



MCUXSDKMIMXRT5XXGSUG

Getting Started with MCUXpresso SDK for EVK-MIMXRT595

Rev. 2.12.0 — 30 June 2022

User guide

Document information

Information	Content
Keywords	MCUXpresso SDK, Getting Started, EVK-MIMXRT595, RT5XX
Abstract	This document describes the steps to get started with MCUXpresso SDK for EVK-MIMXRT595.



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 EVK-MIMXRT595* (document MCUSDKRT595RN).

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

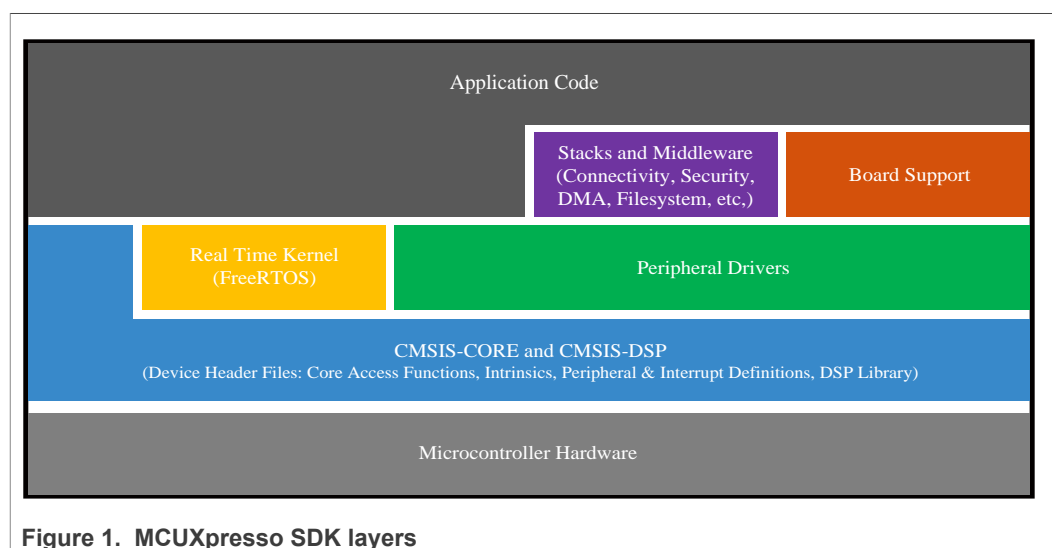


Figure 1. MCUXpresso SDK layers

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.

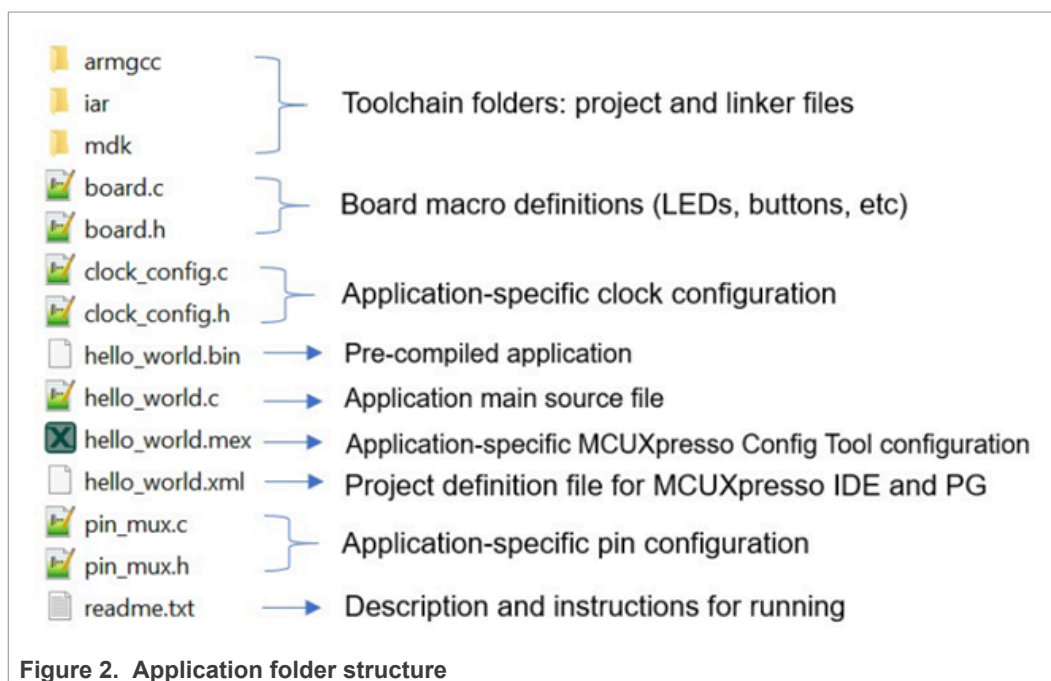
- **driver_examples:** Simple applications that show how to use the MCUXpresso SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral but there are cases where multiple peripherals are used (for example, SPI conversion using DMA).
- **emwin_examples:** Applications that use the emWin GUI widgets.
- **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
- **usb_examples:** Applications that use the USB host/device/OTG stack.
- **wireless_examples:** Applications that use the Zigbee and OpenThread stacks.
- **usb_dongle_examples:** Simple applications to be used on the PCB2459-2 JN5189 USB DONGLE.

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:



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/<device_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 MIMXRT595-EVK 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.

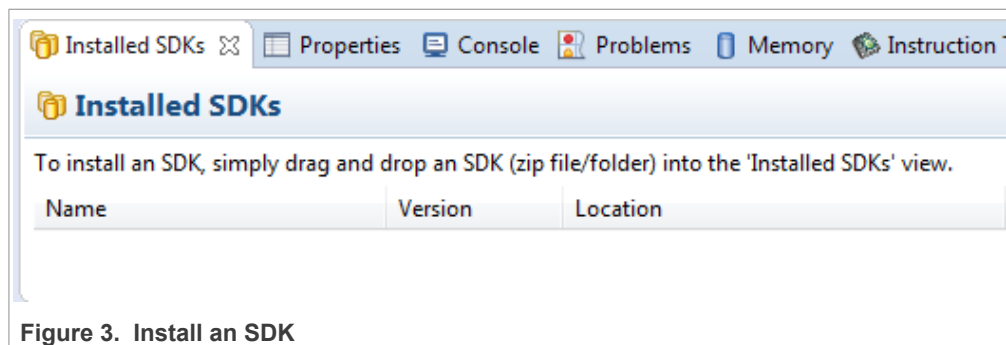


Figure 3. Install an SDK

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

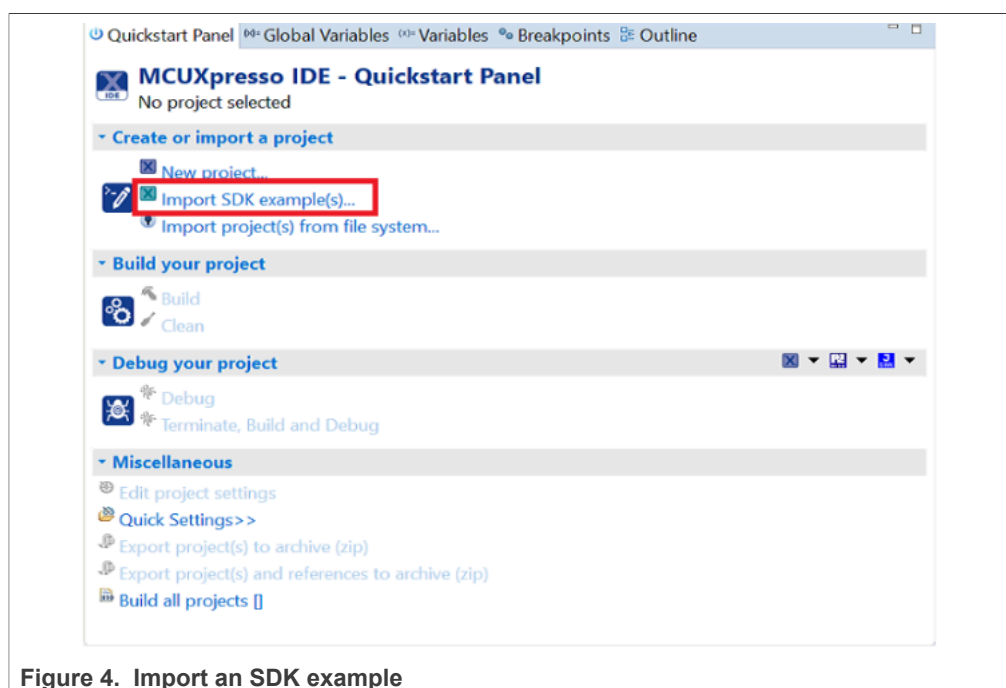


Figure 4. Import an SDK example

3. In the window that appears, expand the **MIMXRT500** folder and select **MIMXRT595S**. Then, select **evkmimxrt595** and click **Next**.
4. Expand the **demo_apps** folder and select **hello_world**. Then, click **Next**.
5. 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**.

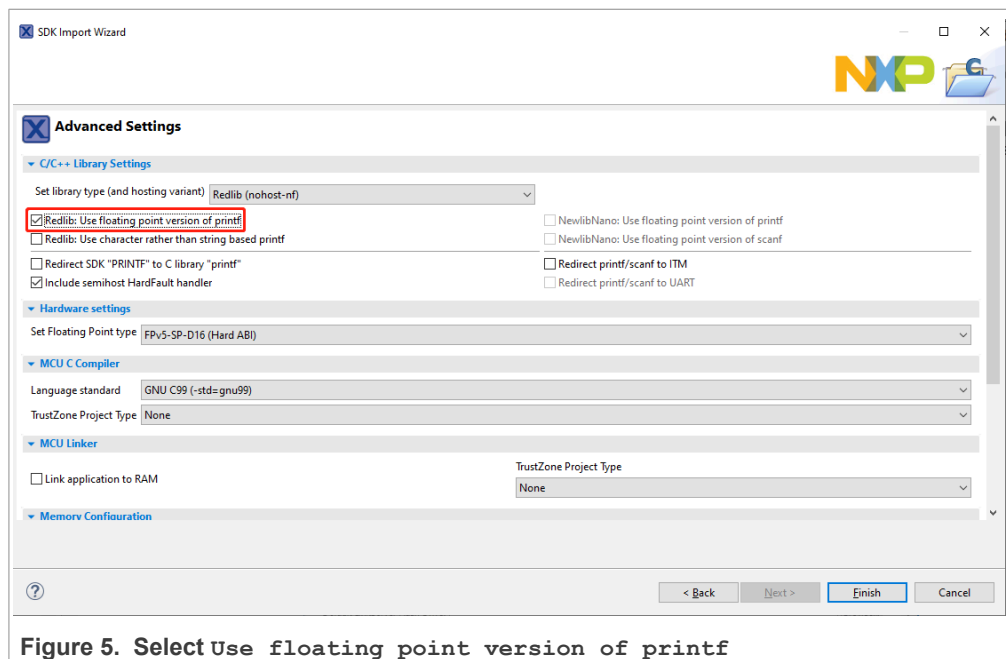


Figure 5. Select Use floating point version of printf

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 11](#) 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 9](#)). 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

