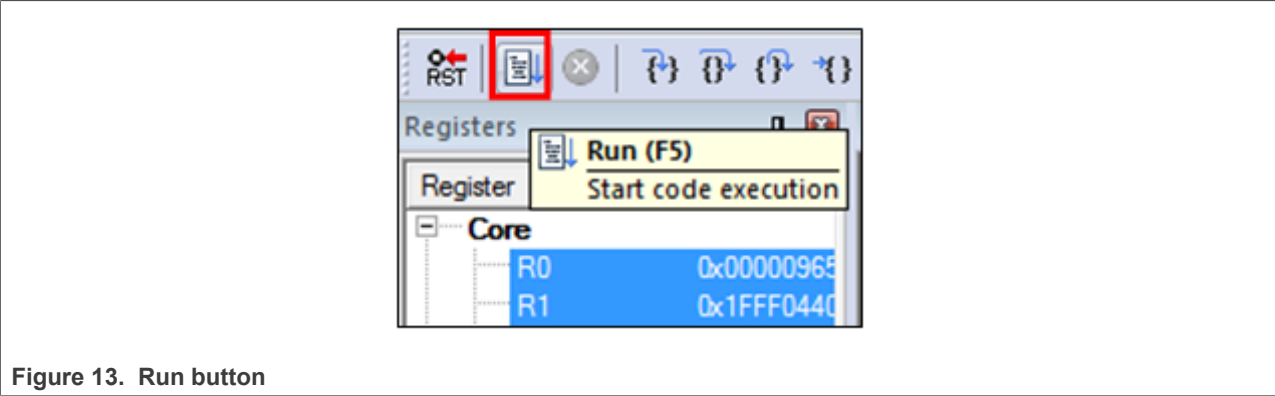
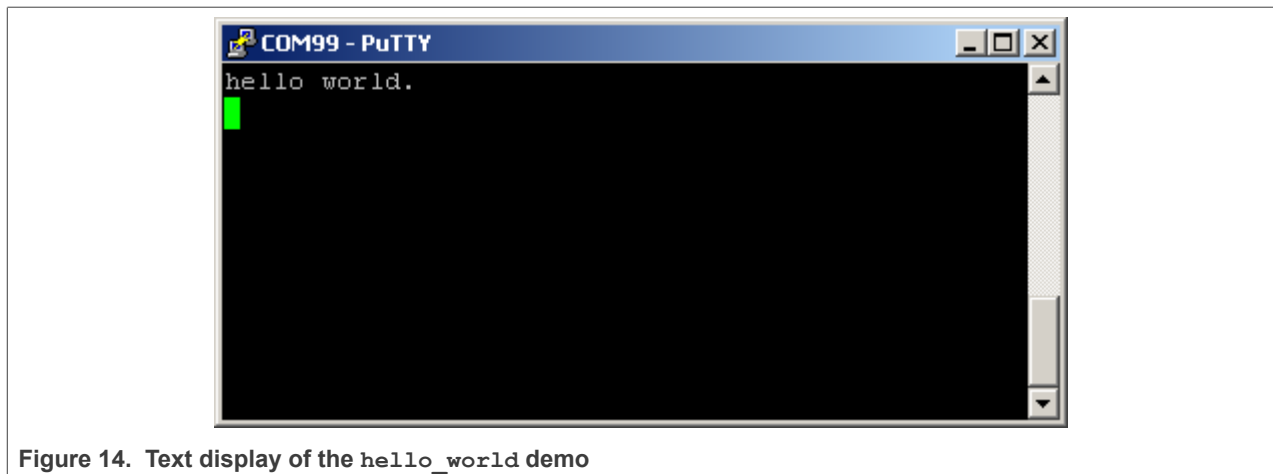


Figure 13. Run button

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.



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 which is used as an example.

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, and so on). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes Supporting MIMXRT1064-EVK* (document MCUXSDKMIMXRT1064RN).

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 [MinGW](https://www.mingw-w64.org/).
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**.

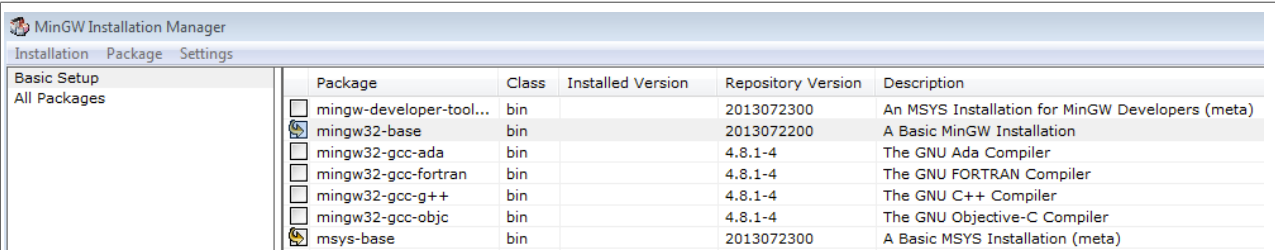


Figure 15. Set up MinGW and MSYS

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

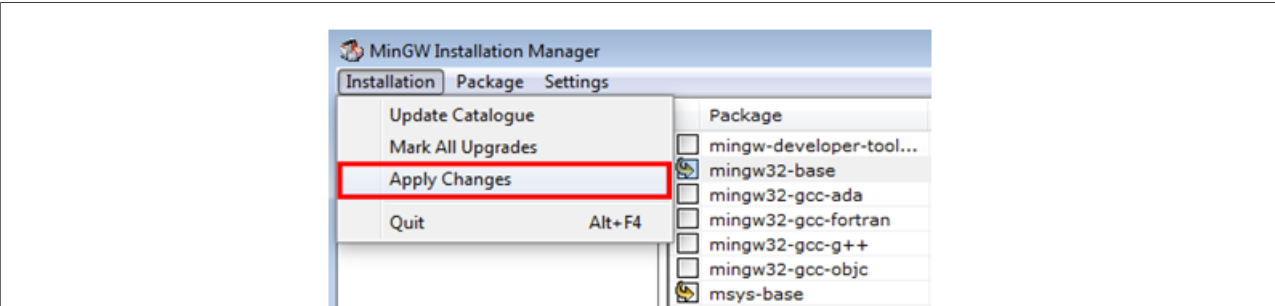


Figure 16. Complete MinGW and MSYS installation

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 work.

Note: If you have C:\MinGW\msys*.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.

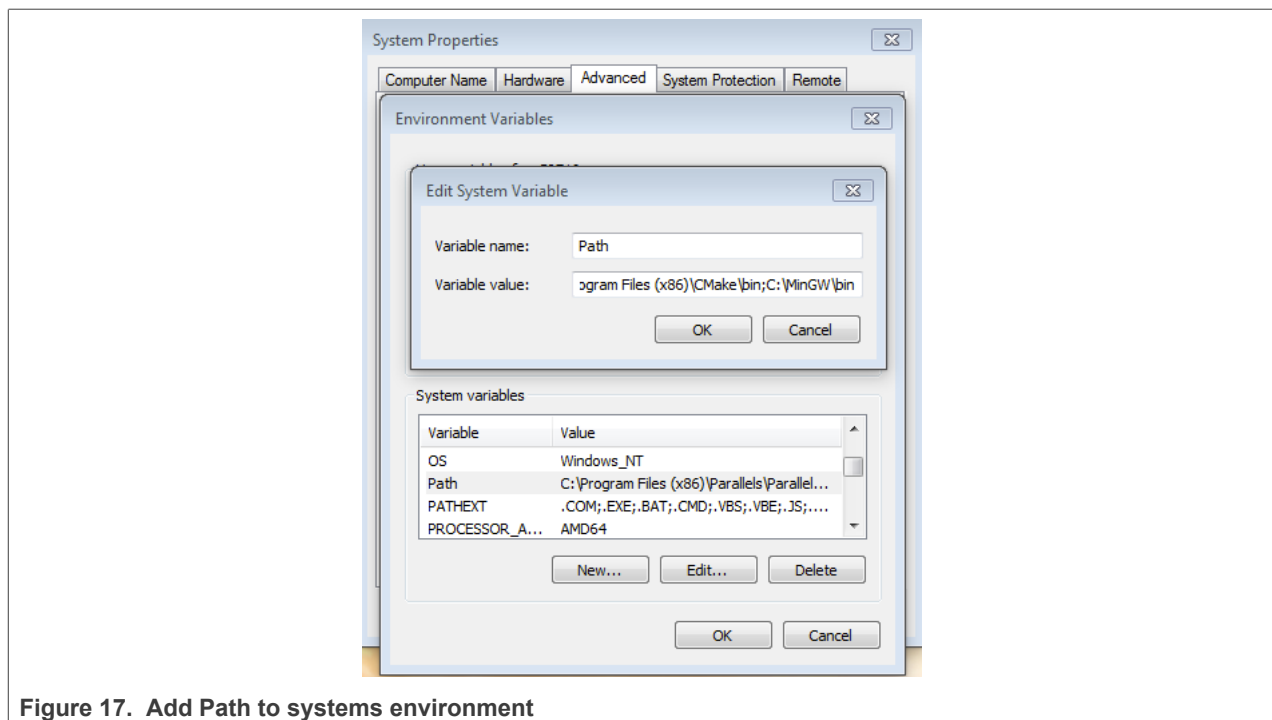


Figure 17. Add Path to systems environment

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:

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

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

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

Figure 18. Convert path to short path

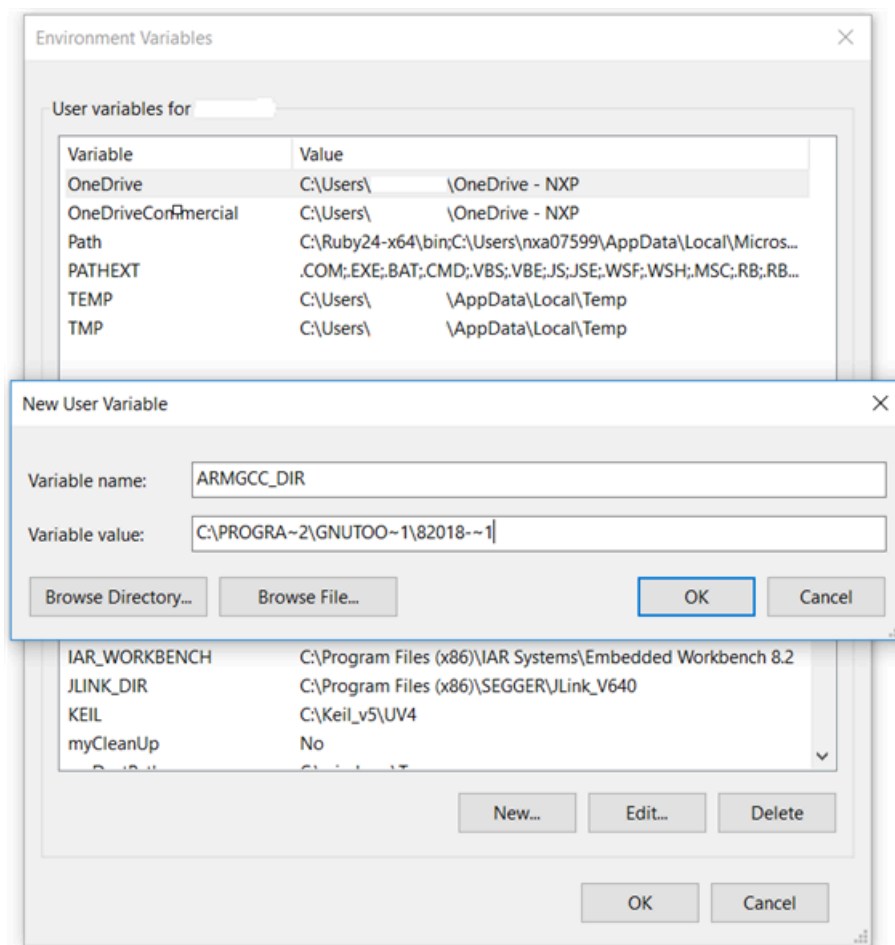


Figure 19. Add ARMGCC_DIR system variable

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.

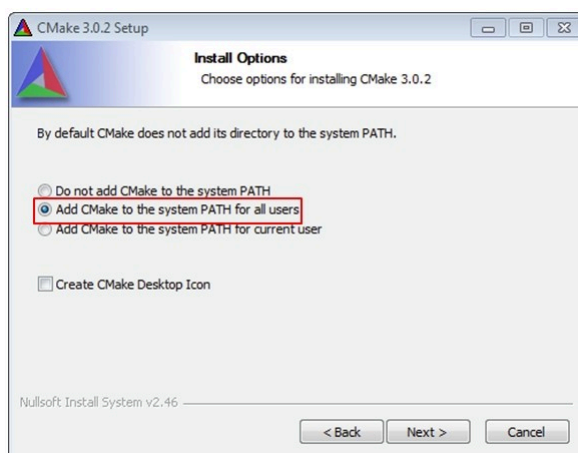


Figure 20. Install CMake

- Follow the remaining instructions of the installer.
- You may need to reboot your system for the PATH changes to take effect.
- 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.

- 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**.

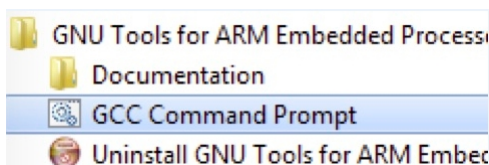


Figure 21. Launch command prompt

- 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/evkmimxrt1064/demo_apps/hello_world/armgcc
```

Note: To change directories, use the `cd` command.

- 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.

```
[ 95%] Building C object CMakeFiles/hello_world.dir/C:/sdk_repo/mcu-sdk-2.0/boards/evkmimxrt1060/xip/evkmimxrt1060_s
