

# Getting Started with MCUXpresso SDK for MIMXRT1021

## 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 the MCUXpresso SDK Release Notes Supporting EVK-MIMXRT1020 (document MCUXSDKMIMXRT102XRN).

For the latest version of this and other MCUXpresso SDK documents, see the MCUXpresso SDK homepage [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#).

### Contents

1	Overview.....	1
2	MCUXpresso SDK board support folders.....	2
3	Run a demo application using IAR.....	4
4	Run a demo using Keil® MDK/µVision.....	7
5	Run a demo using Arm® GCC.....	11
6	Run a demo using MCUXpresso IDE....	19
7	MCUXpresso Config Tools.....	28
8	MCUXpresso IDE New Project Wizard.....	29
9	Appendix A - How to determine COM port.....	29
10	Appendix B - How to add or remove boot header for XIP targets.....	32





**Figure 1. MCUXpresso SDK layers**

## 2 MCUXpresso SDK board support folders

MCUXpresso SDK board support provides example applications for NXP development and evaluation boards for Arm<sup>®</sup> Cortex<sup>®</sup>-M cores, including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside of 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 they contain. These include (but are not limited to):

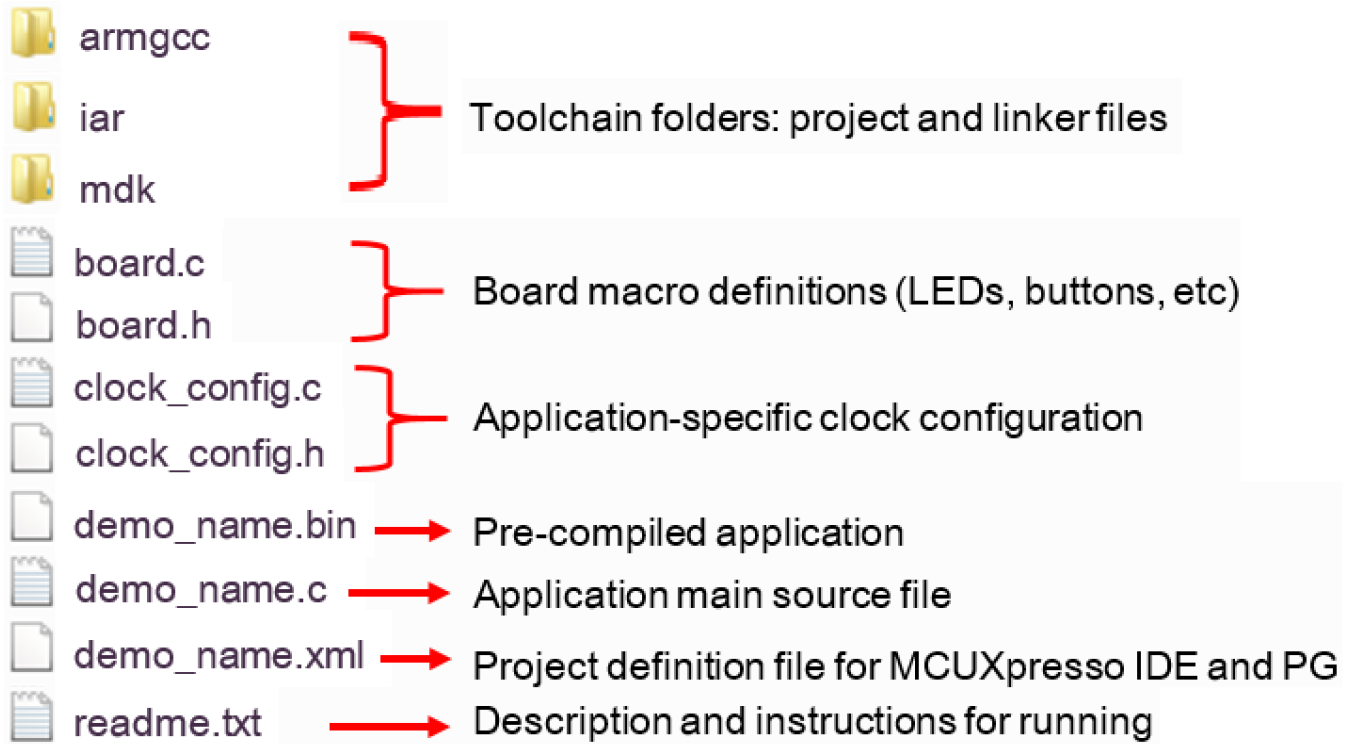
- **demo\_apps**: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- **driver\_examples**: Simple applications intended to concisely illustrate how to use 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 are used (for example, SPI conversion using DMA).
- **rtos\_examples**: Basic FreeRTOS<sup>™</sup> OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers

### 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 the *MCUXpresso SDK API Reference Manual* document (MCUXSDKAPIRM).

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 (except MCUXpresso IDE), 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 things.
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU.
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code. Vector table definitions are here.
- `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. Again, the core files of each of these are shared, so modifying them could have potential impacts on other projects that depend on them.

## 3 Run a demo application using IAR

### 3.1 Build an example application

The following steps guide you through opening the `hello_world` example application. These steps may change slightly for other example applications as some of these applications may have additional layers of folders in their path.

1. If not already done, 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-MIMXRT1020 hardware platform as an example, the `hello_world` workspace is located in

`<install_dir>/boards/evkmimxrt1020/demo_apps/hello_world/iar/hello_world.eww`

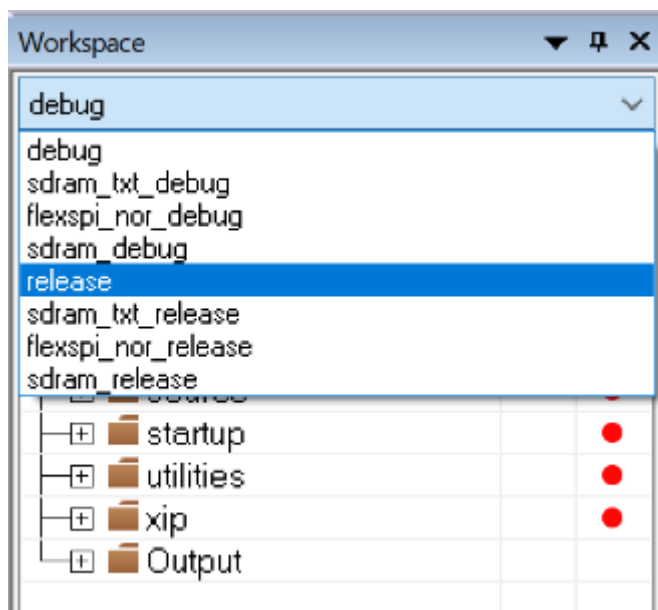
2. Select the desired build target from the drop-down.

There are eight 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.
- **sdram\_debug** - Project configuration is same as 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 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 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 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 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 release target. The linker file is flexspi\_nor linker, where text is put in flash and data put in TCM.

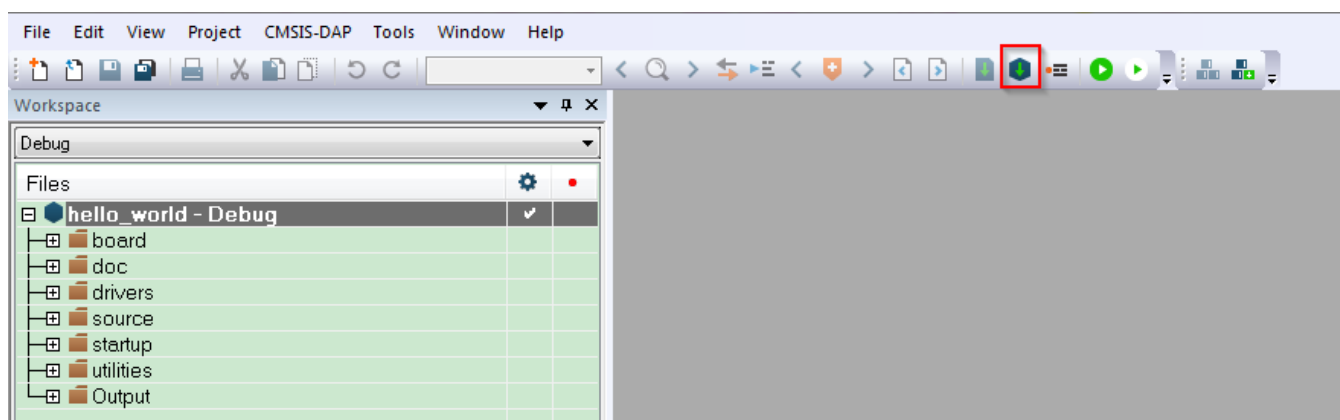
For some examples need large data memory, only `sdram_debug` and `sdram_release` targets are supported.

For this example, select the “`hello_world – Debug`” target.



**Figure 3. Demo build target selection**

3. To build the demo application, click the “Make” button, highlighted in red below.



**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:

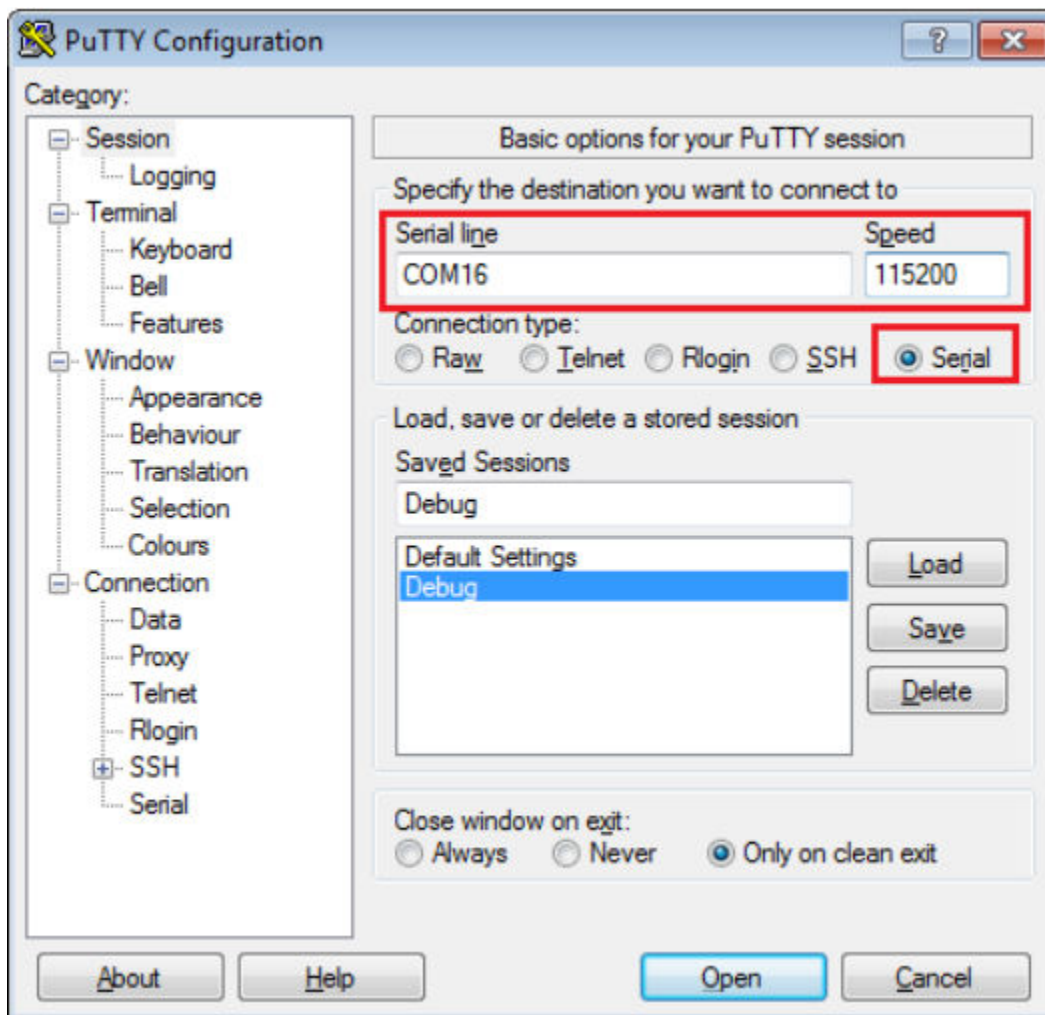
### NOTE

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

1. This board supports the CMSIS-DAP/mbed/DAPLink debug probe by default. Visit [os.mbed.com/handbook/Windows-serialconfiguration](https://os.mbed.com/handbook/Windows-serialconfiguration) 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.