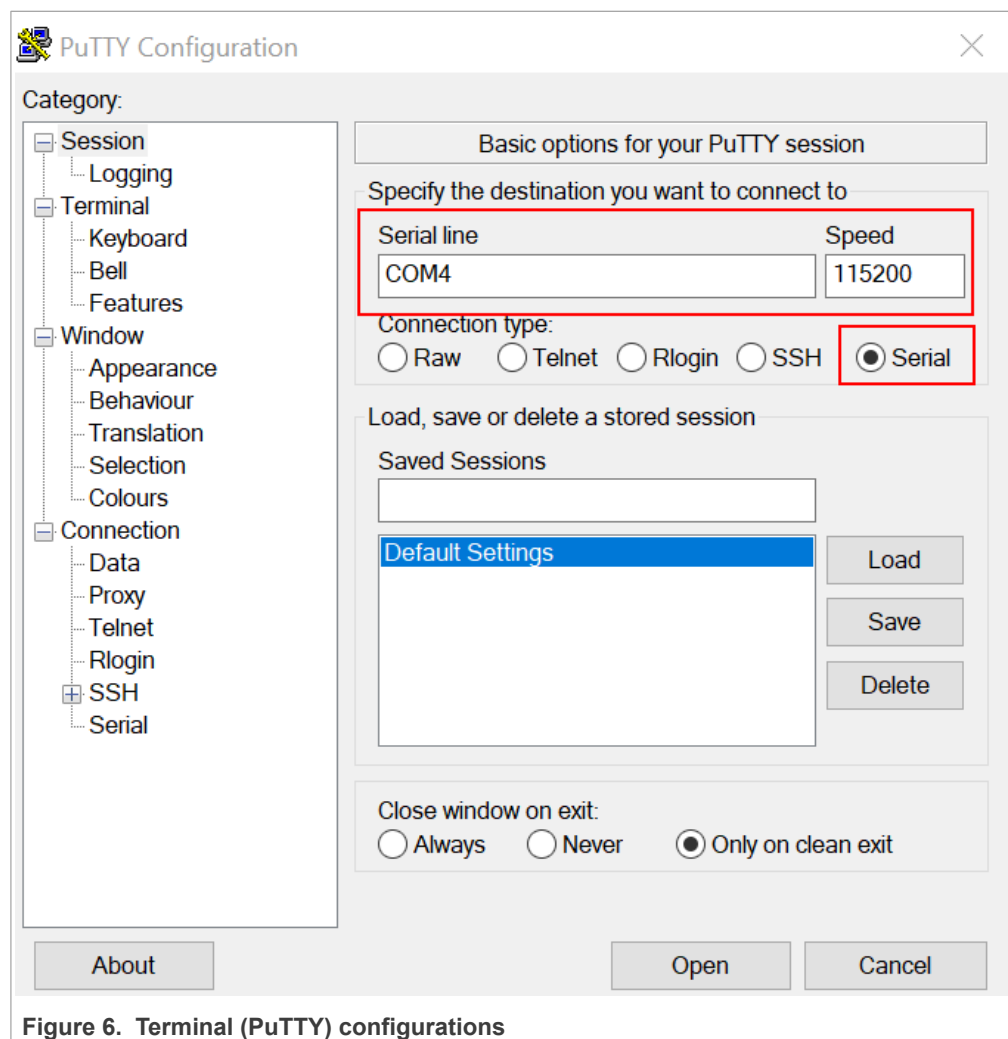


Figure 6. Terminal (PuTTY) configurations

4. On the **Quickstart Panel**, click on **Debug evkmimxrt595_hello_world [Debug]** to launch the debug session.

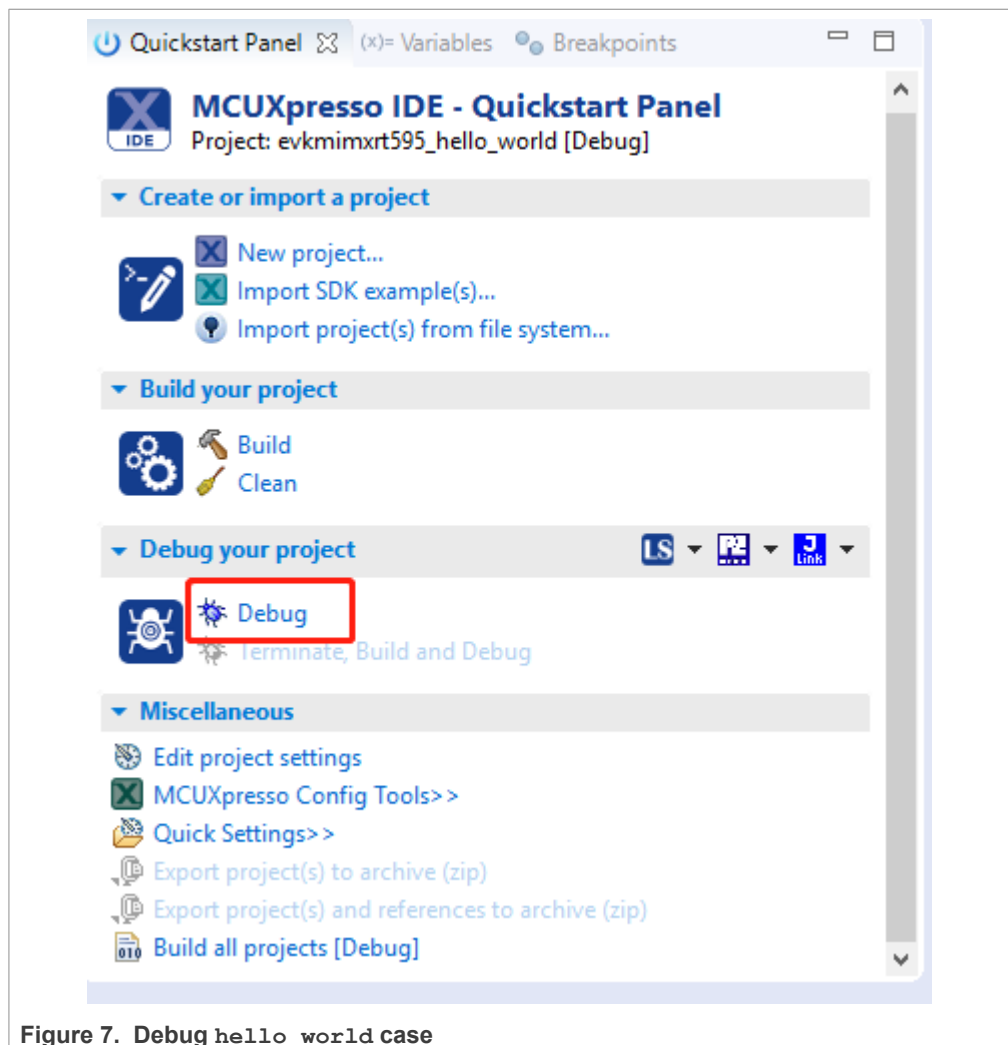


Figure 7. Debug hello_world case

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.)

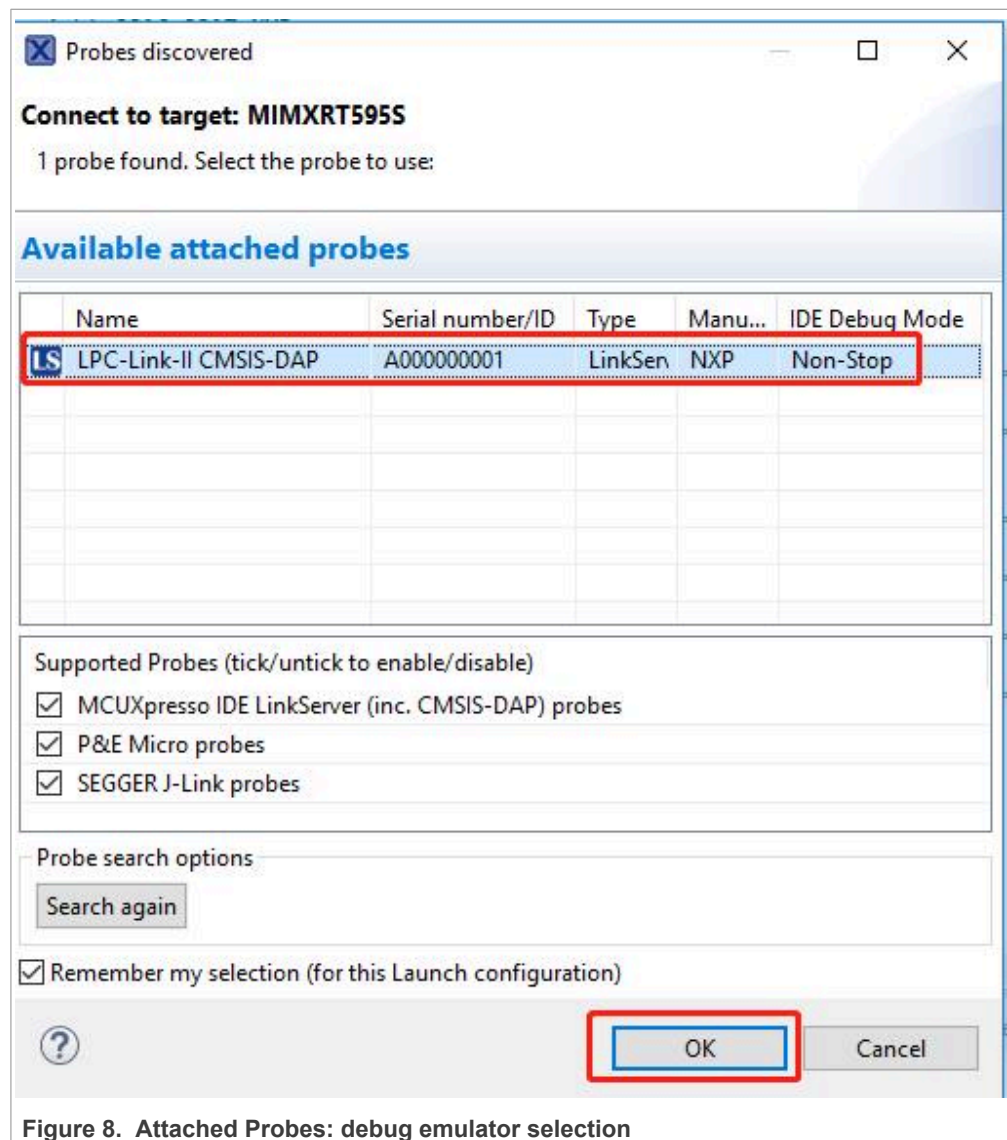


Figure 8. Attached Probes: debug emulator selection

Note: If the debugging application is running in flash, make sure the board is set to FlexSPI flash boot mode.

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

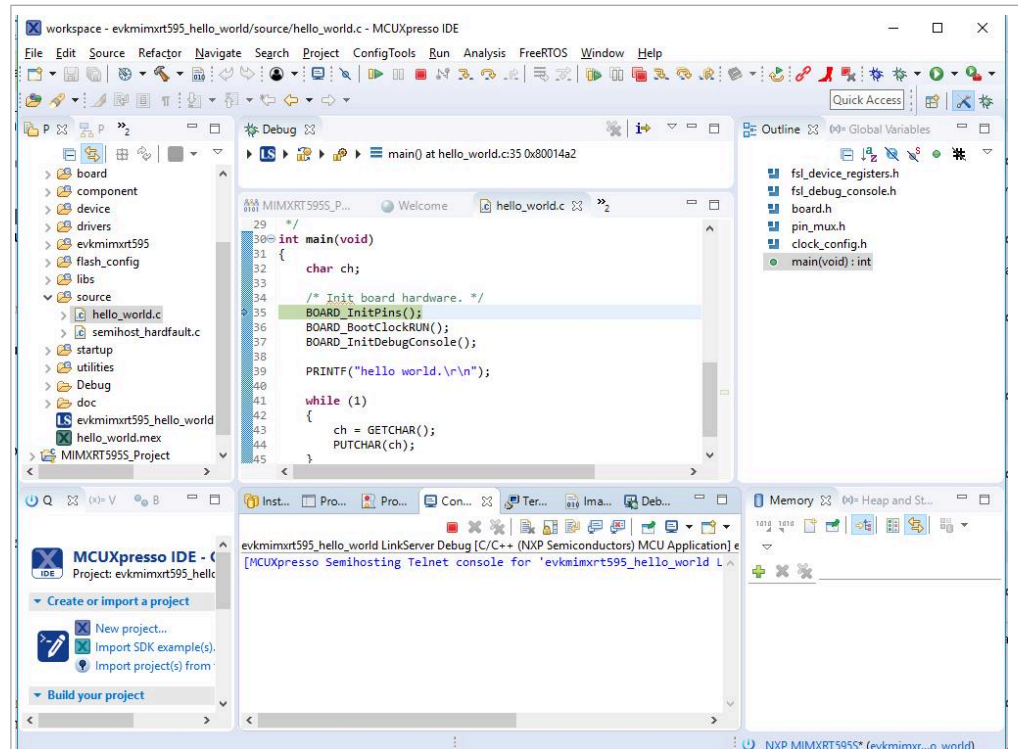


Figure 9. Stop at main() when running debugging

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

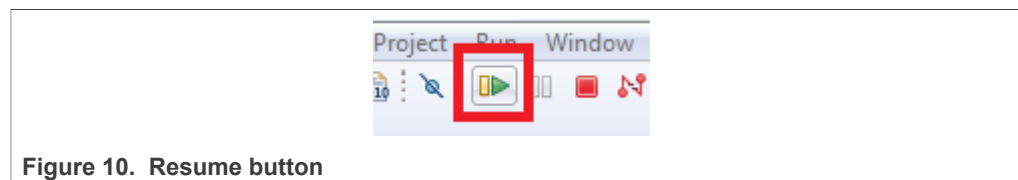


Figure 10. Resume button

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.

