

MCUXSDKRDRW61XGSUG

Getting Started with MCUXpresso SDK for RD-RW61X

Rev. 2.16.000 — 17 June 2024

User guide

Document information

Information	Content
Keywords	RD-RW61X, Getting Started, RW61X, MCUXSDKRDRW61XGSUG
Abstract	This document describes the steps to get started with MCUXpresso SDK for RD-RW61X.



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, a USB host and device stack, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for RD-RW61X* (document MCUXSDKRDRW61XRN).

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

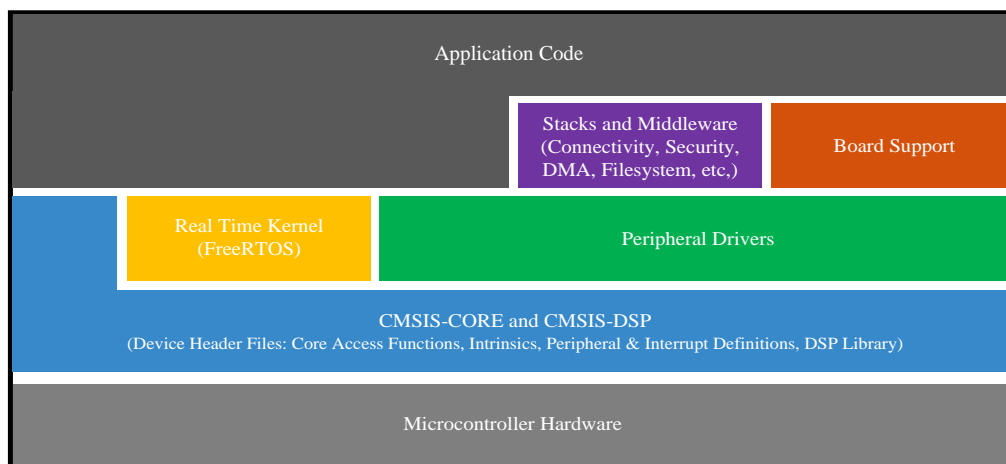


Figure 1. MCUXpresso SDK layers

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, i.mxrt EVK boards, 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

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:

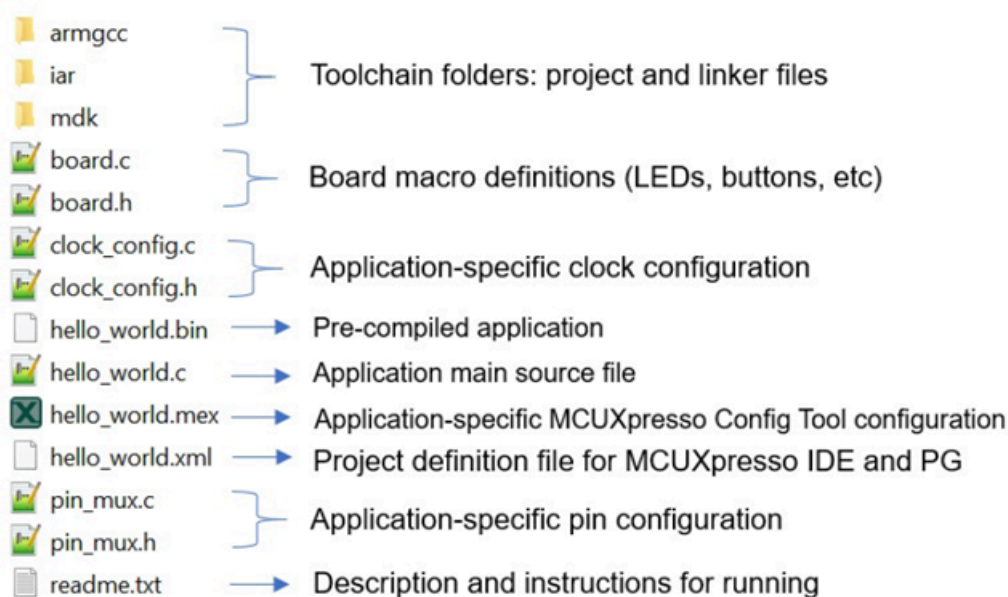


Figure 2. Application folder structure

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

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

3 Run a demo using MCUXpresso IDE

Note: 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 RD-RW61X-BGA hardware platform is used as an example, though these steps can be applied to any example application in the MCUXpresso SDK.

3.1 Select the workspace location

Every time MCUXpresso IDE launches, it prompts the user to select a workspace location. MCUXpresso IDE is built on top of Eclipse which uses workspace to store information about its current configuration, and in some use cases, source files for the projects are in the workspace. The location of the workspace can be anywhere, but it is recommended that the workspace be located outside of the MCUXpresso SDK tree.

3.2 Build an example application

To build an example application, follow these steps.

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

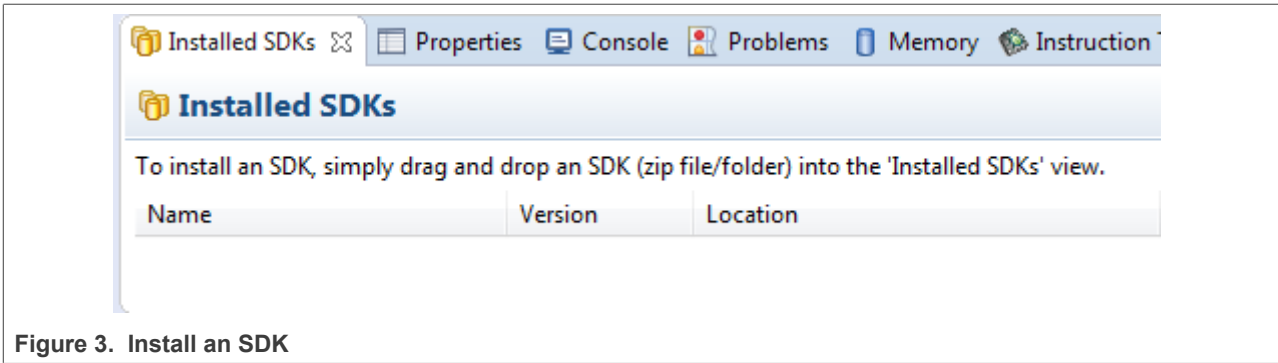


Figure 3. Install an SDK

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

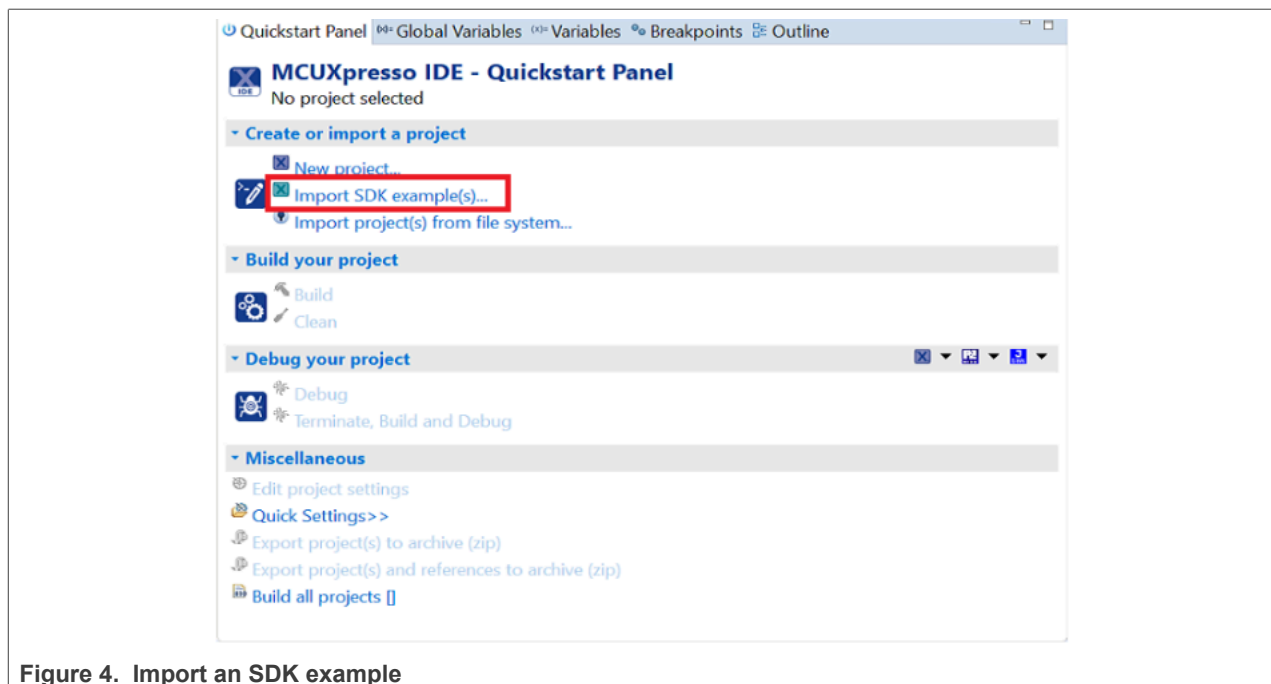


Figure 4. Import an SDK example

3. In the window that appears, expand the **RW61X** folder and select **RW612**.
4. Select **rdrw612bga** and click **Next**.