ram_ini_dcd.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\sdk_repo\mcu-sdk-2.0\boards\evkmimxrt1060\demo_apps\hello_world\armgcc>IF "" == "" (pause )
Press any key to continue . . .
```

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

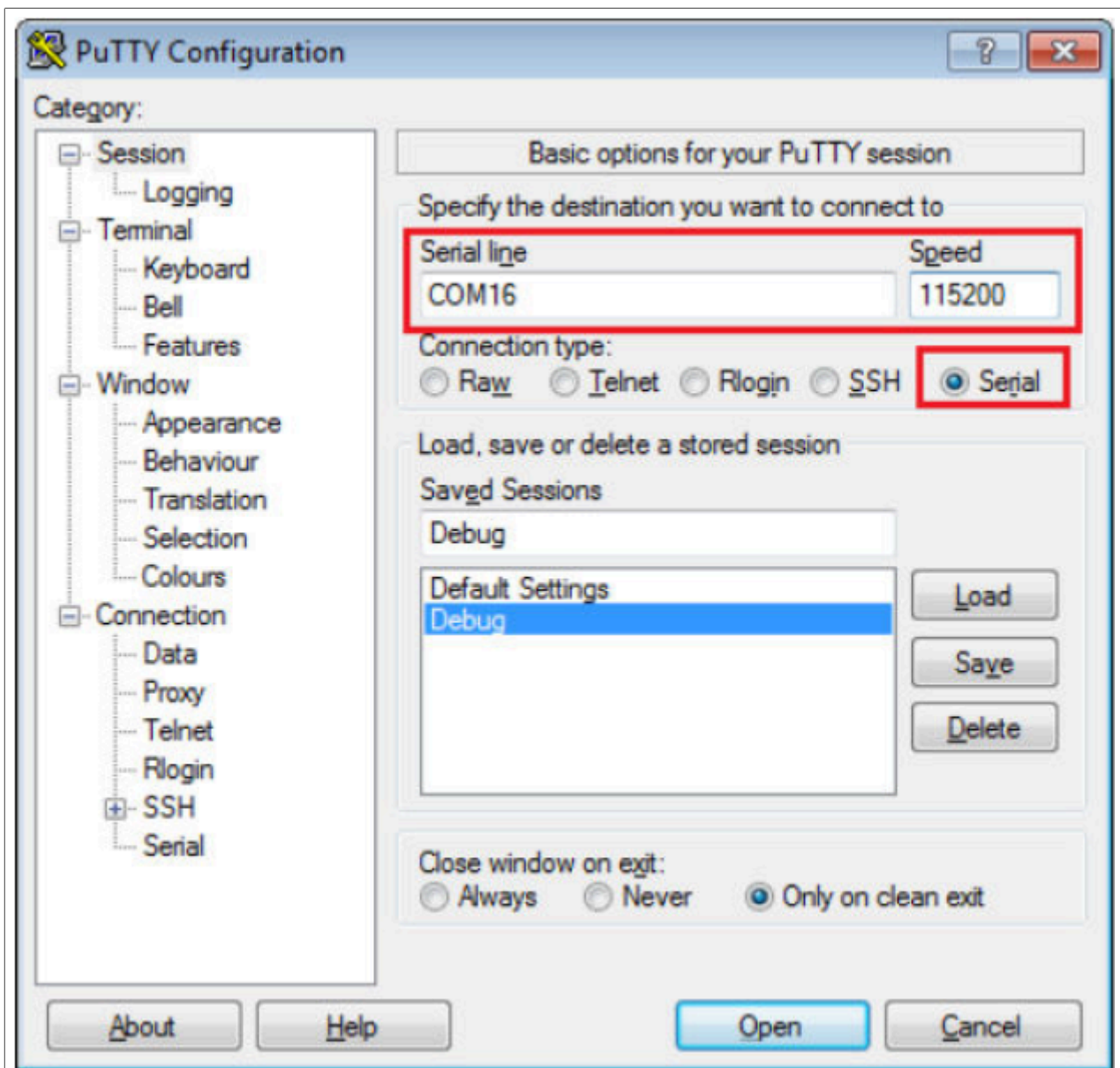


Figure 23. Terminal (PuTTY) configurations

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/evkmimxrt1064/demo_apps/hello_world/evkmimxrt1064_sdram_init.jlinkscript` command
5. The target device selection chosen for this example is **MIMXRT1064DVL6A**.
6. After it is connected, the screen should resemble [Figure 24](#).

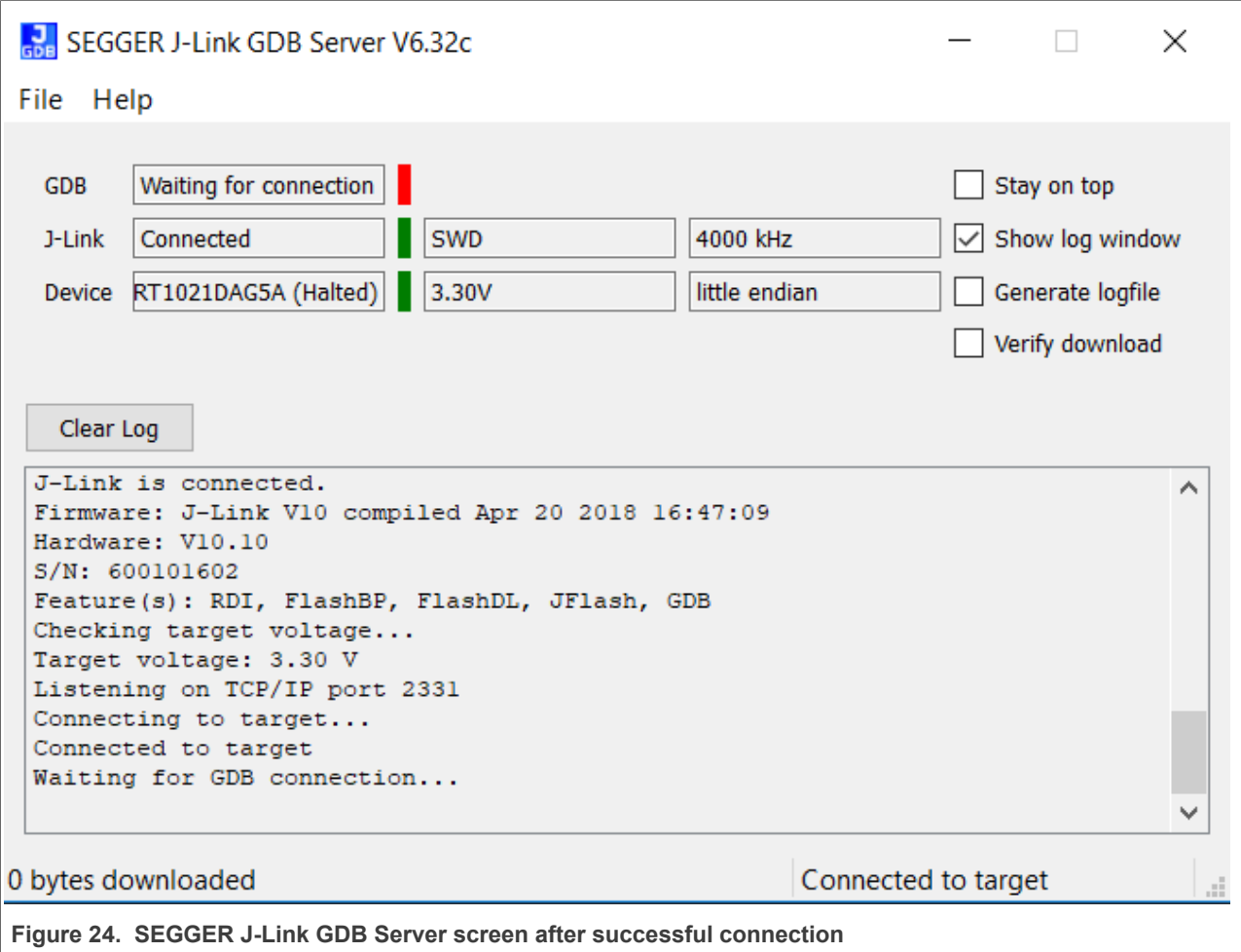


Figure 24. SEGGER J-Link GDB Server screen after successful connection

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**.

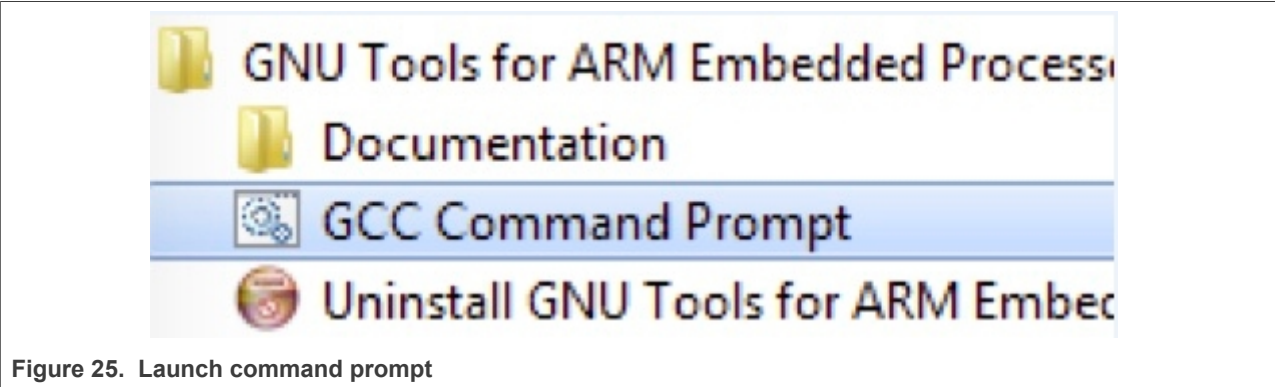


Figure 25. Launch command prompt

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/evkmimxrt1064/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`.