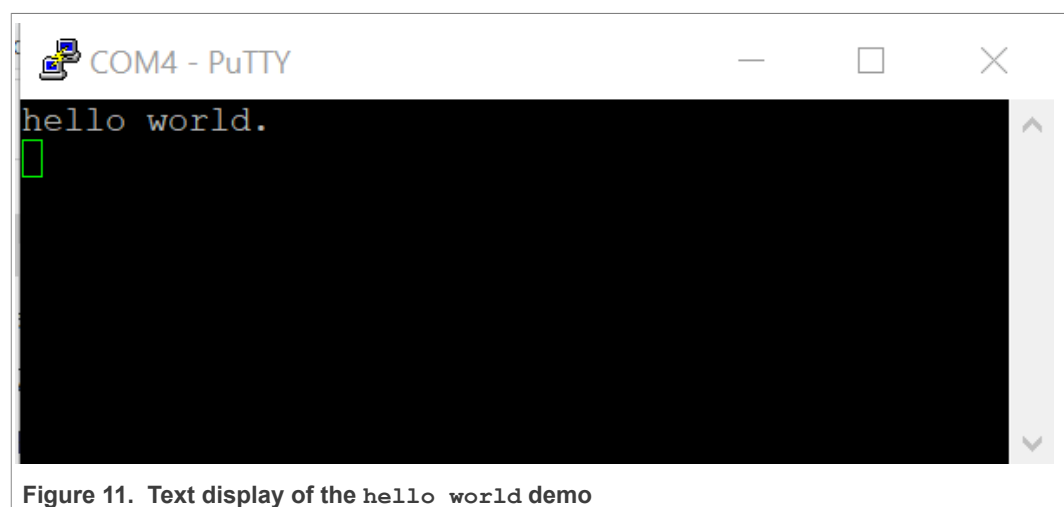


Figure 11. Text display of the `hello_world` demo

3.4 Build a TrustZone example application

This section describes the steps required to configure MCUXpresso IDE to build, run, and debug TrustZone example applications. The trustzone version of the `hello_world` example application targeted for the MIMXRT595-EVK hardware platform is used as an example, though these steps can be applied to any TrustZone example application in the MCUXpresso SDK.

1. TrustZone examples are imported into the workspace in a similar way as single core applications. When the SDK zip package for MIMXRT595-EVK is installed and available in the **Installed SDKs** view, click **Import SDK example(s)...** on the Quickstart Panel. In the window that appears, expand the **MIMXRT500** folder and select **MIMXRT595S**. Then, select **evkmimxrt595** and click **Next**.
2. Expand the `trustzone_examples/` folder and select `hello_world_s`. Because TrustZone examples are linked together, the non-secure project is automatically imported with the secure project, and there is no need to select it explicitly. Then, click **Finish**.

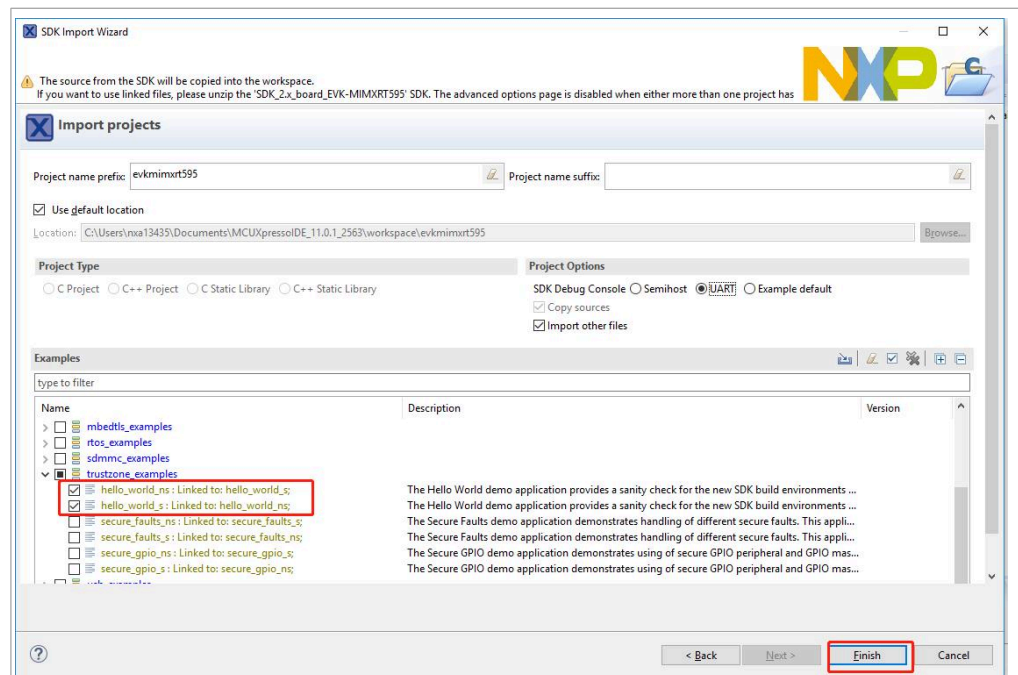


Figure 12. Select the `hello_world` TrustZone example

3. Now, two projects should be imported into the workspace. To start building the TrustZone application, highlight the `evkmimxrt595_hello_world_s` project (TrustZone master project) in the Project Explorer. Then, choose the appropriate build target, **Debug** or **Release**, by clicking the downward facing arrow next to the hammer icon, as shown in Figure 13. For this example, select the **Debug** target.

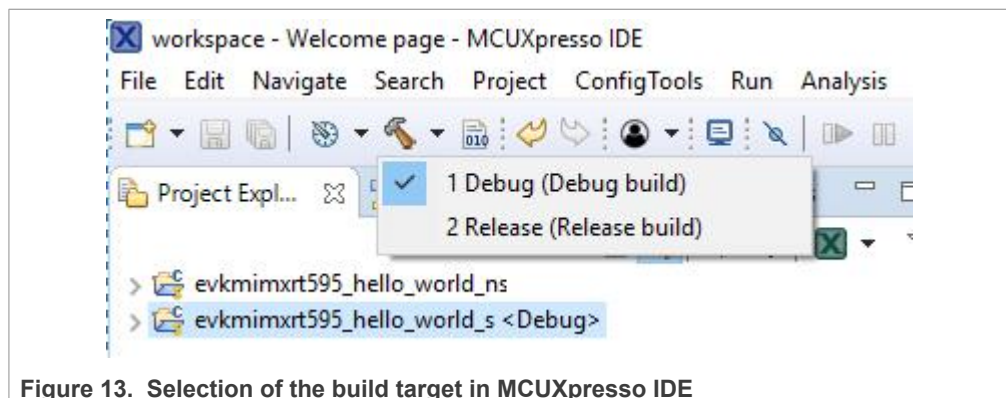


Figure 13. Selection of the build target in MCUXpresso IDE

The project starts building after the build target is selected. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library when running the linker. It is not possible to finish the non-secure project linker when the secure project since CMSE library is not ready.

Note: When the **Release** build is requested, it is necessary to change the build configuration of both the secure and non-secure application projects first. To do this, select both projects in the Project Explorer view by clicking to select the first project, then using shift-click or control-click to select the second project. Right click in the Project Explorer view to display the context-sensitive menu and select **Build Configurations > Set Active >Release**. This is also possible by using the menu item of **Project > Build Configuration >Set Active >Release**. After switching to the **Release** build configuration. Build the application for the secure project first.

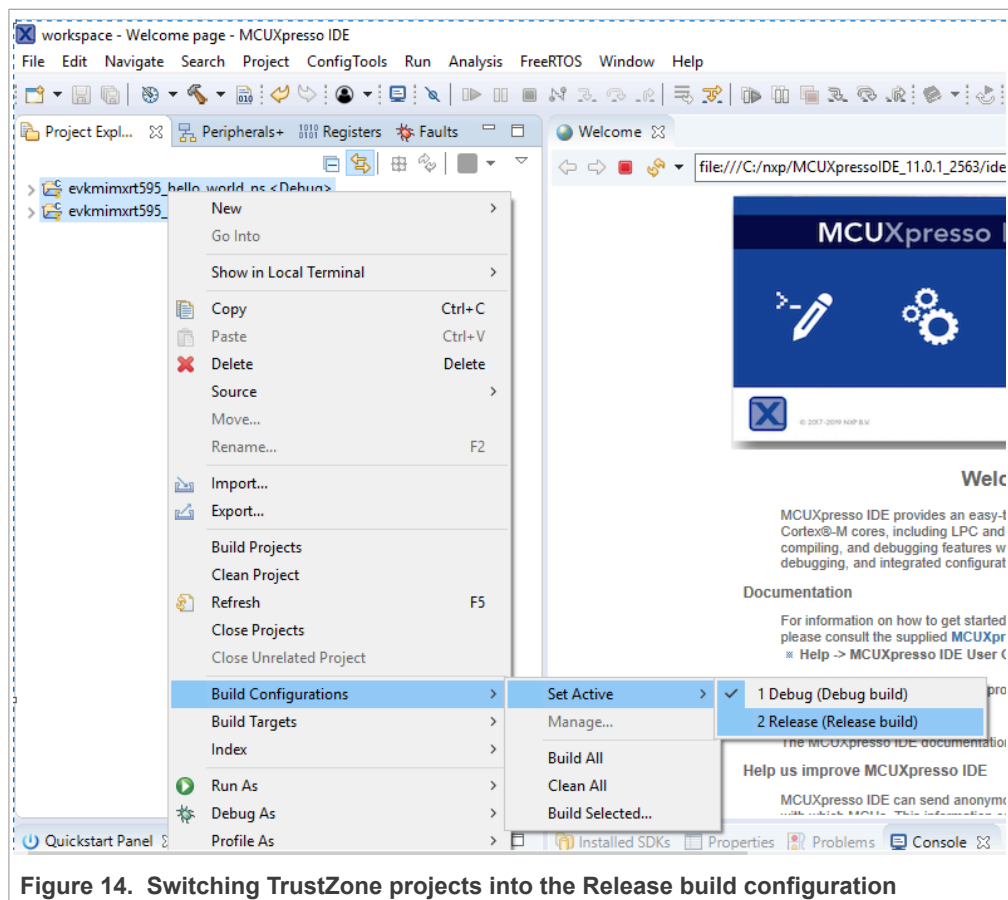


Figure 14. Switching TrustZone projects into the Release build configuration

3.5 Run a TrustZone example application

To download and run the application, perform all steps as described in [Section 3.3](#). These steps are common for single core, and TrustZone applications, ensuring `evkmimxrt595_hello_world_s` is selected for debugging.

In the Quickstart Panel, click **Debug** to launch the second debug session.