## Run a demo application using IAR

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 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 board.h file)
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit



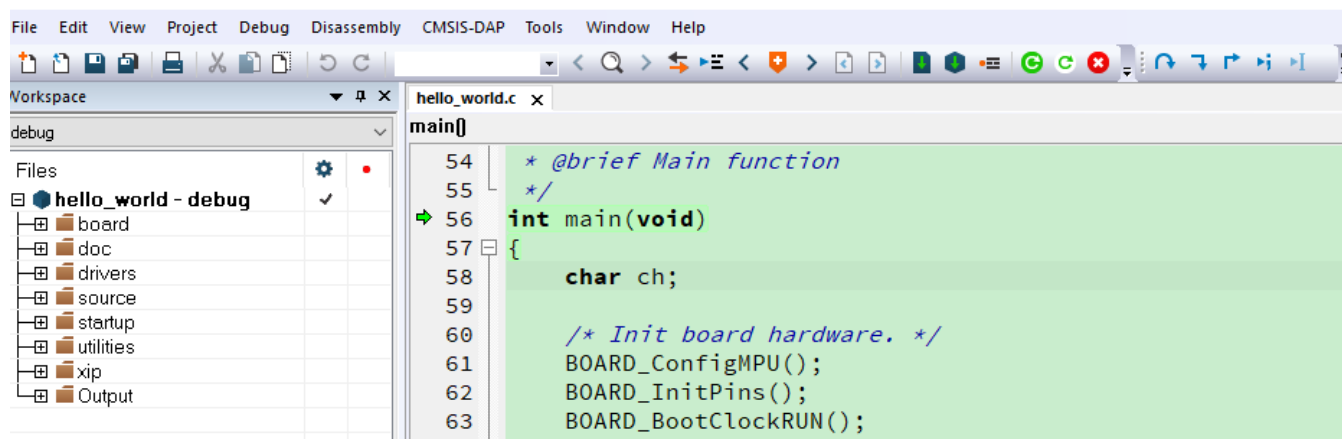
**Figure 5. Terminal (PuTTY) configuration**

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



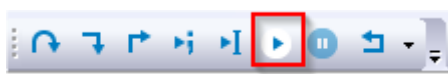
**Figure 6. Download and Debug button**

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



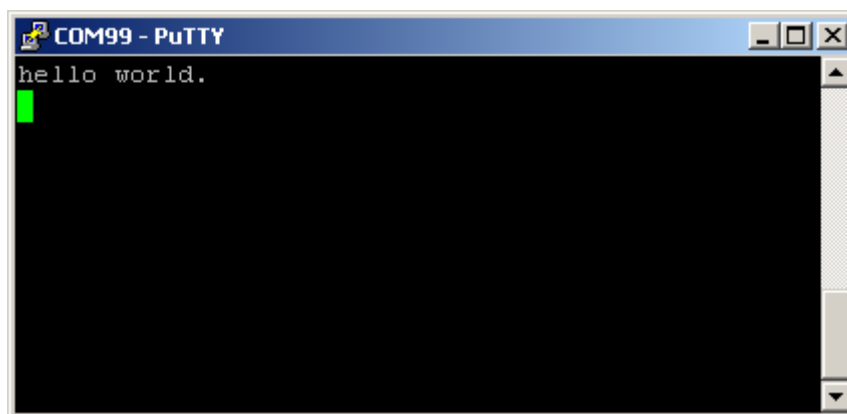
**Figure 7. Stop at main() when running debugging**

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 MIMXRT1020 and MIMXRT1021 packs.
2. After downloading the DFP, double click to install it.

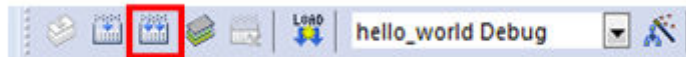
## 4.2 Build an example application

- Open the desired example application workspace in: `<install_dir>/boards/<board_name>/<example_type>/<application_name>/mdk`

The workspace file is named `<demo_name>.uvmpw`, so for this specific example, the actual path is:

`<install_dir>/boards/evkmimxrt1020/demo_apps/hello_world/mdk/hello_world.uvmpw`

- To build the demo project, select the "Rebuild" button, highlighted in red.



**Figure 10. Build the demo**

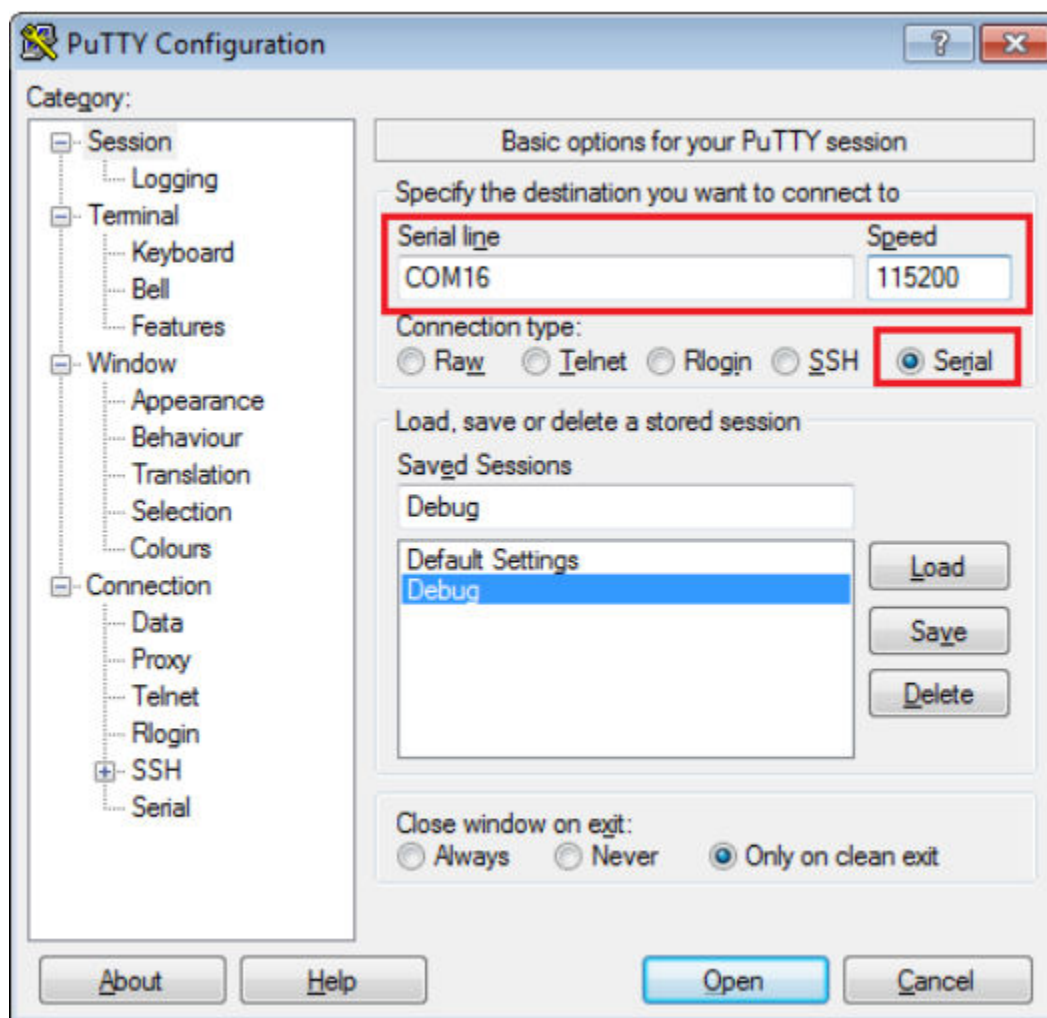
- The build completes without errors.

## 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-serialconfiguration](https://os.mbed.com/handbook/Windows-serialconfiguration) 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 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 board.h file)
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit

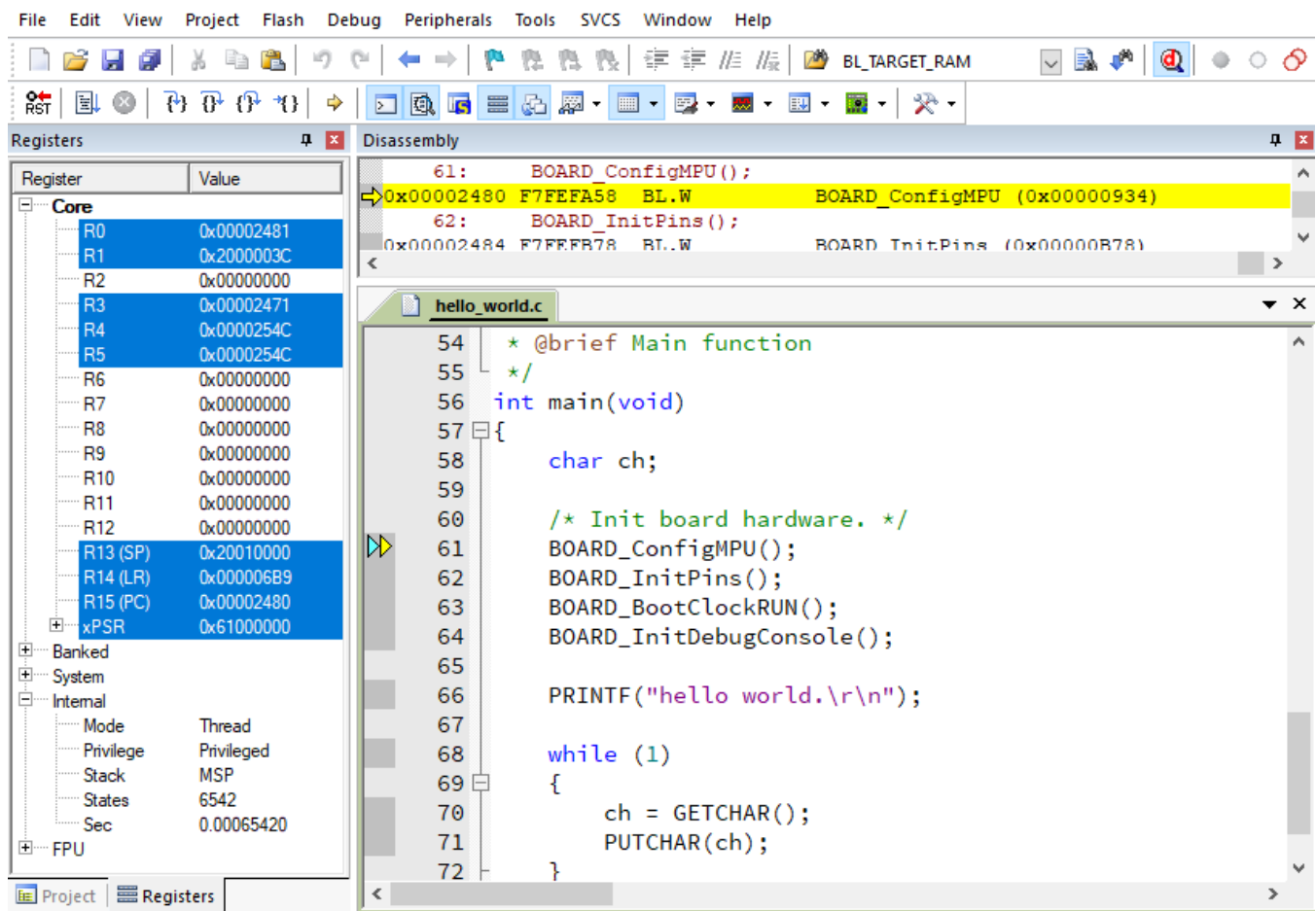




**Figure 11. Terminal (PuTTY) configurations**

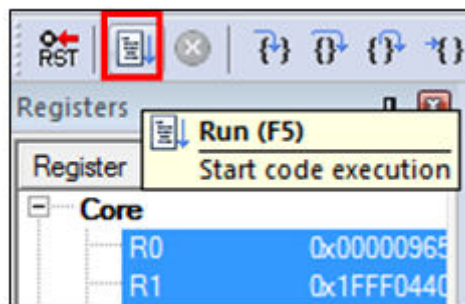
4. To debug the application, click the "load (F8)" button if the flexspi\_nor target is used. Then, click the "Start/Stop Debug Session" button, highlighted in red. If using J-Link as the debugger, "SW" should be selected in project option --> Debug --> Settings --> Debug -->Port.