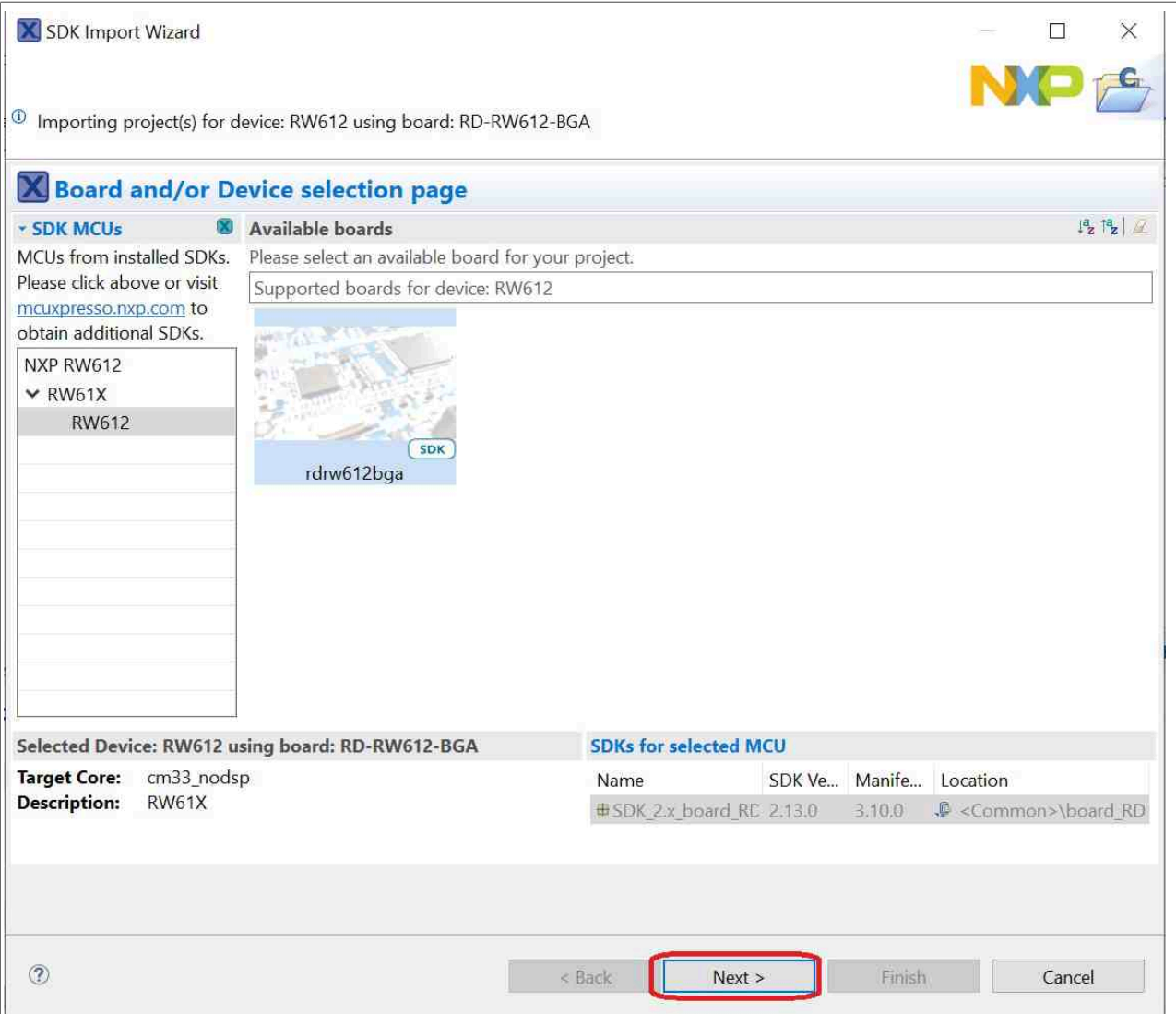


Figure 5. Select RD-RW610-BGA board

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

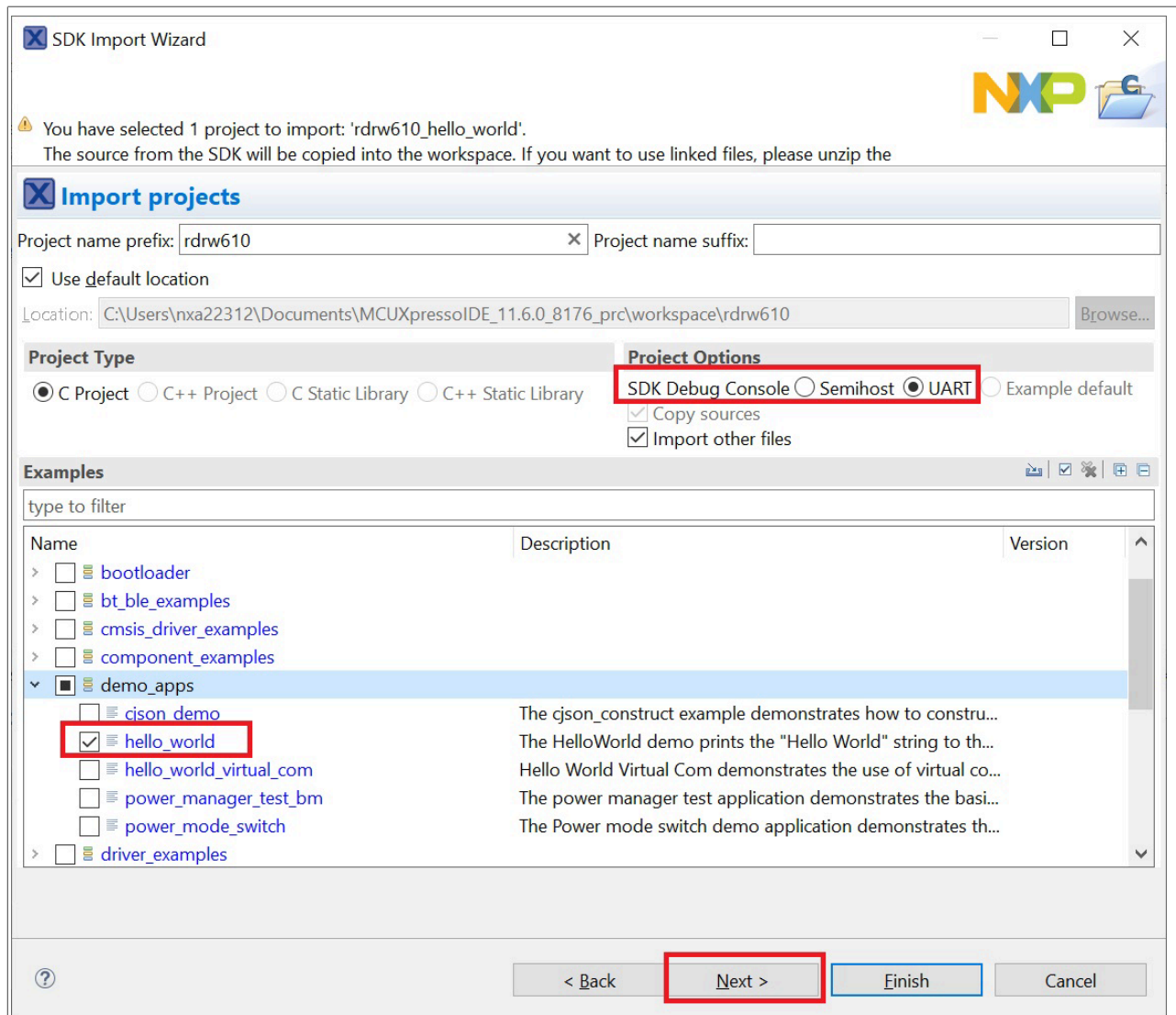


Figure 6. Select hello_world

6. Ensure **Redlib: Use floating point version of printf** is selected if the example prints floating point numbers on the terminal. Otherwise, it is not necessary to select this option. Then, click **Finish**.

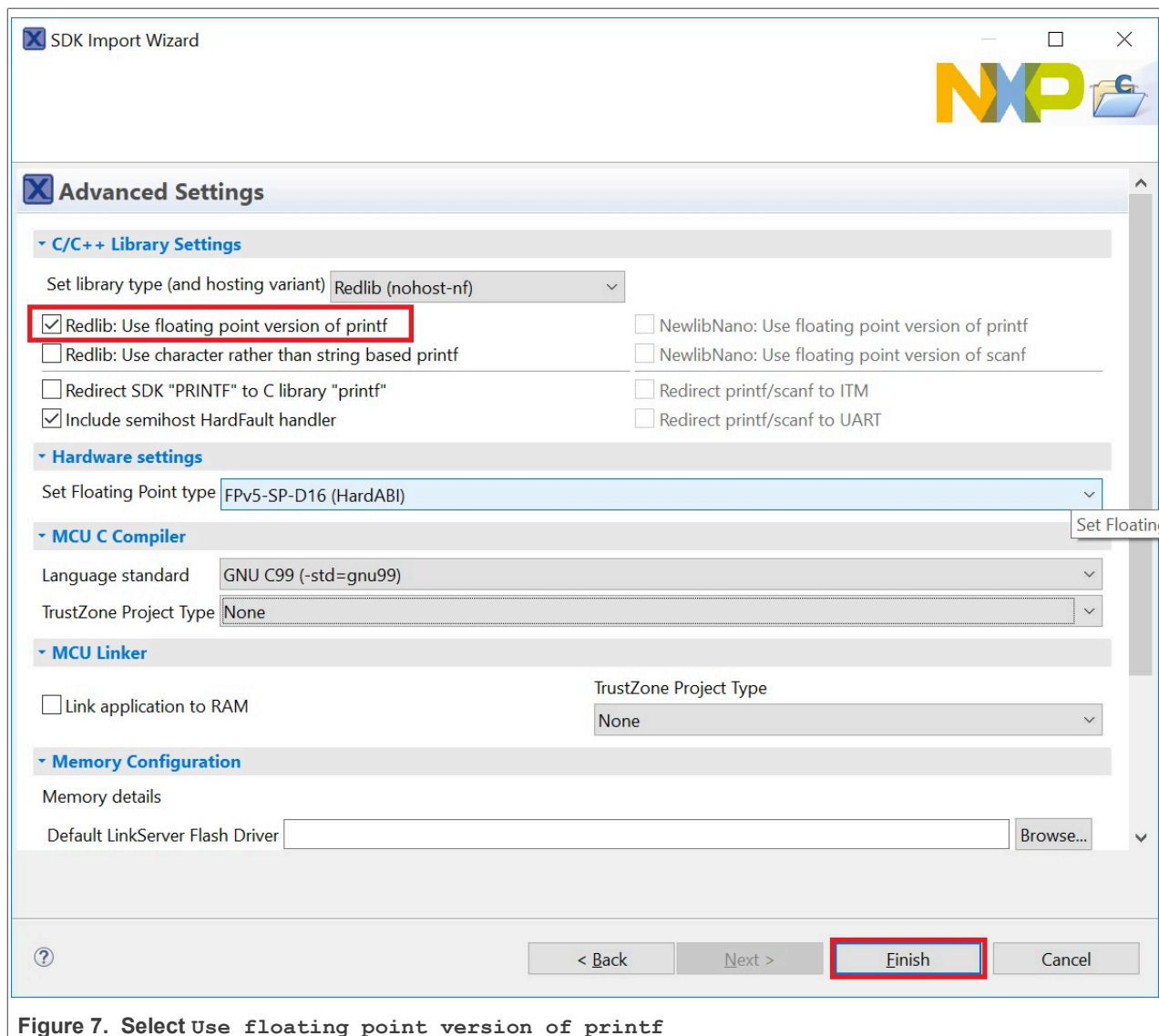


Figure 7. Select Use floating point version of printf

3.3 Run an example application

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

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

1. See the table in [Section 10](#) to determine the debug interface that comes loaded on your specific hardware platform.
 - For boards with CMSIS-DAP/mbd/DAPLink interfaces, visit developer.mbed.org/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.
 - For boards with a P&E Micro interface, see [PE micro](#) to download and install the P&E Micro Hardware Interface Drivers package.

2. Connect the development platform to your PC via a 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 8](#)). 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

