MCUXSDKKE16GSUG

Getting Started with MCUXpresso SDK for FRDM-KE16Z

Rev. 2.10.0 — 10 July 2021

User Guide

Contents

1 Overview

The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease and help accelerate embedded system development of applications based on general purpose, crossover and Bluetooth™-enabled MCUs from NXP. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. The MCUXpresso SDK contains optional RTOS integrations such as FreeRTOS and Azure RTOS, and various other middleware to support rapid development.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for FRDM-KE16Z* (document MCUXSDKKE16RN).

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

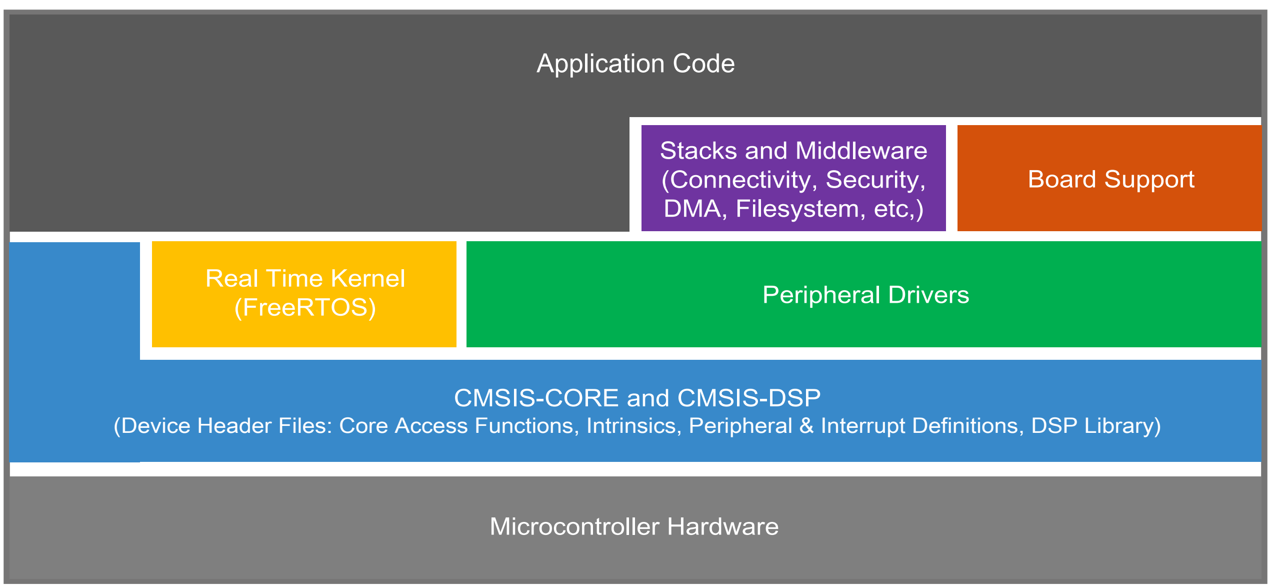

Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support package folders

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

- `cmsis_driver_examples`: Simple applications intended to show how to use CMSIS drivers.
- `demo_apps`: Full-featured applications that highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- `driver_examples`: Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- `rtos_examples`: Basic FreeRTOS$^{\text{TM}}$ 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:

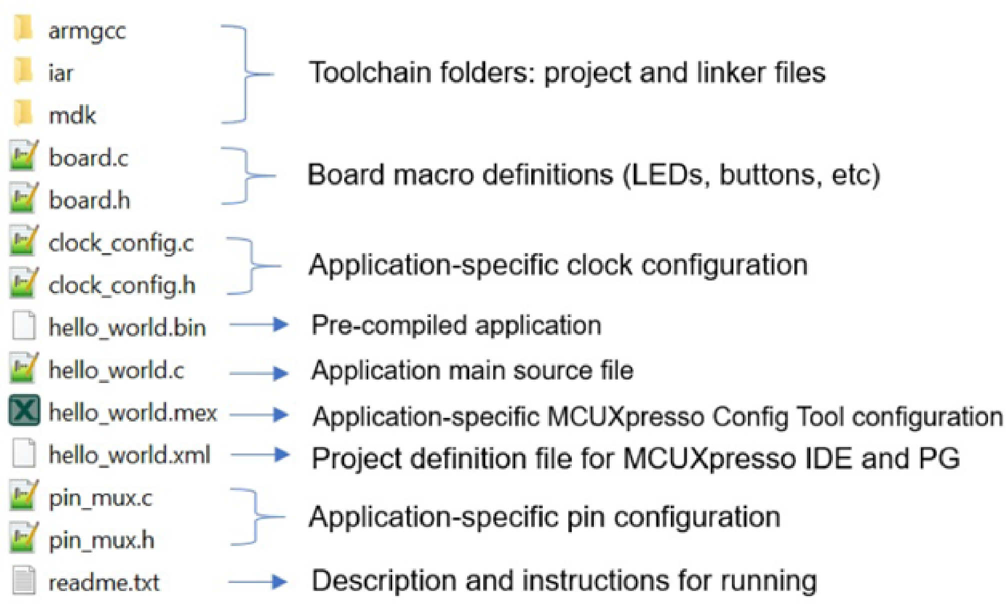

Figure 2. Application folder structure

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>/cmsis_drivers: All the CMSIS drivers for your specific MCU
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU
- devices/<device_name>/<tool_name>: Toolchain-specific startup code, including vector table definitions
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications
- devices/<devices_name>/project: Project template used in CMSIS PACK new project creation

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

3 Run a demo 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 FRDM-KE16Z 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.

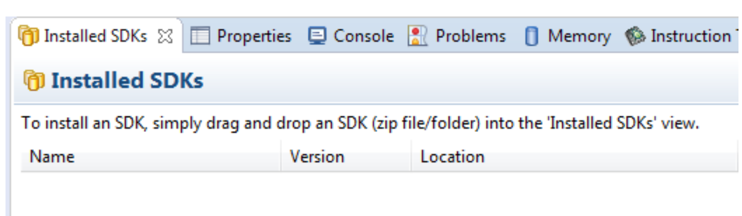

Figure 3. Install an SDK

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

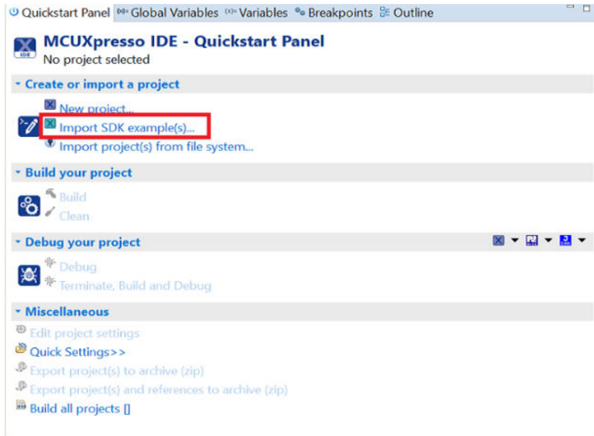

Figure 4. Import an SDK example

3. In the window that appears, expand the **KE1x** folder and select **MKE16Z64xxx4** . Then, select **frdmke16z** and click **Next**.