## Run a demo using Keil® MDK/μVision



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

5. Run the code by clicking the “Run” button to start the application.



**Figure 13. Go button**

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.

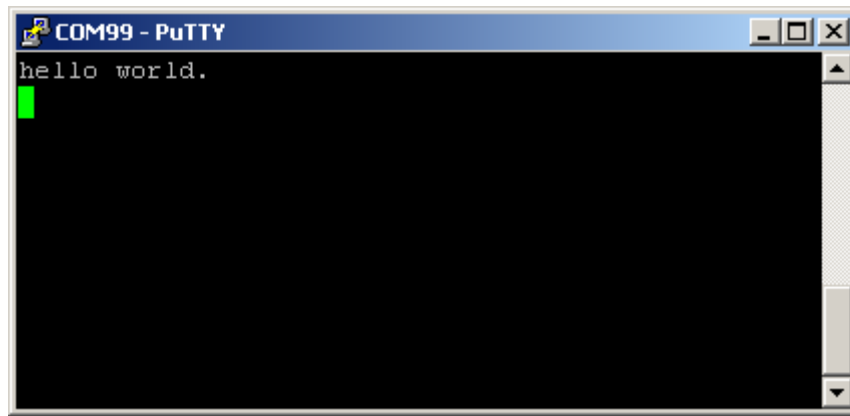


Figure 14. Text display of the hello\_world demo

## 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 as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

### NOTE

ARMGCC version 5.2.2015q4 is used as an example in this document, the latest GCC version for this package is as described in the *MCUXpresso SDK Release Notes* (document MCUXSDKRN).

### 5.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run a 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](http://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 the *MCUXpresso SDK Release Notes Supporting EVK-MIMXRT1020* (document MCUXSDKMIMXRT102XRN).

#### 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 SDK does not utilize 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/](http://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.

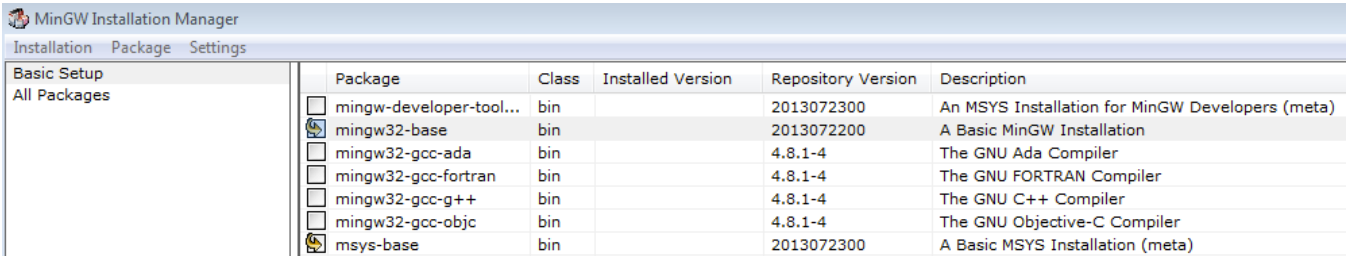


Figure 15. Setup MinGW and MSYS

- 4. Click “Apply Changes” in the “Installation” menu and follow the remaining instructions to complete the installation.

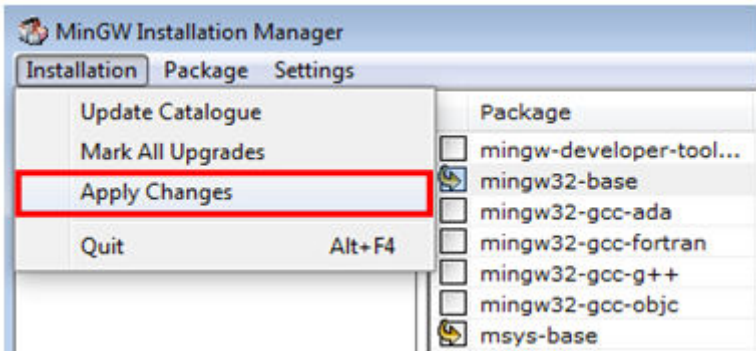


Figure 16. Complete MinGW and MSYS installation

- 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 does not work.

NOTE

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

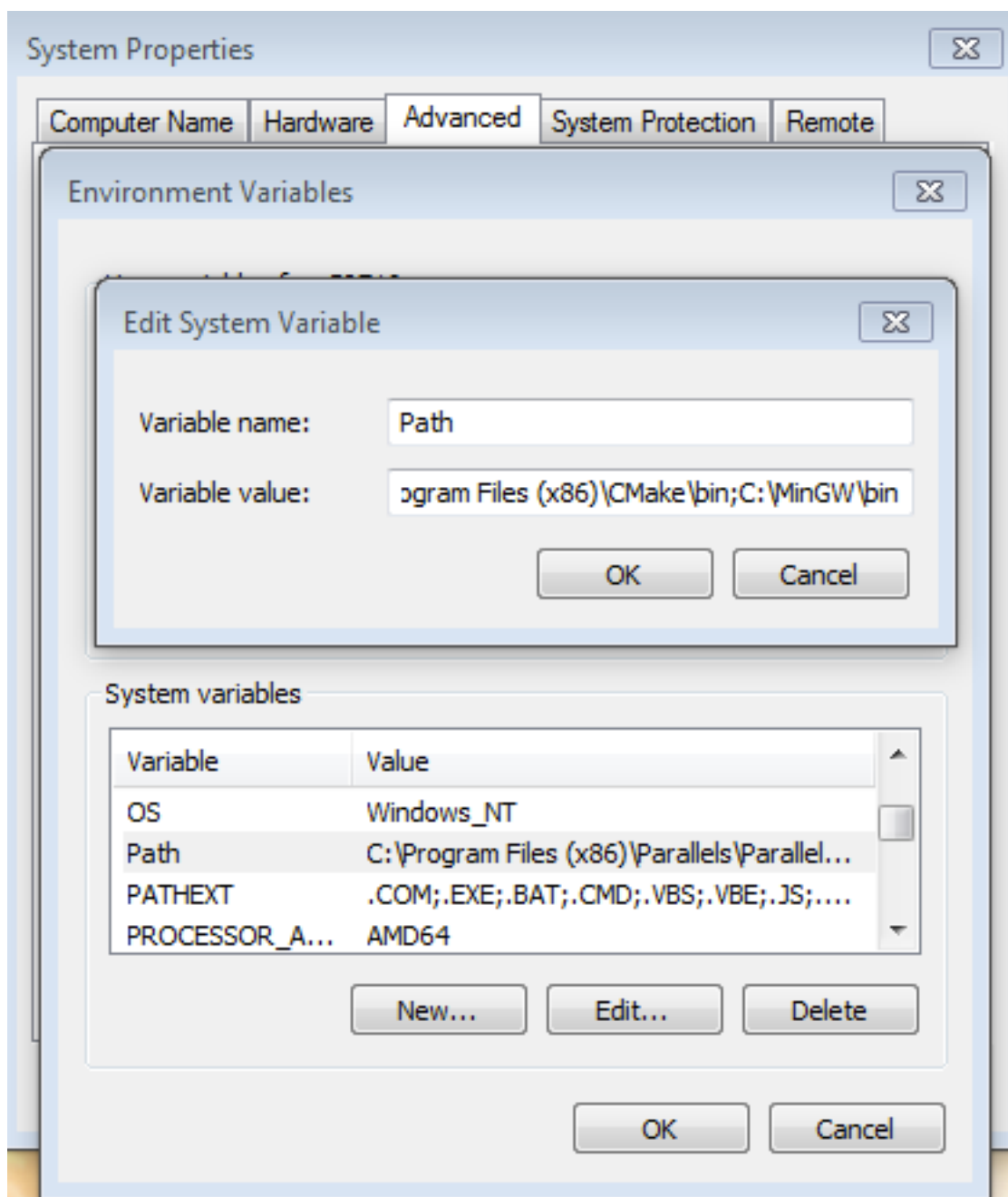
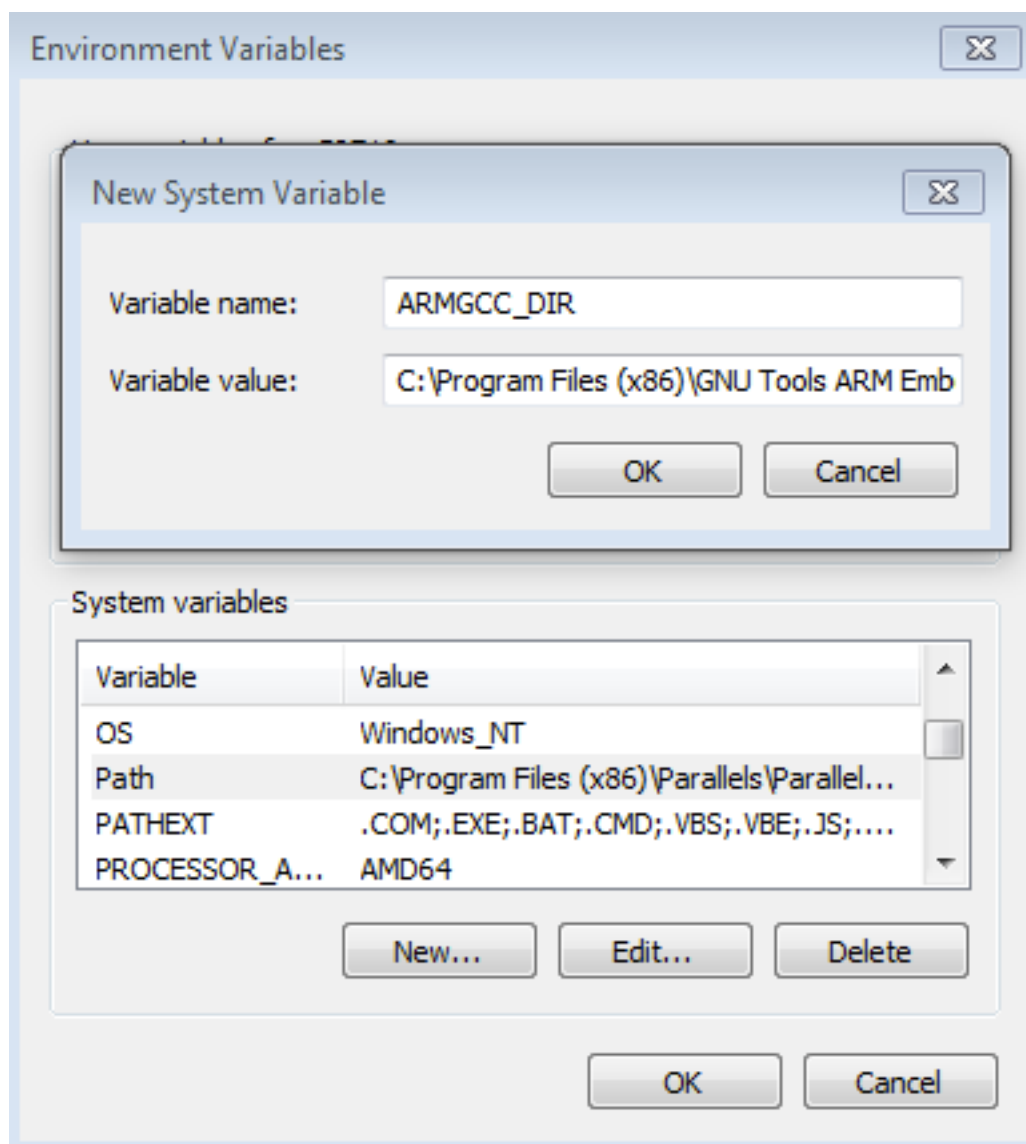