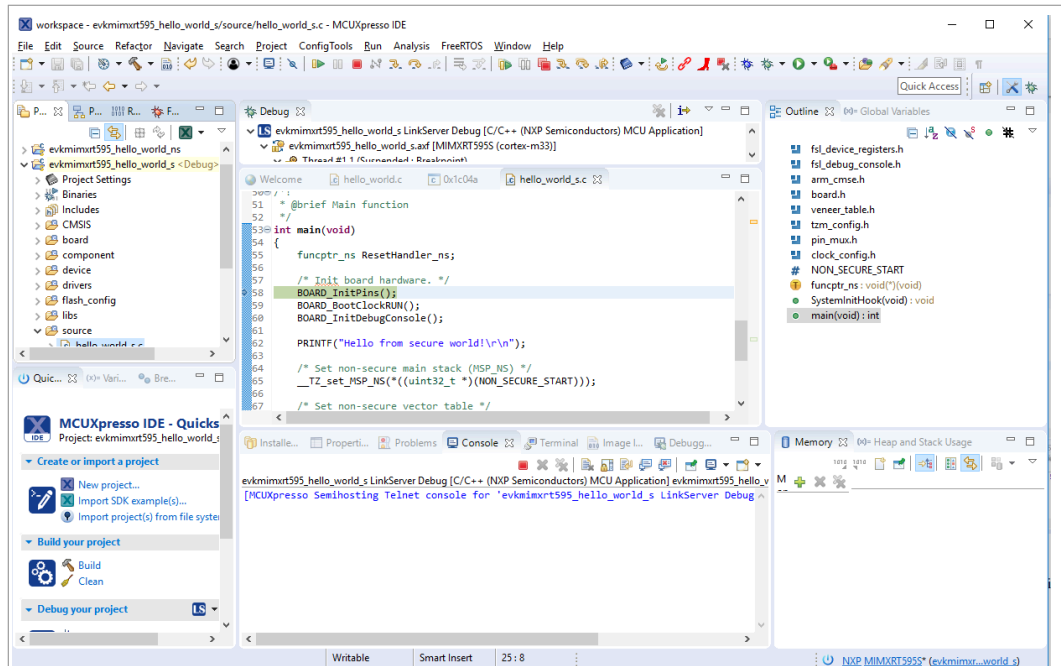


Figure 15. TrustZone debug sessions

Now, the TrustZone sessions should be opened. Click **Resume**. The `hello_world` TrustZone application then starts running, and the secure application starts the non-secure application during run time.

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, 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 MIMXRT595-EVK hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/evkmimxrt595/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**.

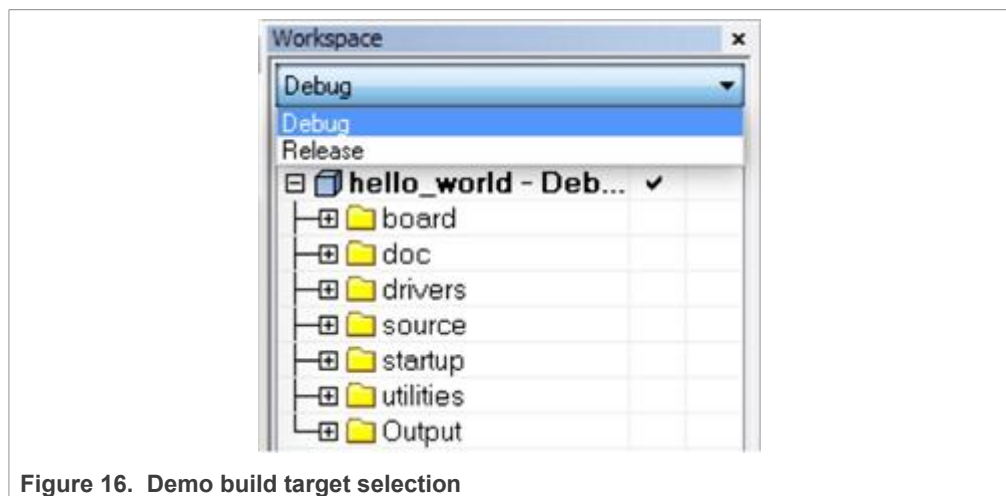


Figure 16. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in [Figure 17](#).

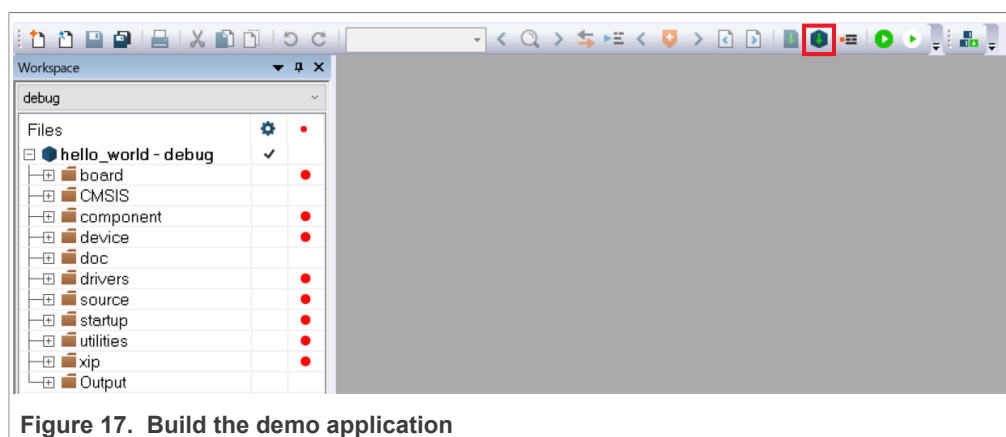


Figure 17. Build the demo application

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 11](#) to determine the debug interface that comes loaded on your specific hardware platform.
 - For boards with P&E Micro interfaces, visit www.pemicro.com/support/downloads_find.cfm and download the P&E Micro Hardware Interface Drivers package.
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 9](#)). Configure the terminal with these settings:
 - a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 - b. No parity
 - c. 8 data bits

d. 1 stop bit

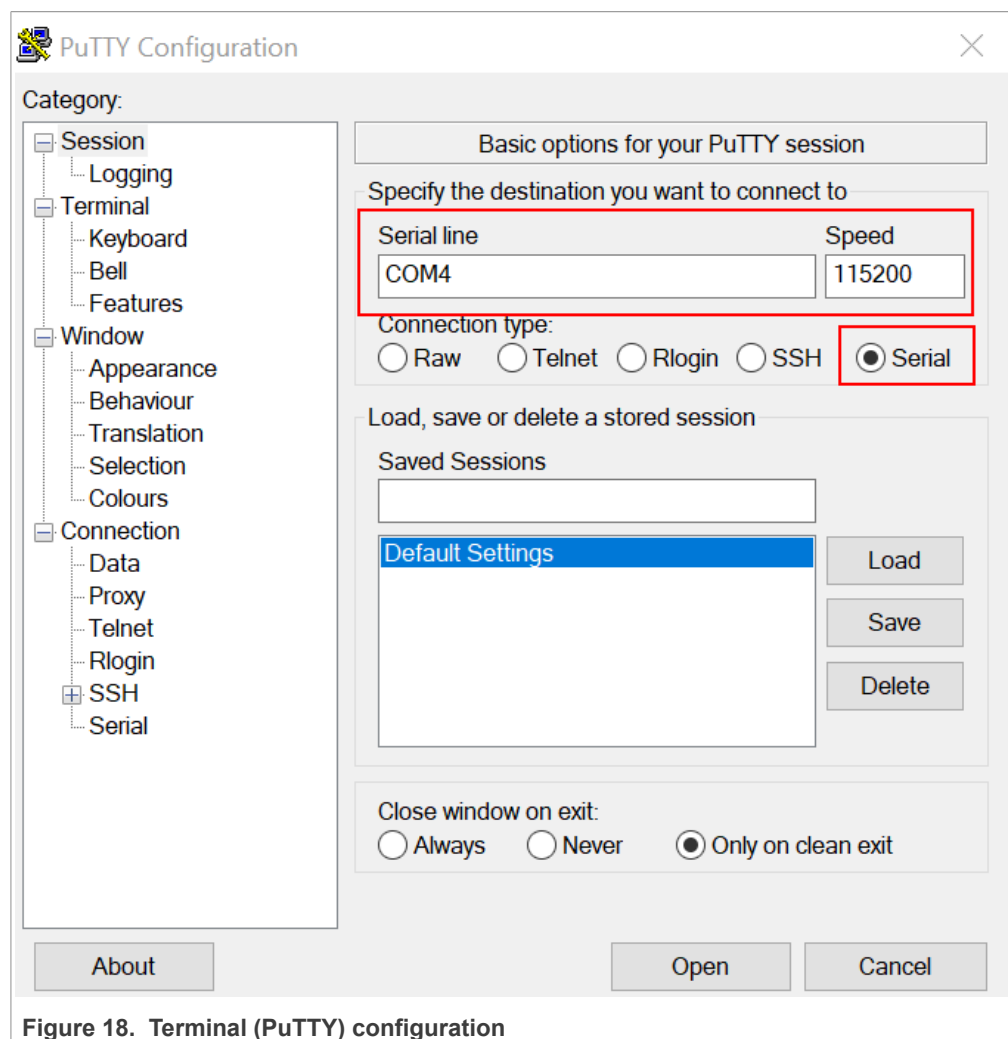


Figure 18. Terminal (PuTTY) configuration

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

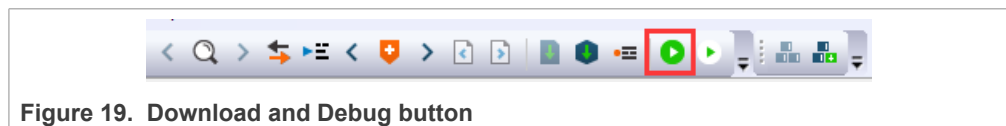
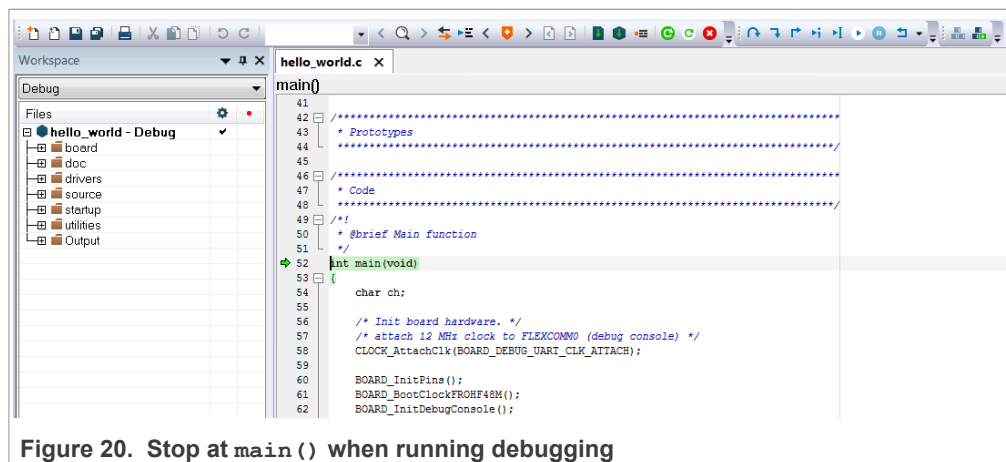


Figure 19. Download and Debug button

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

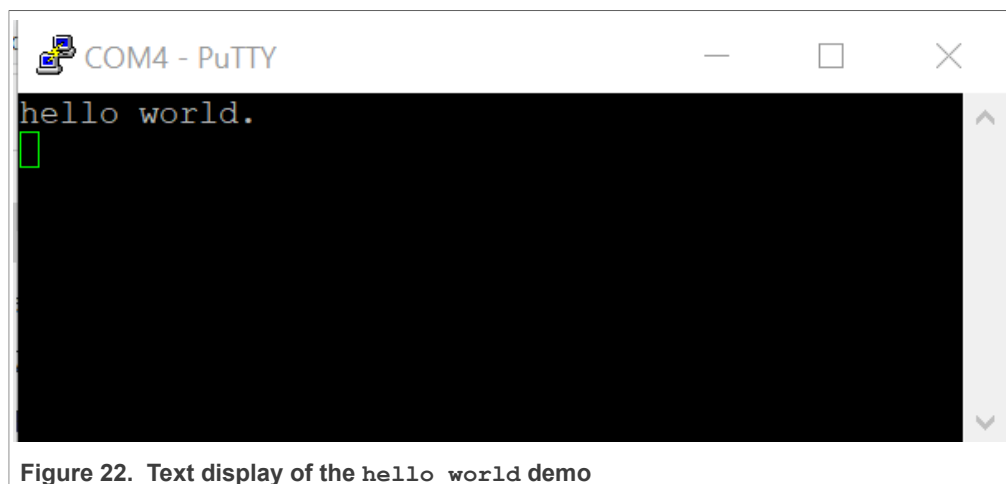
Figure 20. Stop at `main()` when running debugging

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



Figure 21. Go button

- 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 22. Text display of the `hello_world` demo

4.3 Build a TrustZone example application

This section describes the particular steps that need to be done in order to build and run a TrustZone application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/  
<application_name>/[<core_type>]/iar/<application_name>_ns/iar
```

```
<install_dir>/boards/<board_name>/trustzone_examples/  
<application_name>/[<core_type>]/iar/<application_name>_s/iar
```

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World IAR workspaces are located in this folder:

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_ns/iar/hello_world_ns.eww
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_s/iar/hello_world_s.eww
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_s/iar/hello_world.eww
```

This project `hello_world.eww` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another. Build both applications separately by clicking **Make**. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since the CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project since CMSE library is not ready.