```

C:\Program Files (x86)\GNU Tools Arm Embedded\7 2018-q2-update>cd bin
C:\Program Files (x86)\GNU Tools Arm Embedded\7 2018-q2-update\bin>arm-none-eabi-gdb.exe C:\Users\nxf47776\Desktop\RT1064\RT1064_9_28\boards\evkmimxrt1064\demo_apps\hello_world\armgcc\debug\hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2018-q2-update) 8.1.0.20180315-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from C:\Users\nxf47776\Desktop\RT1064\RT1064_9_28\boards\evkmimxrt1064\demo_apps\hello_world\armgcc\debug\hello_world.elf...done.
(gdb)

```

Figure 26. Run `arm-none-eabi-gdb`

10. Run these commands:
- `target remote localhost:2331`
 - `monitor reset`
 - `monitor halt`
 - `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 MIMXRT1064-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 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.

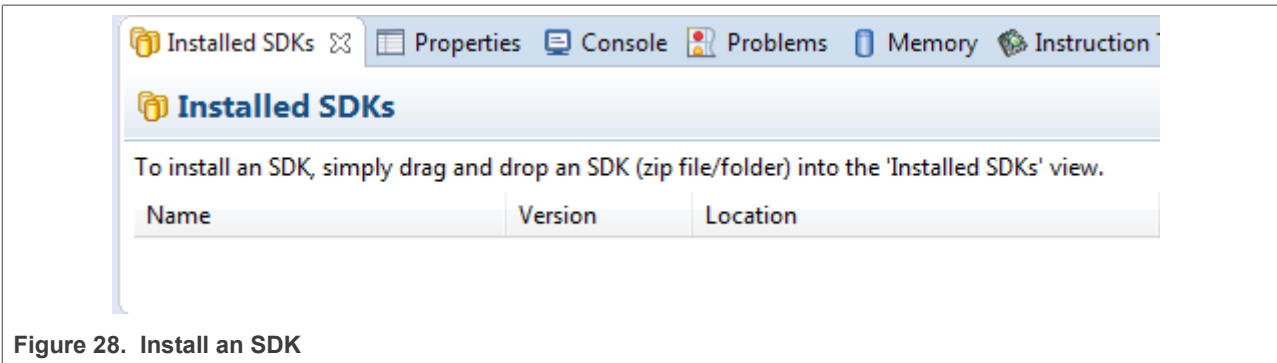


Figure 28. Install an SDK

2. On the **Quickstart Panel**, click **Import SDK example(s)...**, as shown in [Figure 29](#).