Figure 17. Add Path to systems environment

### 5.1.3 Add a new system environment variable for ARMGCC\_DIR

Create a new *system* environment variable and name it ARMGCC\_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is:

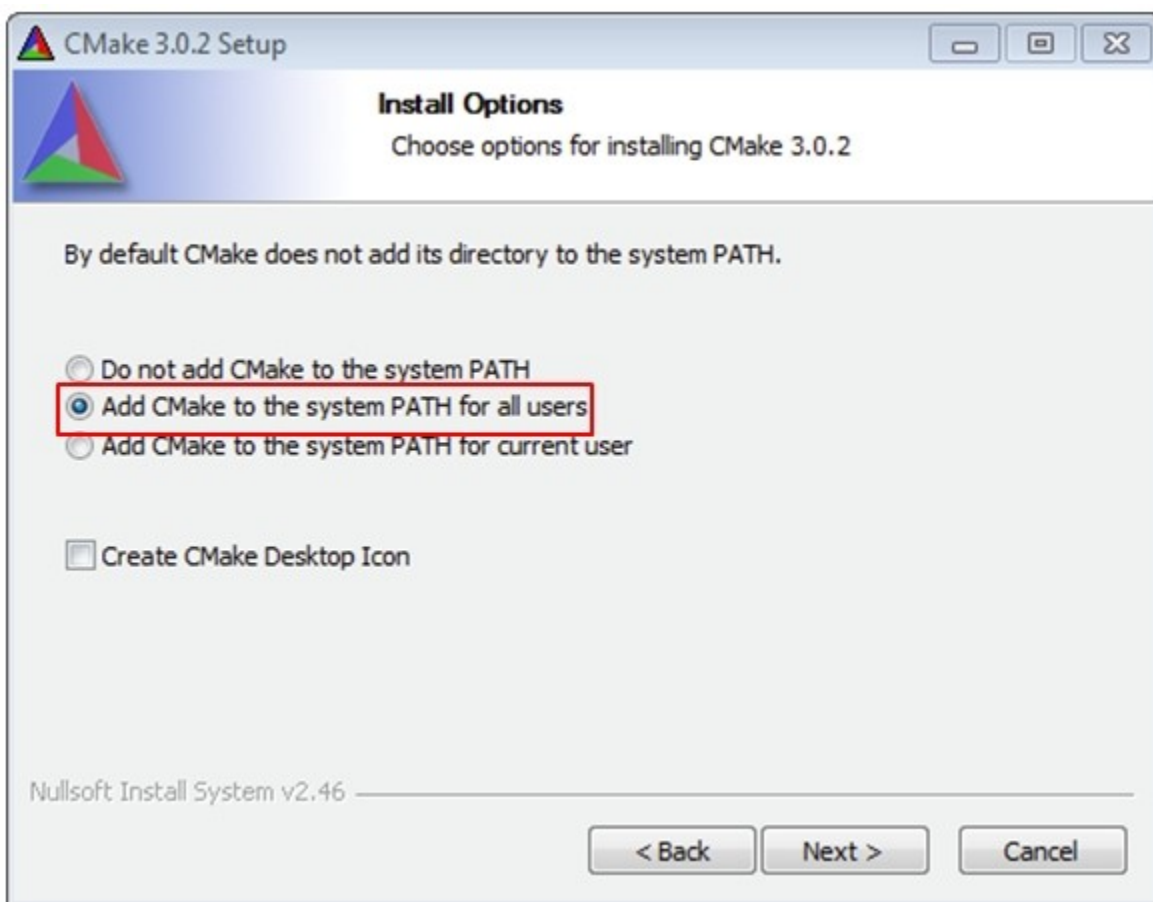
Reference the installation folder of the GNU Arm GCC Embedded tools for the exact path name of your installation.



**Figure 18. Add ARMGCC\_DIR system variable**

### 5.1.4 Install CMake

1. Download CMake 3.0.x from [www.cmake.org/cmake/resources/software.html](http://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 19. 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 20. Launch command prompt**



## Run a demo using Arm® GCC

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

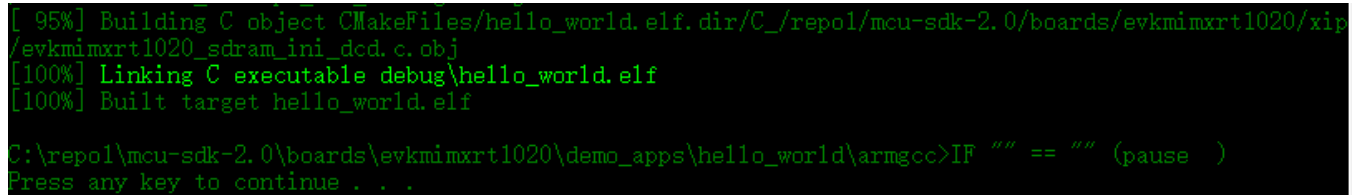
`<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc`

For this example, the exact path is: `<install_dir>/examples/evkmimxrt1020/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 the "build\_debug.bat" file in Windows Explorer to perform the build. The output is shown in this figure:



```
[ 95%] Building C object CMakeFiles/hello_world.elf.dir/C:/repol/mcu-sdk-2.0/boards/evkmimxrt1020/xip/evkmimxrt1020_sdram_ini_dcd.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\repol\mcu-sdk-2.0\boards\evkmimxrt1020\demo_apps\hello_world\armgcc>IF "" == "" (pause )
Press any key to continue . . .
```

Figure 21. hello\_world demo build successful

## 5.3 Run an example application

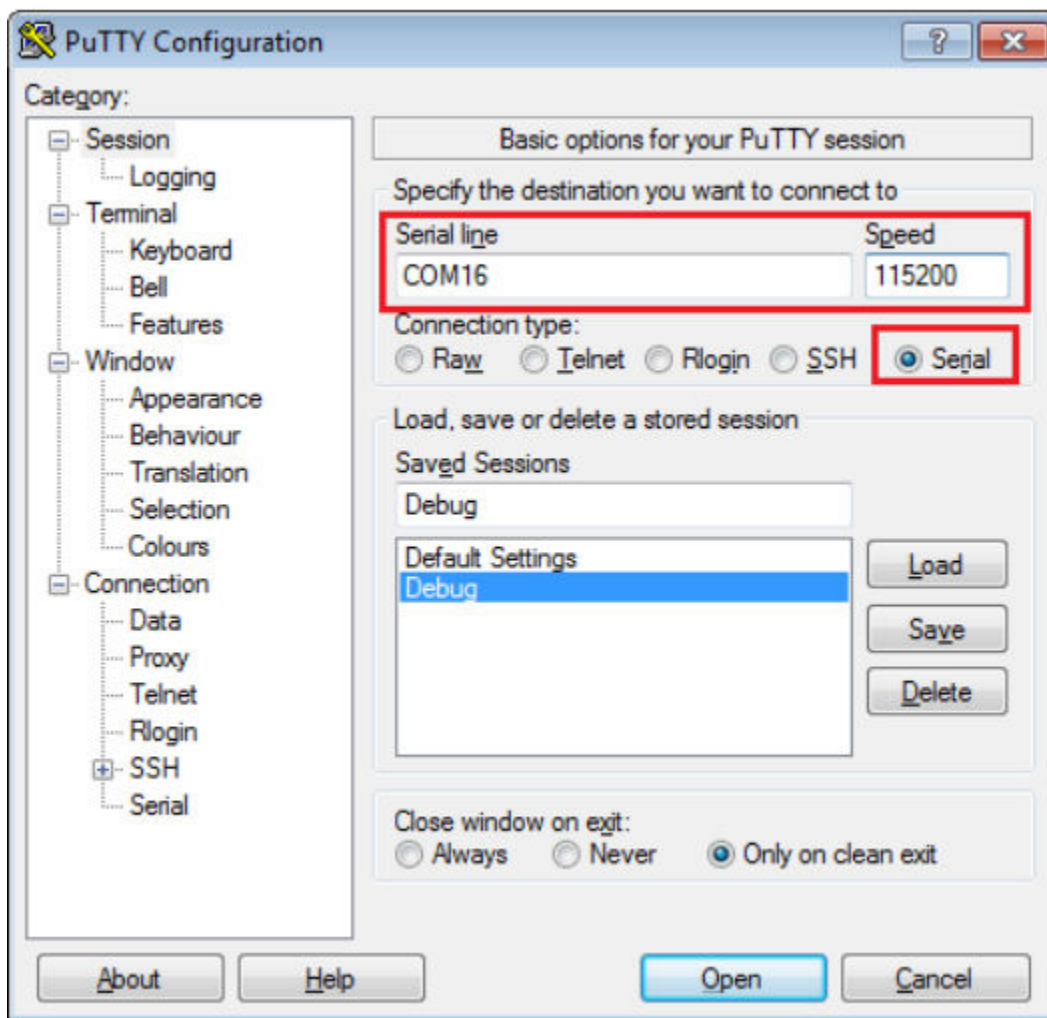
This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, two things must be done:

- 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 that 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:

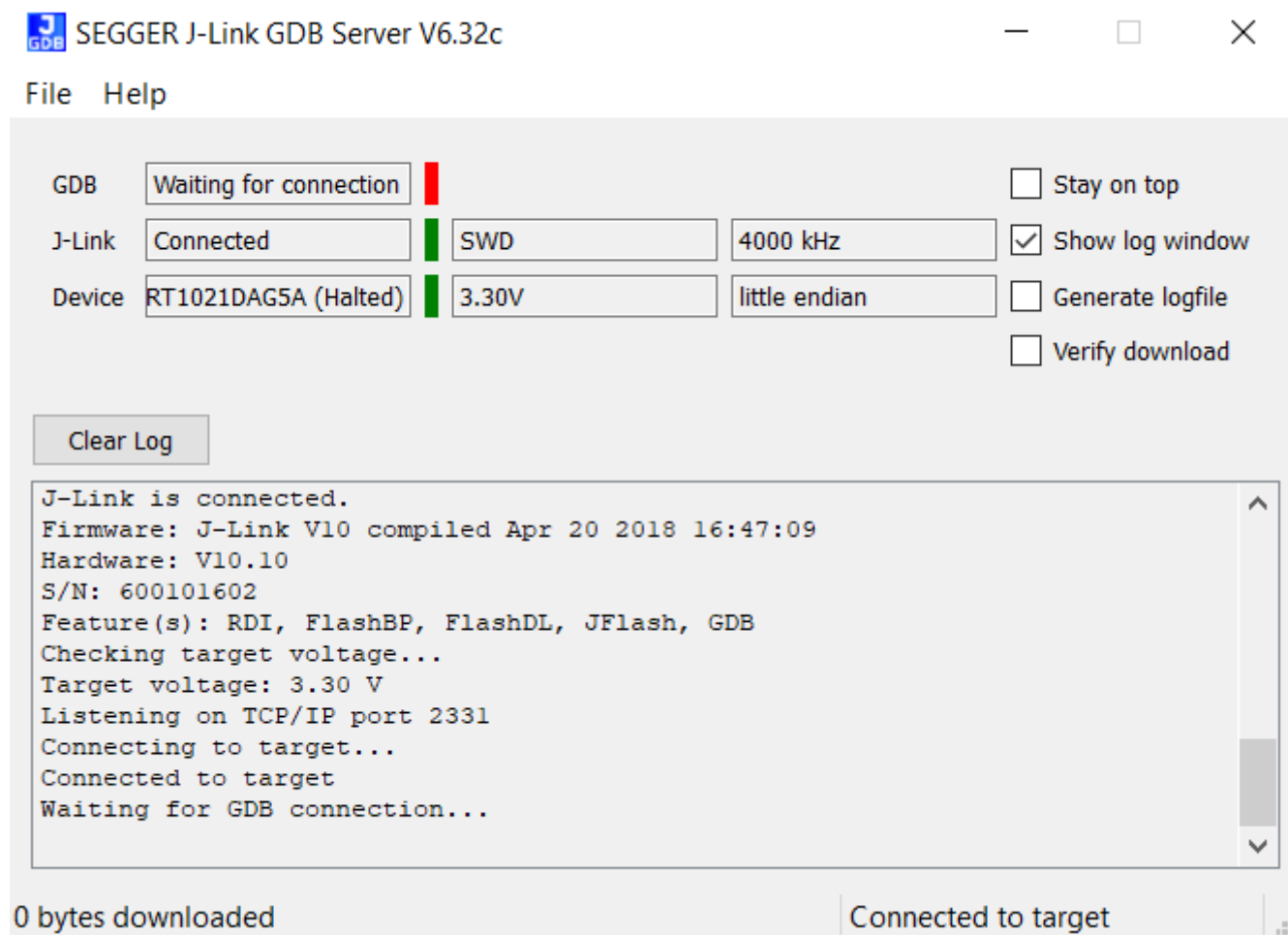
1. This board supports the J-Link debug probe. Before using it, install SEGGER software, which can be downloaded from <http://www.segger.com>
2. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector. If using a standalone J-Link debug pod, 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 board.h file)
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit





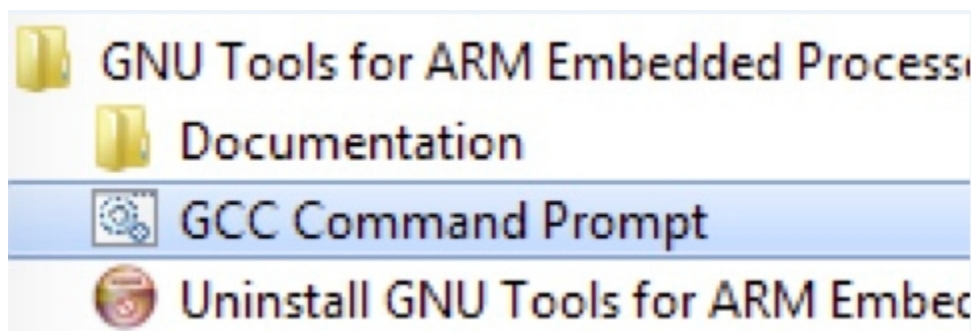
**Figure 22. Terminal (PuTTY) configurations**

4. Open the J-Link GDB Server application. Go to the SEGGER install folder, for example, C:\Program Files (x86)\SEGGER\JLink\_V632c. Open the command windows here, for Debug and Release targets, and use the command "JLinkGDBServer.exe". For the sdram\_debug and sdram\_release targets, use the command "JLinkGDBServer.exe-scriptfile <install\_dir>/boards/ evkmimxrt1020/demo\_apps/hello\_world/evkmimxrt1020\_sdram\_init.jlinkscript".
5. The target device selection chosen for this example is the MIMXRT1021DAG5A.
6. After it is connected, the screen should resemble this figure:



**Figure 23. SEGGER J-Link GDB Server screen after successful connection**

- 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 24. Launch command prompt**

- 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/evkmimxrt1020/demo\_apps/hello\_world/armgcc/debug