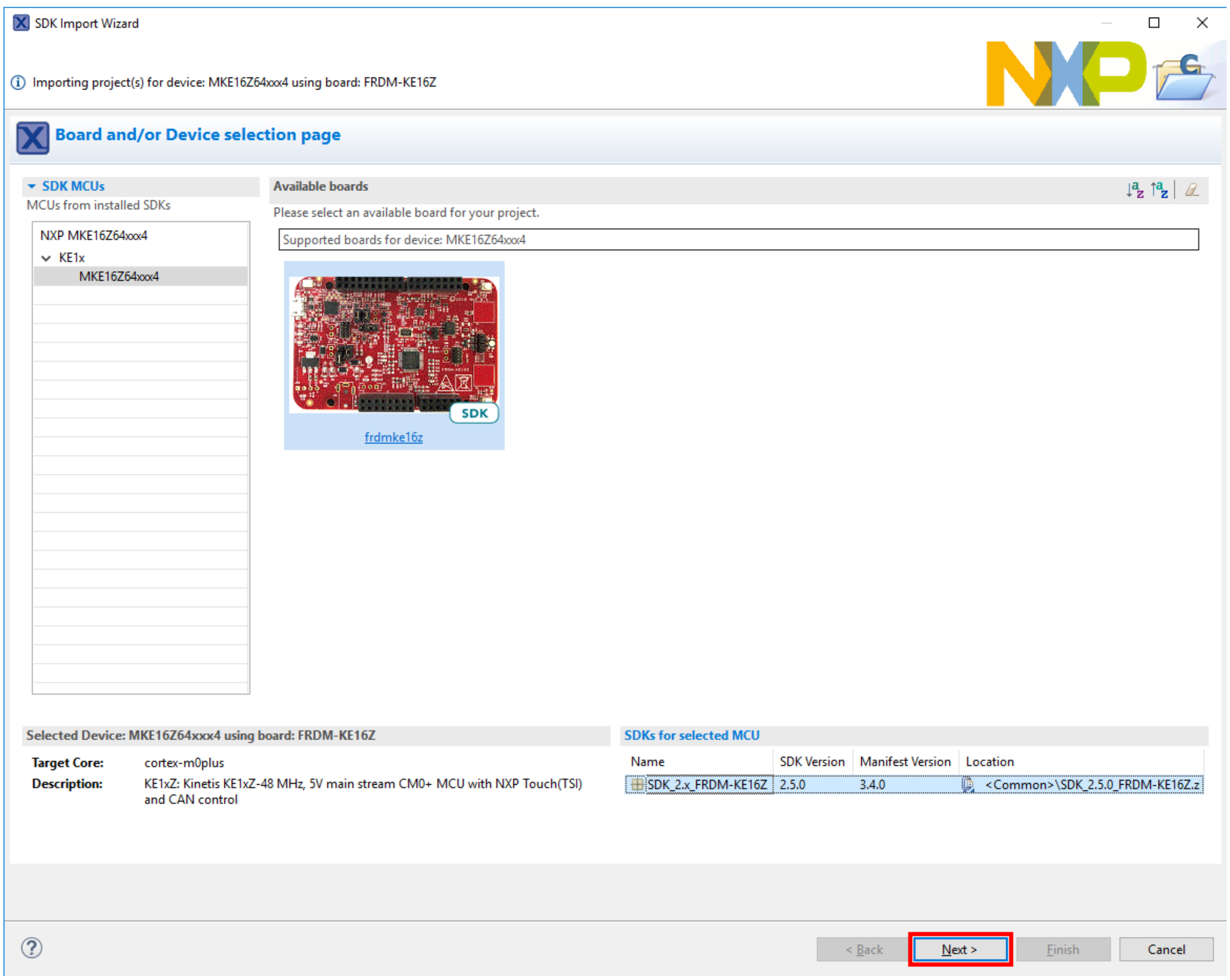

Figure 5. Select FRDM-KE16Z board

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

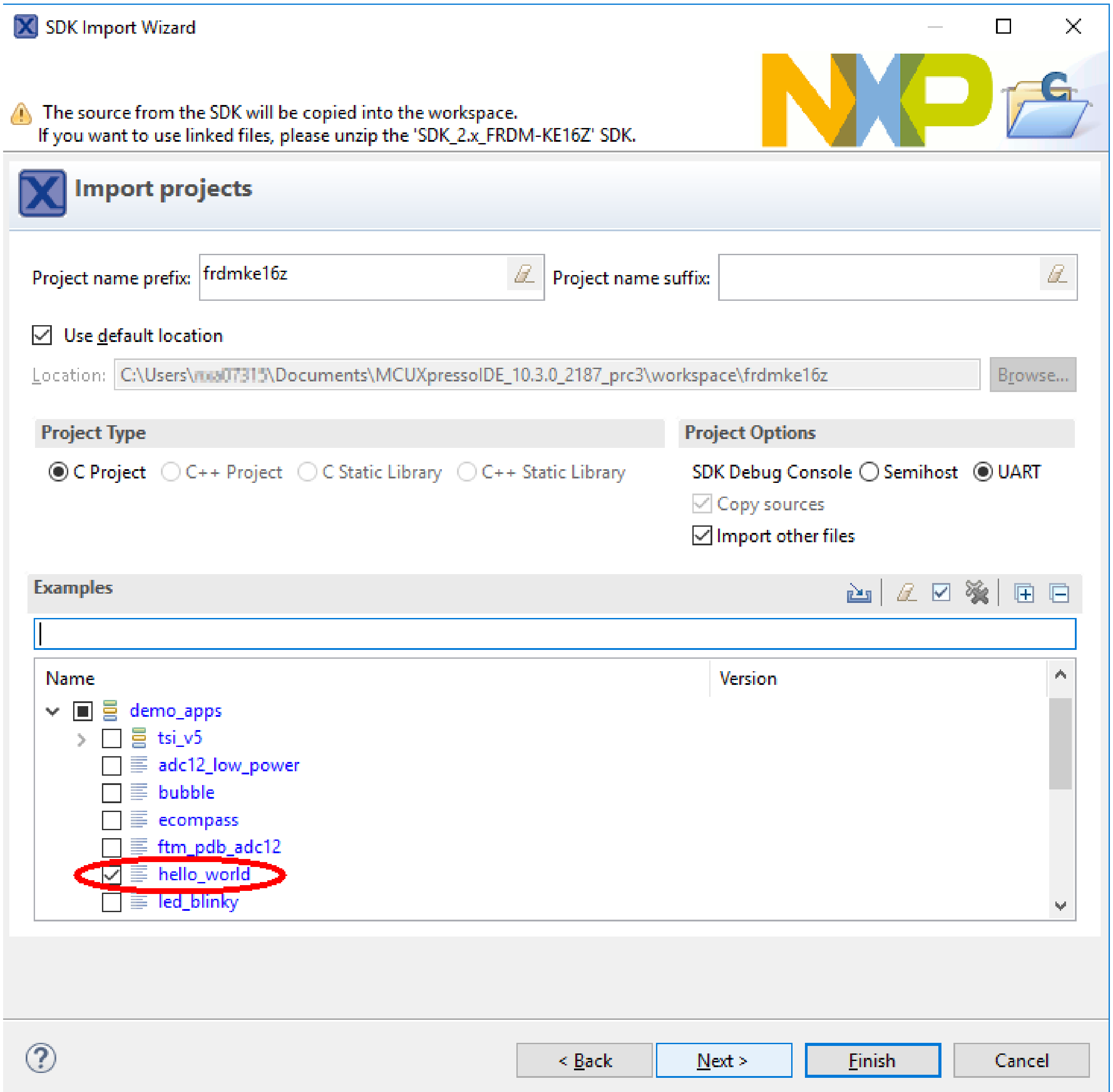

Figure 6. Select `hello_world`

5. Ensure **Redlib: Use floating point version of printf** is selected if the example prints floating point numbers on the terminal for demo applications such as `adc_basic`, `adc_burst`, `adc_dma`, and `adc_interrupt`. Otherwise, it is not necessary to select this option. Then, click **Finish**.