4.4 Run a TrustZone example application

The secure project is configured to download both secure and non-secure output files, so debugging can be fully managed from the secure project. To download and run the TrustZone application, switch to the secure application project and perform steps 1 – 4 as described in *Section 4.2, Run an example application*. These steps are common for both single core, and TrustZone applications in IAR. After clicking **Download and Debug**, both the secure and non-secure image are loaded into the device memory, and the secure application is executed. It stops at the `Rest_Handler` function.

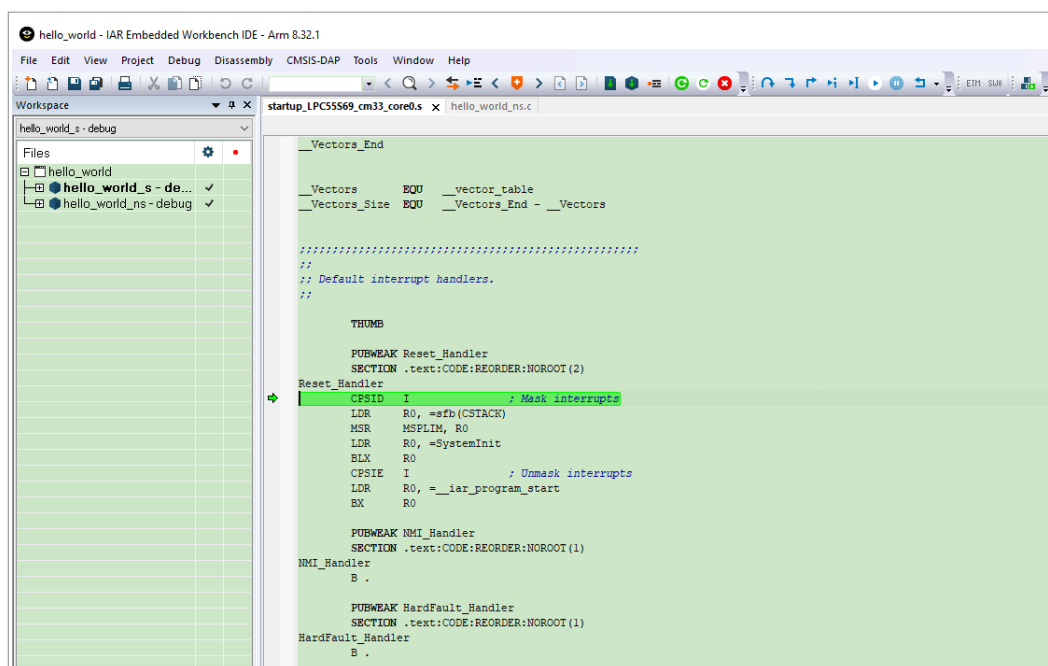


Figure 23. Stop at `Rest_Handler` when running debugging

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



Figure 24. Go button

The TrustZone `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.

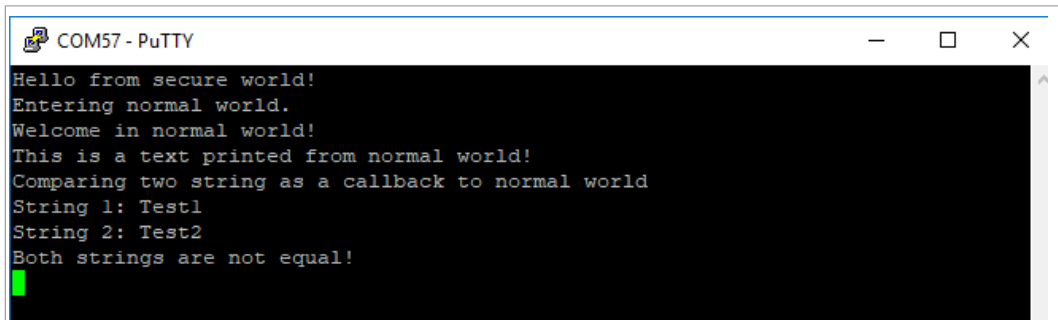


Figure 25. Text display of the trustzone `hello_world` application

Note: If the application is running in RAM (debug/release build target), in **Options > Debugger > Download** tab, disable **Use flash loader(s)**. This can avoid the `_ns` download issue on i.MXRT500.

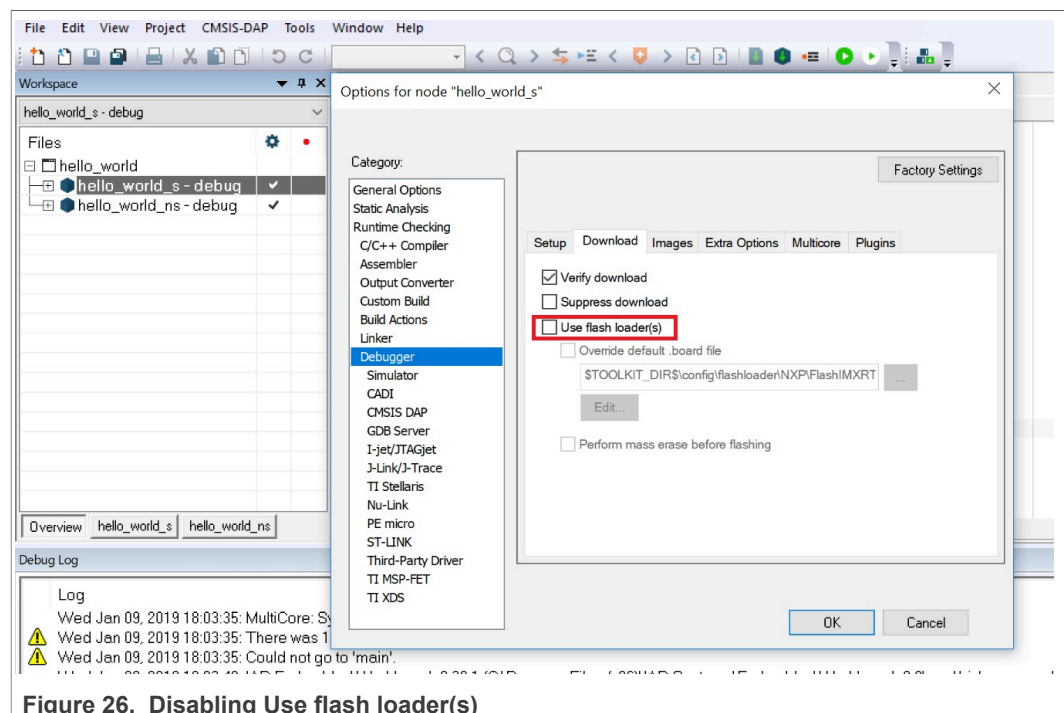


Figure 26. Disabling Use flash loader(s)

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 MIMXRT595-EVK hardware platform which is used as an example.

Note: ARMGCC version 7-2018-q2 is used as an example in this document. The latest GCC version for this package is as described in the MCUXpresso SDK Release Notes (document MCUXSDKMIMXRT5XXRN).

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.

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**.

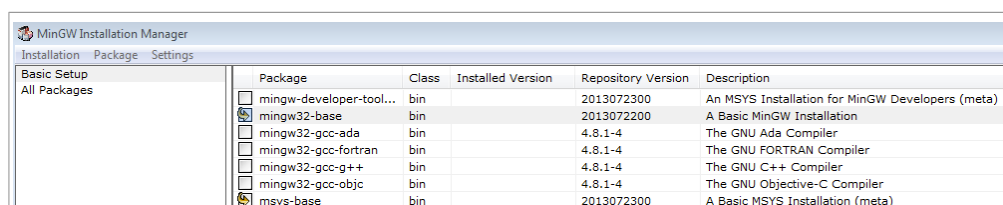
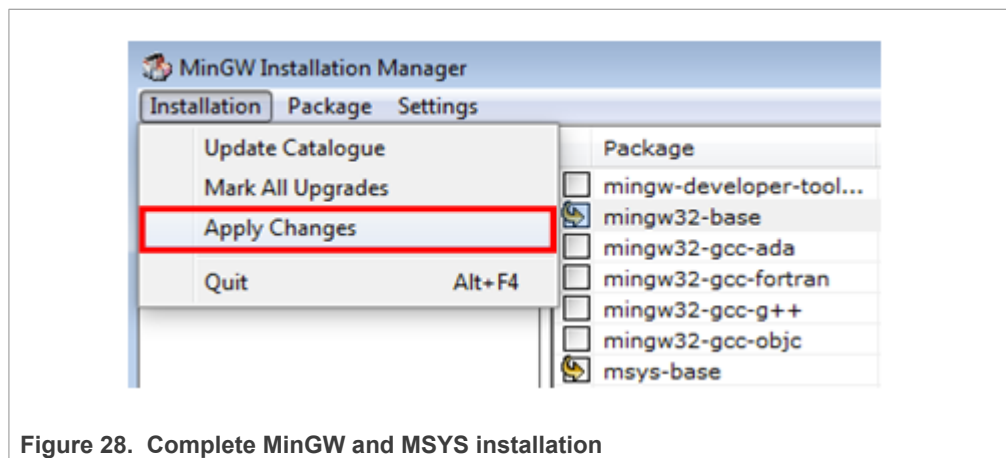