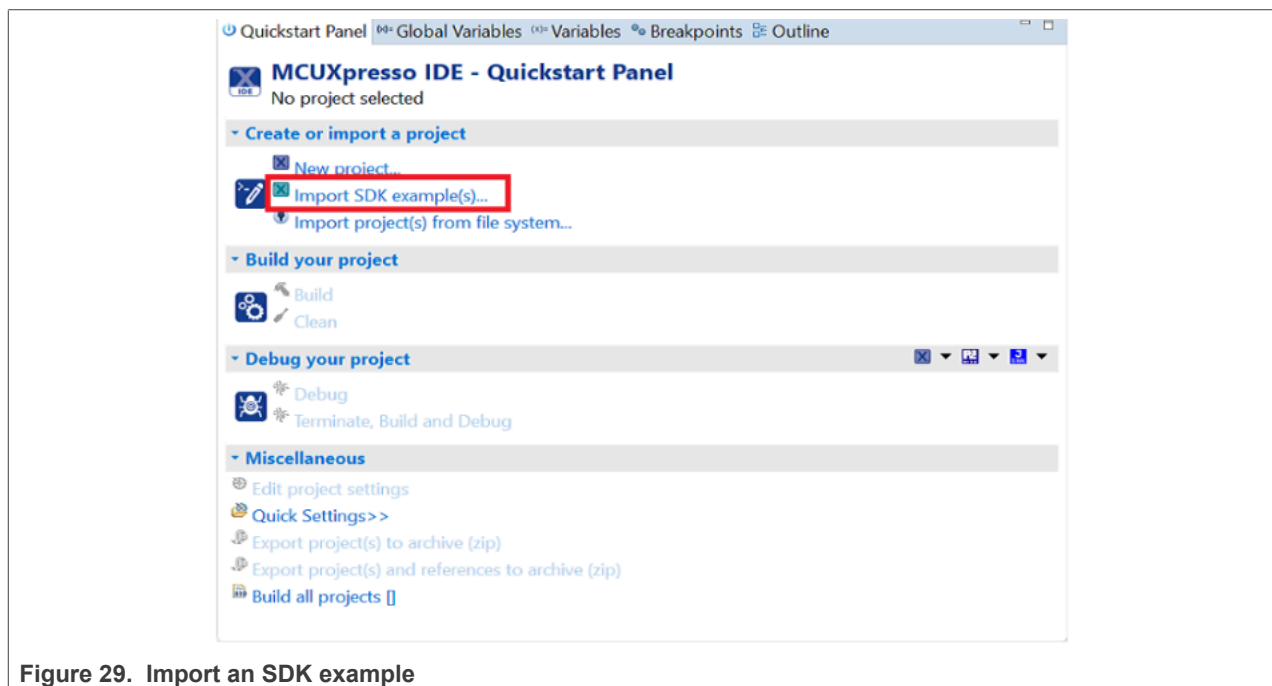


Figure 29. Import an SDK example

3. In the window that appears, expand the **MIMXRT1064** folder and select **MIMXRT1064xxxxx**. Then, select **evkmimxrt1064** and click **Next**, as shown in [Figure 30](#).

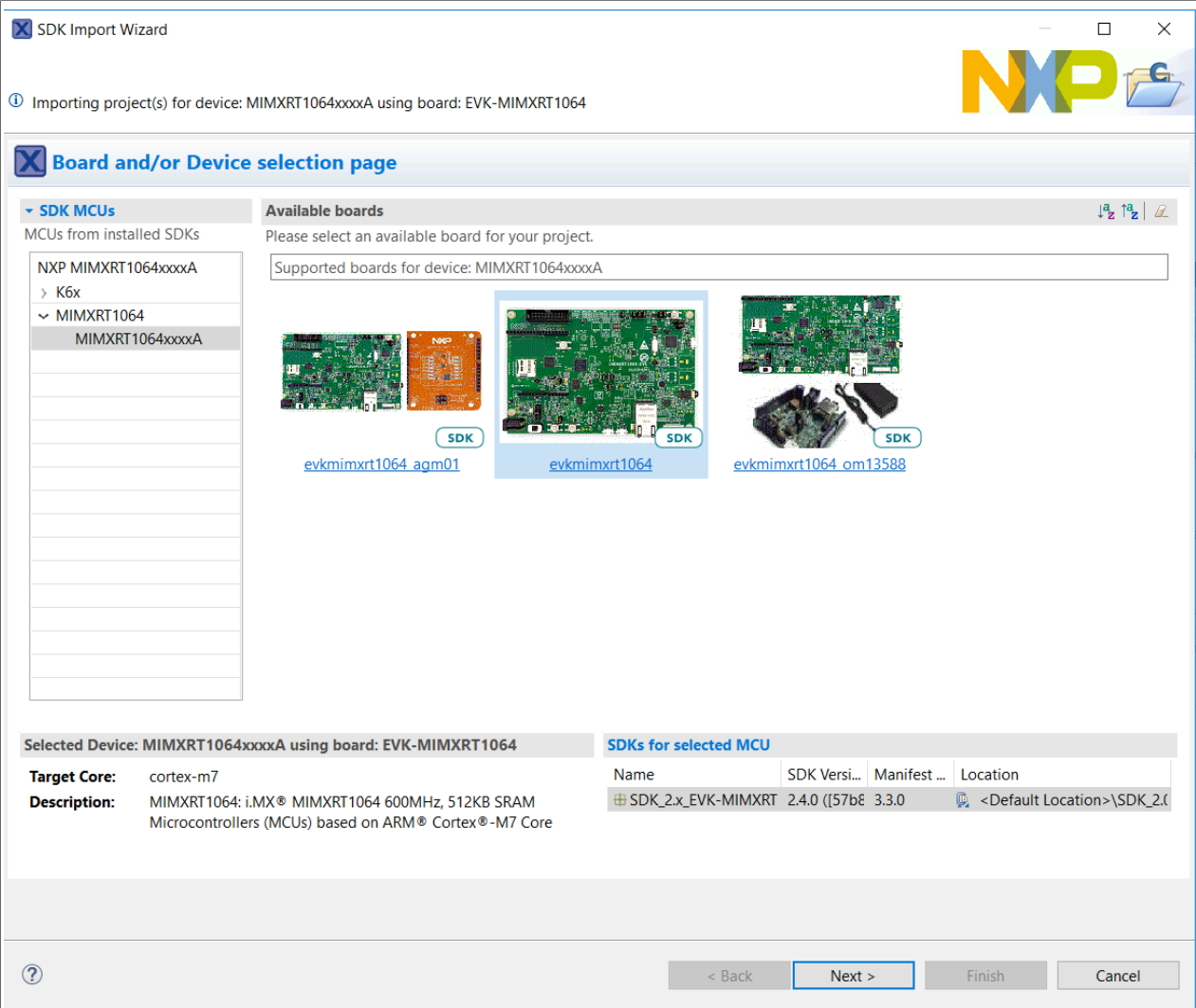


Figure 30. Selecting MIMXRT1064-EVK board

4. Expand the demo_apps folder, select hello_world, and then click **Next**, as shown in [Figure 31](#).

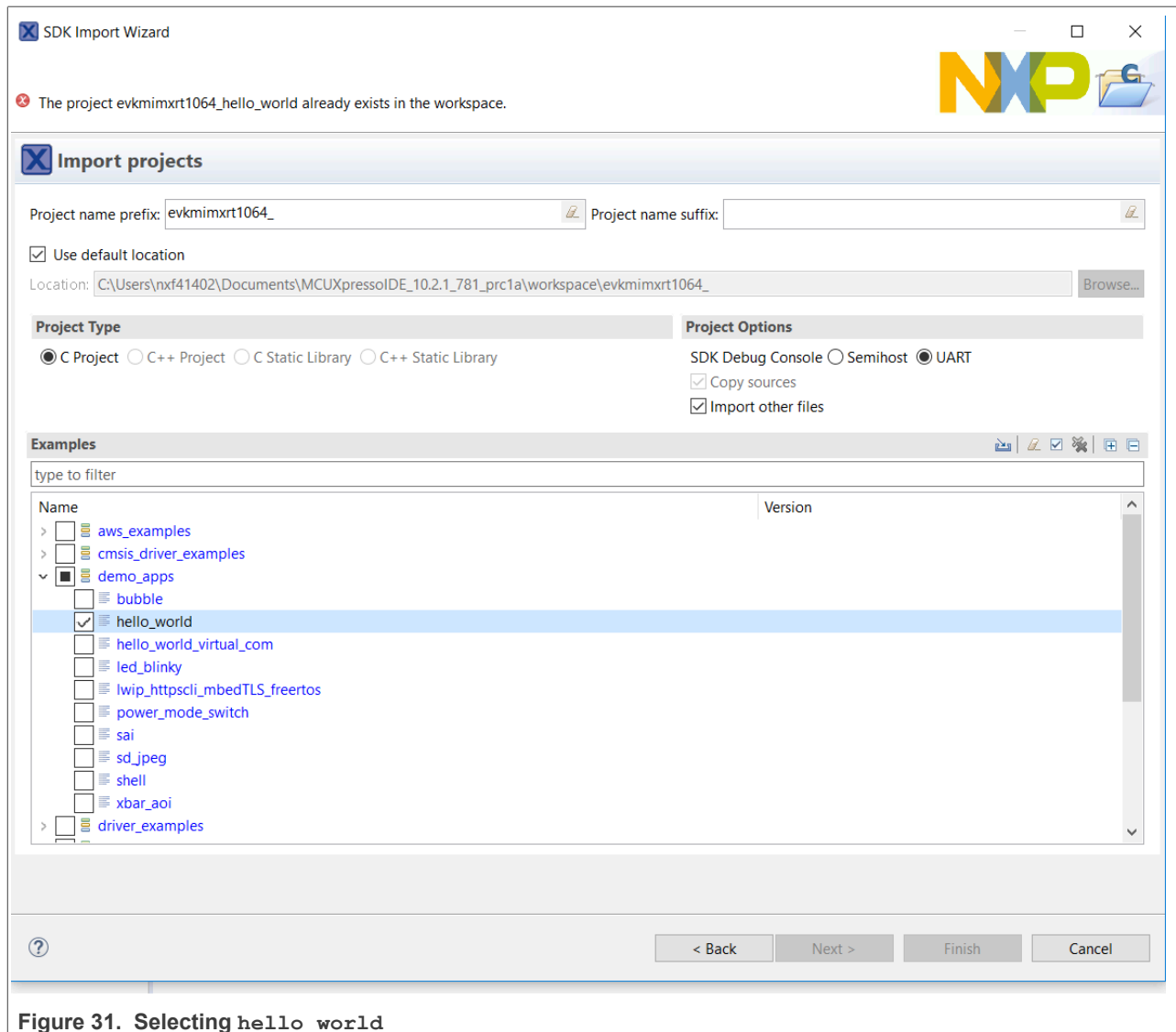
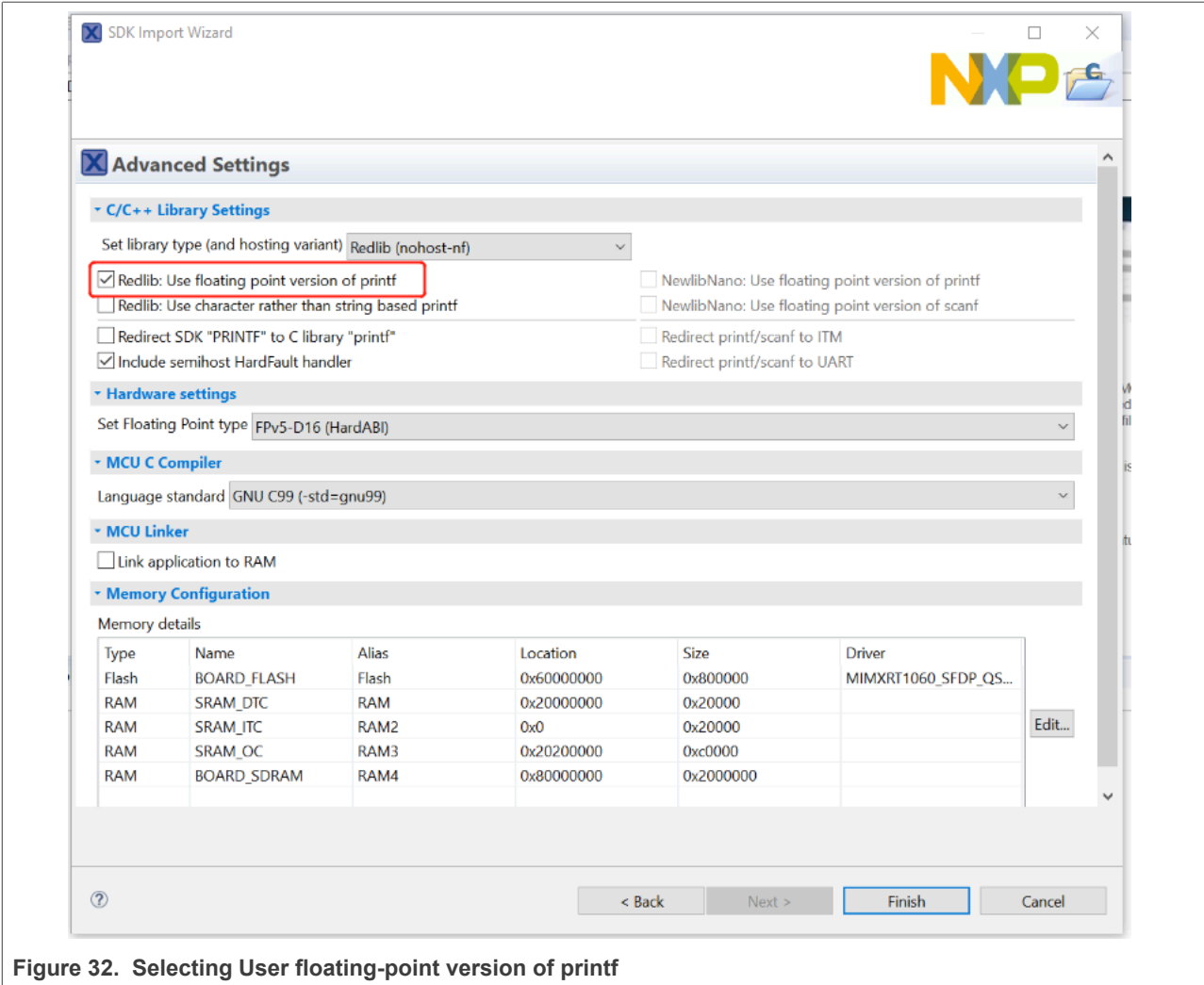


Figure 31. Selecting hello_world

5. Ensure the option **Redlib: Use floating-point version of printf** is selected if the cases print floating-point numbers on the terminal (for demo applications such as `dac32_adc12`, `dac_adc`, `dac_cadc`, `ecompass`, `sai`, `coremark`, `mbdrtls_benchmark`, `wolfssl_benchmark`, and for `mmcau_examples` such as `mmcau_api`). Otherwise, there is no need to select it. Click **Finish**.