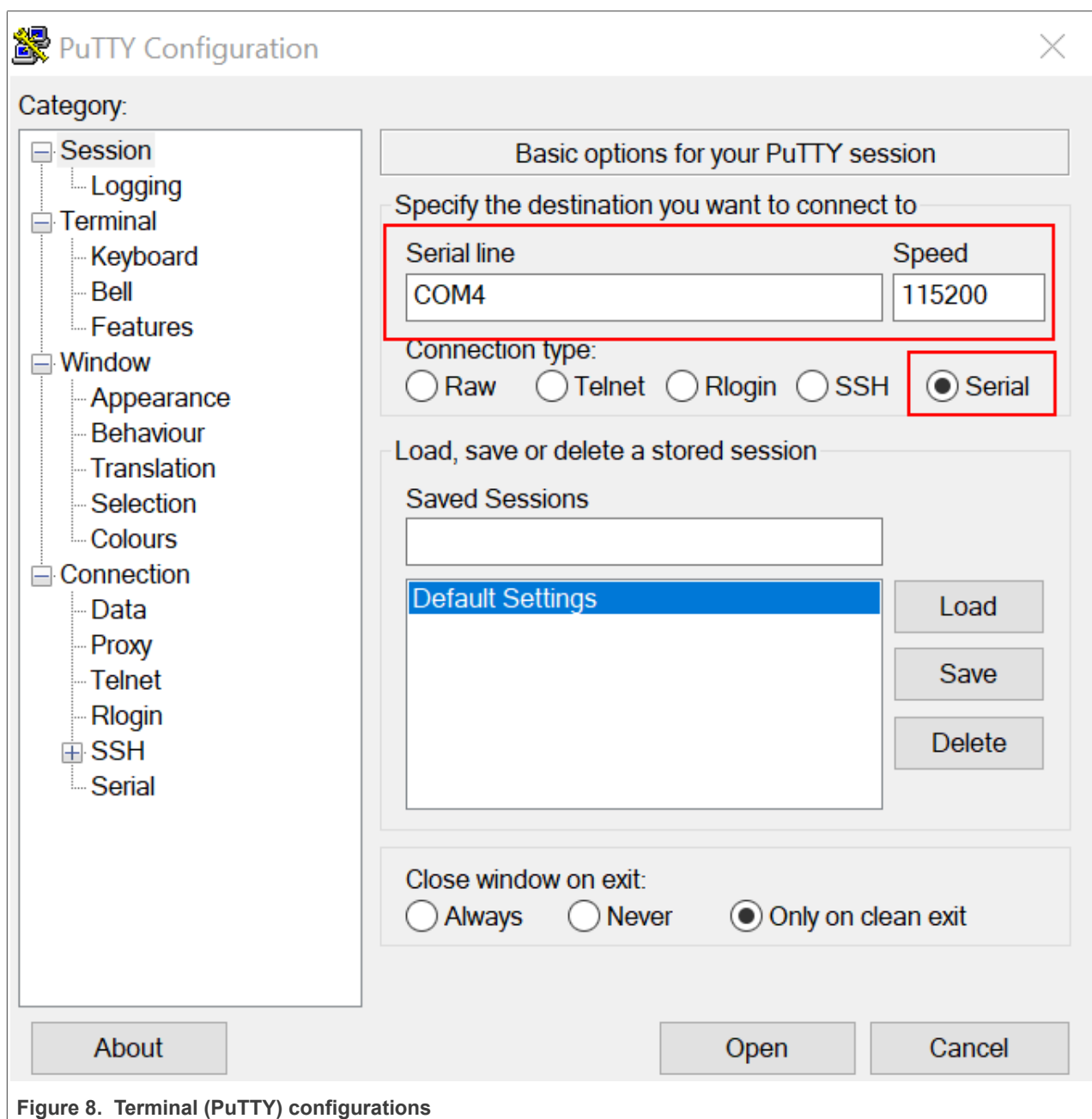


Figure 8. Terminal (PuTTY) configurations

4. On the **Quickstart Panel**, click on **Debug rdrw610_hello_world [Debug]** to launch the debug session.

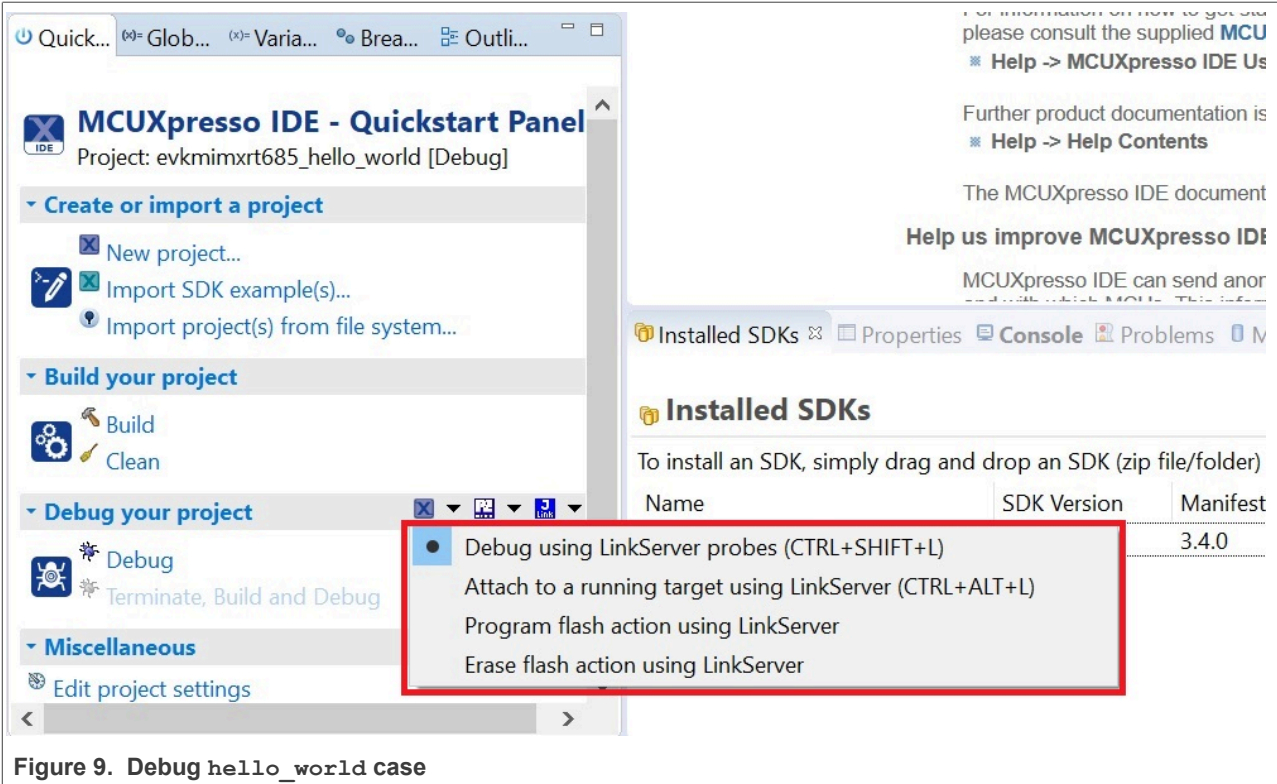


Figure 9. Debug hello_world case

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

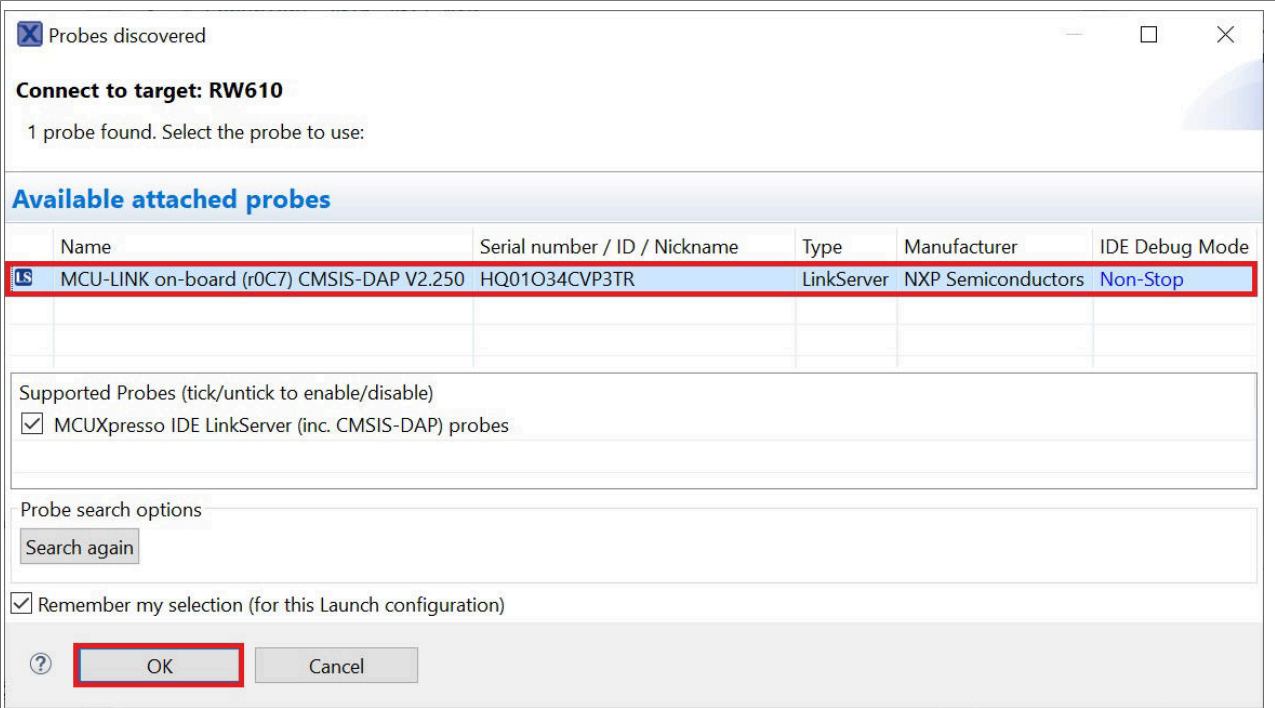


Figure 10. Attached Probes: debug emulator selection

Note: Before debugging, it is recommend that the board is set to FlexSPI flash boot mode (CON[3:0]: [off, off, off, off]). If there is no bootable image in a flash, you can set CON[3:0] to all "on" and use J-Link to debug. In such a case, **Reset before running** must be disabled. Double click the <example> J-Link Debug.launch file, and deselect this option under **JLink Debugger ->Additional Options** menu.

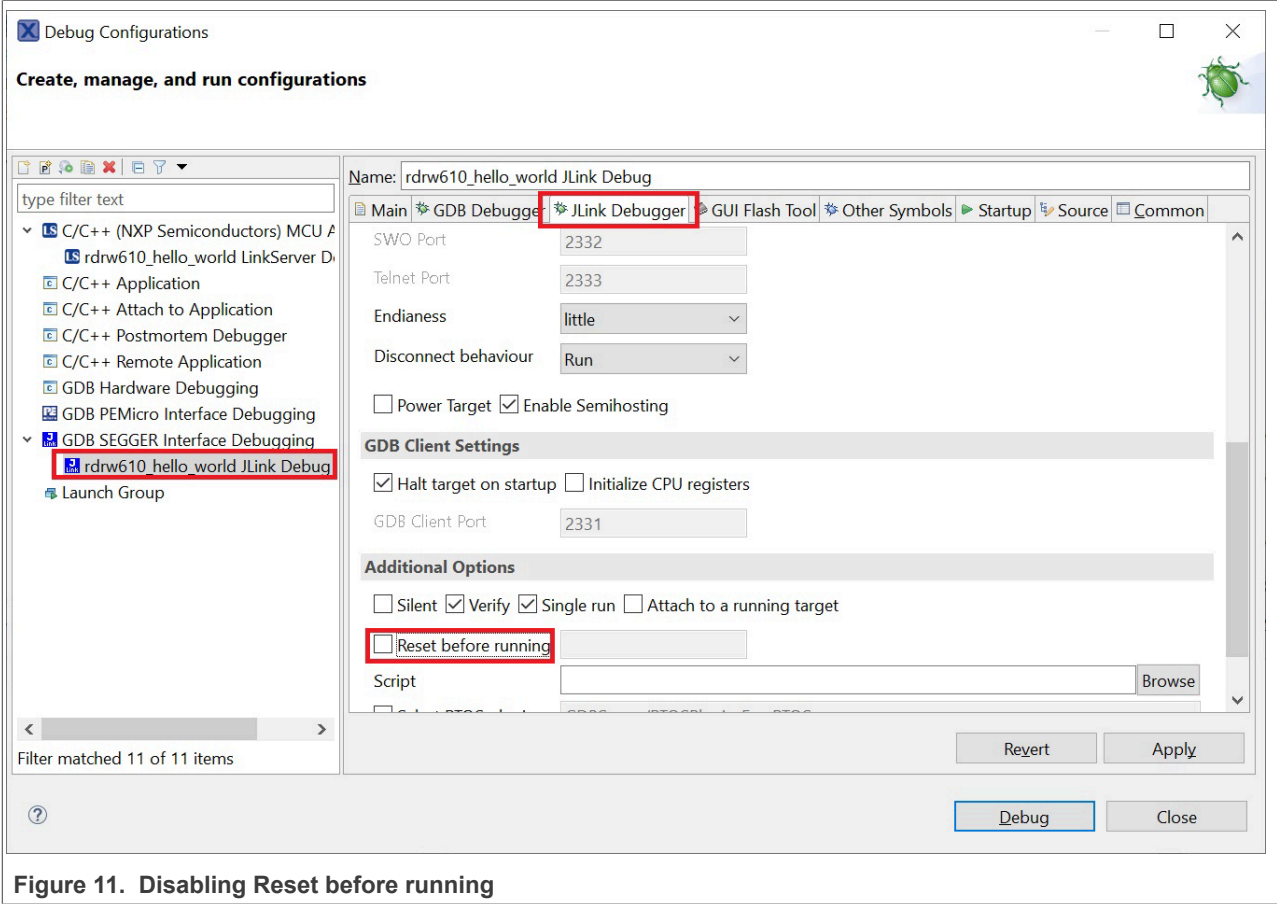


Figure 11. Disabling Reset before running

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

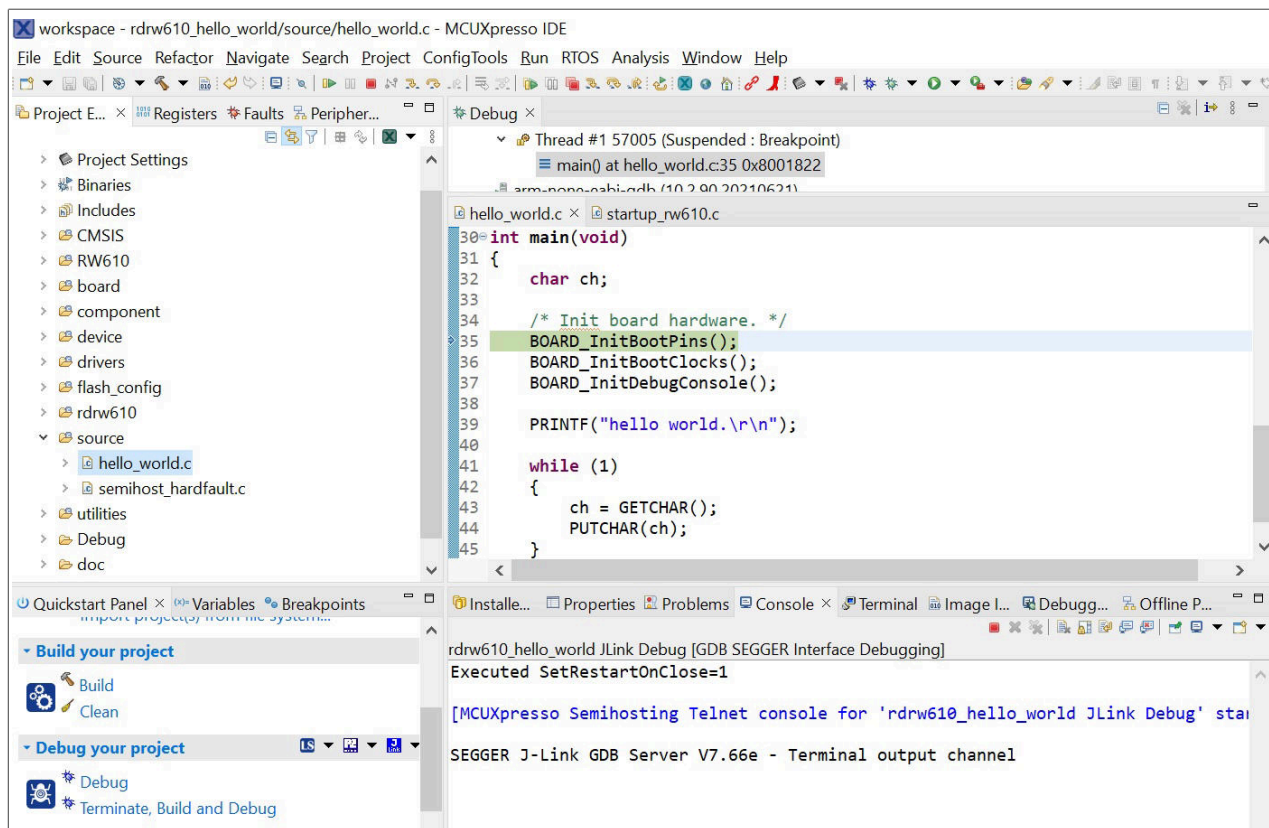


Figure 12. Stop at `main()` when running debugging

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



Figure 13. Resume button

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



Figure 14. Text display of the `hello_world` demo

4 Run a demo application using IAR

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

Note: IAR Embedded Workbench for Arm version 8.40.2 is used in the following example. The IAR toolchain should correspond to the latest supported version, as described in the MCUXpresso SDK Release Notes (document ID: MCUXSDKRN). IAR/Segger still does not support RW61x well. Therefore, contact the support team to get `iar_support_patch_rw610_rfp1.zip` and install the IAR patch before opening the IAR project.

4.1 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 RD-RW61X hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/rdrw61x/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.
For this example, select **hello_world – debug**.

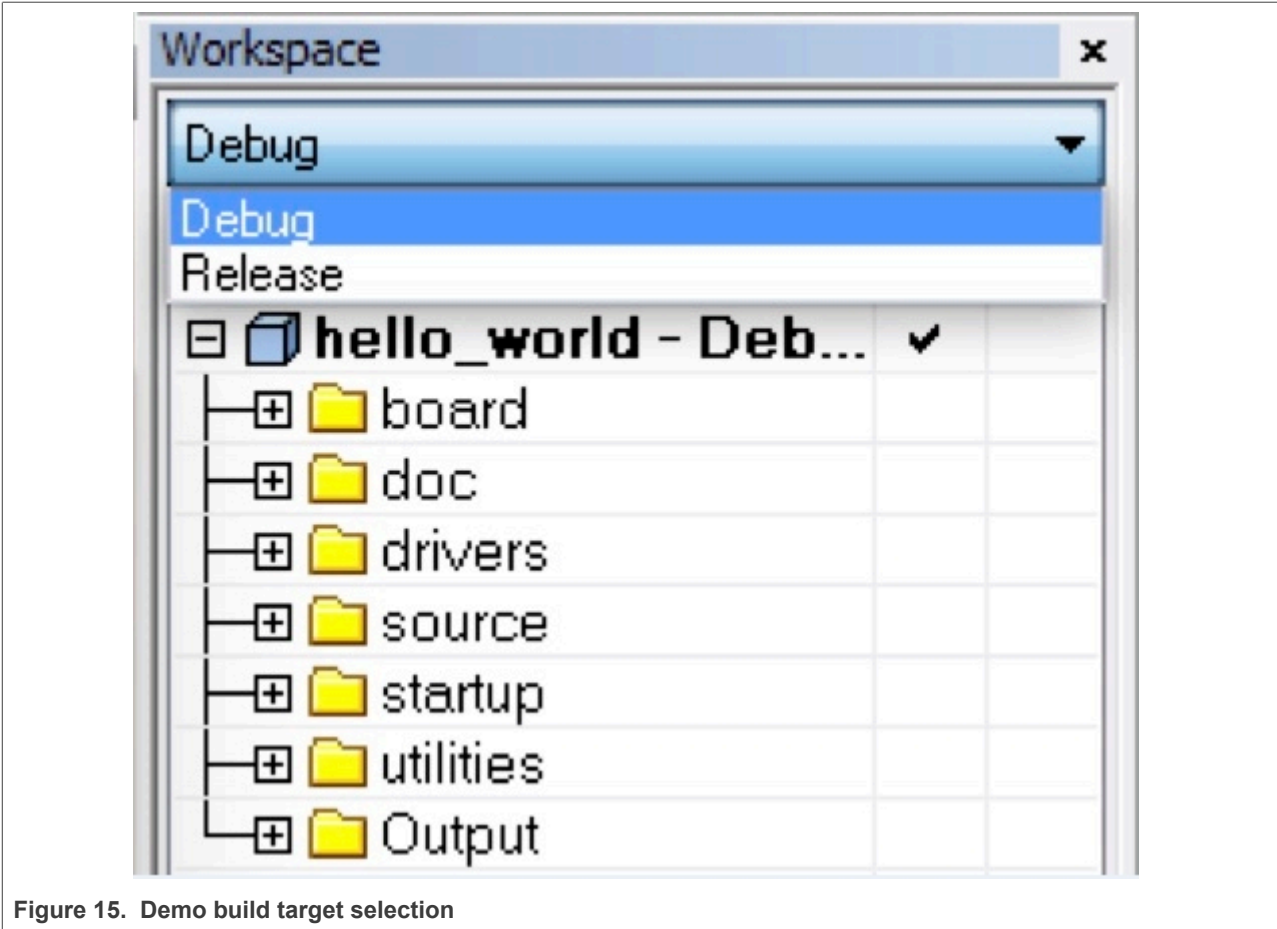


Figure 15. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in [Figure 16](#).