Figure 27. Set up MinGW and MSYS

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

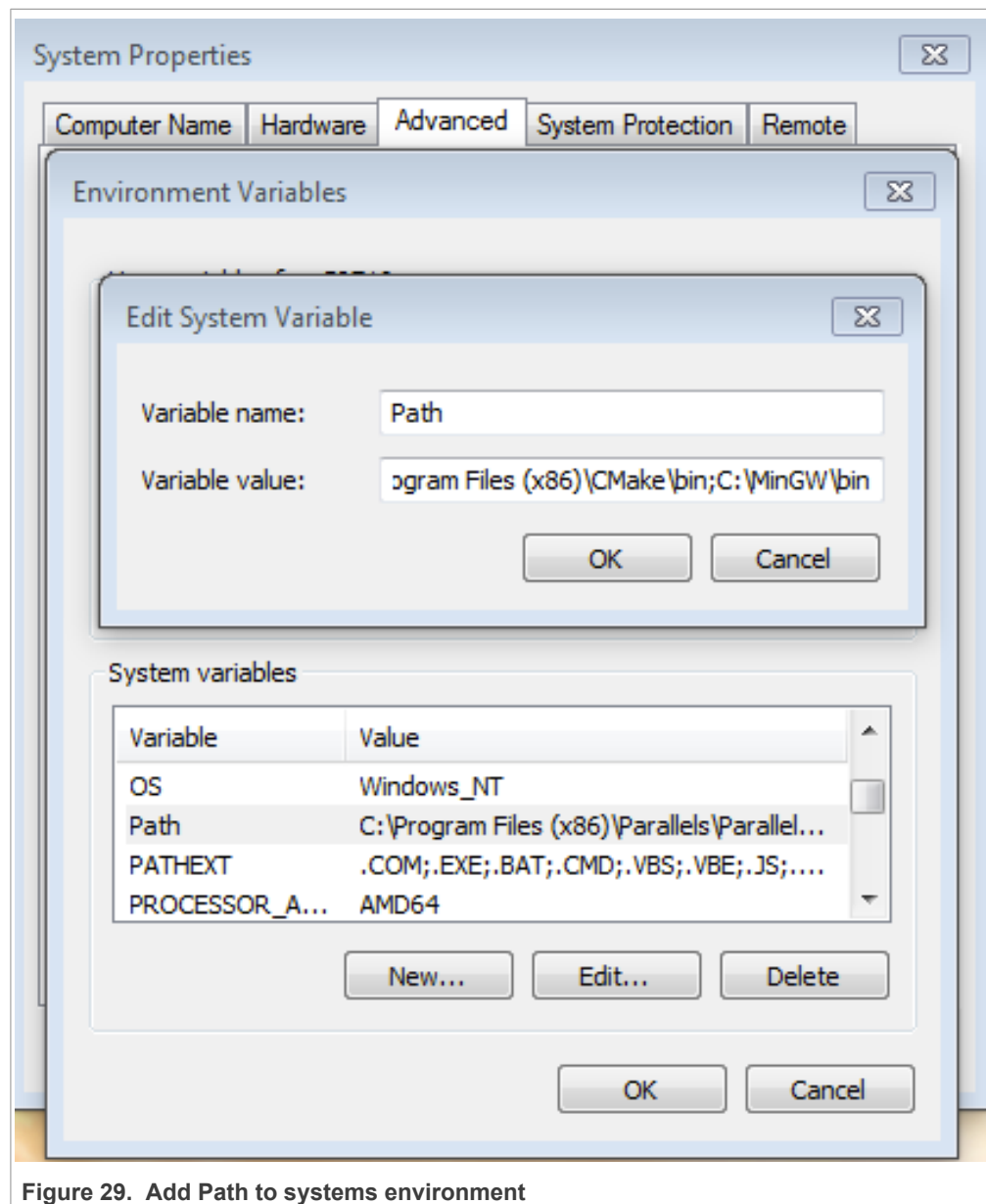


Figure 29. Add Path to systems environment

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

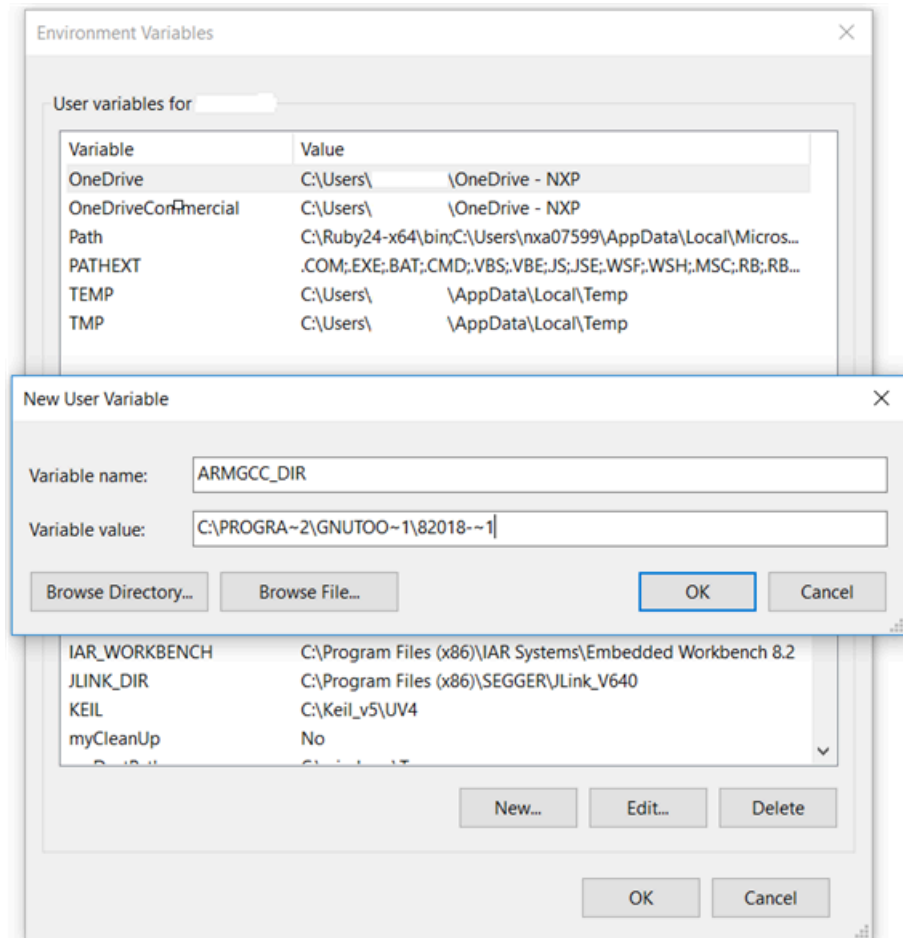


Figure 31. Add ARMGCC_DIR system variable

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**.

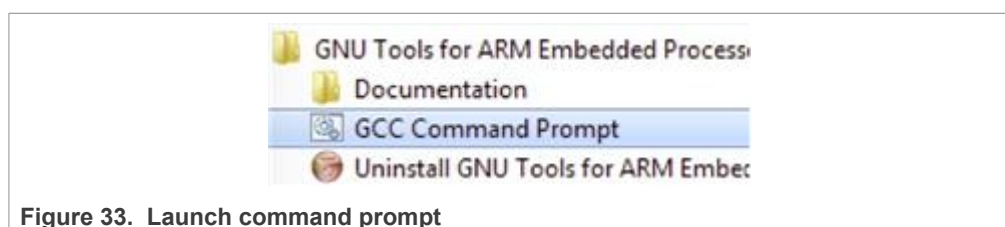


Figure 33. Launch command prompt

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

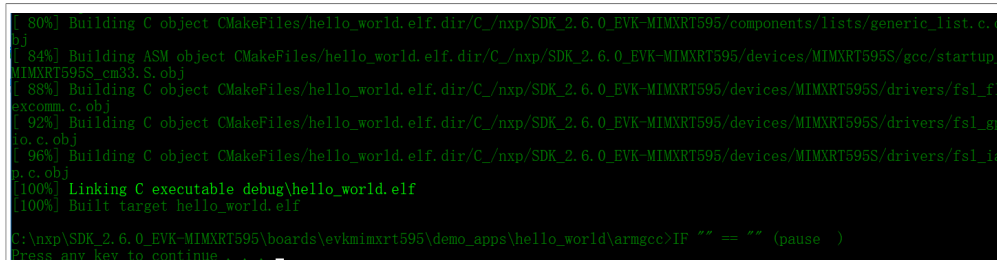
```
<install_dir>/boards/<board_name>/<example_type>/  
<application_name>/armgcc
```


For this example, the exact path is:

```
<install_dir>/examples/evkmimxrt595/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](#).



```
80%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c.o
84%] Building ASM object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startup_
MIMXRT595S_cm33.S.o
88%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_f
xcomm.c.o
92%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_g
io.c.o
96%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_i
a
n.c.o
100%] Linking C executable debug\hello_world.elf
100%] Built target hello_world.elf