9. Run the command “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 2017-q4-major\bin>arm-none-eabi-gdb.exe C:\repo1\mcu-sdk-2.0\boards\evkmimxrt1020\demo_apps\hello_world\armgcc\debug\hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-git
Copyright (C) 2017 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\evkmimxrt1020\demo_apps\hello_world\armgcc\debug\hello_world.elf...done.
(gdb)

```

**Figure 25. 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.

## 6 Run a demo using MCUXpresso IDE

### NOTE

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

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 EVK-MIMXRT1020 platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

**NOTE**

Three macros "XIP\_EXTERNAL\_FLASH=1", "XIP\_BOOT\_HEADER\_ENABLE=1" and "XIP\_BOOT\_HEADER\_DCD\_ENABLE=1" are set by default 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.

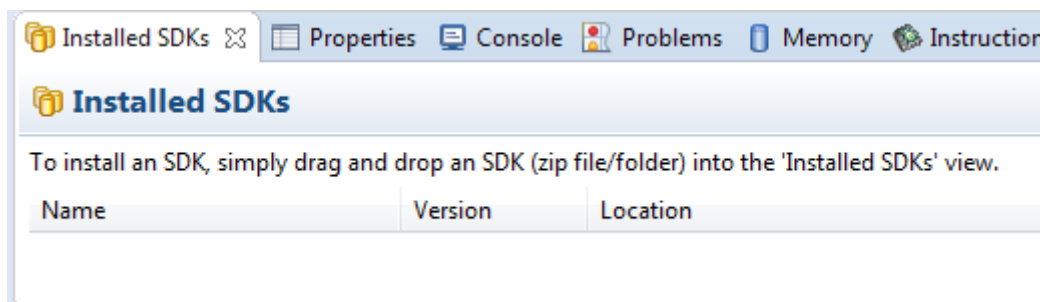
## 6.1 Select the workspace location

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 in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be outside of the MCUXpresso SDK tree.

## 6.2 Build an example application

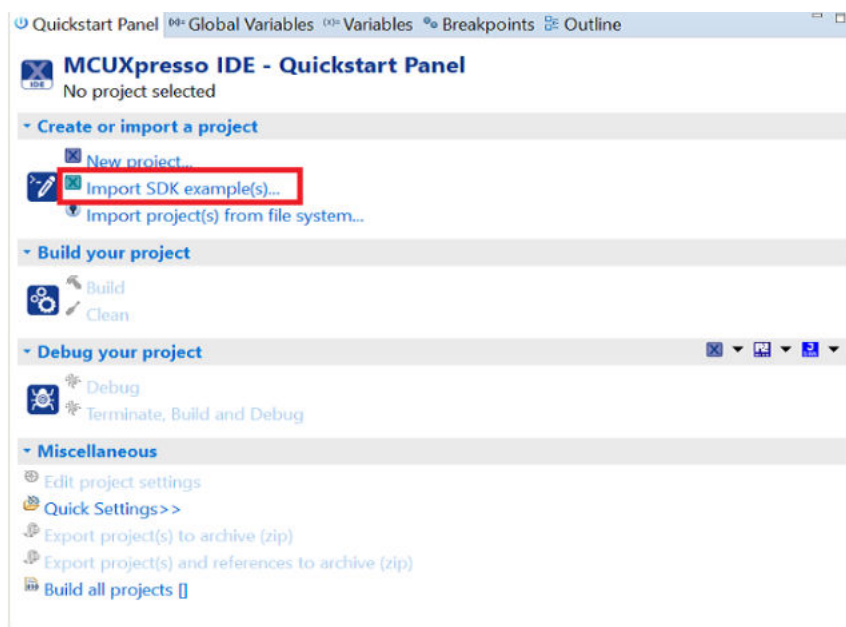
To build an example application, follow these steps.

1. Drag and drop the SDK zip file into the “Installed SDKs” view to install an SDK. In the window that appears, click the “OK” button and wait until the import has finished.



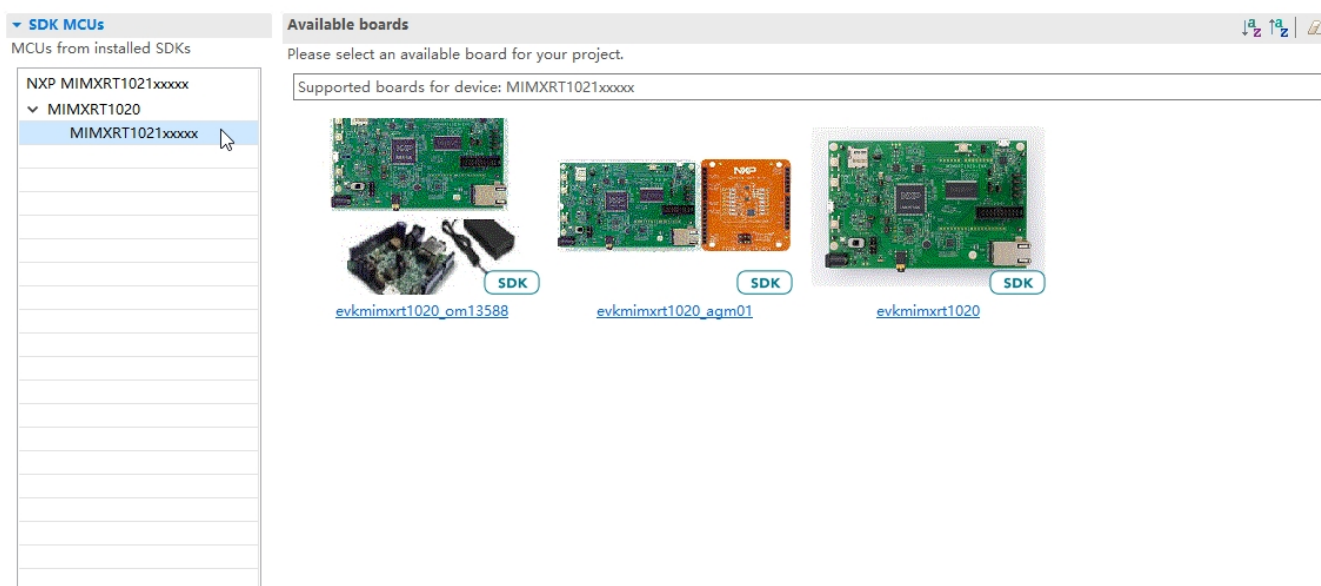
**Figure 26. Install an SDK**

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



**Figure 27. Import an SDK example**

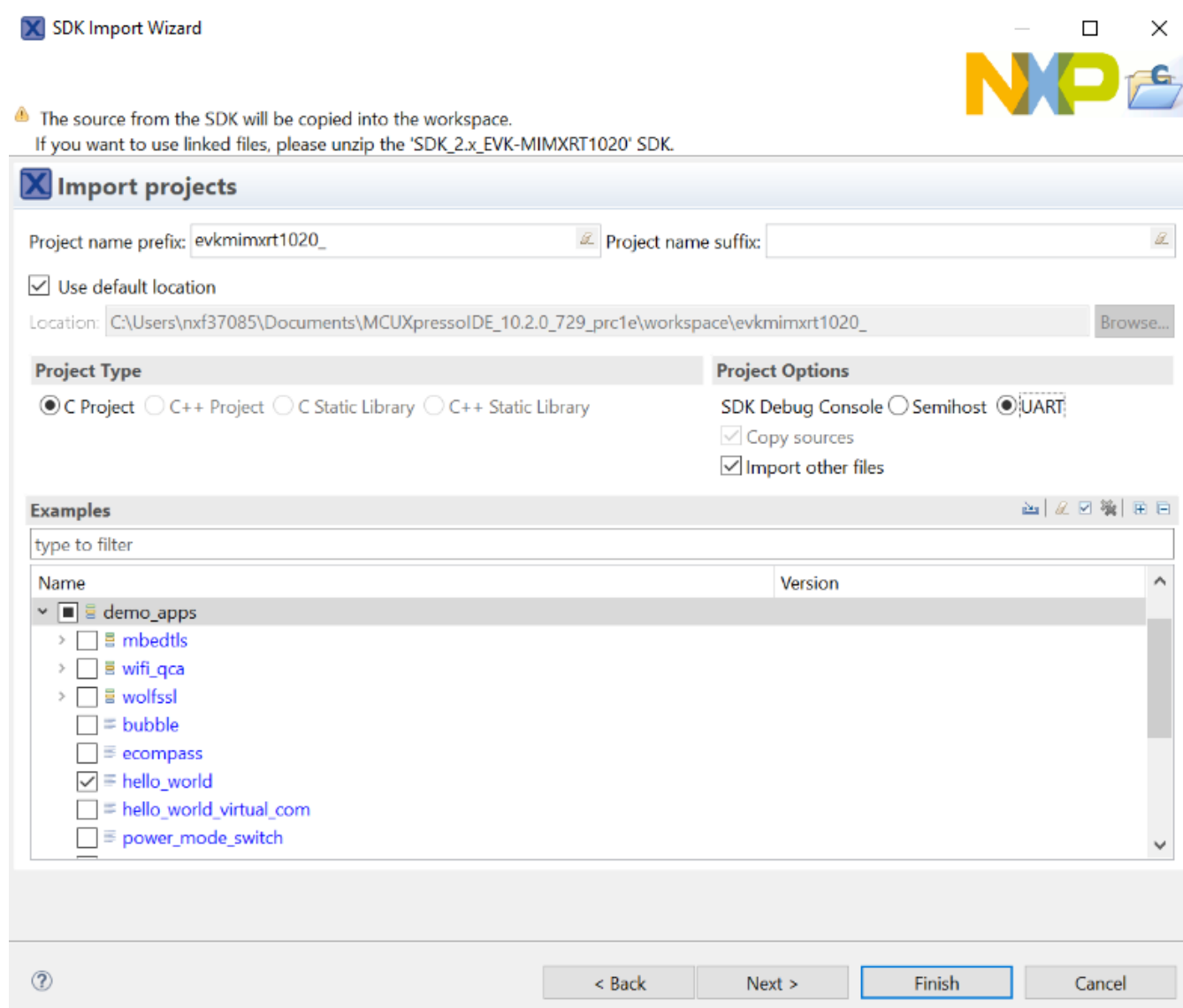
3. In the window that appears, expand the “MIMXRT1020” folder and select MIMXRT1021xxxxx. Then, select “evkbimxrt1020” and click the “Next” button.



**Figure 28. Select EVK-MIMXRT1020 board**

4. Expand the “demo\_apps” folder and select “hello\_world”. Then, click the "Next" button.

## Run a demo using MCUXpresso IDE



**Figure 29. Select "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, mbedtls\_benchmark, wolfssl\_benchmark, and for mmcau\_examples such as mmcau\_api). Otherwise, there is no need to select it. Click the "Finish" button.



**Advanced Settings**

**C/C++ Library Settings**

Set library type (and hosting variant) Redlib (nohost-nf)

☐ Redlib: Use floating point version of printf
 ☐ NewlibNano: Use floating point version of printf  
☐ Redlib: Use character rather than string based printf
 ☐ NewlibNano: Use floating point version of scanf  
☐ Redirect SDK "PRINTF" to C library "printf"  
☒ Include semihost HardFault handler
 ☐ Redirect printf/scanf to ITM  
☐ Redirect printf/scanf to UART

**Hardware settings**

Set Floating Point type FPv5-SP-D16 (HardABI)

**MCU C Compiler**

Language standard GNU C99 (-std=gnu99)

**MCU Linker**

☐ Link application to RAM

**Memory Configuration**

Memory details

Type	Name	Alias	Location	Size	Driver
Flash	BOARD_FLASH	Flash	0x60000000	0x800000	MIMXRT1020-EVK_IS25LP064.cfx