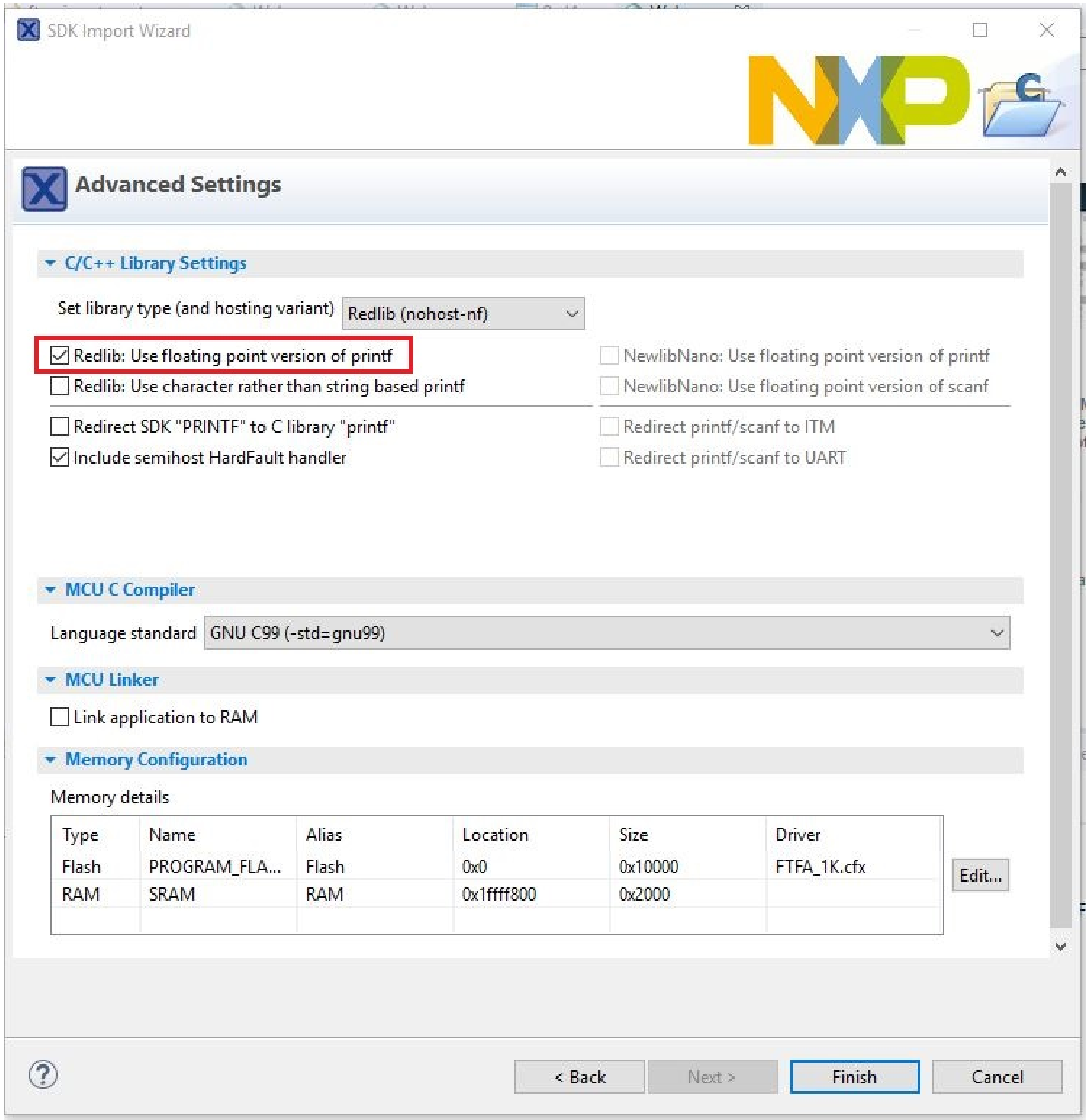

Figure 7. Select `Use floating point version of printf`

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

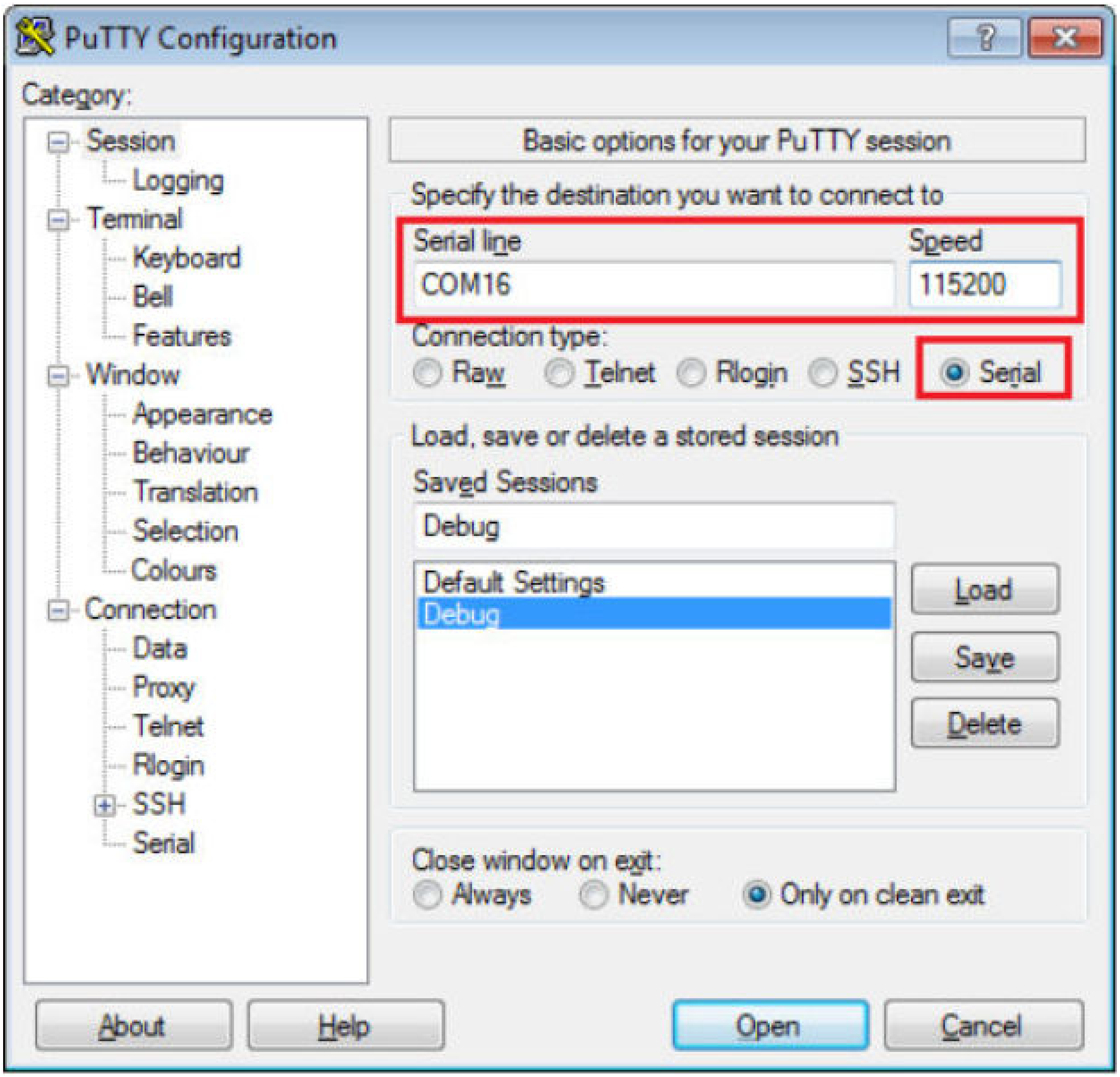

Figure 8. Terminal (PuTTY) configurations

4. On the **Quickstart Panel**, click on **Debug** **`frdmke16z_demo_apps_hello_world [Debug]`** to launch the debug session.

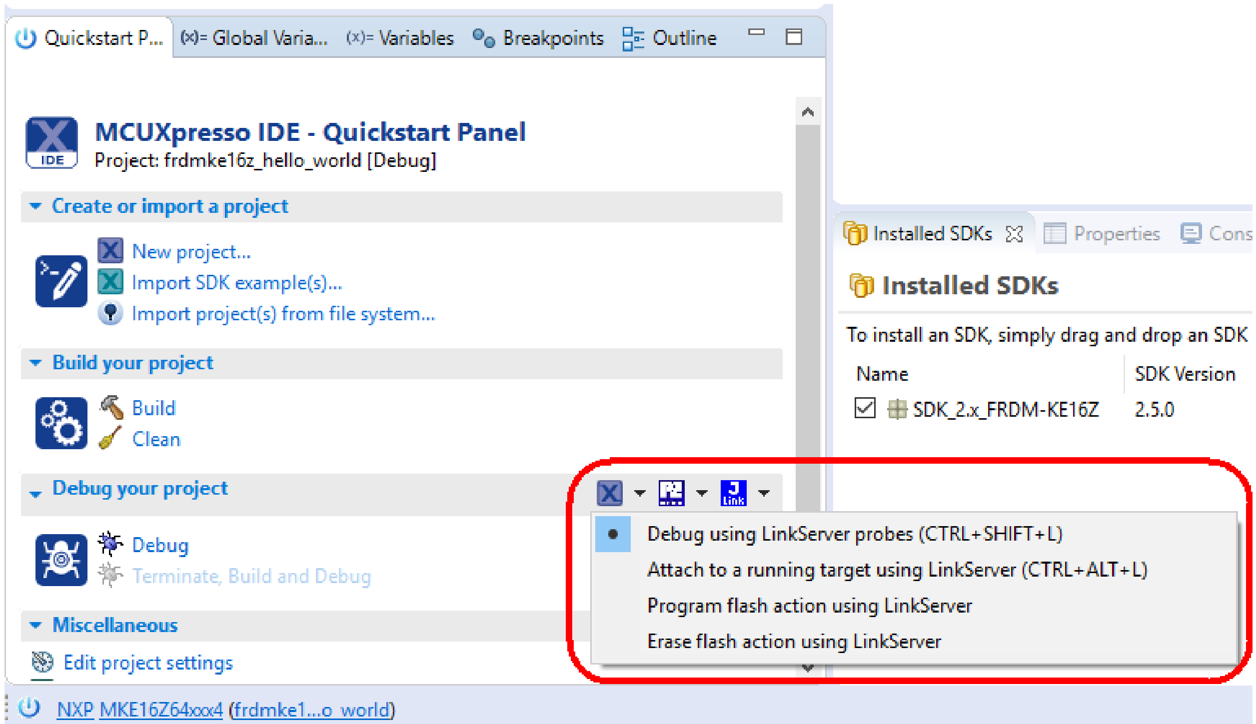

Figure 9. Debug `hello_world` case

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.)