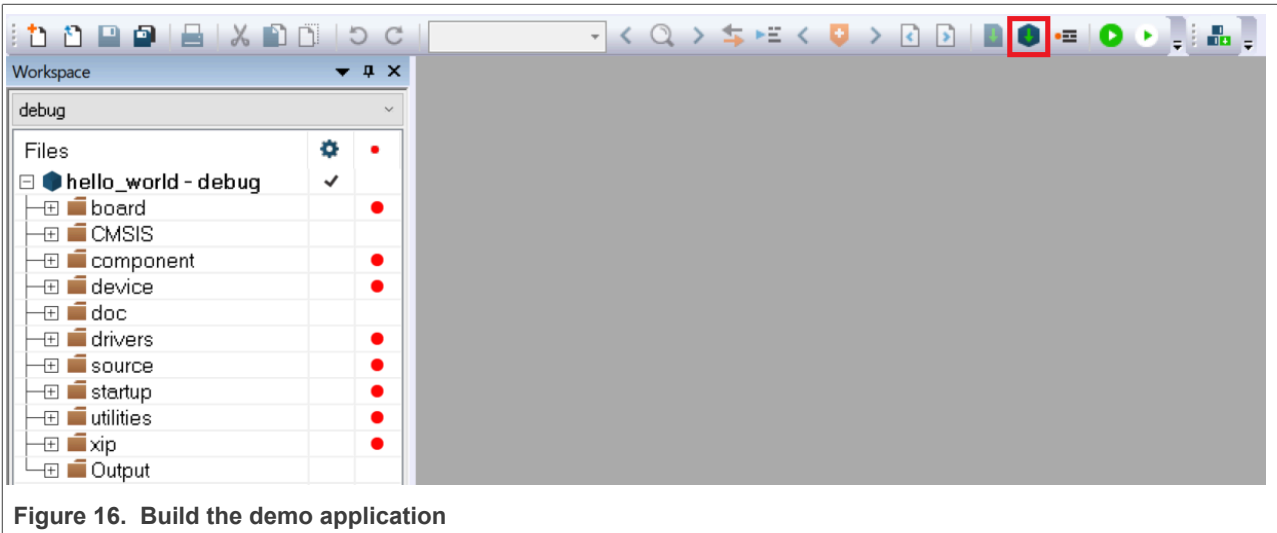


Figure 16. Build the demo application

4. The build completes without errors.

4.2 Run an example application

To download and run the application, perform these steps:

1. See the table in [Section 10](#) to determine the debug interface that comes loaded on your specific hardware platform.
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 COM port (to determine the COM port number, see [Section 8](#)). Configure the terminal with these settings:
 - a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit

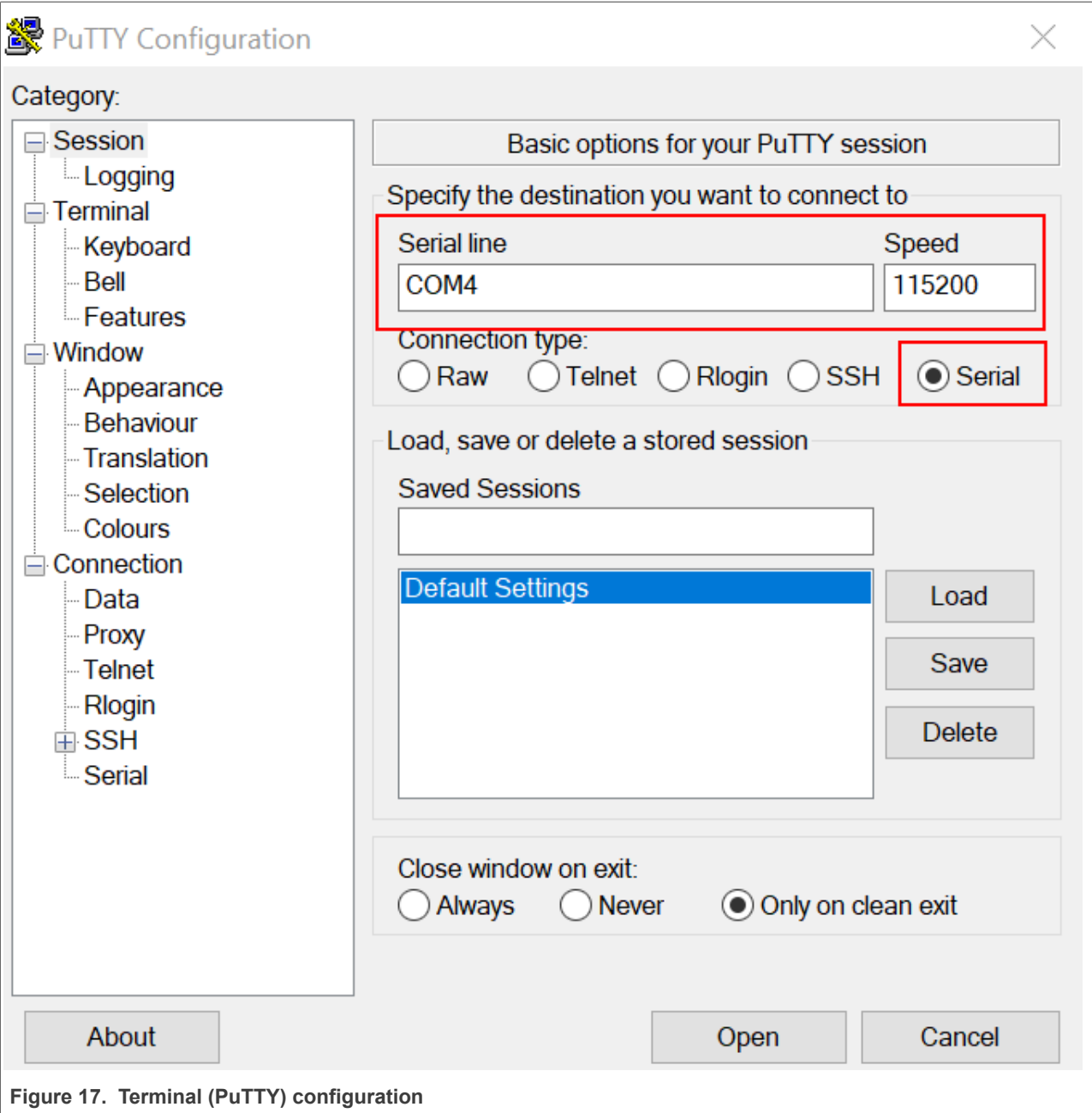


Figure 17. Terminal (PuTTY) configuration

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

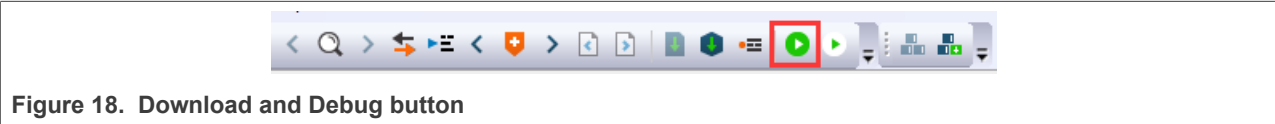


Figure 18. Download and Debug button

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



Figure 19. Stop at main() when running debugging

6. Run the code by clicking the **Go** button.



Figure 20. Go button

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



Figure 21. Text display of the hello_world demo

4.3 IAR RAM debugging notes

In the SDK examples, there are many build configurations. For example, debug, release, flash_debug, and flash_release. The debug/release configuration demonstrates the application running in RAM, while flash_debug/flash_release runs in the flash (XIP).

XIP debugging is clean with the "boot from flash" setting on the board because the debugging has the same sequence as a normal boot.

The debugging steps are as follows:

1. The debugger downloads the image to the flash.
2. The debugger resets the device and boots from ROM.
3. The debugger halts the device and waits for the command from the user.

However, with RAM debugging with the *boot from flash* setting on the board, the situation is complicated.

1. The debugger downloads the image to the RAM.
2. The debugger resets the device and boots it from ROM.
Note: *The image downloaded for debugging and the image running from the flash are different. The images are different because the image in flash might be written by the previous debugging.*
3. The debugger halts the device and sets the PC register value to the entry address of the downloaded image.

The problem here is that step 2 might overwrite the RAM written in step 1. The other issue is that the hardware status is undetermined when PC is set to the start of the RAM application.

To avoid the RAM debugging issue, MCUX IDE/armgcc/MDK provides a plain load feature. To get a clean debugging status, the RAM image is downloaded to the flash.

IAR needs a special setting to achieve the plain load feature as follows:

1. Right-click the project and select **Options**.

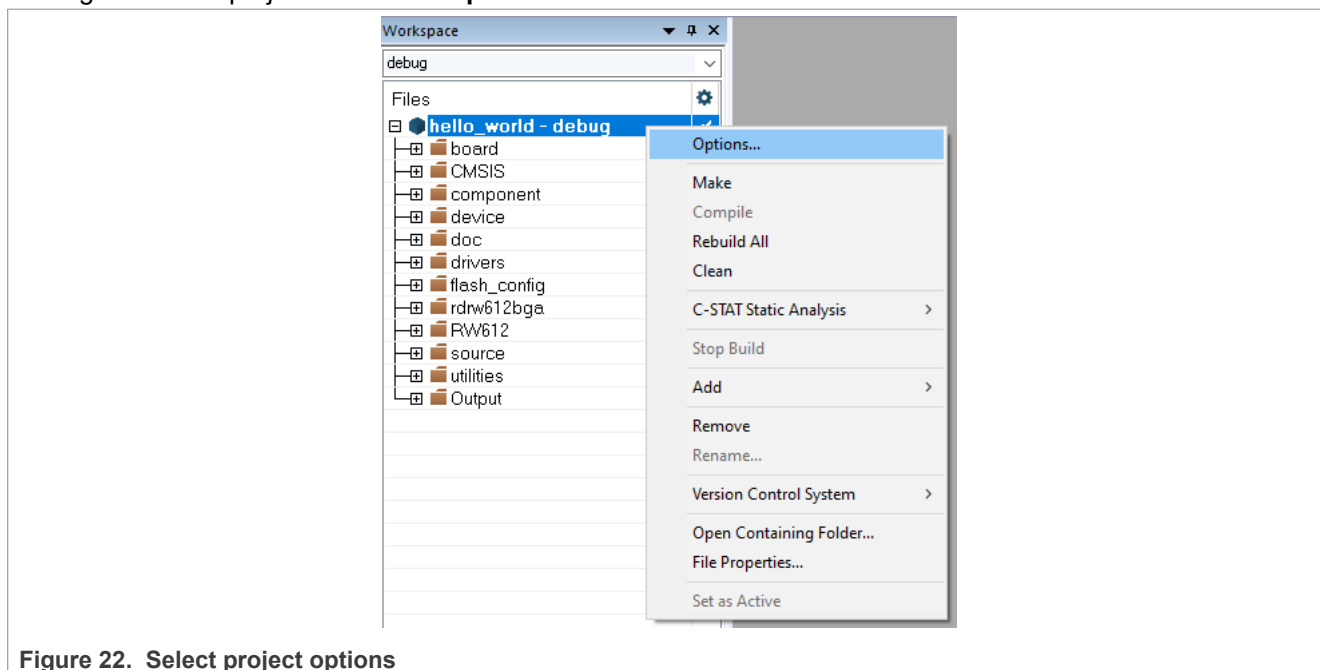


Figure 22. Select project options

2. Select **Debugger > Download** in the **Options for node "<project_name>"** dialog box.
3. Clear the **Verify download** checkbox because the download address is different from the real address.
4. Select the **Override default.board file** option.
5. Copy the default *.board file to the IAR workspace folder.
6. Change the *.board file path.
7. Click the **Edit** button.