C:/nxp/SDK_2.6.0_EVK-MIMXRT595/boards/evkmimxrt595/demo_apps/hello_world/armgcc/IF "" == "" (pause )
Press any key to continue
```

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 perform this exercise, make sure that either:

- The OpenSDA interface on your board is programmed with the J-Link OpenSDA firmware. To determine if your board supports OpenSDA, see [Section 11](#). For instructions on reprogramming the OpenSDA interface, see [Section 12](#). If your board does not support OpenSDA, a standalone J-Link pod is required.
- You have a standalone J-Link pod that is connected to the debug interface of your board. Note that some hardware platforms require hardware modification in order to function correctly with an external debug interface.

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

1. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
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 9](#)). Configure the terminal with these settings:
 - a. 115200 or 9600 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit

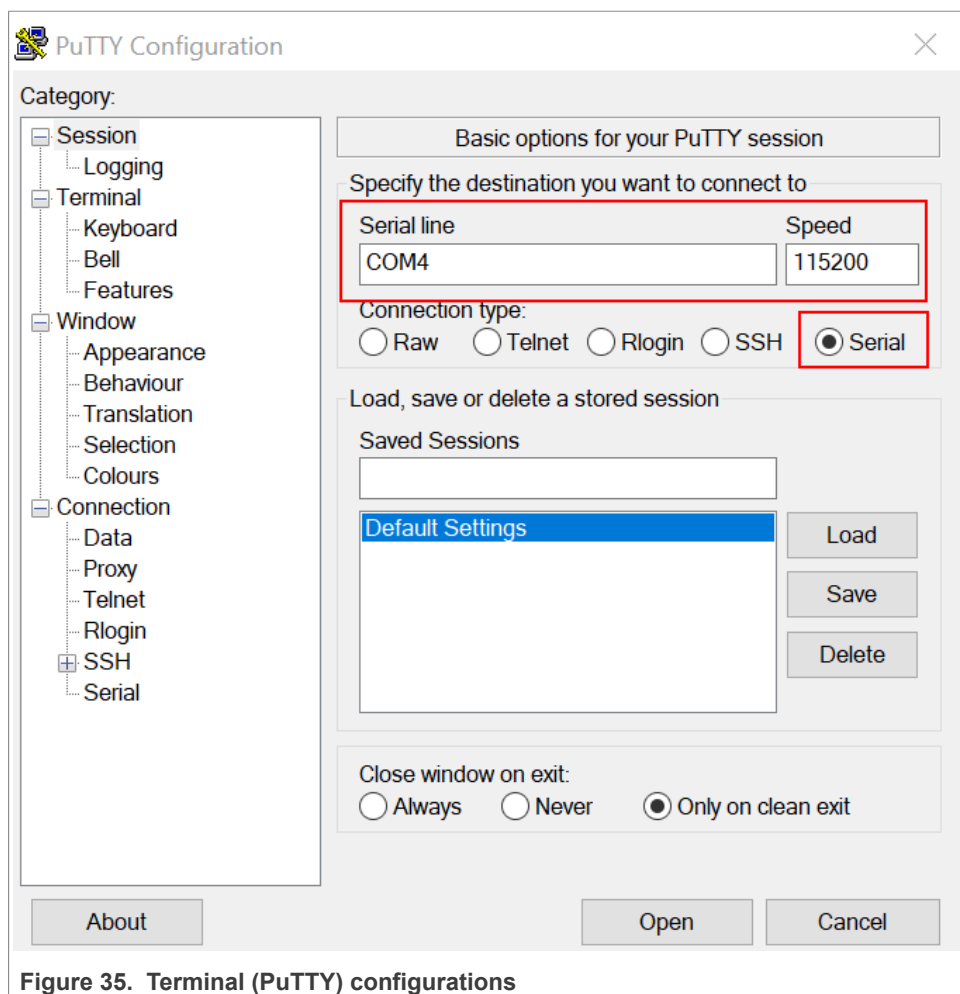


Figure 35. Terminal (PuTTY) configurations

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. Modify the settings as shown in [Figure 36](#). The target device selection chosen for this example is **MIMXRT595_M33**.

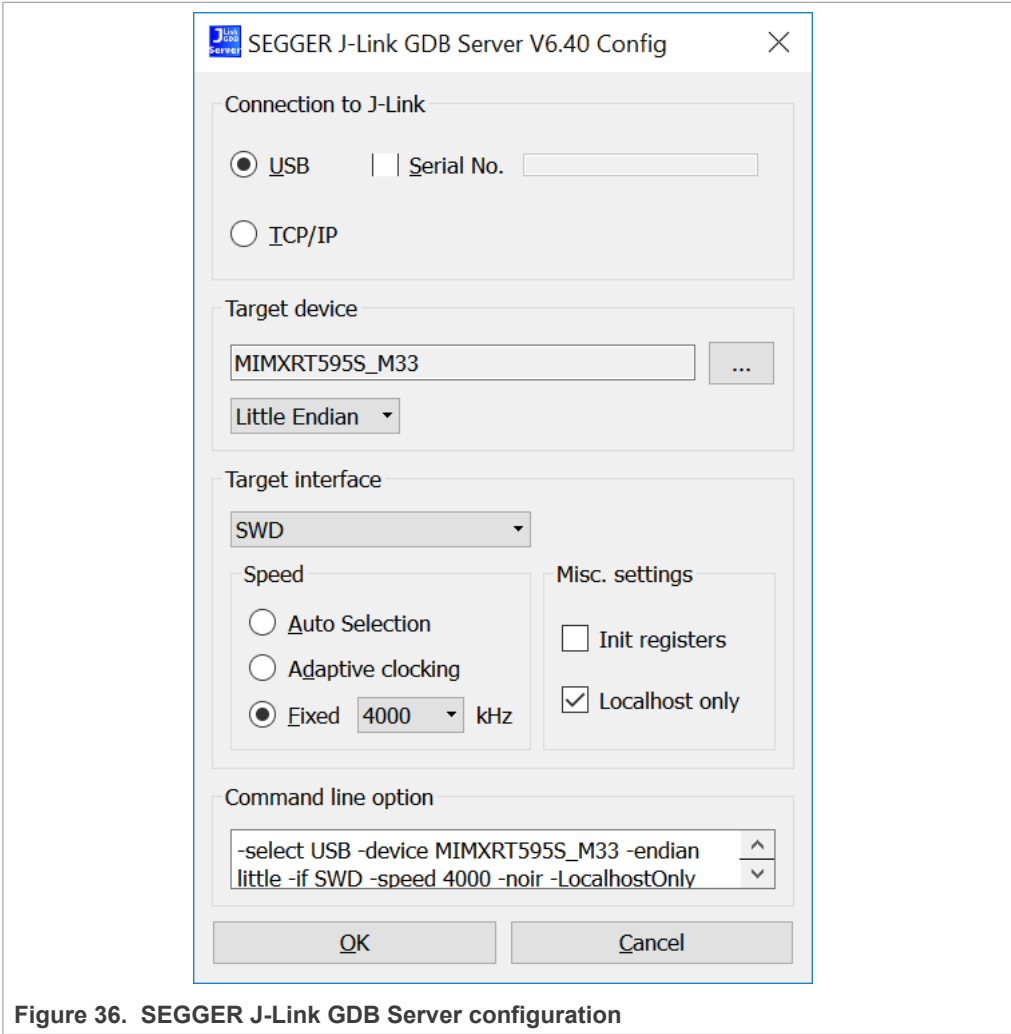


Figure 36. SEGGER J-Link GDB Server configuration

5. After it is connected, the screen should be as shown in [Figure 37](#).



Figure 37. 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 **Start** menu of the Windows operating system, go to **Programs->GNU Tools Arm Embedded <version>** and select **GCC Command Prompt**.

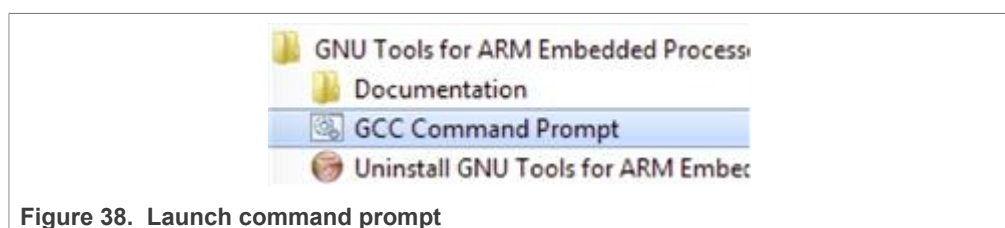
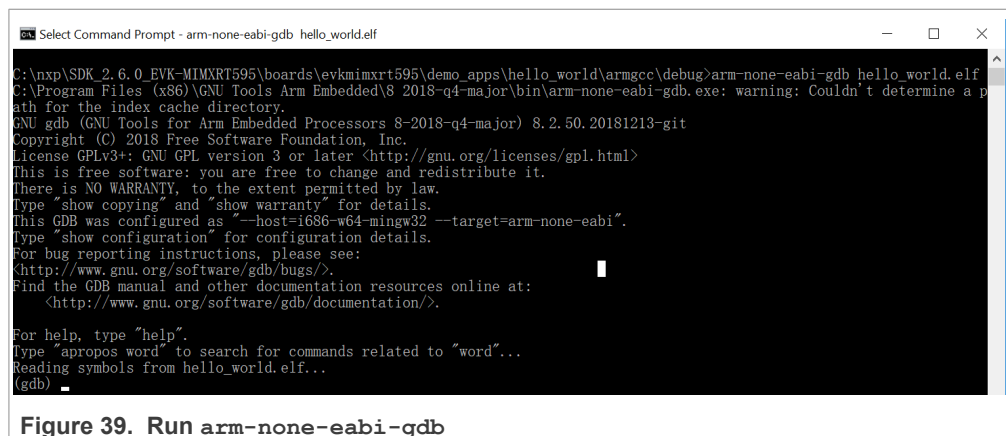


Figure 38. Launch command prompt

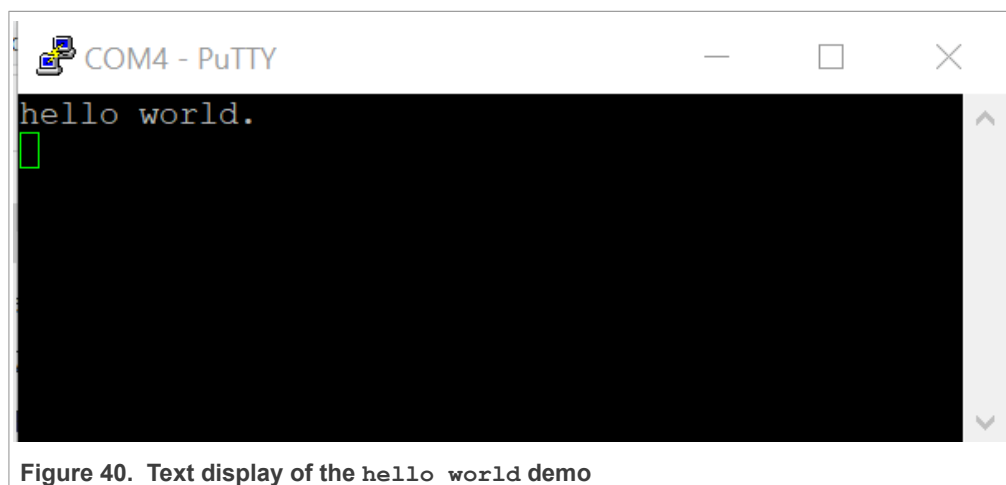
7. Change to the directory that contains the example application output. The output can be found in using one of these paths, depending on the build target selected:
`<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/debug`
`<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/release`
 For this example, the path is: `<install_dir>/boards/evkmimxrt595/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`.



Note: Make sure the board is set to FlexSPI flash boot mode before debugging.

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

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



5.4 Build a TrustZone example application

This section describes the steps to build and run a TrustZone application. The demo application build scripts are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/  
<application_name>/[<core_type>]/<application_name>_ns/armgcc
```

```
<install_dir>/boards/<board_name>/trustzone_examples/  
<application_name>/[<core_type>]/<application_name>_s/armgcc
```

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World GCC build scripts are located in this folder:

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_ns/armgcc/build_debug.bat
```

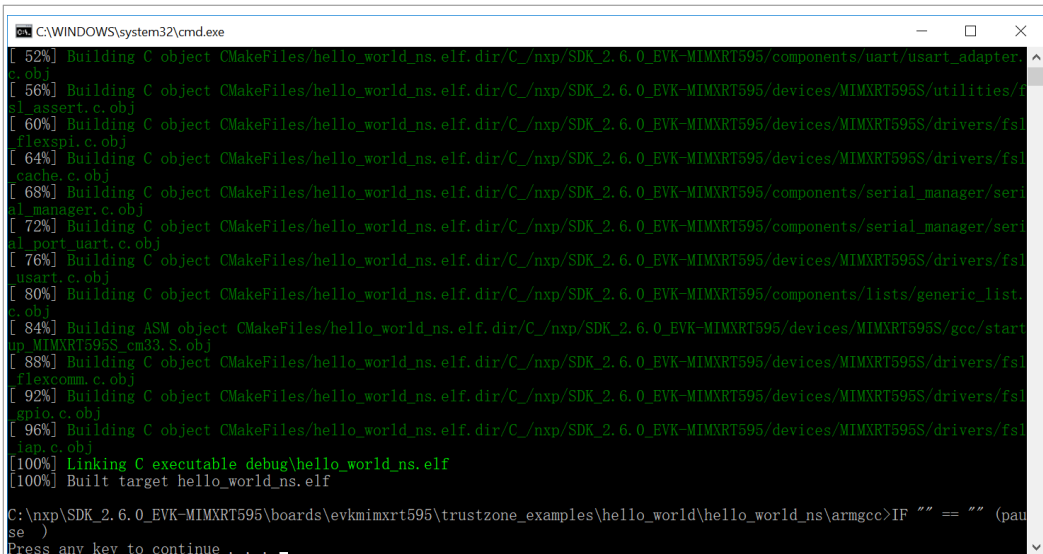
```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_s/armgcc/build_debug.bat
```

Build both applications separately, following steps for single core examples as described in [Section 6.2, "Build an example application"](#). It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project, since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because the CMSE library is not ready.

```
C:\WINDOWS\system32\cmd.exe
[ 55%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fs
_assert.c.obj
[ 59%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c
.obj
[ 62%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
lexspi.c.obj
[ 66%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
ache.c.obj
[ 70%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
_manager.c.obj
[ 74%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/seria
_port_uart.c.obj
[ 77%] Building ASM object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startu
p_MIMXRT595S_cm33.S.obj
[ 81%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c
.obj
[ 85%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
uart.c.obj
[ 88%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
lexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
pio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_s.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_
ap.c.obj
[100%] Linking C executable debug\hello_world_s.elf
[100%] Built target hello_world_s.elf

C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_s\armgcc>IF "" == "" (paus
e)
Press any key to continue . . .
```

Figure 41. hello_world_s example build successful



```

C:\WINDOWS\system32\cmd.exe
[ 52%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/uart/usart_adapter.c.obj
[ 56%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/utilities/fsl_assert.c.obj
[ 60%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexspi.c.obj
[ 64%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_cache.c.obj
[ 68%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_manager.c.obj
[ 72%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/serial_manager/serial_port_uart.c.obj
[ 76%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_usart.c.obj
[ 80%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/components/lists/generic_list.c.obj
[ 84%] Building ASM object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/gcc/startup_MIMXRT595S_cm33.S.obj
[ 88%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_flexcomm.c.obj
[ 92%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_gpio.c.obj
[ 96%] Building C object CMakeFiles/hello_world_ns.elf.dir/C:/nxp/SDK_2.6.0_EVK-MIMXRT595/devices/MIMXRT595S/drivers/fsl_iap.c.obj
[100%] Linking C executable debug\hello_world_ns.elf
[100%] Built target hello_world_ns.elf