RAM	SRAM_DTC	RAM	0x20000000	0x10000	
RAM	SRAM_ITC	RAM2	0x0	0x10000	
RAM	SRAM_OC	RAM3	0x20200000	0x20000	
RAM	BOARD_SDRAM	RAM4	0x80000000	0x2000000	

Edit...

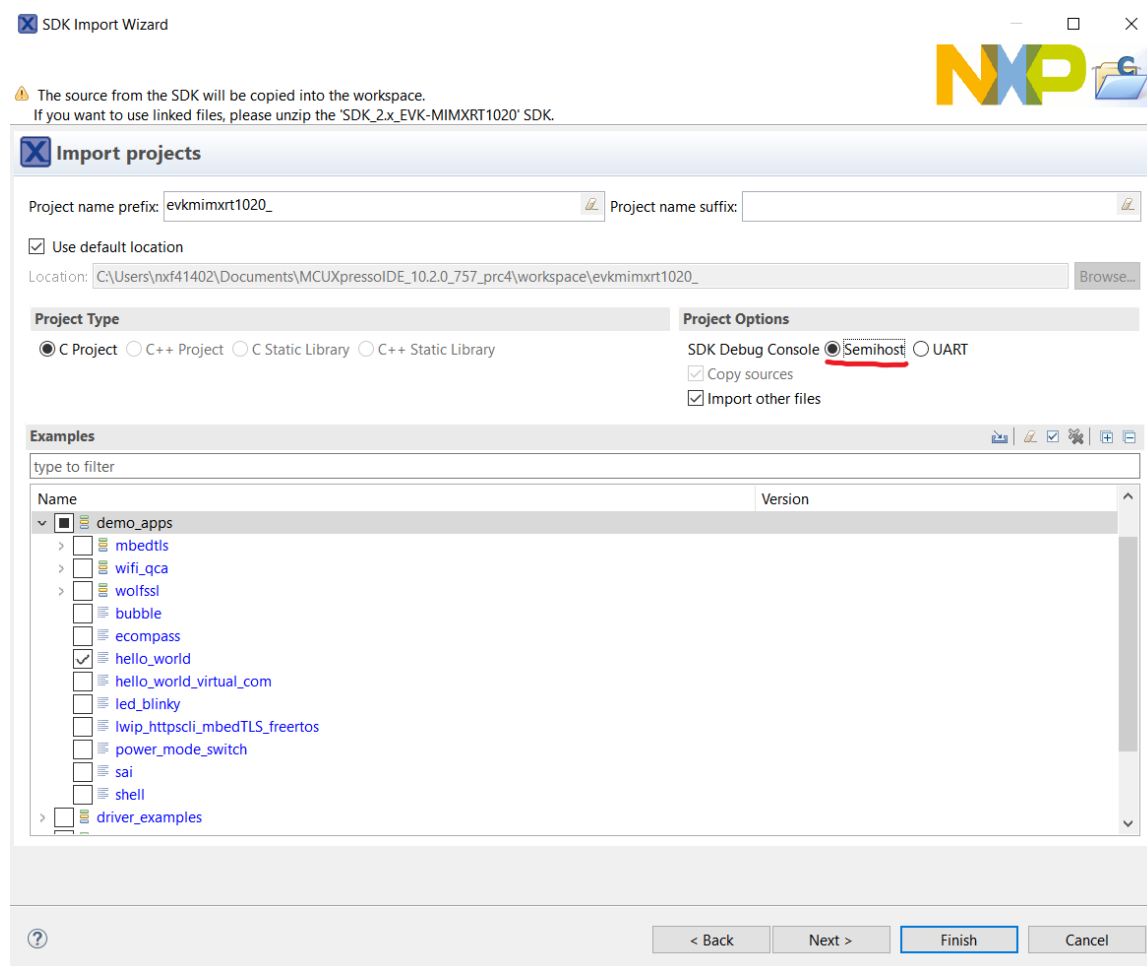
?
< Back
Next >
Finish
Cancel

**Figure 30. Select "User floating print version of printf"**

**NOTE**

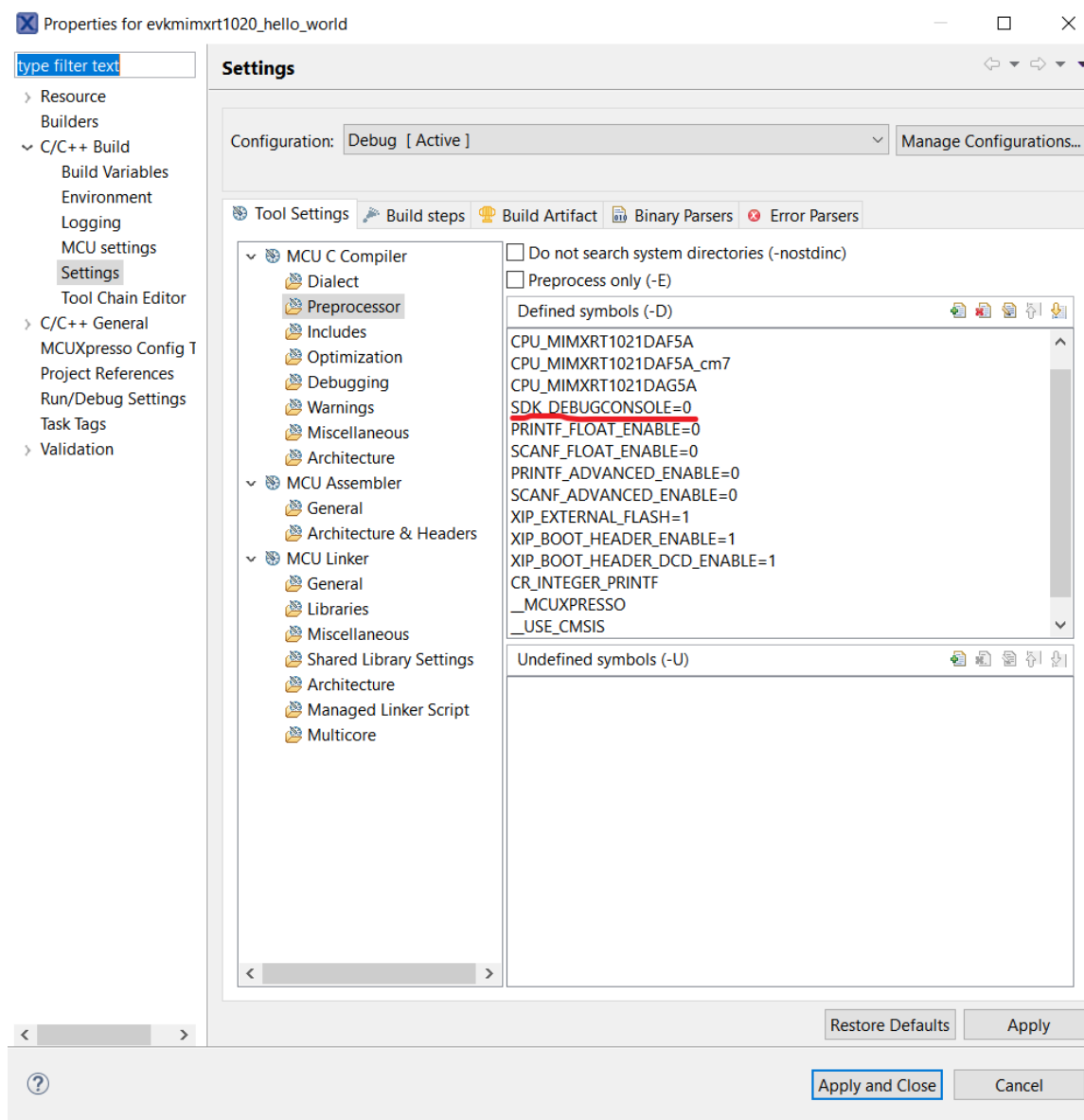
If you want to use semihost to print log, first select the "Semihost" button when importing projects. Then, change the value of "SDK\_DEBUGCONSOLE" from "1" to "0" in Properties.

## Run a demo using MCUXpresso IDE



**Figure 31. Select "Semihost"**





**Figure 32. Setting "SDK\_DEBUGCONSOLE"**

- On the *Quickstart Panel*, click "'build evkmimxrt1020\_demo\_apps\_hello\_world' [Debug]".

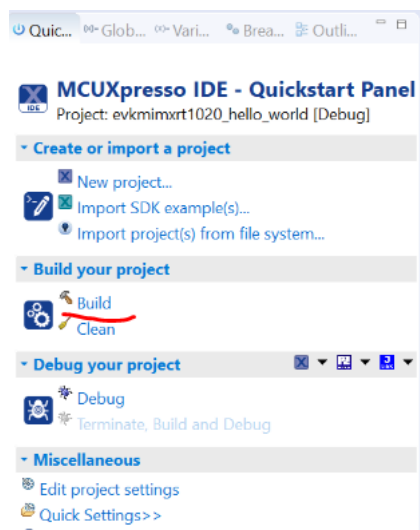


Figure 33. Build hello world case

## 6.3 Run an example application

For more information on debug probe support in the MCUXpresso IDE, visit [community.nxp.com](http://community.nxp.com).

To download and run the application, perform these steps:

### NOTE

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

1. On the *Quickstart Panel*, click on "'Debug evkmimxrt1020\_demo\_apps\_hello\_world' [Debug]'".

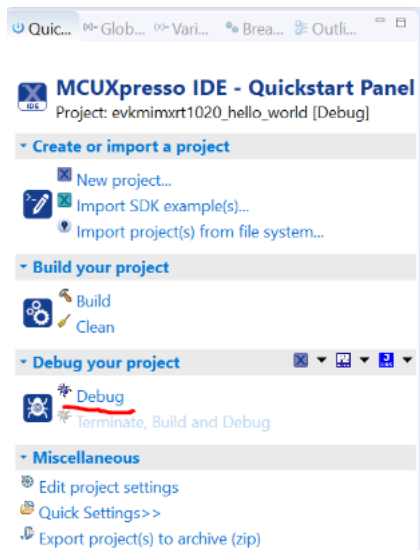
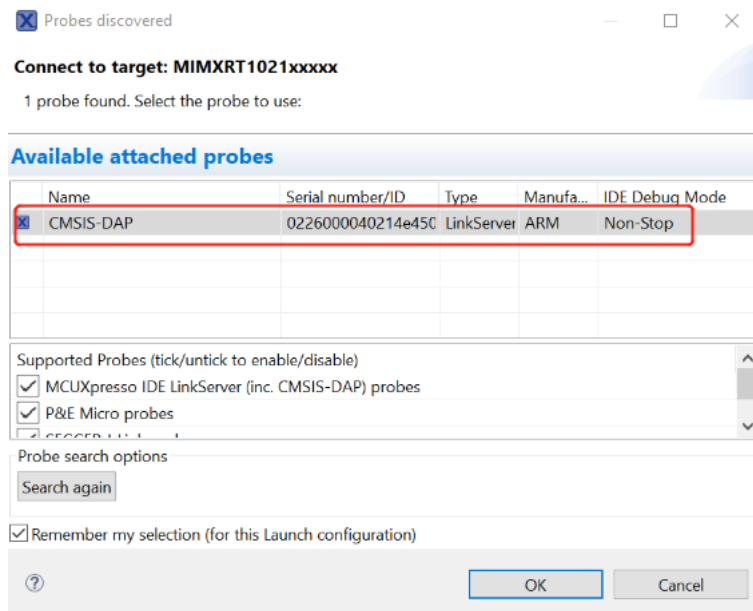


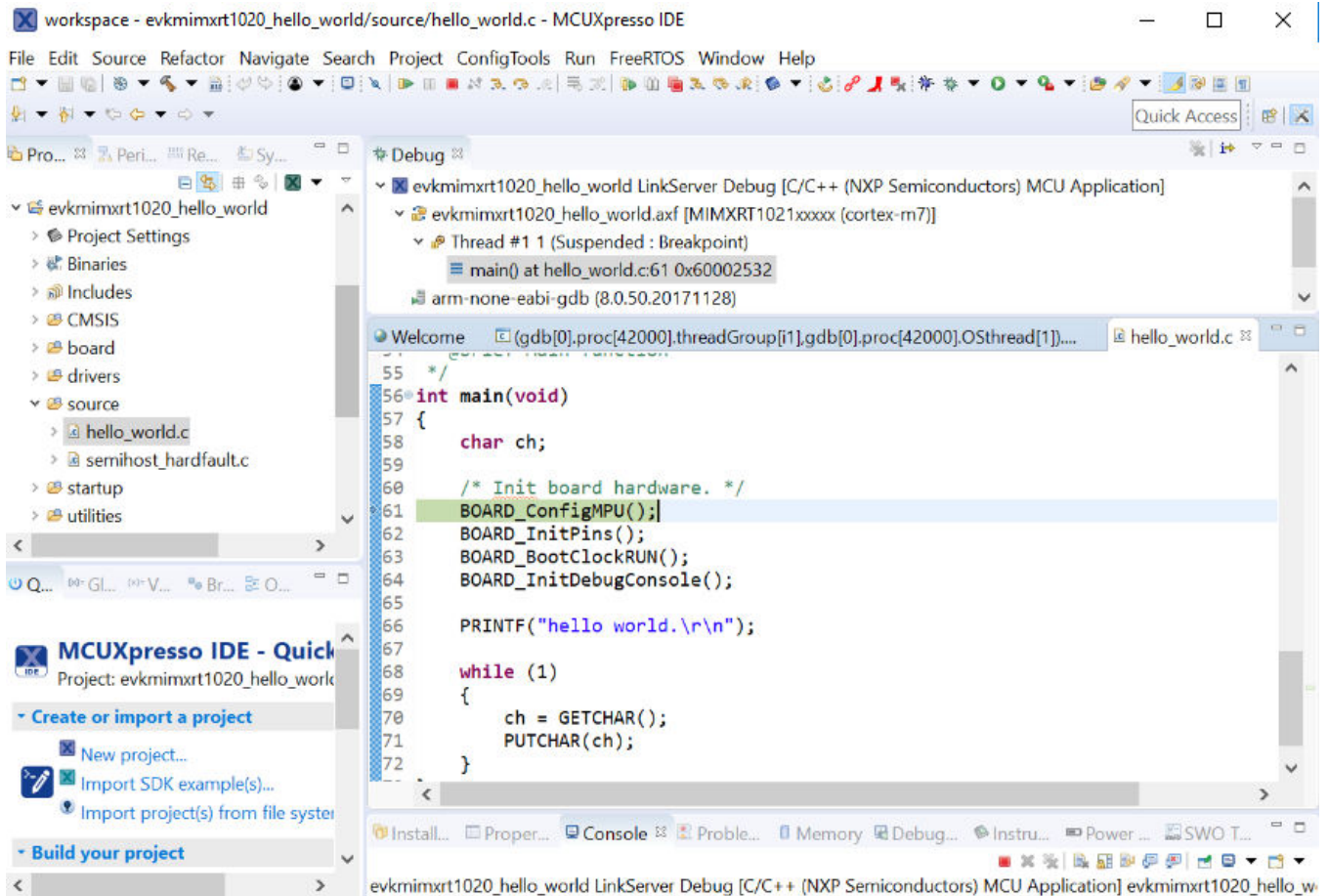
Figure 34. Debug "hello\_world" case

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 the "OK" button. (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.

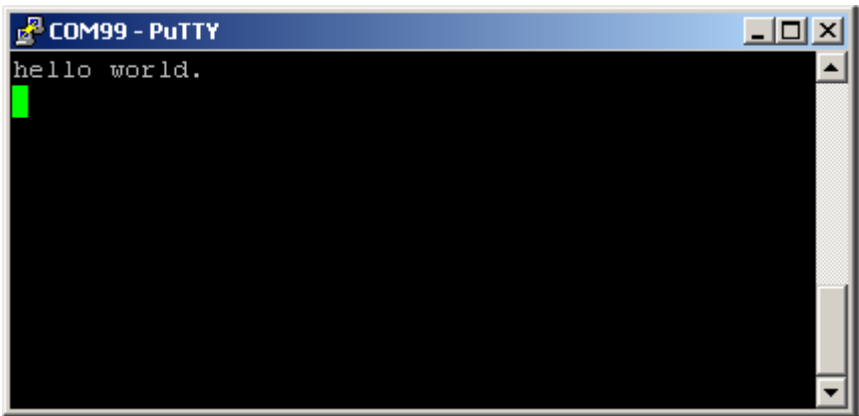






Figure 38. Text display of the hello\_world demo

## 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.

The MCUXpresso Config Tools consist of the following:

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	
Project Cloner	Allows creation of the standalone projects from SDK examples.	

MCUXpresso Config Tools can be accessed in the following products:

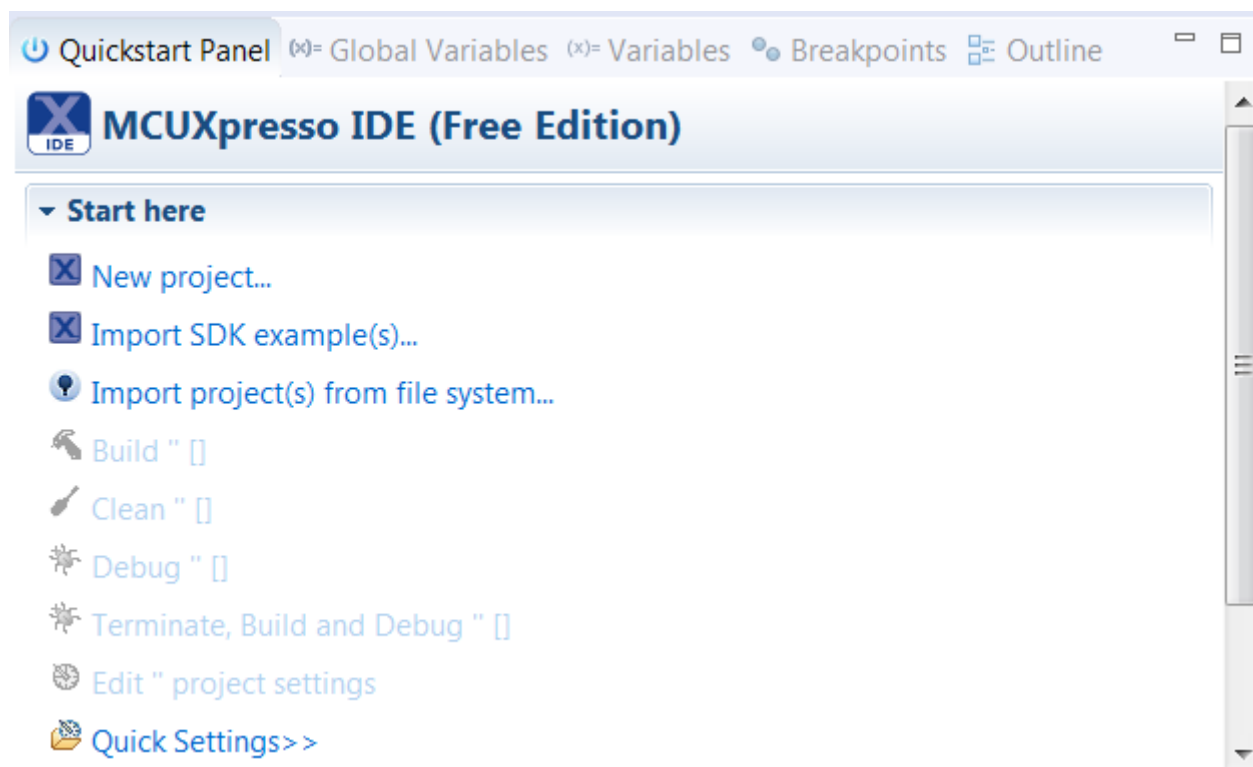
- **Integrated** in the MCUXpresso IDE. Config tools are integrated with the compiler and debugger, so this represents the easiest way to begin that development.
- **Standalone version** available for download from [www.nxp.com](http://www.nxp.com). Recommended for customers using IAR Embedded Workbench, Keil MDK  $\mu$ Vision, or Arm GCC.
- **Online version** available on [mcuxpresso.nxp.com](http://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 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), offers the flexibility to select/change many builds, includes a library, and provides source code options. The source code is organized as software components, categorized as driver, 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 the “New project” option, shown in the below figure.



**Figure 39. MCUXpresso IDE Quickstart Panel**

For more details of the usage of new project wizard, see the “MCUXpresso\_IDE\_User\_Guide.pdf” in the MCUXpresso IDE installation folder.