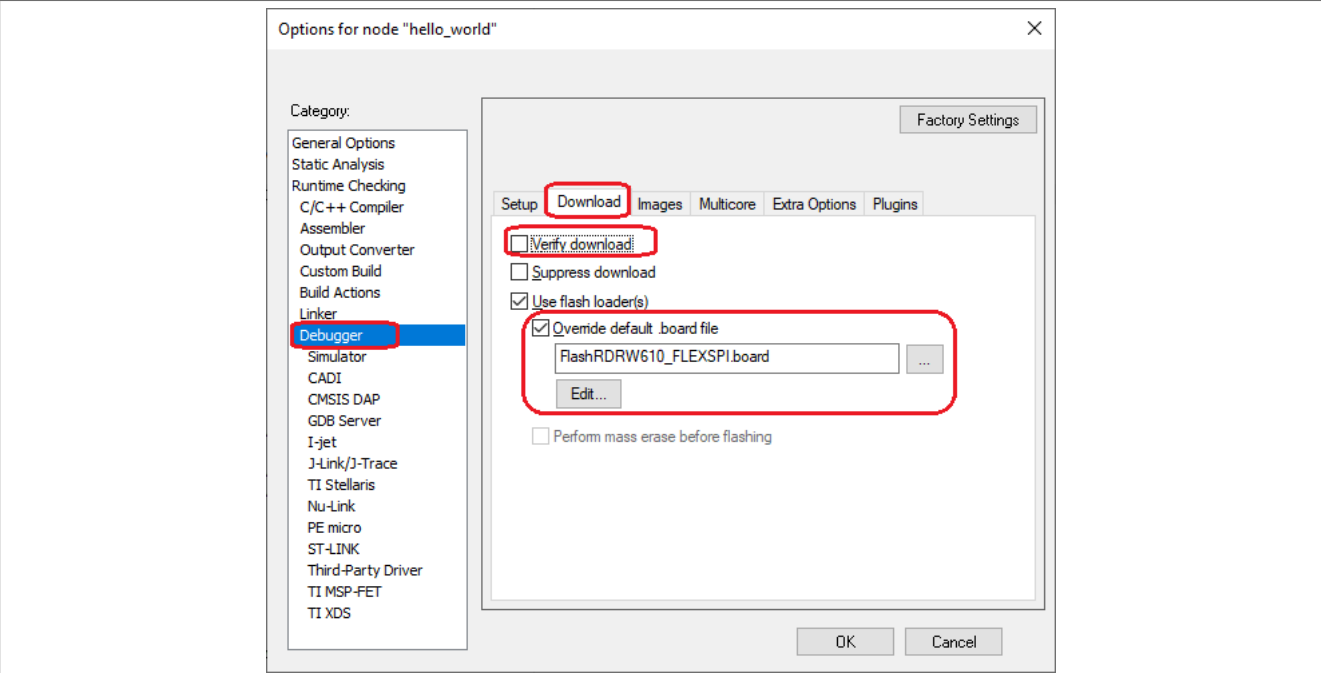


Figure 23. Edit debugger options

8. Create a range for RAM.

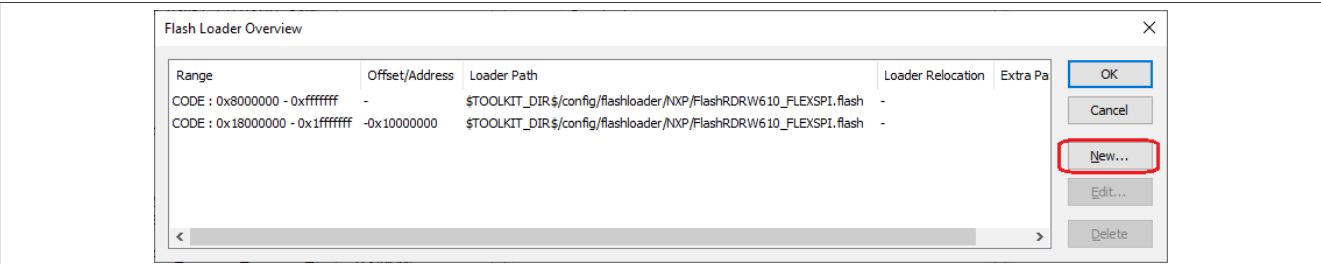


Figure 24. Flashloader overview

9. Set the memory range, relocate offset, and flashloader path.

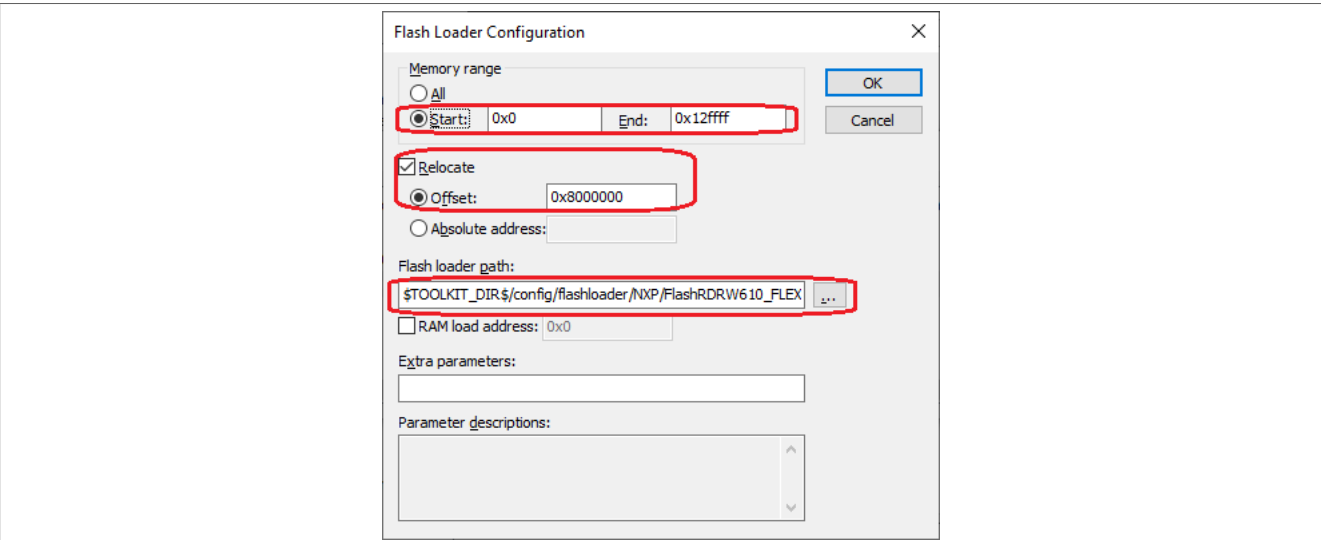


Figure 25. Configure flashloader

The new RAM range appears in the flashloader overview.

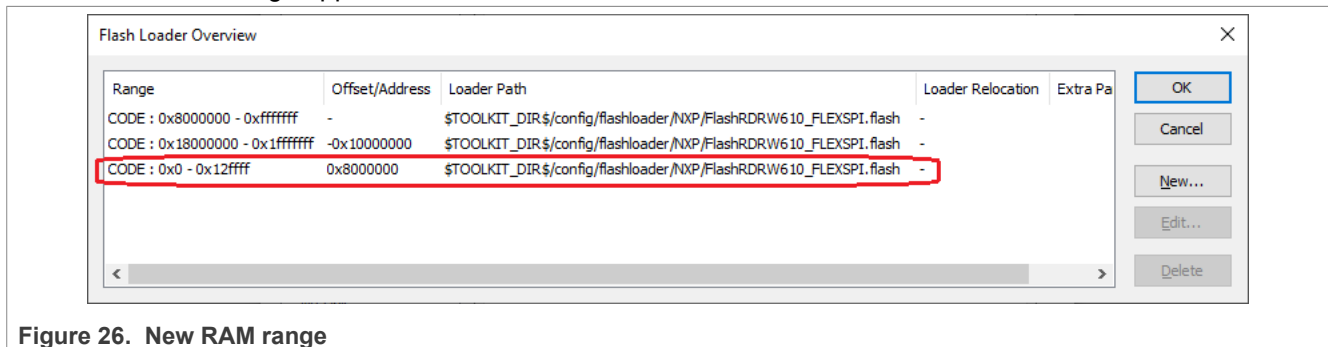


Figure 26. New RAM range

- Follow the instructions in [Section 4.2](#) and now it is safe to debug the RAM application. As a result, the downloaded RAM application can also boot on reset.

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 RD-RW61X hardware platform which is used as an example.

Note: ARMGCC version 8-2019-q3 is used as an example in this document. The latest GCC version for this package is as described in the MCUXpresso SDK Release Notes Supporting RD-RW61X (document MCUXSDKRDRW61XRN). IAR/Segger officially does not support RW61x. Therefore, contact the support team to get `iar_segger_support_patch_rw610_prc.zip` and install the JLink patch before debugging with gdb.

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 GNU Arm Embedded Toolchain. 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*.

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.

- Download the latest MinGW mingw-get-setup installer from [MinGW](#).
- 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.

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

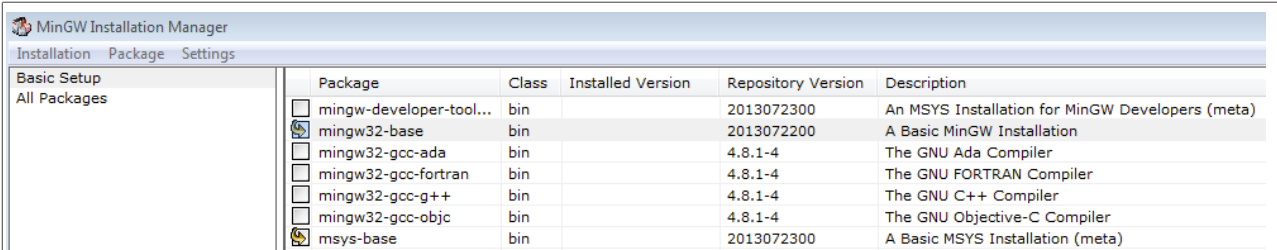


Figure 27. Set up MinGW and MSYS

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

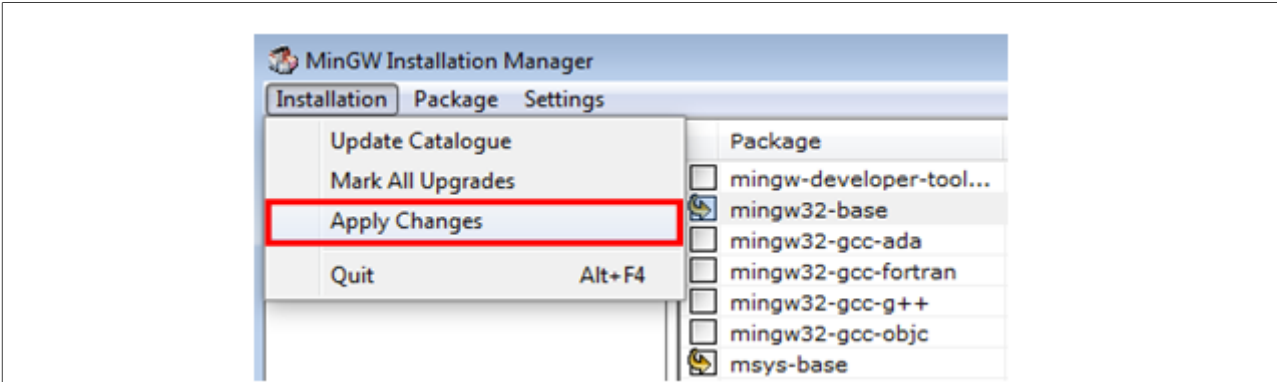


Figure 28. Complete MinGW and MSYS installation

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.