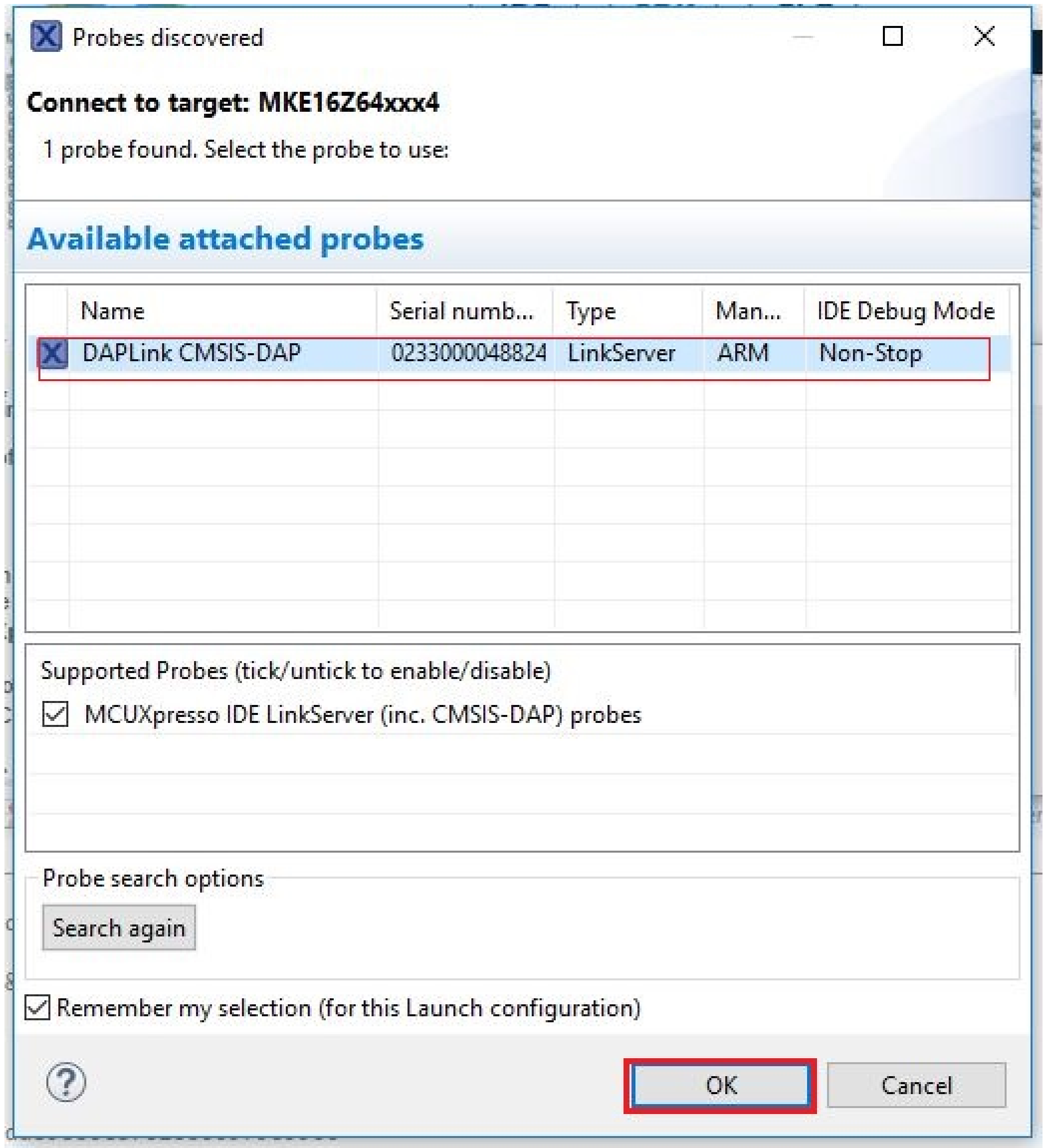

Figure 10. Attached Probes: debug emulator selection

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

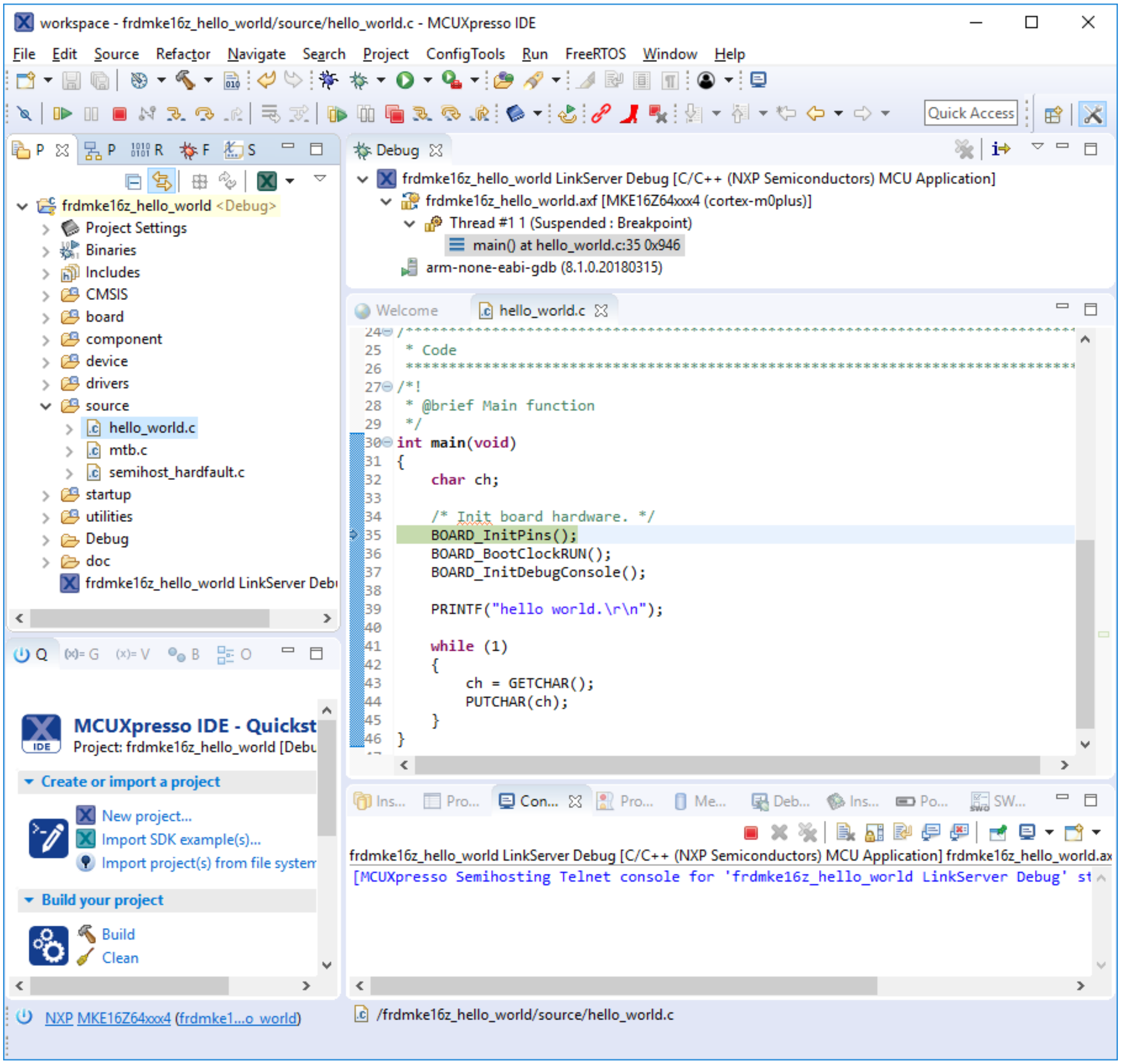

Figure 11. Stop at `main()` when running debugging

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

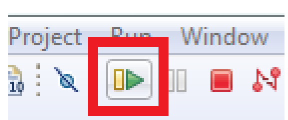

Figure 12. Resume button

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.

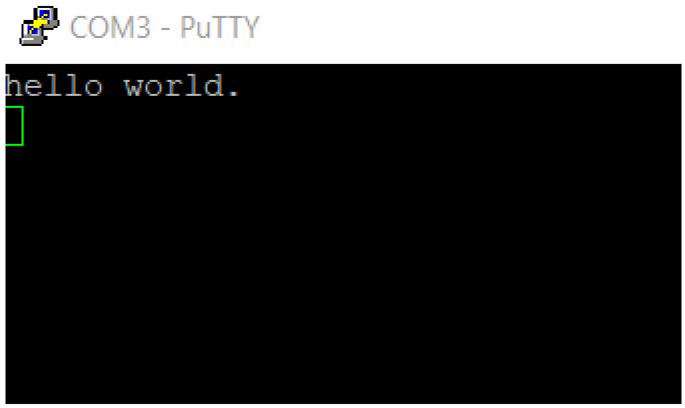

Figure 13. Text display of the **`hello_world`** demo

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 8.32.1 is used in the following example, and the IAR toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes* (document ID: MCUXSDKRN).

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 FRDM-KE16Z Freedom hardware platform as an example, the `hello_world` workspace is located in:

   ```
   <install_dir>/boards/frdmke16z/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**.

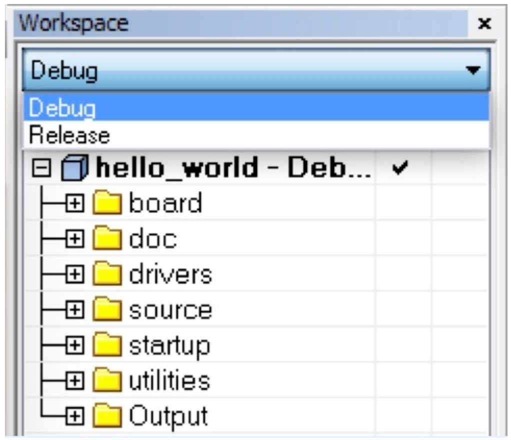

Figure 14. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 15.

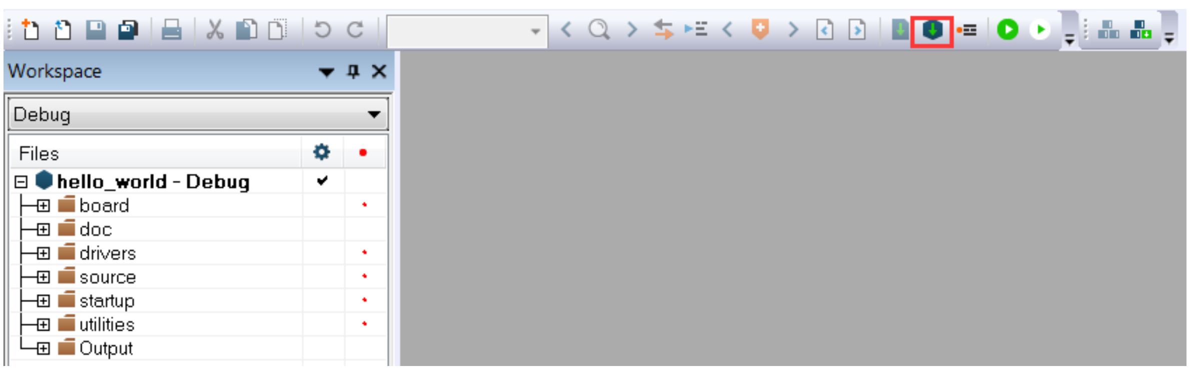

Figure 15. Build the demo application

4. The build completes without errors.

4.2 Run an example application

To download and run the application, perform these steps:

1. Connect the development platform to your PC via USB cable.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see How to determine COM port). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

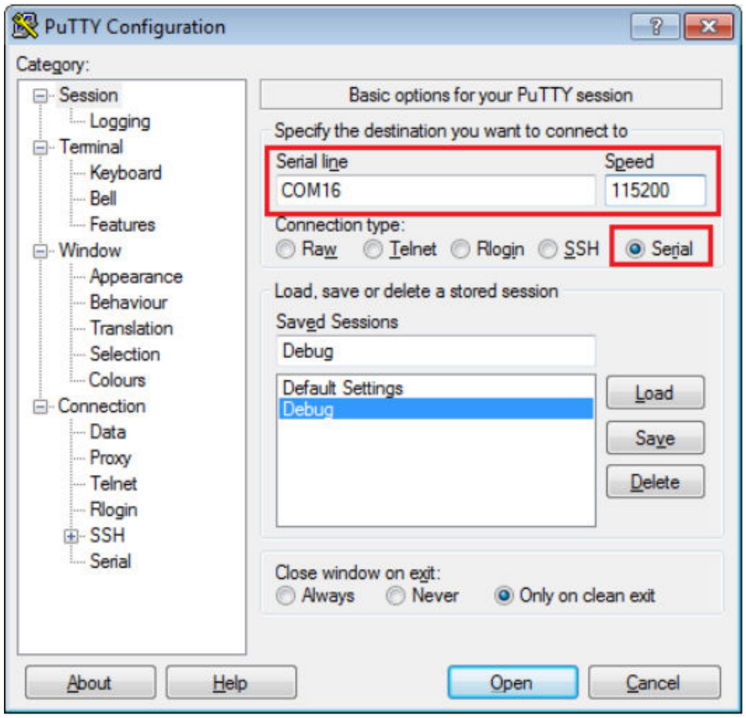

Figure 16. Terminal (PuTTY) configuration

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

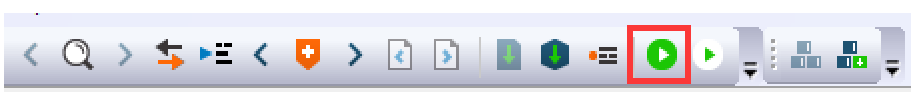

Figure 17. Download and Debug button

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

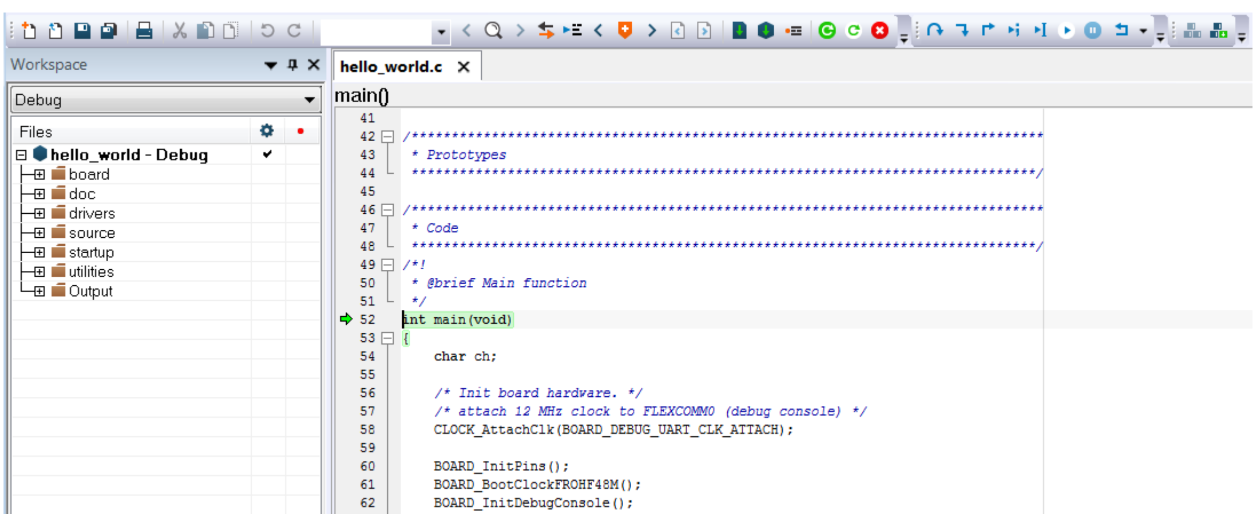

Figure 18. Stop at `main()` when running debugging

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

Figure 19. Go button

6. 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.

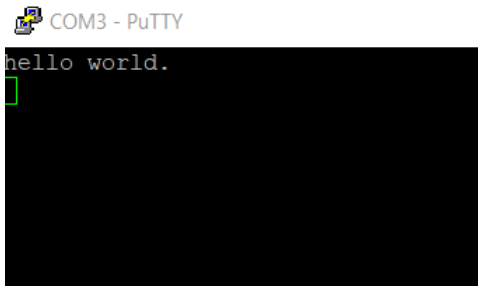

Figure 20. Text display of the **`hello_world`** demo

5 Run a demo using Keil® MDK/µVision

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the FRDM-KE16Z Freedom hardware platform is used as an example, although these steps can be applied to any demo or example application in the MCUXpresso SDK.

5.1 Install CMSIS device pack

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

1. Open the MDK IDE, which is called µVision. In the IDE, select the **Pack Installer** icon.

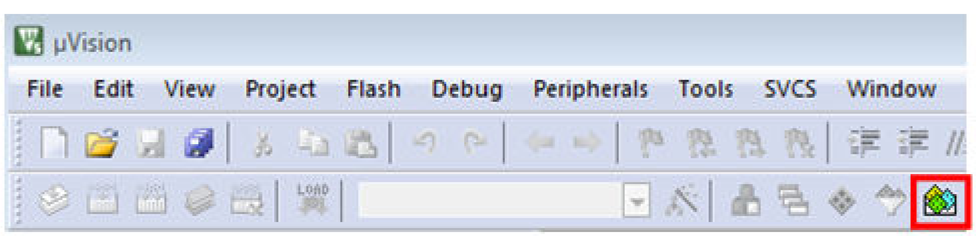

Figure 21. Launch the Pack Installer

2. After the installation finishes, close the Pack Installer window and return to the µVision IDE.

5.2 Build an example application

1. Open the desired example application workspace in:

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

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