Note: If you want to use semihost to print log, first select the **Semihost** button when importing projects, as shown in [Figure 33](#).

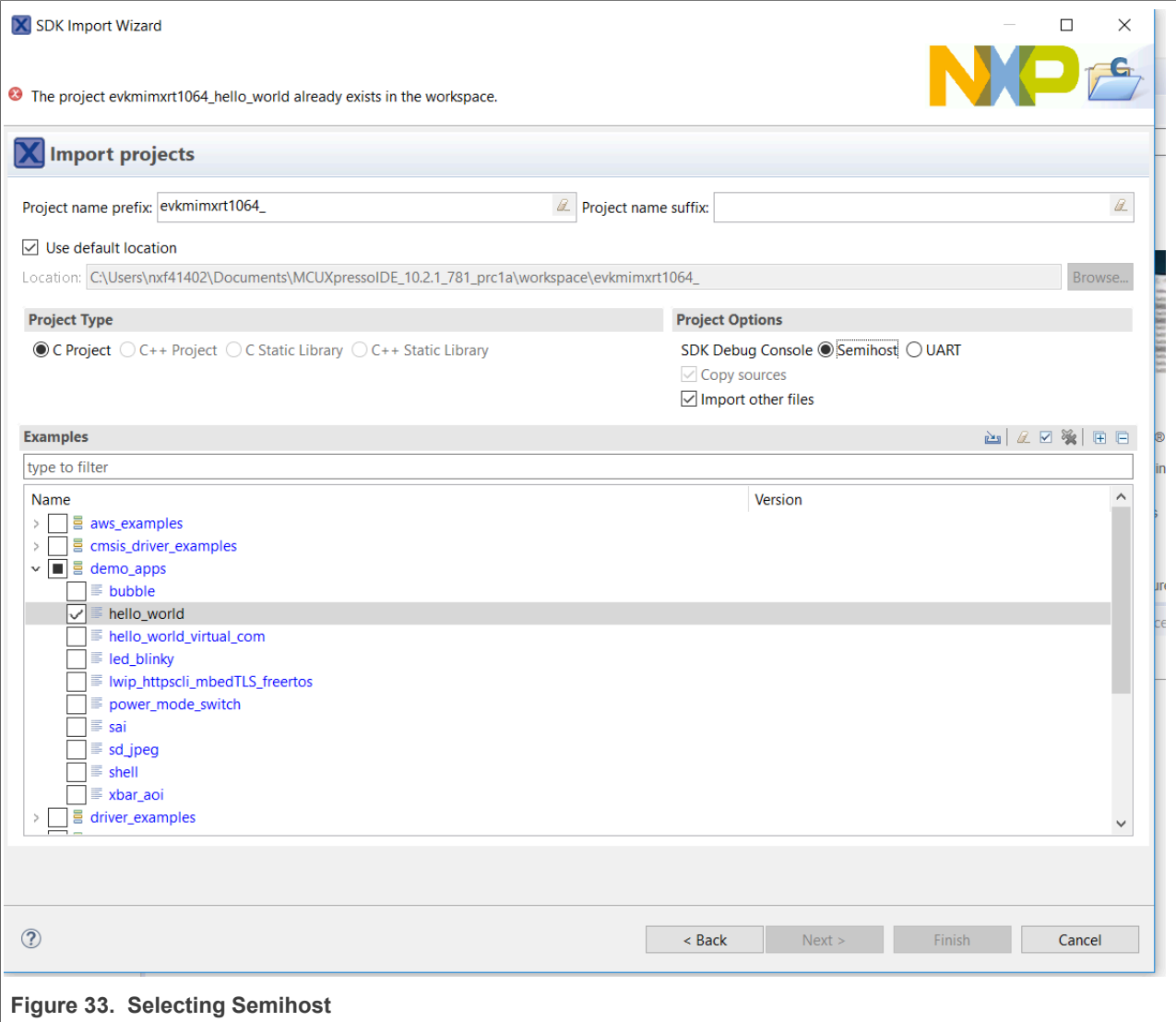


Figure 33. Selecting Semihost

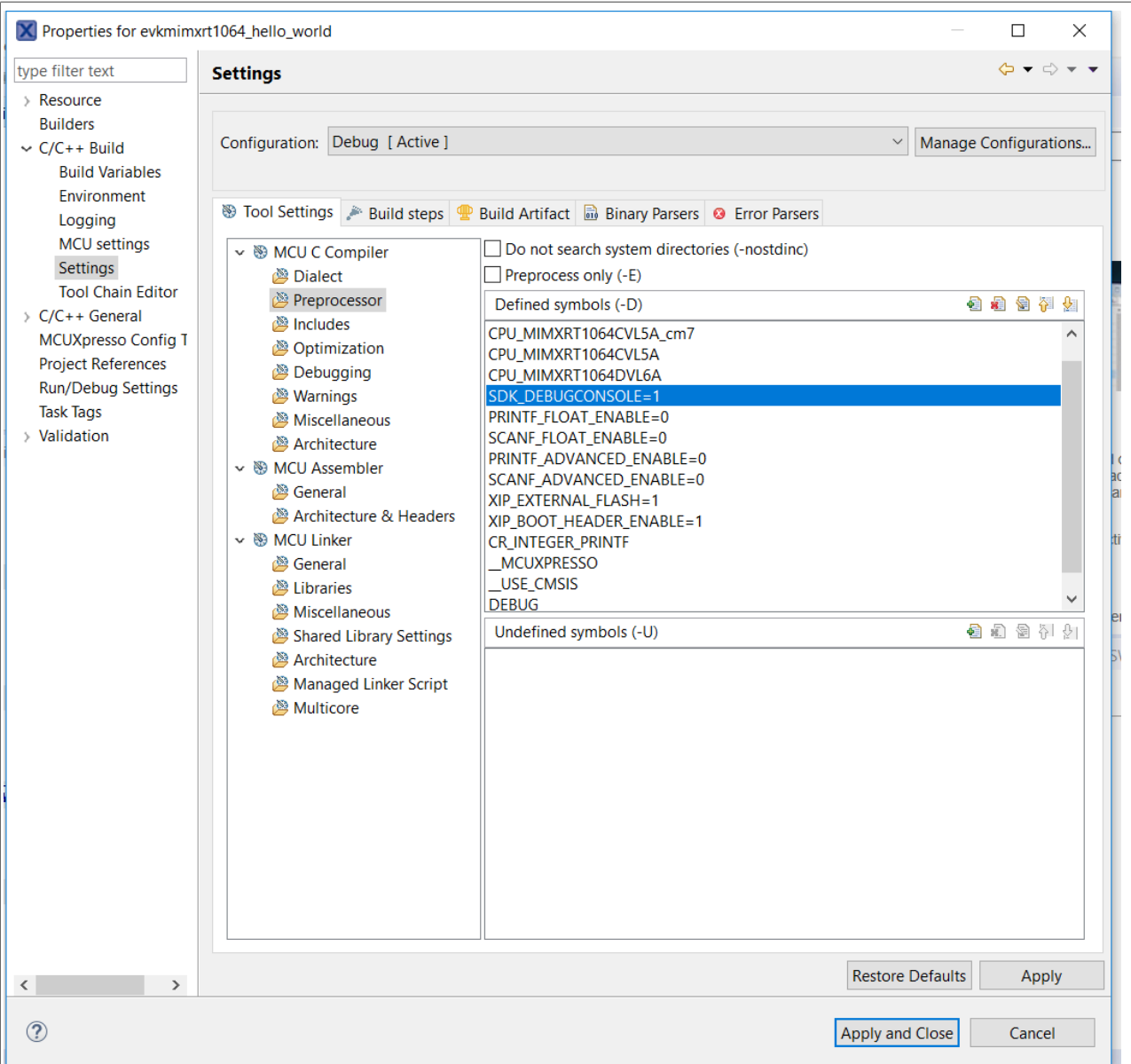


Figure 34. Setting "SDK_DEBUGCONSOLE"

6. On the **Quickstart** panel, click **build evkmimxrt1064_demo_apps_hello_world [Debug]**, as shown in [Figure 35](#).

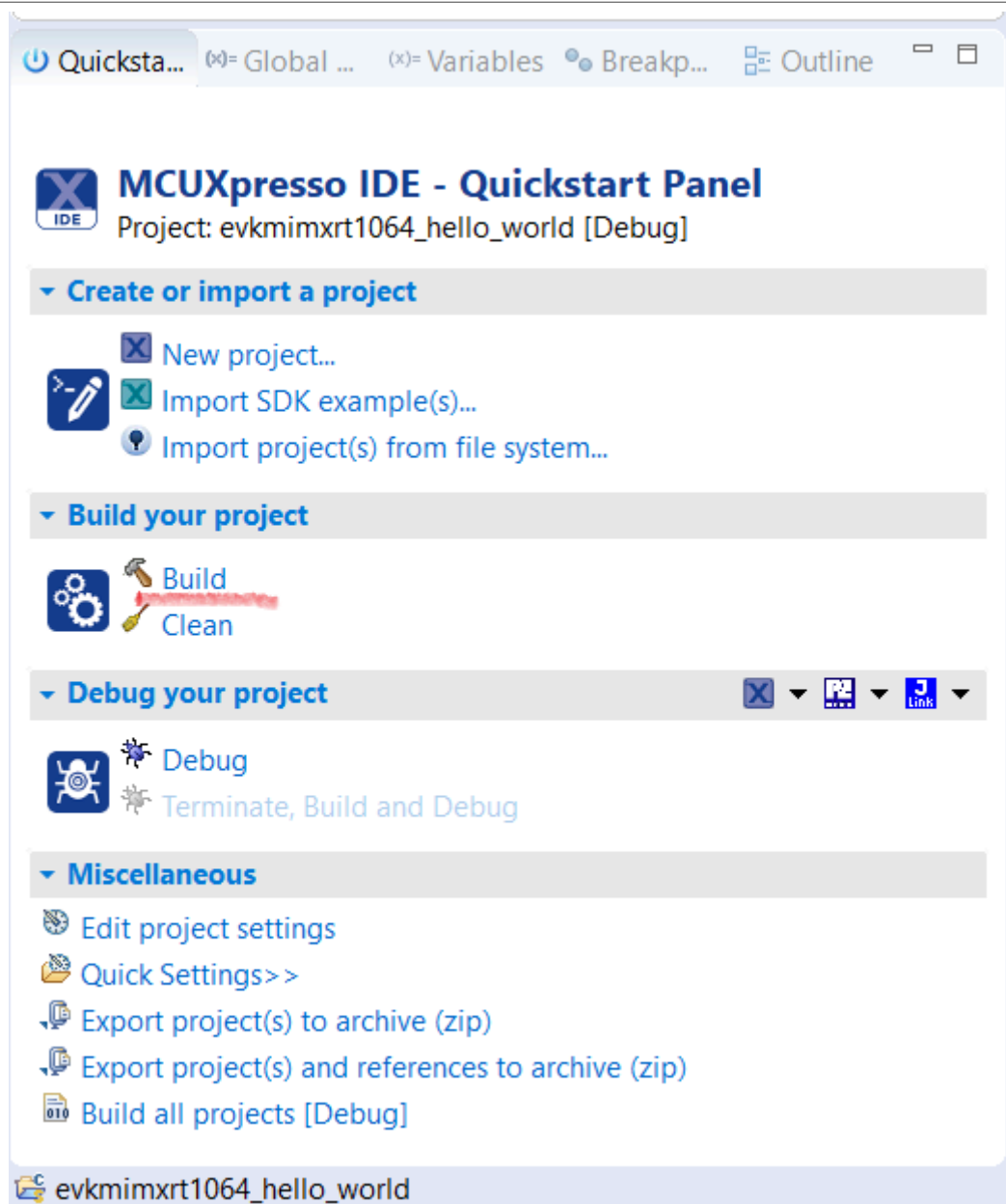


Figure 35. Building hello world case

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:

1. On the **Quickstart Panel**, click **Debug evkmimxrt1064_demo_apps_hello_world [Debug]**.

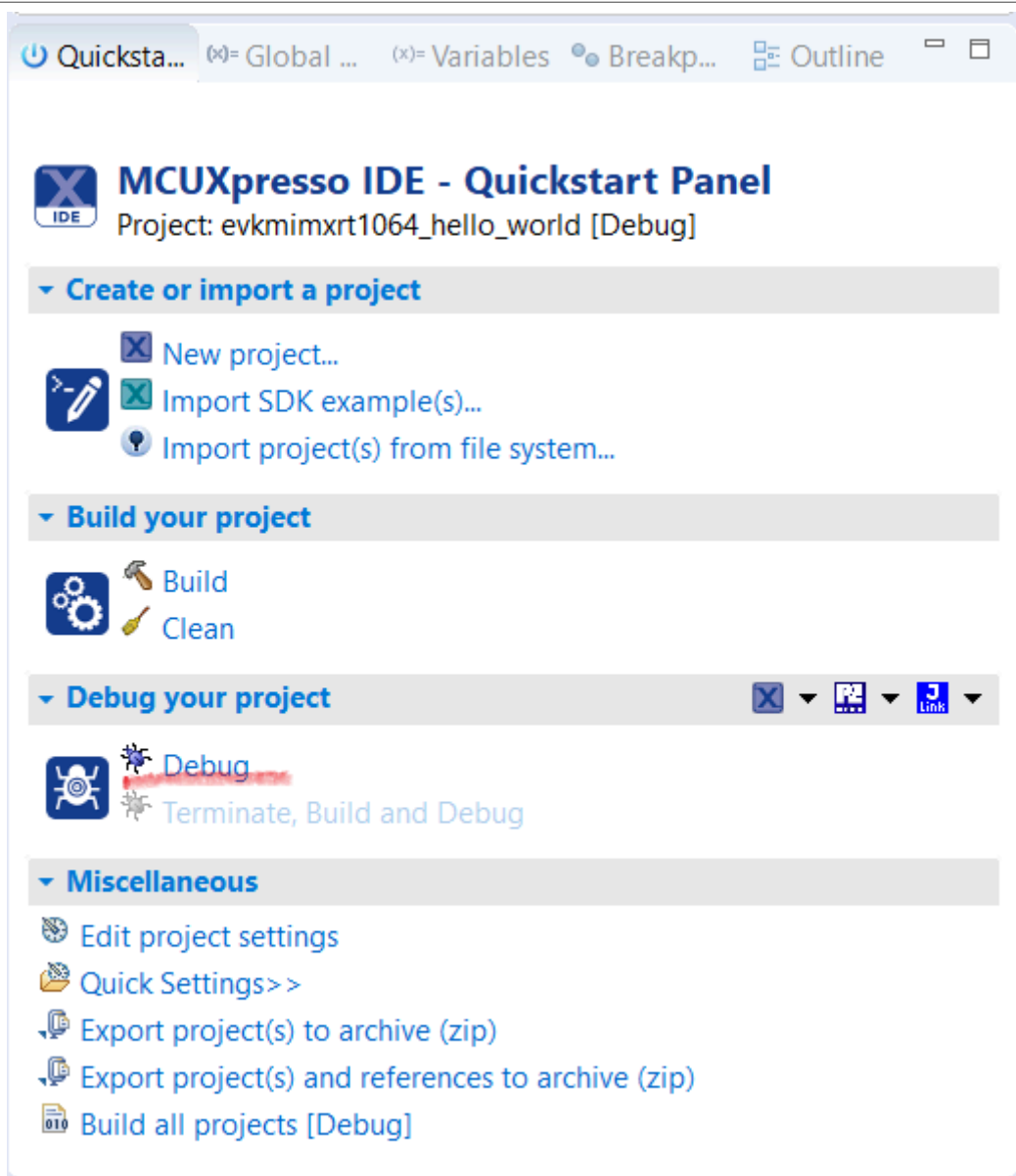


Figure 36. Debugging hello_world case

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.)

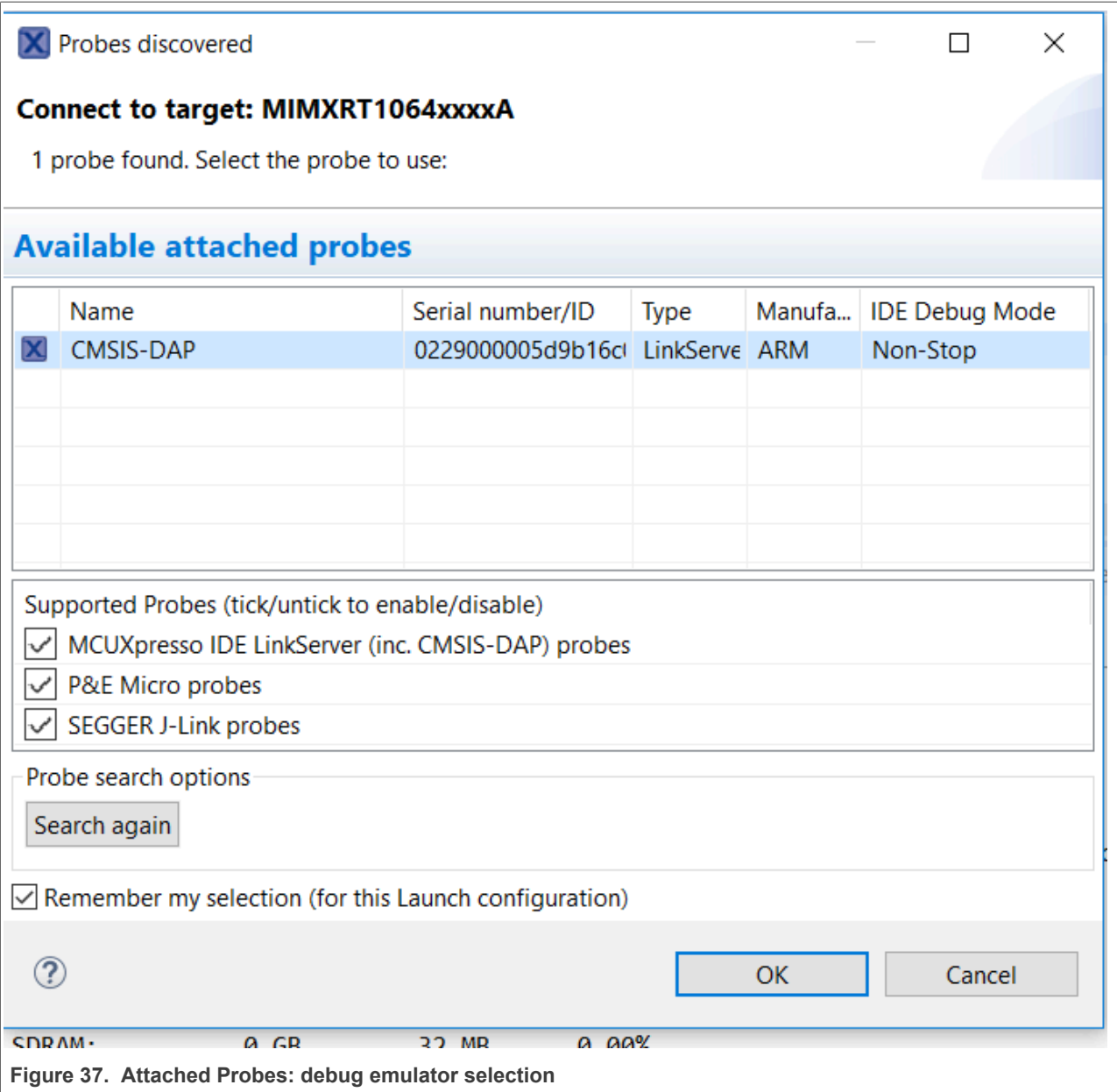
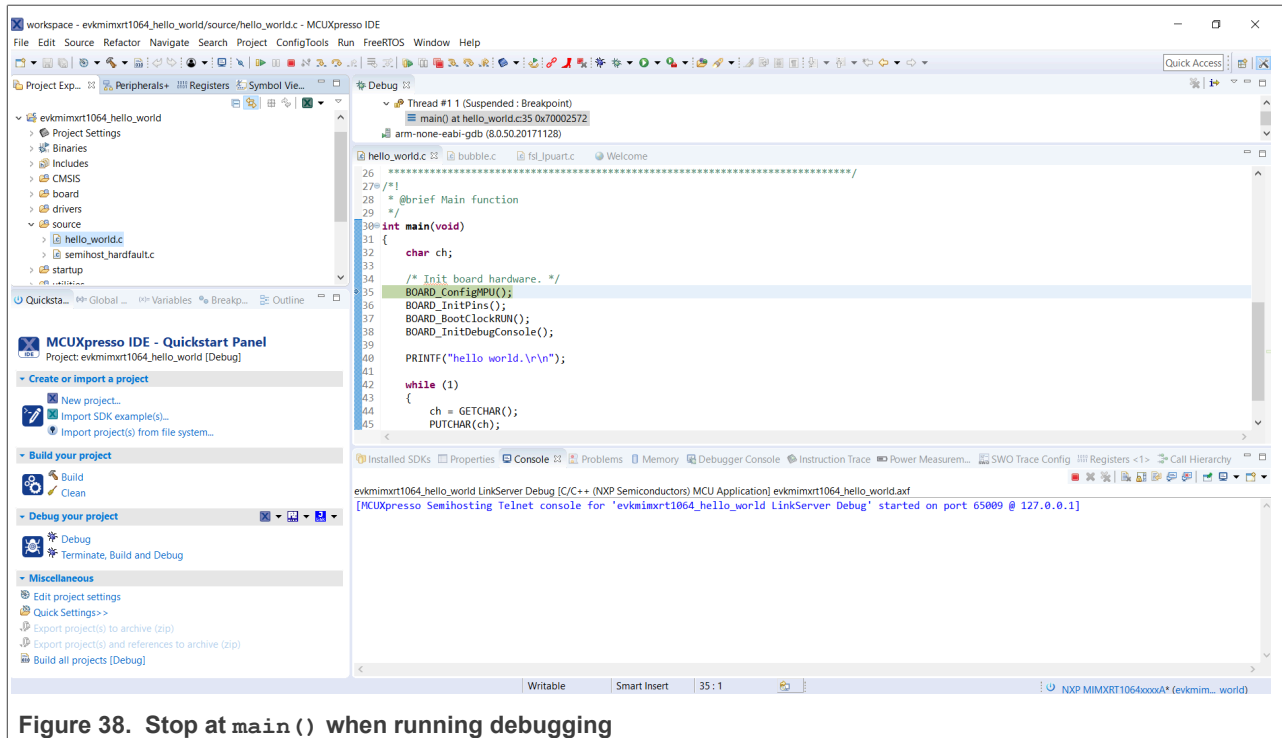


Figure 37. Attached Probes: debug emulator selection

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