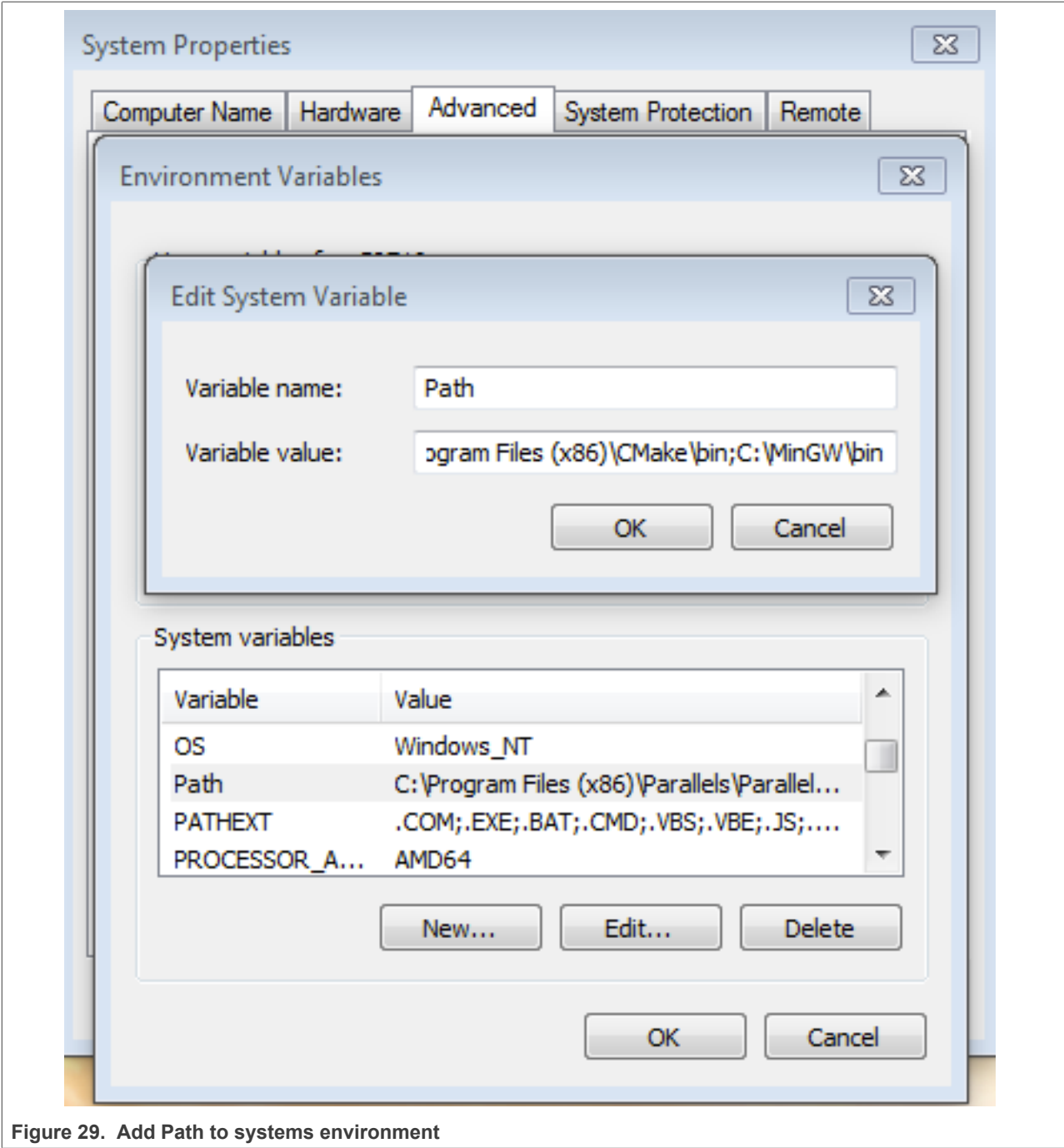


Figure 29. Add Path to systems environment

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 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.

```
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 30. Convert path to short path

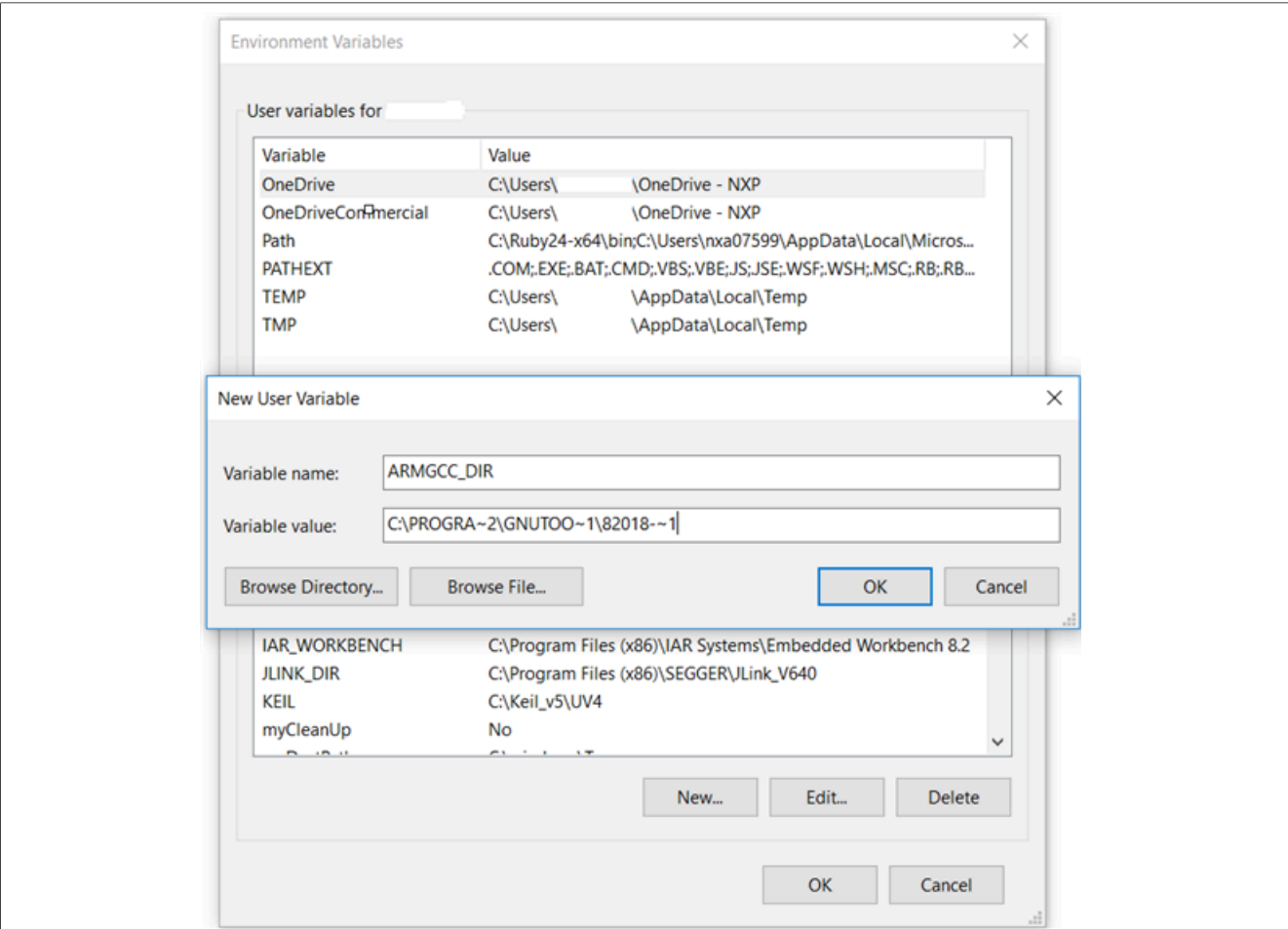


Figure 31. Add ARMGCC_DIR system variable

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.

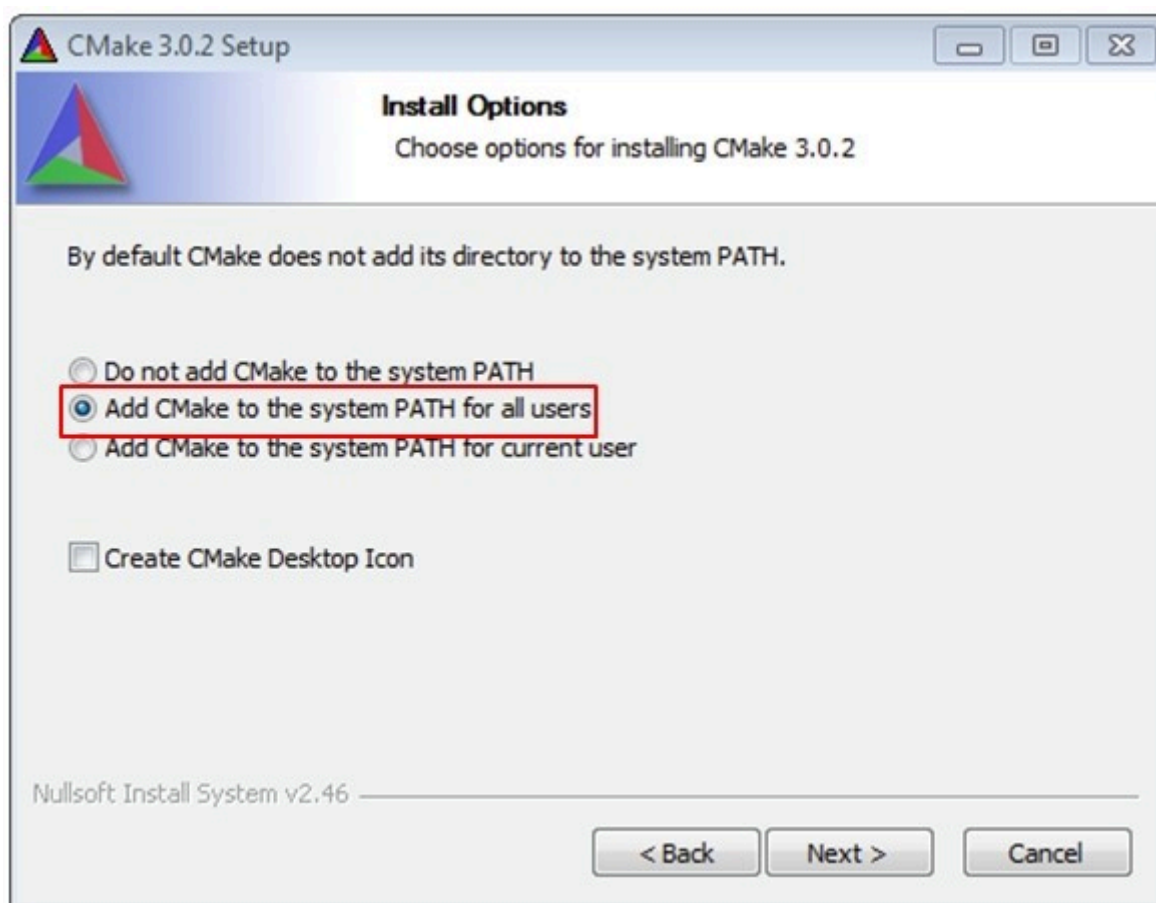


Figure 32. Install CMake

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**.

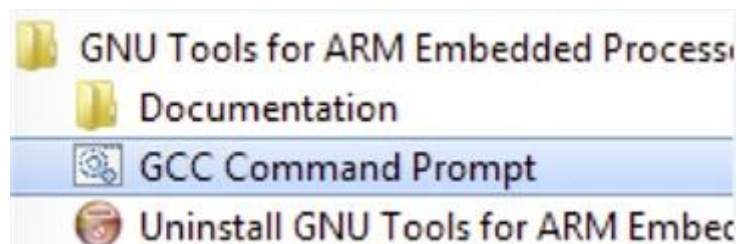


Figure 33. Launch command prompt

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/rdrw610/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 34](#).

```
[ 90%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2_11_0_RDRW610/boards/rdrw610/demo_apps/hello_world/armgcc/hello_world.c.o
[ 95%]
Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2_11_0_RDRW610/boards/rdrw610/demo_apps/hello_world/armgcc/hello_world.c.o
[100%] Linking C executable debug\hello_world.elf
Memory region      Used Size  Region Size  %age Used
  m_interrupts:      580 B      640 B      90.63%
    m_text:        16768 B    785792 B      2.13%
    m_data:         2648 B     448 KB      0.58%
[100%] Built target hello_world.elf
```

Figure 34. hello_world build successful

5.3 Run an example application

This section describes steps to run a demo application using J-Link GDB Server application. To update the on-board LPC-Link2 debugger to Jlink firmware, see [Section 11](#).

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

1. Connect the development platform to your PC via USB cable between the LPC-Link2 USB connector and the PC USB connector. If using a standalone J-Link debug pod, connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see [Section 8](#)). 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

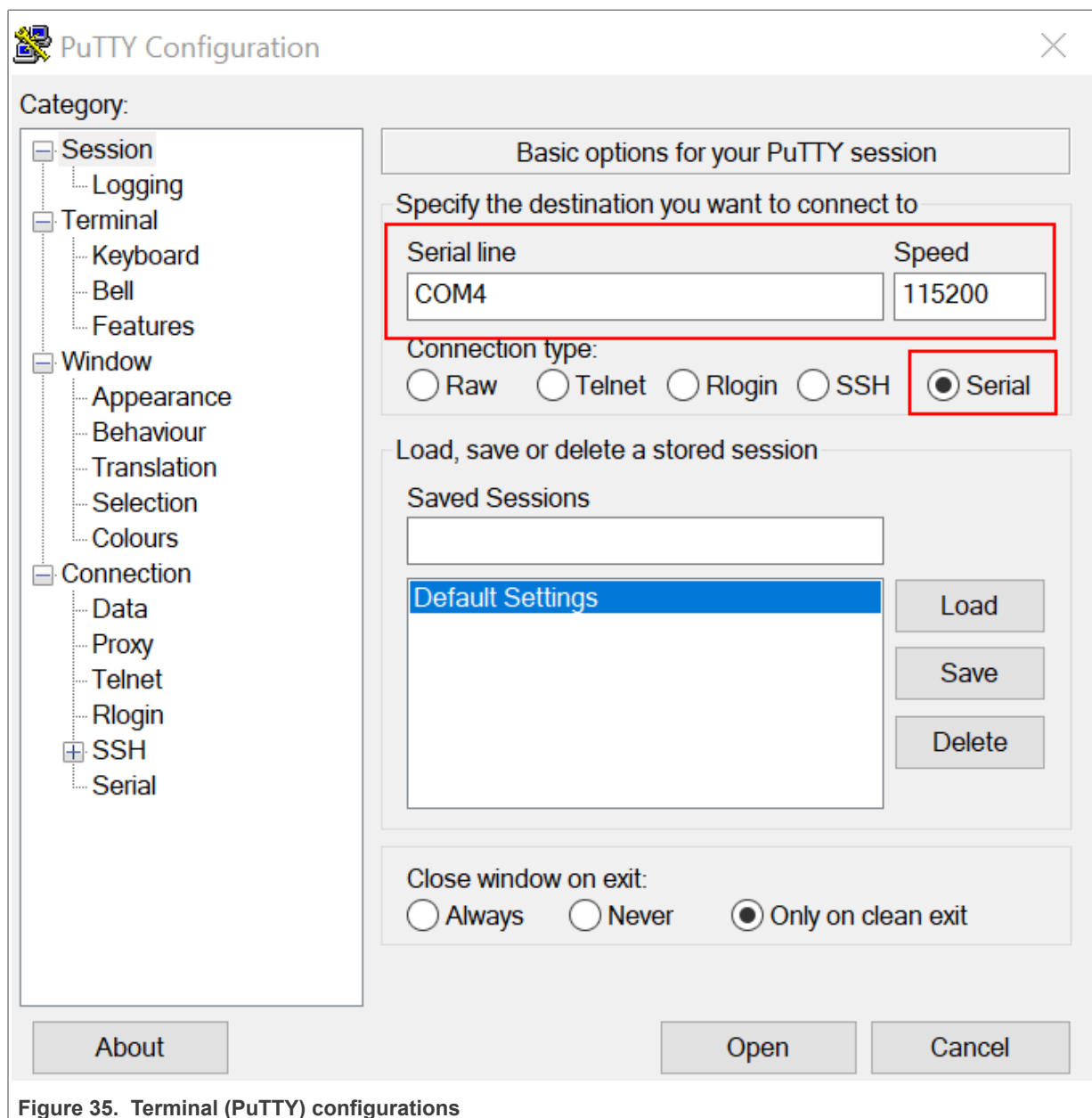


Figure 35. Terminal (PuTTY) configurations

Note: Before debugging, it is recommend that the board is set to the FlexSPI flash boot mode (CON[3:0]: [off, off, off, off]). If there is no bootable image in a flash, set CON[3:0] to all "on" and download the image. Change the flash boot mode and gdb debug again.

3. Open the J-Link GDB Server application.
4. Select "RW610" as the target device and "SWD" as the target interface.
5. After it is connected, the screen appears as in [Figure 36](#).

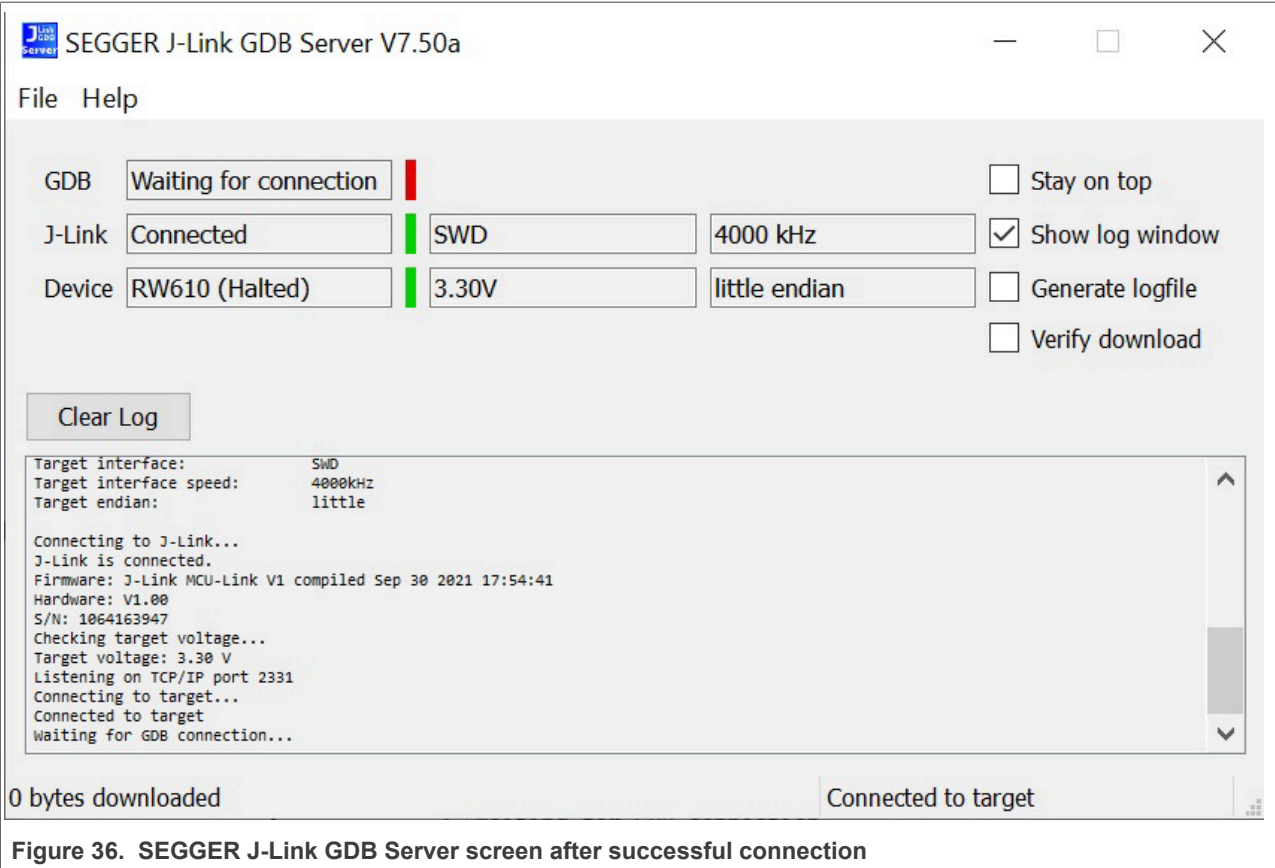


Figure 36. SEGGER J-Link GDB Server screen after successful connection

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

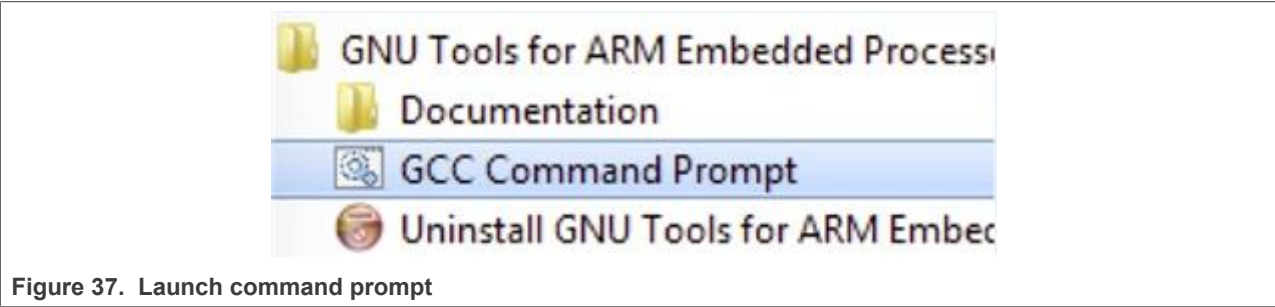


Figure 37. Launch command prompt

7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:

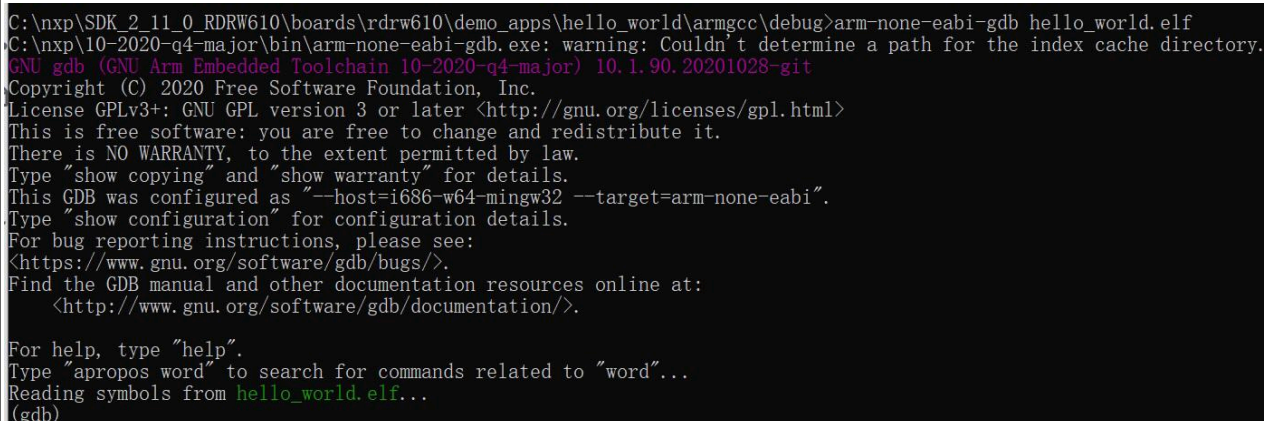
```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/  
debug
```

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

For this example, the path is:

```
<install_dir>/boards/rdrw61x/demo_apps/hello_world/armgcc/debug
```

8. Run the `arm-none-eabi-gdb.exe <application_name>.elf` command. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.



```
C:\nxp\SDK_2_11_0_RDRW610\boards\rdrw610\demo_apps\hello_world\armgcc\debug>arm-none-eabi-gdb hello_world.elf
C:\nxp\10-2020-q4-major\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 10-2020-q4-major) 10.1.90.20201028-git
Copyright (C) 2020 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:
<https://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 hello_world.elf...
(gdb)
```

Figure 38. Run `arm-none-eabi-gdb`

9. Run these commands:
 - a. `target remote localhost:2331`
 - b. `monitor reset`
 - c. `load`
 - d. `monitor reset`
10. The application is now downloaded and halted at the watch point. 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 does not appear, check your terminal settings and connections.

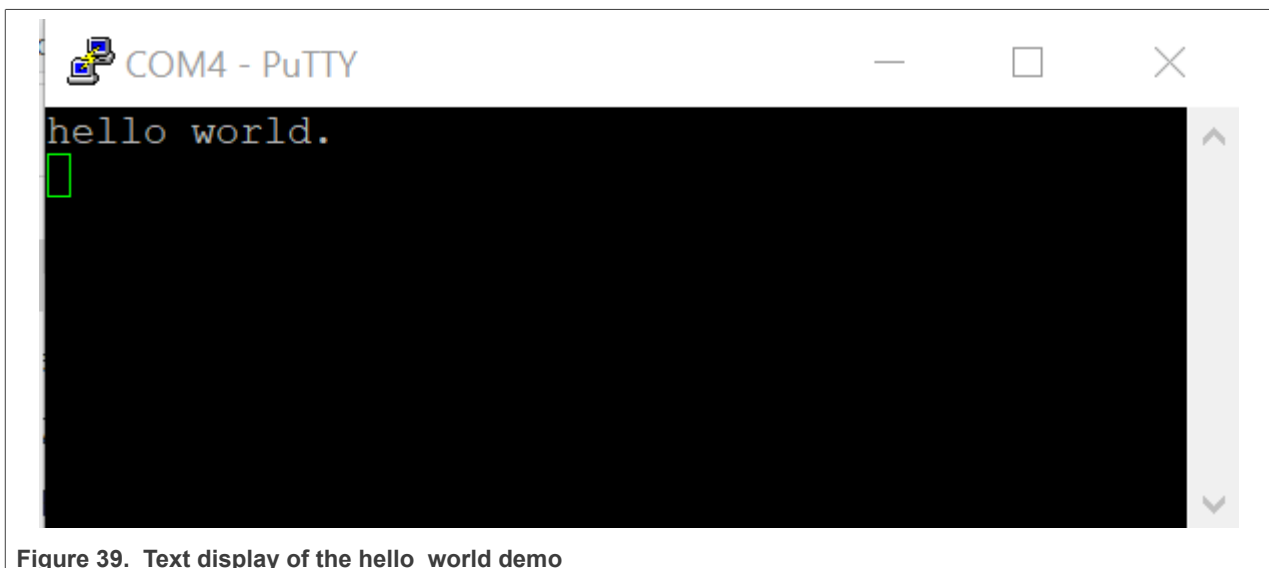


Figure 39. Text display of the `hello_world` demo

6 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.

6.1 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 RW612 CMSIS pack.

1. Download the RW612 pack.
2. After downloading the DFP, double click to install it.

6.2 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/rdrw612bga/demo_apps/hello_world/mdk/hello_world.uvmpw
```

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

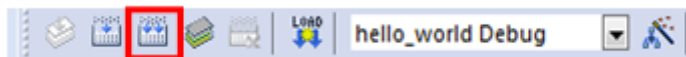


Figure 40. Build the demo

3. The build completes without errors.

6.3 Run an example application

To download and run the application, perform these steps:

1. Reference the table in [Section 10](#) to determine the debug interface that comes loaded on your specific hardware platform.
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 8](#)). Configure the terminal with these settings:
 - a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit

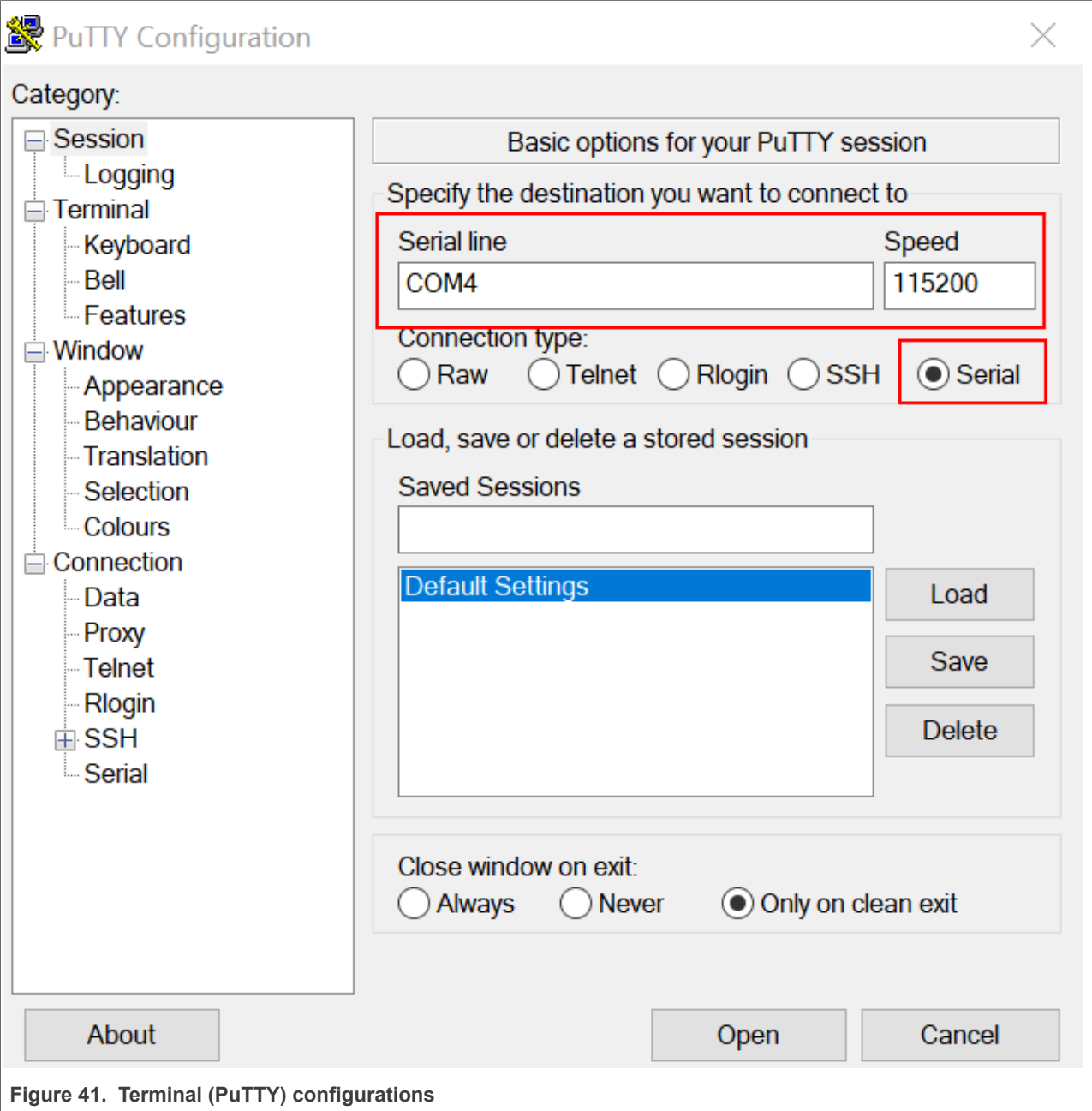


Figure 41. Terminal (PuTTY) configurations

4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button.

Note: Make sure that the board is set to FlexSPI flash boot mode (U38 all off) before using Keil debug.

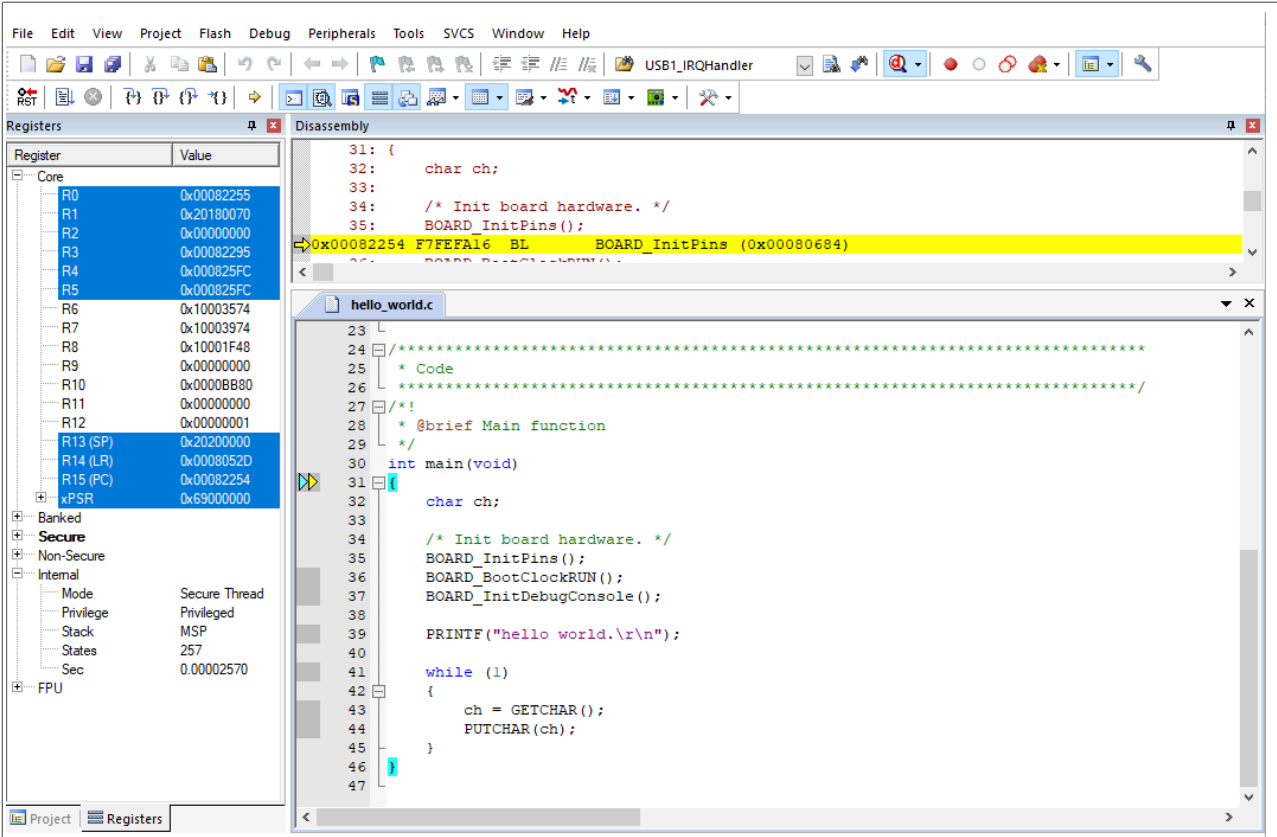


Figure 42. Stop at main () when run debugging

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

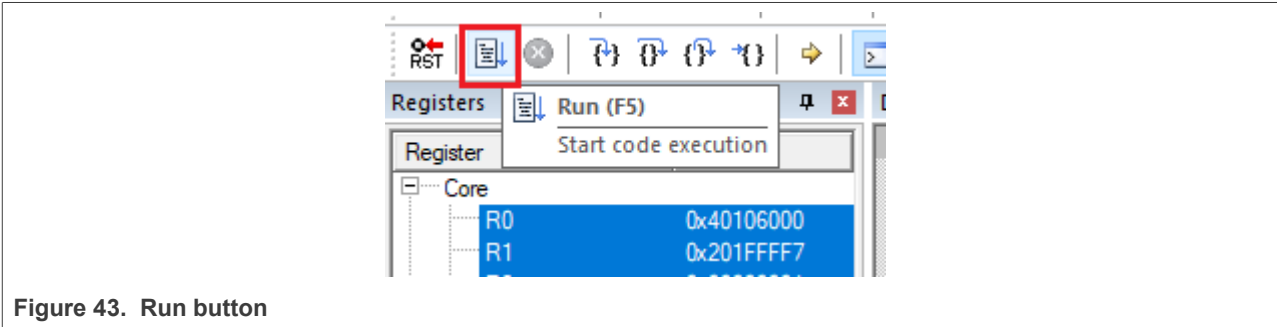


Figure 43. Run button

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.