```
<install_dir>/boards/frdmke16z/demo_apps/hello_world/mdk/hello_world.uvmpw
```

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

Figure 22. Build the demo

3. The build completes without errors.

5.3 Run an example application

To download and run the application, perform these steps:

1. See the table in Default debug interfaces to determine the debug interface that comes loaded on your specific hardware platform.
   - For boards with the CMSIS-DAP/mbed/DAPLink interface, visit mbed 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, visit www.pemicro.com/support/downloads_find.cfm and download and install the P&E Micro Hardware Interface Drivers package.
   - If using J-Link either a standalone debug pod or OpenSDA, install the J-Link software (drivers and utilities) from www.segger.com/jlink-software.html.
2. Connect the development platform to your PC via USB cable using OpenSDA USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm and connect to the debug serial port number (to determine the COM port number, see How to determine COM port). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

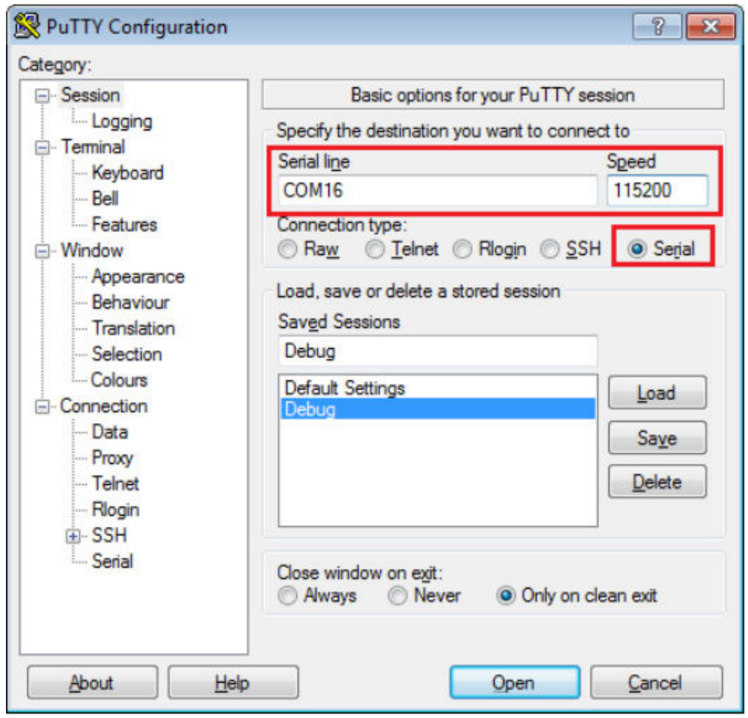

Figure 23. Terminal (PuTTY) configurations

4. In µVision, after the application is built, click the **Download** button to download the application to the target.

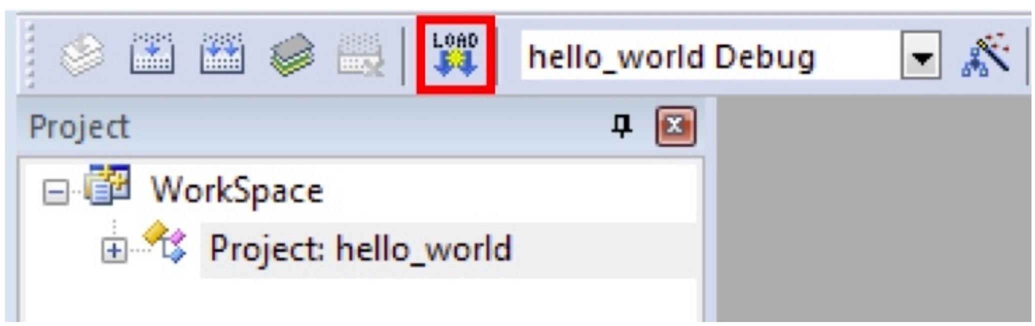

Figure 24. Download button

5. After clicking the **Download** button, the application downloads to the target and is running. To debug the application, click the **Start/Stop Debug Session** button, highlighted in red.

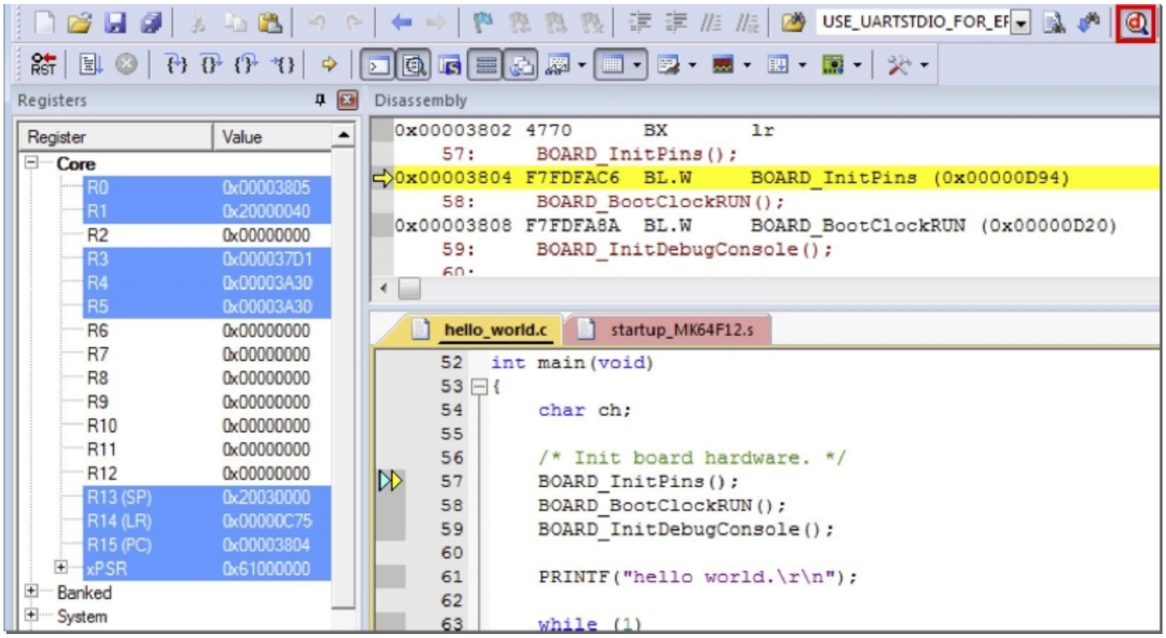

Figure 25. Stop at `main()` when run debugging

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

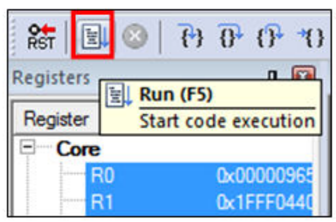

Figure 26. Go button

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.

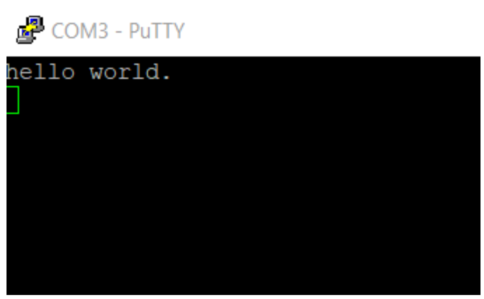

Figure 27. Text display of the hello_world demo

6 Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The `hello_world` demo application is targeted for the FRDM-KE16Z Freedom hardware platform which is used as an example.

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

6.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run an MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK. There are many ways to use Arm GCC tools, but this example focuses on a Windows operating system environment.

6.1.1 Install GCC Arm Embedded tool chain

Download and run the installer from GNU Arm Embedded Toolchain. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes*.

6.1.2 Install MinGW (only required on Windows OS)

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

1. Download the latest MinGW mingw-get-setup installer from MinGW.
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**.

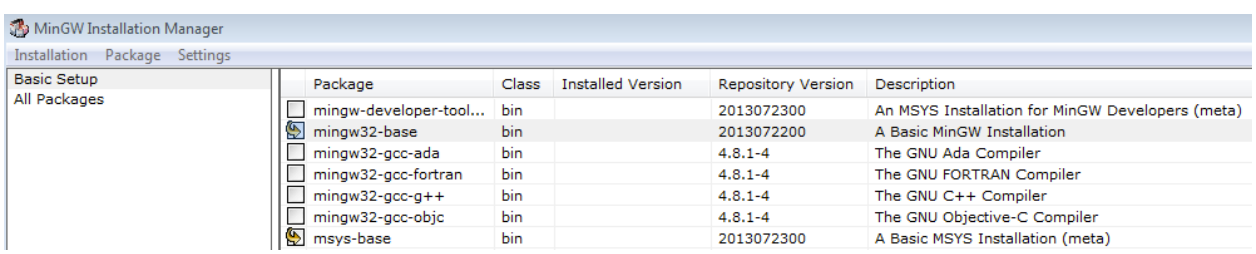

Figure 28. Set up MinGW and MSYS

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

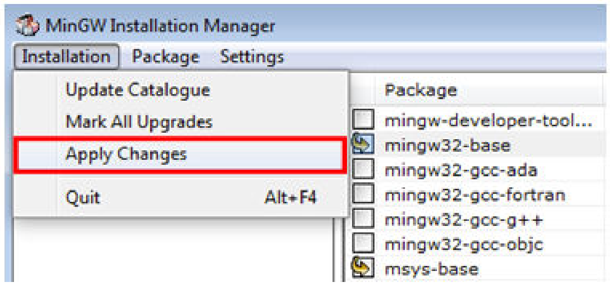

Figure 29. Complete MinGW and MSYS installation

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.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.

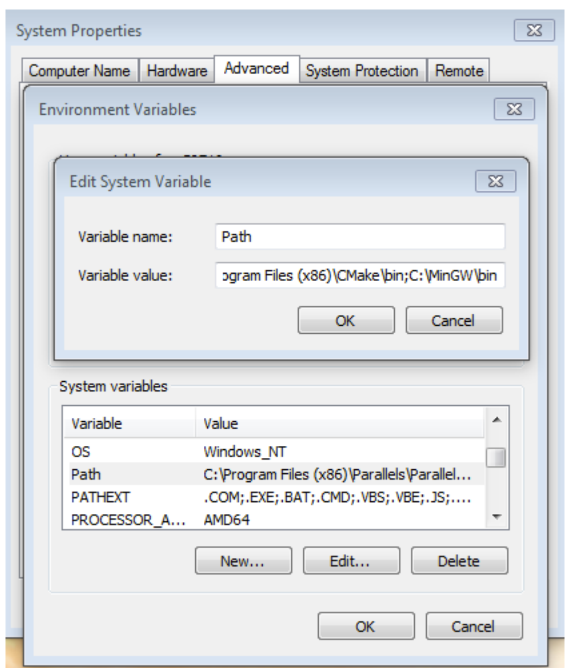

Figure 30. Add Path to systems environment

6.1.3 Add a new system environment variable for ARMGCC_DIR

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