Figure 38. Stop at `main()` when running debugging

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

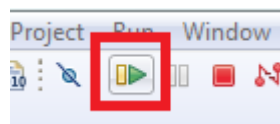
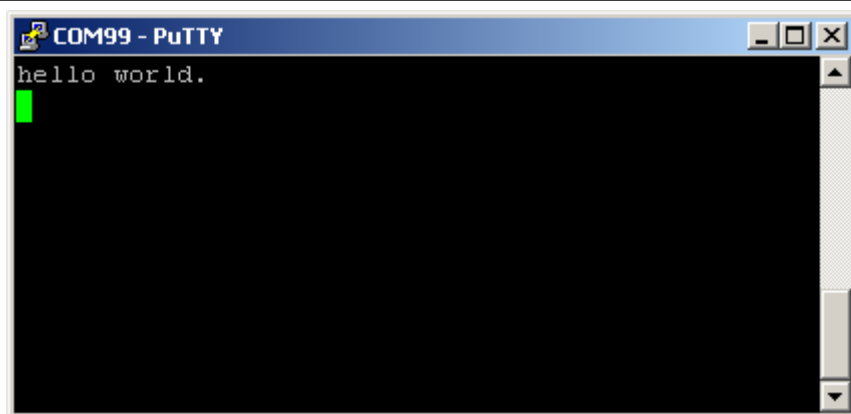


Figure 39. Resume button

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






Figure 40. 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 set up various on-chip peripherals prior to 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 doing 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 41](#).



Figure 41. 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.

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. **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 the **Start** menu and type **Device Manager** in the search bar.
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.

10 How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default `PIT_IRQHandler` define in `startup_DEVICE.s`, application code like `app.c` can be implement like:

```
c
void PIT_IRQHandler(void)
{
    // Your code
}
```


When application file is CPP file, like app.cpp, then `extern "C"` should be used to ensure the function prototype alignment.

```
cpp
extern "C" {
    void PIT_IRQHandler(void);
}
void PIT_IRQHandler(void)
{
    // Your code
}
```

11 How to add or remove boot header for XIP targets

The MCUXpresso SDK for i.MX 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:

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

Take `hello_world` as an example:

- IAR

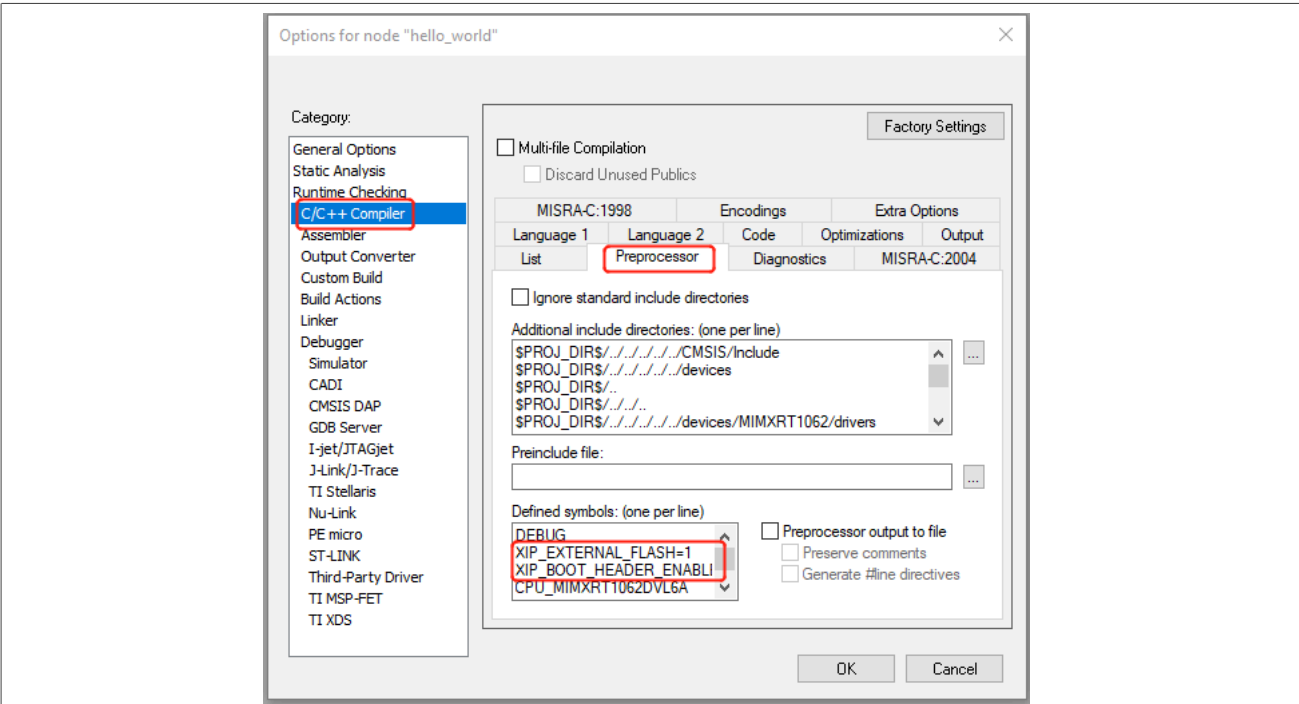


Figure 42. Options node IAR

- MDK

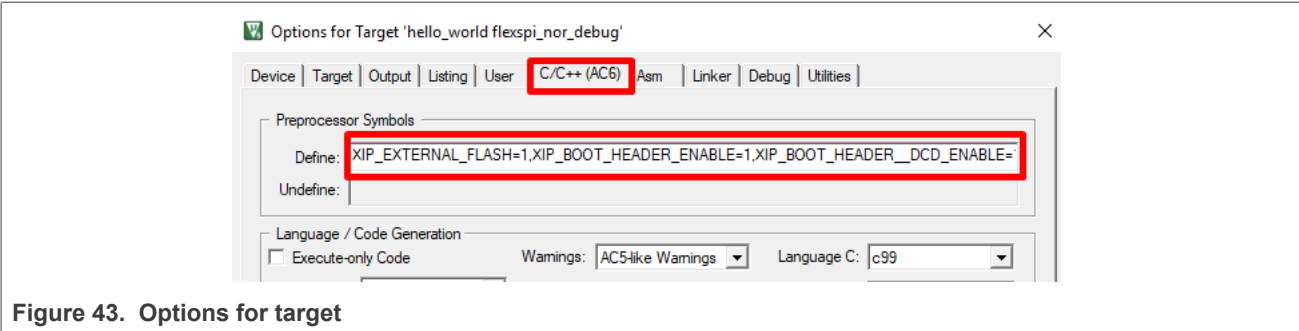


Figure 43. Options for target

- **ARMGCC**
Change the configuration in CMakeLists.txt.
- **MCUX**

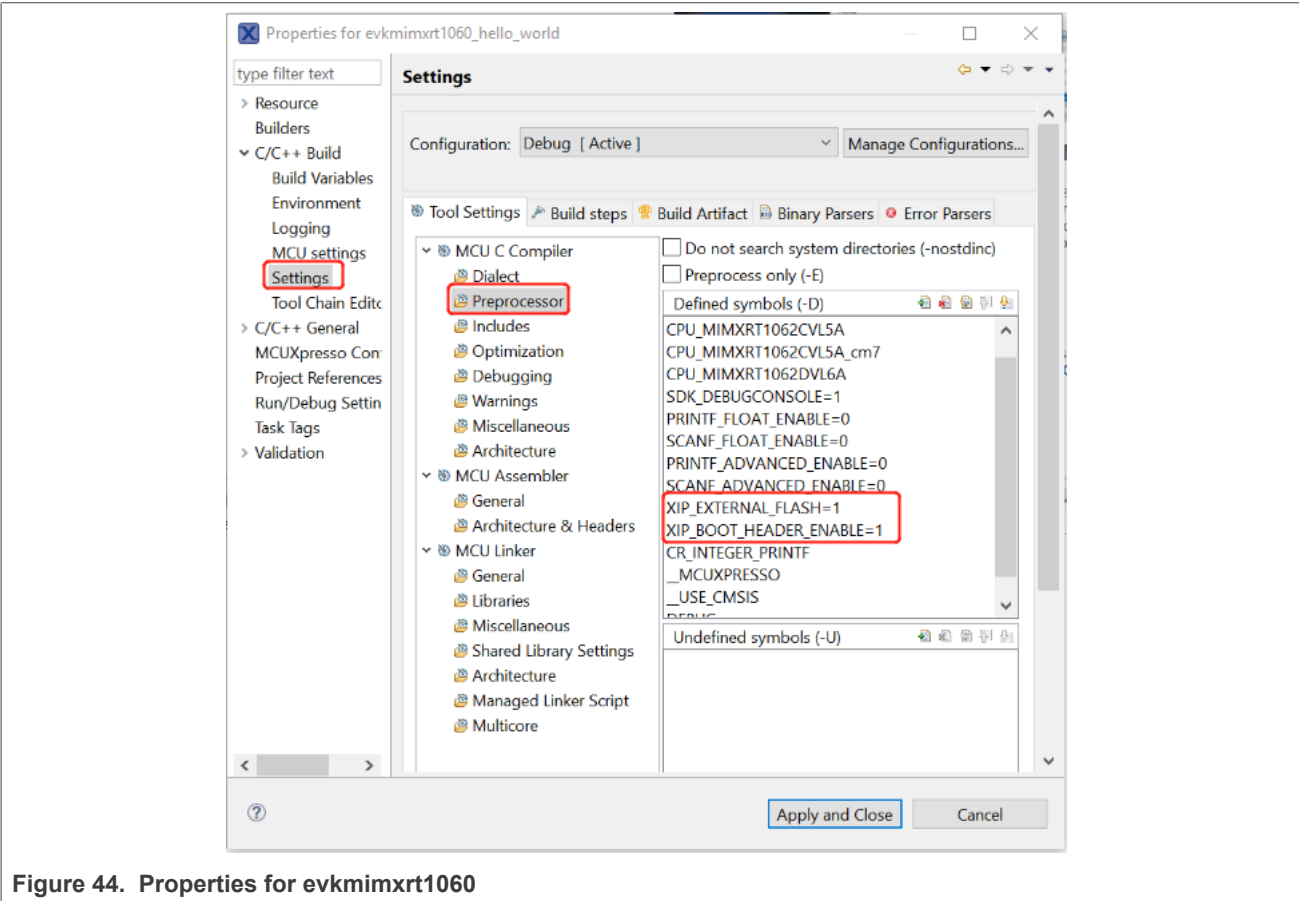


Figure 44. Properties for evkmimxrt1060

12 Revision history

This table summarizes the revisions to this document.

Table 2. Revision history

Revision number	Date	Substantive changes
2.13.0	22 December 2022	Updates for MCUXpresso SDK v2.13.0

Table 2. Revision history ...continued

Revision number	Date	Substantive changes
2.14.0	27 July 2023	Updates for MCUXpresso SDK v2.14.0
2.15.000	10 January 2024	Updated for MCUXpresso SDK v2.15.000

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.

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.

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.

Kinetis — is a trademark of NXP B.V.

Tower — is a trademark of NXP B.V.

Tables

Tab. 1.	MCUXpresso Config Tools	31	Tab. 2.	Revision history	34
---------	-------------------------------	----	---------	------------------------	----

Figures

Fig. 1.	MCUXpresso SDK layers	2	Fig. 24.	SEGGER J-Link GDB Server screen after successful connection	18
Fig. 2.	Application folder structure	3	Fig. 25.	Launch command prompt	18
Fig. 3.	Demo build target selection	5	Fig. 26.	Run arm-none-eabi-gdb	19
Fig. 4.	Build the demo application	5	Fig. 27.	Text display of the hello_world demo	20
Fig. 5.	Terminal (PuTTY) configuration	6	Fig. 28.	Install an SDK	20
Fig. 6.	Download and Debug button	6	Fig. 29.	Import an SDK example	21
Fig. 7.	Stop at main() when running debugging	7	Fig. 30.	Selecting MIMXRT1064-EVK board	22
Fig. 8.	Go button	7	Fig. 31.	Selecting hello_world	23
Fig. 9.	Text display of the hello_world demo	7	Fig. 32.	Selecting User floating-point version of printf	24
Fig. 10.	Build the demo	8	Fig. 33.	Selecting Semihost	25
Fig. 11.	Terminal (PuTTY) configurations	9	Fig. 34.	Setting "SDK_DEBUGCONSOLE"	26
Fig. 12.	Stop at main() when run debugging	10	Fig. 35.	Building hello world case	27
Fig. 13.	Run button	10	Fig. 36.	Debugging hello_world case	28
Fig. 14.	Text display of the hello_world demo	11	Fig. 37.	Attached Probes: debug emulator selection	29
Fig. 15.	Set up MinGW and MSYS	12	Fig. 38.	Stop at main() when running debugging	30
Fig. 16.	Complete MinGW and MSYS installation	12	Fig. 39.	Resume button	30
Fig. 17.	Add Path to systems environment	13	Fig. 40.	Text display of the hello_world demo	30
Fig. 18.	Convert path to short path	13	Fig. 41.	MCUXpresso IDE Quickstart Panel	32
Fig. 19.	Add ARMGCC_DIR system variable	14	Fig. 42.	Options node IAR	33
Fig. 20.	Install CMake	15	Fig. 43.	Options for target	34
Fig. 21.	Launch command prompt	15	Fig. 44.	Properties for evkmimxrt1060	34
Fig. 22.	hello_world demo build successful	15			
Fig. 23.	Terminal (PuTTY) configurations	17			

Contents

1	Overview	2
2	MCUXpresso SDK board support	
	package folders	2
2.1	Example application structure	3
2.2	Locating example application source files	3
3	Run a demo application using IAR	4
3.1	Build an example application	4
3.2	Run an example application	5
4	Run a demo using Keil MDK/μVision	7
4.1	Install CMSIS device pack	7
4.2	Build an example application	8
4.3	Run an example application	8
5	Run a demo using Arm GCC	11
5.1	Set up toolchain	11
5.1.1	Install GCC Arm Embedded tool chain	11
5.1.2	Install MinGW (only required on Windows OS)	11
5.1.3	Add a new system environment variable for ARMGCC_DIR	13
5.1.4	Install CMake	14
5.2	Build an example application	15
5.3	Run an example application	16
6	Run a demo using MCUXpresso IDE	20
6.1	Select the workspace location	20
6.2	Build an example application	20
6.3	Run an example application	27
7	MCUXpresso Config Tools	31
8	MCUXpresso IDE New Project Wizard	31
9	How to determine COM port	32
10	How to define IRQ handler in CPP files	32
11	How to add or remove boot header for XIP targets	33
12	Revision history	34
	Legal information	36

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.