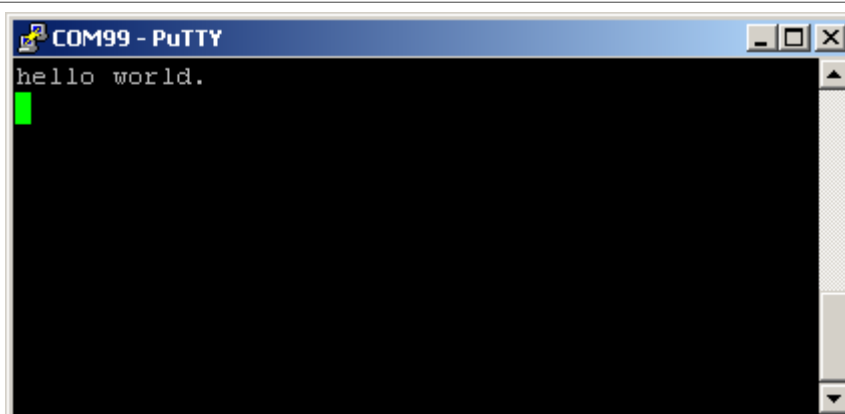


Figure 44. Text display of the `hello_world` demo

7 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 45](#).



Figure 45. 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.

8 How to determine COM port

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

1. **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.
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.

9 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
}
```

10 Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with a variety of factory programmed debug interface configurations. [Table 1](#) lists the hardware platforms supported by the MCUXpresso SDK, their default debug interface, and any version information that helps differentiate a specific interface configuration.

Note: The [OpenSDA details](#) column in [Table 1](#) is not applicable to LPC.

Table 1. Hardware platforms supported by SDK

Hardware platform	Default interface	OpenSDA details
EVK-MC56F83000	P&E Micro OSJTAG	N/A
EVK-MIMXRT595	CMSIS-DAP	N/A
EVK-MIMXRT685	CMSIS-DAP	N/A
FRDM-K22F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-K28F	DAPLink	OpenSDA v2.1
FRDM-K32L2A4S	CMSIS-DAP	OpenSDA v2.1
FRDM-K32L2B	CMSIS-DAP	OpenSDA v2.1
FRDM-K32W042	CMSIS-DAP	N/A
FRDM-K64F	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
FRDM-K66F	J-Link OpenSDA	OpenSDA v2.1
FRDM-K82F	CMSIS-DAP	OpenSDA v2.1
FRDM-KE15Z	DAPLink	OpenSDA v2.1
FRDM-KE16Z	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.2
FRDM-KL02Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL03Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL25Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL26Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL27Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL28Z	P&E Micro OpenSDA	OpenSDA v2.1
FRDM-KL43Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL46Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KL81Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KL82Z	CMSIS-DAP	OpenSDA v2.0
FRDM-KV10Z	CMSIS-DAP	OpenSDA v2.1
FRDM-KV11Z	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KV31F	P&E Micro OpenSDA	OpenSDA v1.0
FRDM-KW24	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.0
HVP-KE18F	DAPLink	OpenSDA v2.2
HVP-KV46F150M	P&E Micro OpenSDA	OpenSDA v1
HVP-KV11Z75M	CMSIS-DAP	OpenSDA v2.1
HVP-KV58F	CMSIS-DAP	OpenSDA v2.1
HVP-KV31F120M	P&E Micro OpenSDA	OpenSDA v1
JN5189DK6	CMSIS-DAP	N/A

Table 1. Hardware platforms supported by SDK...continued

Hardware platform	Default interface	OpenSDA details
LPC54018 IoT Module	N/A	N/A
LPCXpresso54018	CMSIS-DAP	N/A
LPCXpresso54102	CMSIS-DAP	N/A
LPCXpresso54114	CMSIS-DAP	N/A
LPCXpresso51U68	CMSIS-DAP	N/A
LPCXpresso54608	CMSIS-DAP	N/A
LPCXpresso54618	CMSIS-DAP	N/A
LPCXpresso54628	CMSIS-DAP	N/A
LPCXpresso54S018M	CMSIS-DAP	N/A
LPCXpresso55s16	CMSIS-DAP	N/A
LPCXpresso55s28	CMSIS-DAP	N/A
LPCXpresso55s69	CMSIS-DAP	N/A
MAPS-KS22	J-Link OpenSDA	OpenSDA v2.0
MIMXRT1170-EVK	CMSIS-DAP	N/A
RD-RW612-BGA	CMSIS-DAP	N/A
RD-RW612-QFN	CMSIS-DAP	N/A
TWR-K21D50M	P&E Micro OSJTAG	N/A
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K24F120M	CMSIS-DAP/mbed	OpenSDA v2.1
TWR-K60D100M	P&E Micro OSJTAG	N/A
TWR-K64D120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K64F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K65D180M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K80F150M	CMSIS-DAP	OpenSDA v2.1
TWR-K81F150M	CMSIS-DAP	OpenSDA v2.1
TWR-KE18F	DAPLink	OpenSDA v2.1
TWR-KL28Z72M	P&E Micro OpenSDA	OpenSDA v2.1
TWR-KL43Z48M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KL81Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KL82Z72M	CMSIS-DAP	OpenSDA v2.0
TWR-KM34Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KM35Z75M	DAPLink	OpenSDA v2.2
TWR-KV10Z32	P&E Micro OpenSDA	OpenSDA v1.0

Table 1. Hardware platforms supported by SDK...continued

Hardware platform	Default interface	OpenSDA details
TWR-KV11Z75M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV31F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV46F150M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-KV58F220M	CMSIS-DAP	OpenSDA v2.1
TWR-KW24D512	P&E Micro OpenSDA	OpenSDA v1.0
USB-KW24D512	N/A External probe	N/A
USB-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater

11 Updating debugger firmware

11.1 Updating OpenSDA firmware

Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. For reference, OpenSDA firmware files can be found at the links below:

- **J-Link:** Download appropriate image from www.segger.com/opensda.html. Choose the appropriate J-Link binary based on the table in [Table 1](#). Any OpenSDA v1.0 interface should use the standard OpenSDA download (in other words, the one with no version). For OpenSDA 2.0 or 2.1, select the corresponding binary.
- **P&E Micro:** Downloading P&E Micro OpenSDA firmware images requires registration with P&E Micro (www.pemicro.com).

Perform the following steps to update the OpenSDA firmware on your board for Windows and Linux OS users:

1. Unplug the board's USB cable.
2. Press the **Reset** button on the board. While still holding the button, plug the USB cable back into the board.
3. When the board re-enumerates, it shows up as a disk drive called **MAINTENANCE**.

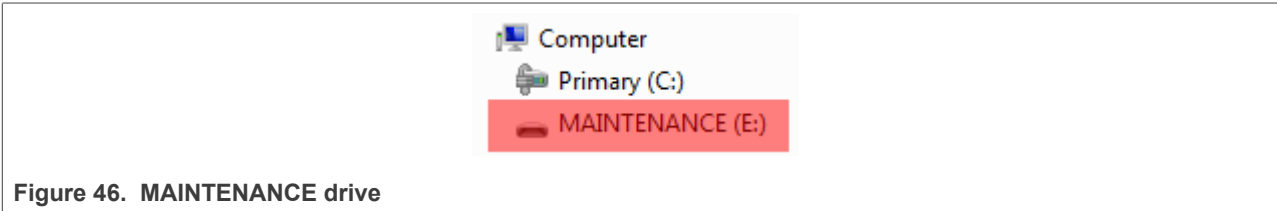


Figure 46. MAINTENANCE drive

4. Drag and drop the new firmware image onto the MAINTENANCE drive.
Note: If for any reason the firmware update fails, the board can always re-enter maintenance mode by holding down **Reset** button and power cycling.

These steps show how to update the OpenSDA firmware on your board for Mac OS users.

1. Unplug the board's USB cable.
2. Press the **Reset** button of the board. While still holding the button, plug the USB cable back into the board.
3. For boards with OpenSDA v2.0 or v2.1, it shows up as a disk drive called **BOOTLOADER** in **Finder**. Boards with OpenSDA v1.0 may or may not show up depending on the bootloader version. If you see the drive in **Finder**, proceed to the next step. If you do not see the drive in Finder, use a PC with Windows OS 7 or an earlier version to either update the OpenSDA firmware, or update the OpenSDA bootloader to version 1.11 or later. The bootloader update instructions and image can be obtained from P&E Microcomputer website.

- 4. For OpenSDA v2.1 and OpenSDA v1.0 (with bootloader 1.11 or later) users, drag the new firmware image onto the BOOTLOADER drive in **Finder**.
- 5. For OpenSDA v2.0 users, type these commands in a Terminal window:

```
> sudo mount -u -w -o sync /Volumes/BOOTLOADER
> cp -X <path to update file> /Volumes/BOOTLOADER
```

Note: If for any reason the firmware update fails, the board can always re-enter bootloader mode by holding down the **Reset** button and power cycling.

11.2 Updating MCU-Link firmware

The LPCXpresso and RW61X hardware platform comes with a CMSIS-DAP-compatible debug interface (known as LPC-Link2). This firmware in this debug interface may be updated using the host computer utility called MCU-Link. This typically used when switching between the default debugger protocol (CMSIS-DAP) to SEGGER J-Link, or for updating this firmware with new releases of these. This section contains the steps to re-program the debug probe firmware.

Note: If MCUXpresso IDE is used and the jumper making DFULink is installed on the board (JP5 on some boards, but consult the board user manual or schematic for specific jumper number), LPC-Link2 debug probe boots to DFU mode, and MCUXpresso IDE automatically downloads the CMSIS-DAP firmware to the probe before flash memory programming (after clicking **Debug**). Using DFU mode ensures most up-to-date/compatible firmware is used with MCUXpresso IDE.

NXP provides the MCU-Link utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto RD-RW61X or LPCXpresso boards. The utility can be downloaded from <https://www.nxp.com/design/microcontrollers-developer-resources/mcu-link-debug-probe:MCU-LINK>.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in the MCU-Link user guide (<https://www.nxp.com/design/microcontrollers-developer-resources/mcu-link-debug-probe:MCU-LINK>, select the documentation tab).

- 1. Install the MCU-Link utility.
- 2. Unplug the board's USB cable.
- 3. Make the DFU link (install the jumper labelled DFULink, JP10 on RD-RW61X-BGA board).
- 4. Connect the probe to the host via USB (use Link USB connector).
- 5. Open a command shell and call the appropriate script located in the MCU-Link installation directory (<MCU-Link install dir>).
 - a. To program CMSIS-DAP debug firmware: <LPCScript install dir>/scripts/program_CMSIS
 - b. To program J-Link debug firmware: <LPCScript install dir>/scripts/program_JLINK
- 6. Remove DFU link (remove the jumper installed in [Step 3](#)).
- 7. Re-power the board by removing the USB cable and plugging it in again.

12 Revision history

This table summarizes revisions to this document.

Table 2. Revision history

Document ID	Release Date	Description
MCUXSDKRDRW61XGSUG v2.16.000	17 June 2024	Initial release for MCUXpresso SDK 2.16.000

Legal information

Definitions

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

Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

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

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

Trademarks

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

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

Tables

Tab. 1.	Hardware platforms supported by SDK	35	Tab. 2.	Revision history	38
---------	---	----	---------	------------------------	----

Figures

Fig. 1.	MCUXpresso SDK layers	2	Fig. 25.	Configure flashloader	20
Fig. 2.	Application folder structure	3	Fig. 26.	New RAM range	21
Fig. 3.	Install an SDK	4	Fig. 27.	Set up MinGW and MSYS	22
Fig. 4.	Import an SDK example	5	Fig. 28.	Complete MinGW and MSYS installation	22
Fig. 5.	Select RD-RW610-BGA board	6	Fig. 29.	Add Path to systems environment	23
Fig. 6.	Select hello_world	7	Fig. 30.	Convert path to short path	24
Fig. 7.	Select Use floating point version of printf	8	Fig. 31.	Add ARMGCC_DIR system variable	24
Fig. 8.	Terminal (PuTTY) configurations	9	Fig. 32.	Install CMake	25
Fig. 9.	Debug hello_world case	10	Fig. 33.	Launch command prompt	25
Fig. 10.	Attached Probes: debug emulator selection	11	Fig. 34.	hello_world build successful	26
Fig. 11.	Disabling Reset before running	12	Fig. 35.	Terminal (PuTTY) configurations	27
Fig. 12.	Stop at main() when running debugging	13	Fig. 36.	SEGGER J-Link GDB Server screen after successful connection	28
Fig. 13.	Resume button	13	Fig. 37.	Launch command prompt	28
Fig. 14.	Text display of the hello_world demo	14	Fig. 38.	Run arm-none-eabi-gdb	29
Fig. 15.	Demo build target selection	15	Fig. 39.	Text display of the hello_world demo	29
Fig. 16.	Build the demo application	15	Fig. 40.	Build the demo	30
Fig. 17.	Terminal (PuTTY) configuration	17	Fig. 41.	Terminal (PuTTY) configurations	31
Fig. 18.	Download and Debug button	17	Fig. 42.	Stop at main() when run debugging	32
Fig. 19.	Stop at main() when running debugging	18	Fig. 43.	Run button	32
Fig. 20.	Go button	18	Fig. 44.	Text display of the hello_world demo	33
Fig. 21.	Text display of the hello_world demo	18	Fig. 45.	MCUXpresso IDE Quickstart Panel	33
Fig. 22.	Select project options	19	Fig. 46.	MAINTENANCE drive	37
Fig. 23.	Edit debugger options	20			
Fig. 24.	Flashloader overview	20			

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 using MCUXpresso IDE 4

3.1 Select the workspace location 4

3.2 Build an example application 4

3.3 Run an example application 8

4 Run a demo application using IAR 14

4.1 Build an example application 14

4.2 Run an example application 16

4.3 IAR RAM debugging notes 18

5 Run a demo using Arm GCC 21

5.1 Set up toolchain 21

5.1.1 Install GCC Arm Embedded tool chain 21

5.1.2 Install MinGW (only required on Windows
OS) 21

5.1.3 Add a new system environment variable for
ARMGCC_DIR 23

5.1.4 Install CMake 24

5.2 Build an example application 25

5.3 Run an example application 26

6 Run a demo using Keil MDK/µVision 29

6.1 Install CMSIS device pack 30

6.2 Build an example application 30

6.3 Run an example application 30

7 MCUXpresso IDE New Project Wizard 33

8 How to determine COM port 34

9 How to define IRQ handler in CPP files 34

10 Default debug interfaces 34

11 Updating debugger firmware 37

11.1 Updating OpenSDA firmware 37

11.2 Updating MCU-Link firmware 38

12 Revision history 38

Legal information 39

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