```
C:\Program Files (x86)\GNU Tools  Arm Embedded\8 2018-q4-major
```

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

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

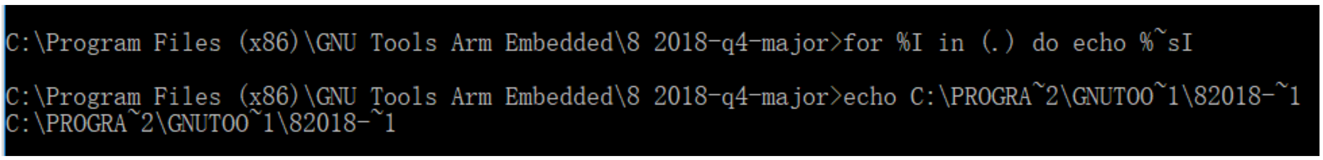

Figure 31. Convert path to short path

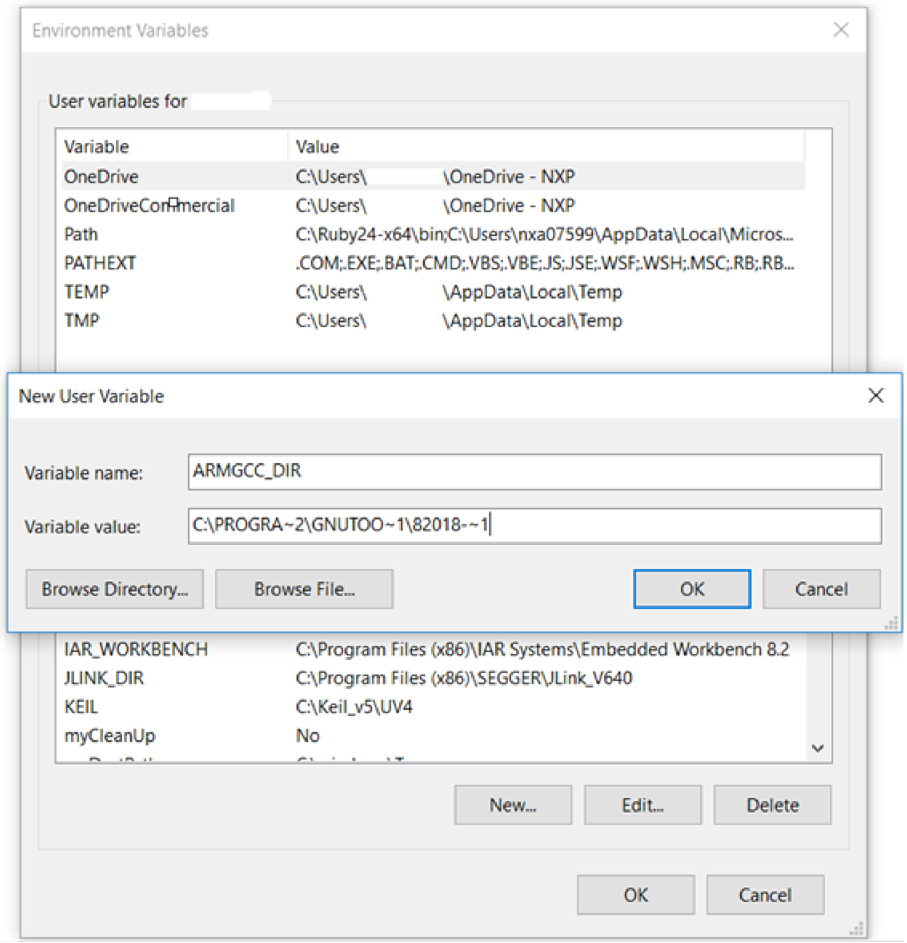

Figure 32. Add ARMGCC_DIR system variable

6.1.4 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.

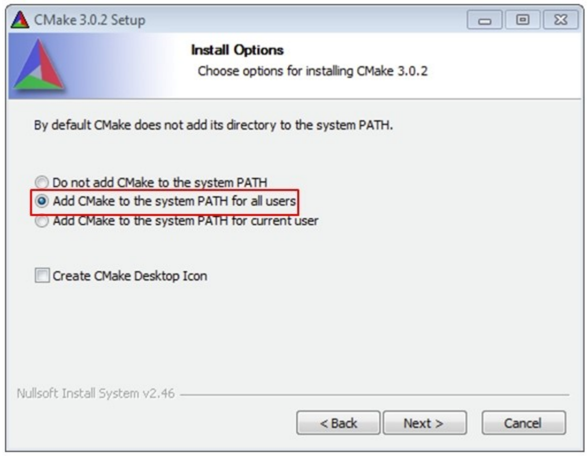

Figure 33. Install CMake

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

6.2 Build an example application

To build an example application, 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**.

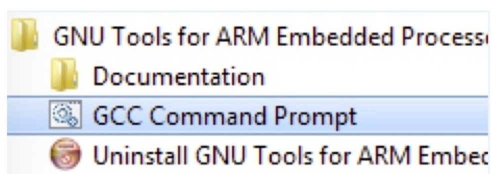

Figure 34. Launch command prompt

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>/boards/frdmke16z/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 35.

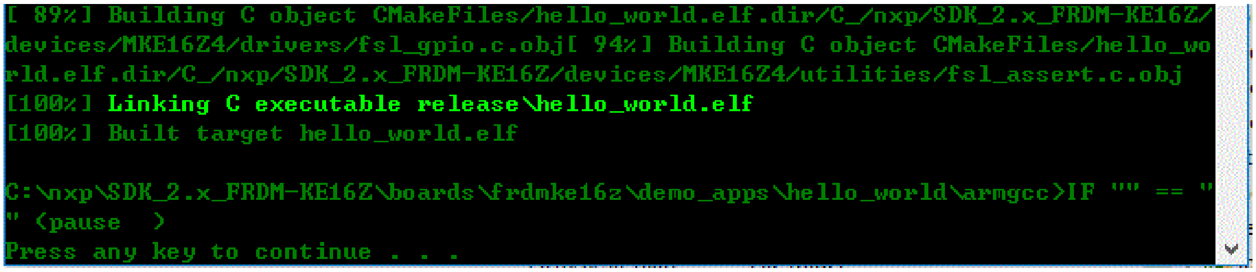

Figure 35. `hello_world` build successful

6.3 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.

NOTE

J-Link GDB Server application is not supported for TFM examples. Use CMSIS DAP instead of J-Link for flashing and debugging TFM examples.

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 How to determine COM port). Configure the terminal with these settings:
   a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in `board.h` file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

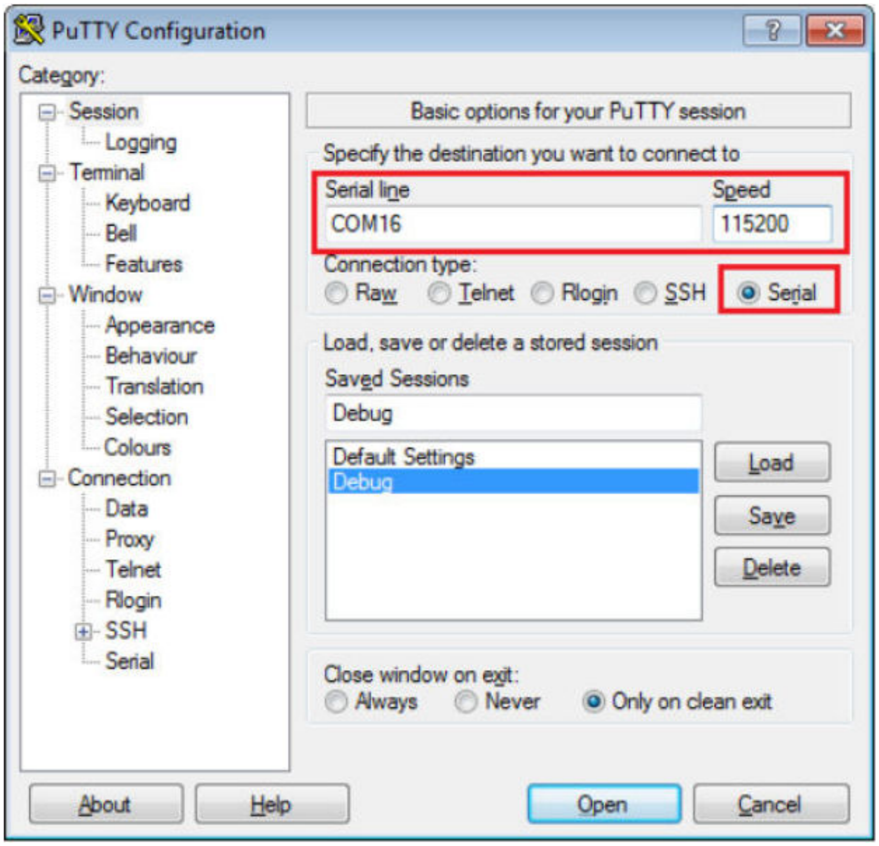

Figure 36. Terminal (PuTTY) configurations

NOTE

Make sure the board is set to FlexSPI flash boot mode (ISP2: ISP1: ISP0 = ON, OFF, ON) before use GDB debug.

3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting **Programs -> SEGGER -> J-Link <version> J-Link GDB Server**.
4. Open the J-Link GDB Server application. Go to the SEGGER install folder. For example, *C:\Program Files(x86)\SEGGER\JLink_Vxxx*. Open the command windows. Use the JLinkGDBServer.exe -device MIMXRT685S_M33 -if SWD -scriptfile: <install_dir>/boards/<board_name>/<example_type>/<application_name>/evkmimxrt685.JLinkScript command.
5. Modify the settings as shown below. The target device selection chosen for this example is **LPC55S06**.
6. Modify the settings as shown below. The target device selection chosen for this example is **LPC55S06CP**.
7. After it is connected, the screen should look like this figure:

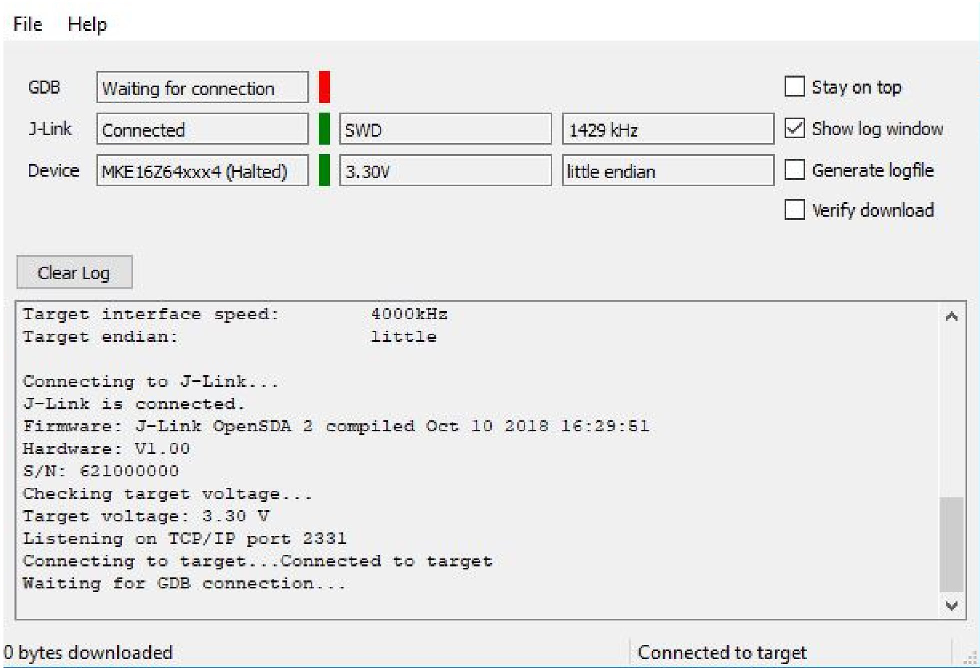

Figure 37. SEGGER J-Link GDB Server screen after successful connection

8. 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**.

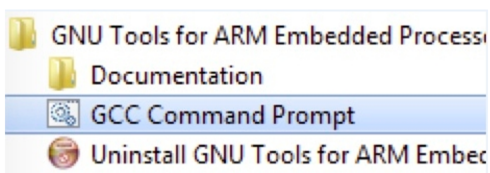

Figure 38. Launch command prompt

9. 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/frdmke16z/demo_apps/hello_world/armgcc/debug
```