## 9 Appendix A - How to determine COM port

## Appendix A - How to determine COM port

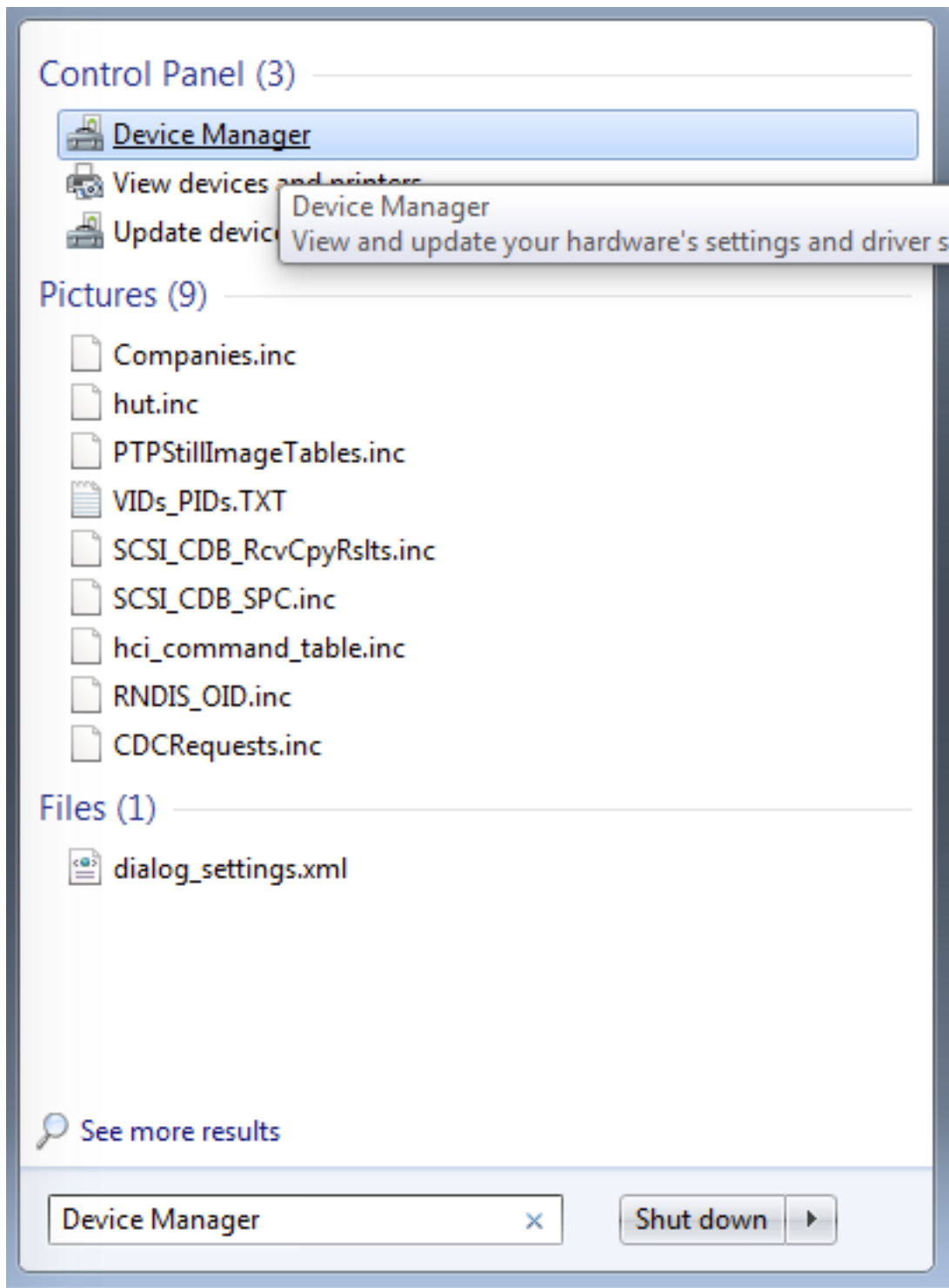
This section describes the steps necessary to determine the debug COM port number of your NXP hardware development platform.

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, another is for Cortex M4.

2. **Windows:** 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 below:



**Figure 40. Device manager**

3. 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:

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

The MCUXpresso SDK for i.MX RT1020 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.

**Table 2. Macros added in flexspi\_nor**

<b>XIP_EXTERNAL_FLASH</b>	1: Exclude the code which changes the clock of flexspi.
	0: Make no changes.
<b>XIP_BOOT_HEADER_ENABLE</b>	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.
<b>XIP_BOOT_HEADER_DCD_ENABLE</b>	1: Add device configuration data to the image.
	0: Do <b>NOT</b> add device configuration data to the image.

- The following table shows the different effect on the built image with a different combination of these macros:

		<b>XIP_BOOT_HEADER_DCD_ENABLE=1</b>	<b>XIP_BOOT_HEADER_DCD_ENABLE=0</b>
<b>XIP_EXTERNAL_FLASH=1</b>	<b>XIP_BOOT_HEADER_ENABLE=1</b>	<ul style="list-style-type: none"> <li>Can be programmed to qspi flash by IDE and can run after POR reset if qspi flash is the boot source.</li> <li>SDRAM will be initialized.</li> </ul>	<ul style="list-style-type: none"> <li>Can be programmed to qspi flash by IDE, and can run after POR reset if qspi flash is the boot source.</li> <li>SDRAM will <b>NOT</b> be initialized.</li> </ul>
	<b>XIP_BOOT_HEADER_ENABLE=0</b>	<ul style="list-style-type: none"> <li><b>CANNOT</b> run after POR reset if it is programmed by IDE, even if qspi flash is the boot source.</li> </ul>	

*Table continues on the next page...*

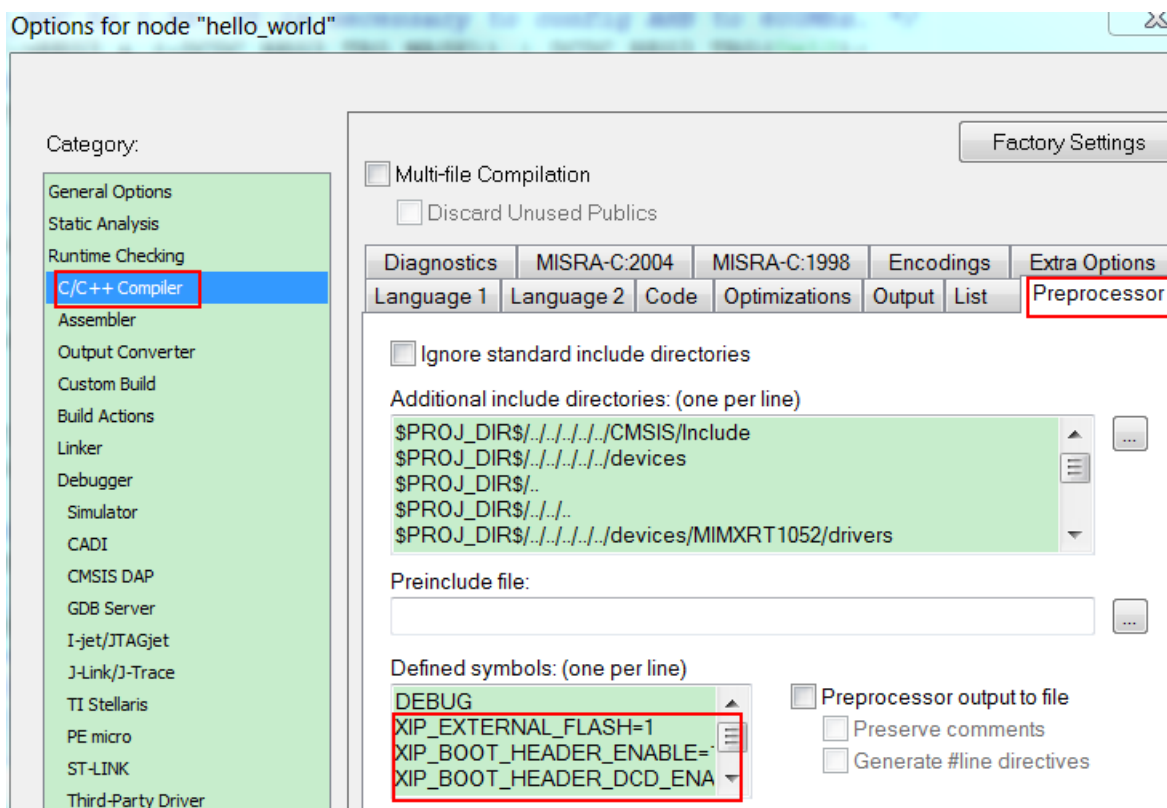


XIP_EXTERNAL_FLASH=0	<ul style="list-style-type: none"> <li>This image <b>CANNOT</b> complete XIP because when this macro is set to 1, it excludes the code, which changes the clock for FLEXSPI.</li> </ul>	
----------------------	---	--

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

Take hello\_world as an example:

**IAR:**



**Figure 41. Options node IAR**

**MDK:**

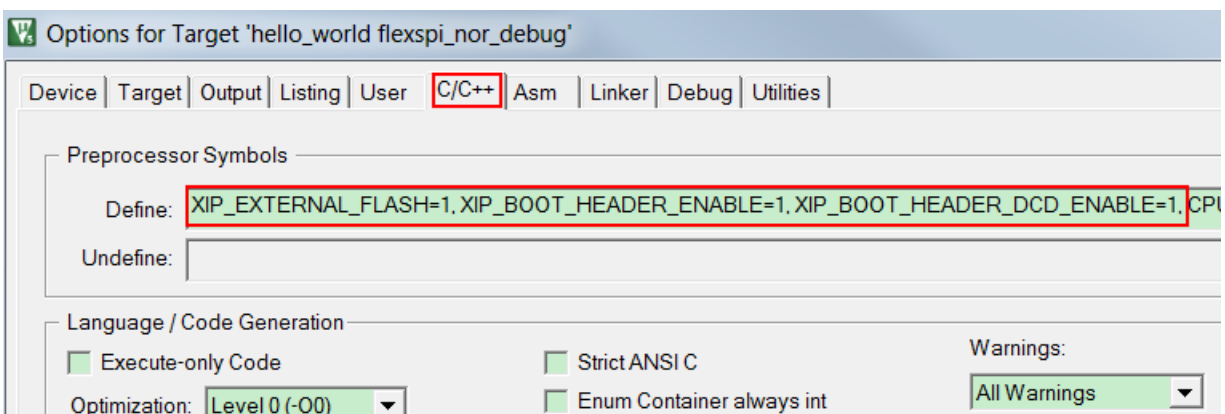


Figure 42. Options for target

#### 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 43. Change configuration CMakeLists.txt

#### MCUX:

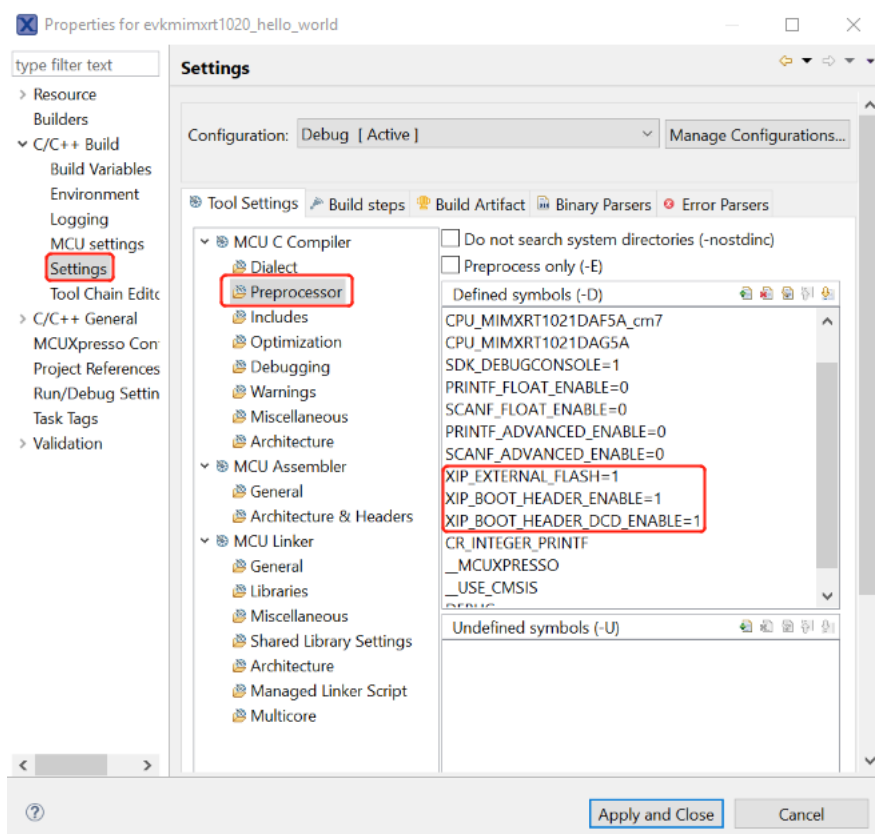


Figure 44. Properties for evkbimxrt1020

**How to Reach Us:****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

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

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

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

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, and  $\mu$ Vision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2018 NXP B.V.

Document Number MCUXSDKMIMXRT102XGSUG  
Revision 0, 06/2018