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

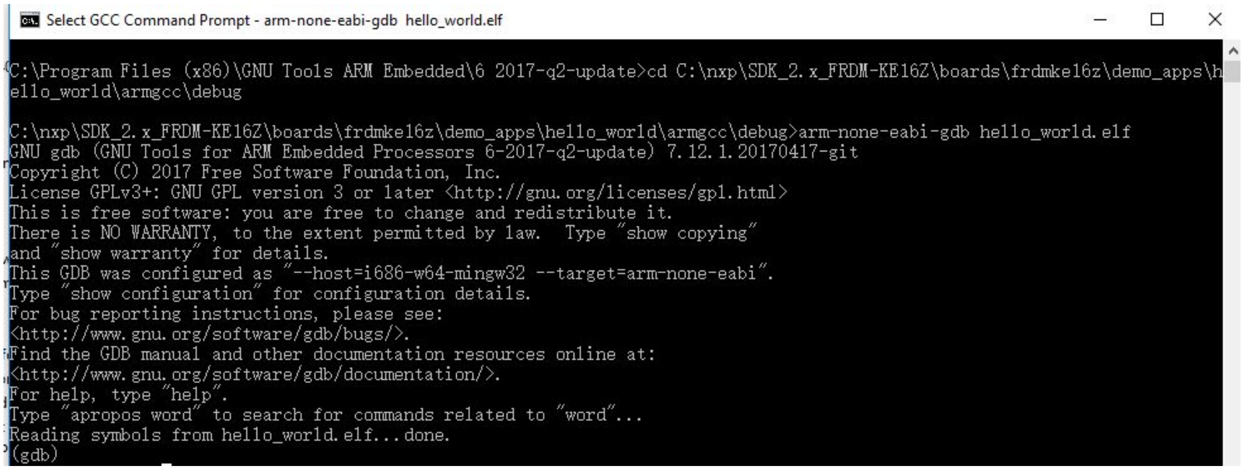

Figure 39. Run **`arm-none-eabi-gdb`**

11. Run these commands:

    a. `target remote localhost:2331`

    b. `monitor reset`

    c. `monitor halt`

    d. `load`

12. 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.

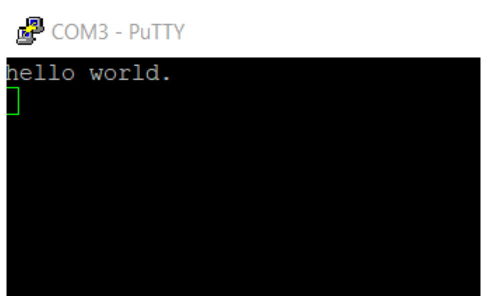

Figure 40. Text display of the hello_world demo

7 MCUXpresso Config Tools

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

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

Table 1. MCUXpresso Config Tools

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

MCUXpresso Config Tools can be accessed in the following products:

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

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

8 MCUXpresso IDE New Project Wizard

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

To use the wizard, start the MCUXpresso IDE. This is located in the **QuickStart Panel** at the bottom left of the MCUXpresso IDE window. Select **New project**, as shown in Figure 41.

Figure 41. MCUXpresso IDE Quickstart Panel

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

9 How to determine COM port

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

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

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

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

2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click 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.

   a. **OpenSDA – CMSIS-DAP/mbed/DAPLink** interface:

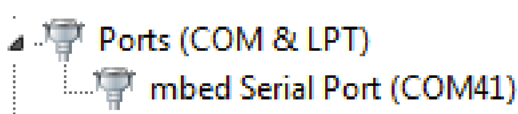

Figure 42. OpenSDA – CMSIS-DAP/mbed/DAPLink interface

   b. **OpenSDA – P&E Micro**:

Figure 43. OpenSDA – P&E Micro

c. OpenSDA – J-Link:

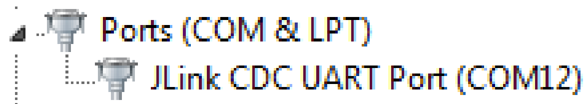

Figure 44. OpenSDA – J-Link

d. P&E Micro OSJTAG:

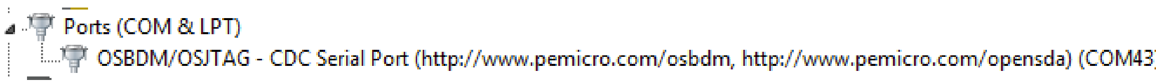

Figure 45. P&E Micro OSJTAG

e. MRB-KW01:

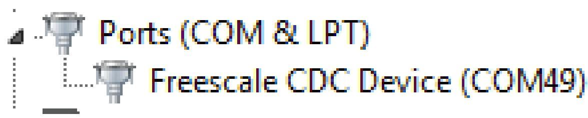

Figure 46. MRB-KW01

10 How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to

override the default IRQ handler. For example, to override the default `PIT_IRQHandler` define in `startup_DEVICE.s`, application code like app.c can be implement like:

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

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

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

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

11 Default debug interfaces

The MCUXpresso SDK supports various hardware platforms that come loaded with a variety of factory programmed debug interface configurations. The following table 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 of the following table is not applicable to LPC.

Table 2. Hardware platforms supported by SDK

Hardware platform	Default interface	OpenSDA details
FRDM-KE16Z	CMSIS-DAP/mbed/DAPLink	OpenSDA v2.2

12 Updating debugger firmware

12.1 Updating OpenSDA firmware

Any NXP hardware platform that comes with an OpenSDA-compatible debug interface has the ability to update the OpenSDA firmware. This typically means switching from the default application (either CMSIS-DAP/mbed/DAPLink or P&E Micro) to a SEGGER J-Link. This section contains the steps to switch the OpenSDA firmware to a J-Link interface. However, the steps can be applied to restoring the original image also. 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 Default debug interfaces. 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.
- CMSIS-DAP/mbed/DAPLink: DAPLink OpenSDA firmware is available at www.nxp.com/opensda.
- 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**.

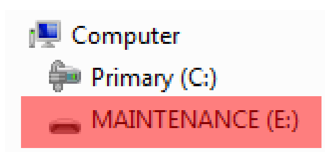

Figure 47. MAINTENANCE drive

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.

13 Revision history

This table summarizes revisions to this document.

Table 3. Revision history

Revision number	Date	Substantive changes
0	February 2018	Initial Release
1	June 2019	Updated for MCUXpresso SDK v2.8.0
2	15 January 2021	Updated for MCUXpresso SDK v2.9.0
2.10.0	10 July 2021	Updated for MCUXpresso SDK v2.10.0

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 10 July 2021

Document identifier: MCUXSDKKE16GSUG