C:\nxp\SDK_2.6.0_EVK-MIMXRT595\boards\evkmimxrt595\trustzone_examples\hello_world\hello_world_ns\armgcc>IF "" == "" (pause)
Press any key to continue . . .

```

Figure 42. hello_world_ns example build successful

5.5 Run a TrustZone example application

When running a TrustZone application, the same prerequisites for J-Link/J-Link OpenSDA firmware, and the serial console as for the single core application, apply, as described in [Section 6.3](#).

To download and run the TrustZone application, perform steps 1 to 10, as described in [Section 5.3](#). These steps are common for both single core and trustzone applications in Arm GCC.

Then, run these commands:

1. arm-none-eabi-gdb.exe
2. target remote localhost:2331
3. monitor reset
4. monitor halt
5. monitor exec SetFlashDLNoRMWThreshold = 0x20000
6. load <install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_ns/armgcc/debug/hello_world_ns.elf
7. load <install_dir>/boards/evkmimxrt595/trustzone_examples/hello_world/hello_world_s/armgcc/debug/hello_world_s.elf
8. monitor reset

The application is now downloaded and halted. Execute the `c` command to start the demo application.

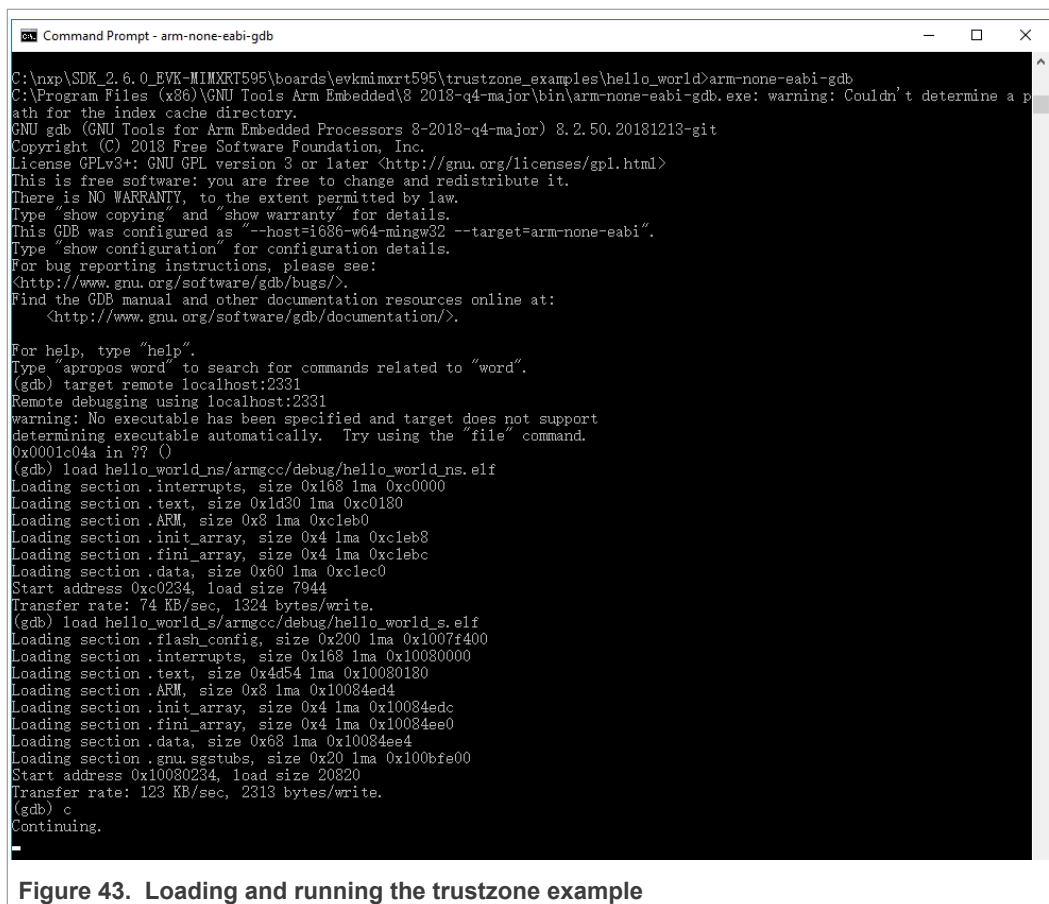


Figure 43. Loading and running the trustzone example

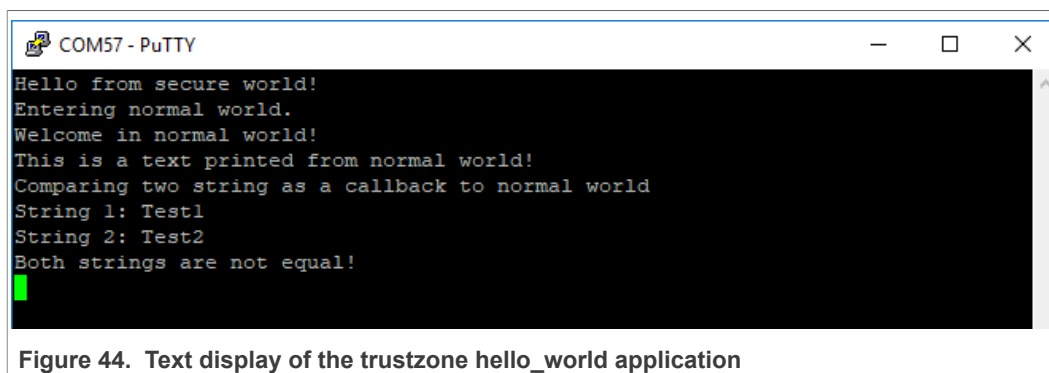


Figure 44. Text display of the trustzone hello_world application

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 MIMXRT595S CMSIS pack.

1. Download the MIMXRT595S CMSIS 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/evkmimxrt595s/demo_apps/hello_world/mdk/  
hello_world.uvmpw
```

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

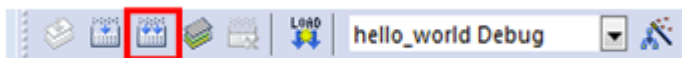


Figure 45. Build the demo

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 11](#) 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 9](#)). Configure the terminal with these settings:
 - a. 115200 or 9600 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in the board.h file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit

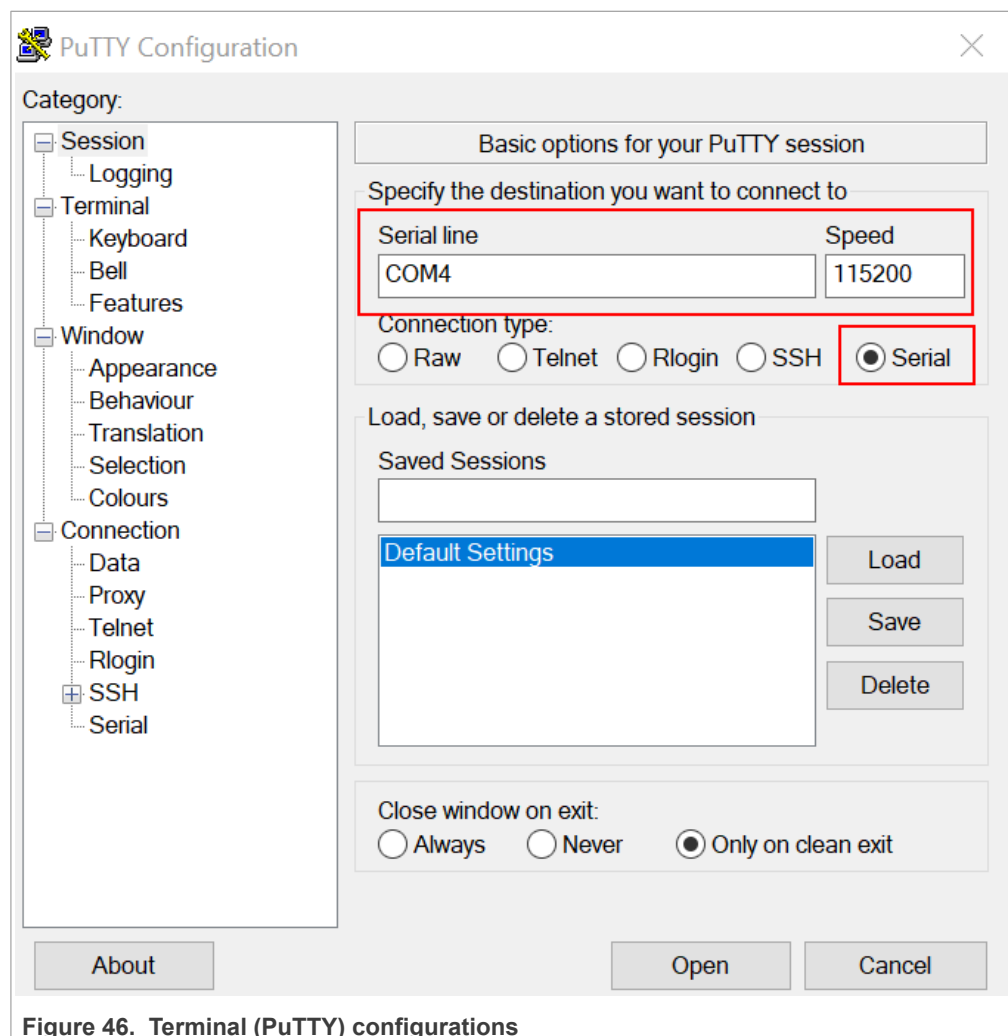


Figure 46. Terminal (PuTTY) configurations

4. To debug the application, click **load** (or press the F8 key). Then, click the **Start/Stop Debug Session** button, highlighted in red in [Figure 47](#). If using **J-Link** as the debugger, click **Project option > Debug > Settings > Debug > Port**, and select **SW**.
Note: When debugging with *jlink*, it expects one *jlinkscript* file named *JLinkSettings.JLinkScript* in the folder where the *uVision* project files are located. For details, see *Segger Wiki*. For the contents in this *JLinkScript*, use contents in *evkmimxrt1020_sdram_init.jlinkscript*.

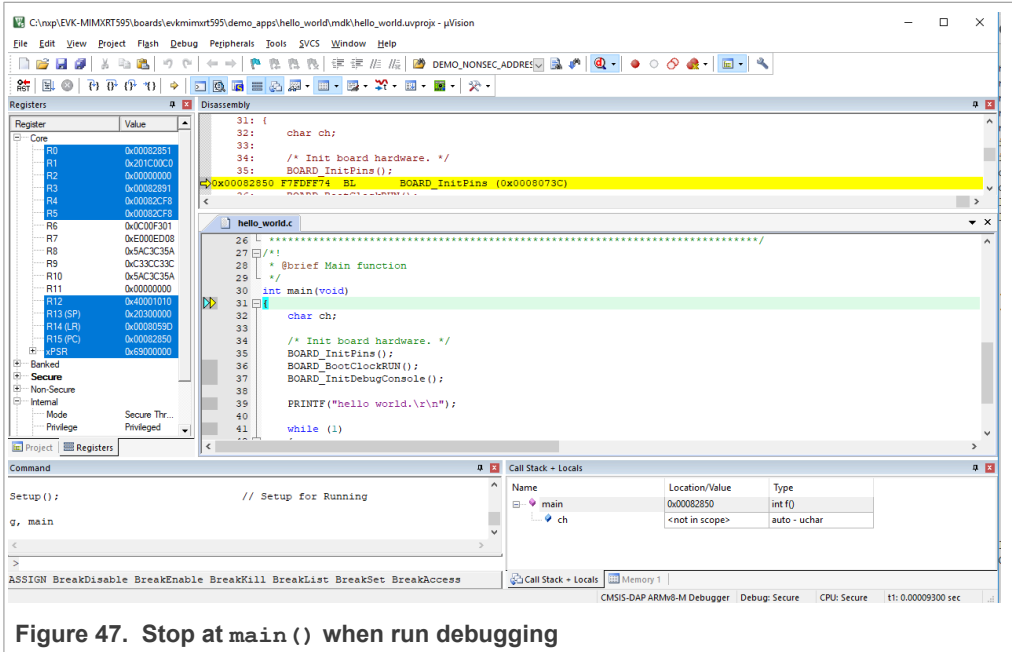


Figure 47. Stop at main () when run debugging

- Note:** Make sure the board is set to FlexSPI flash boot mode before debugging.
5. Run the code by clicking **Run** to start the application, as shown in [Figure 48](#).

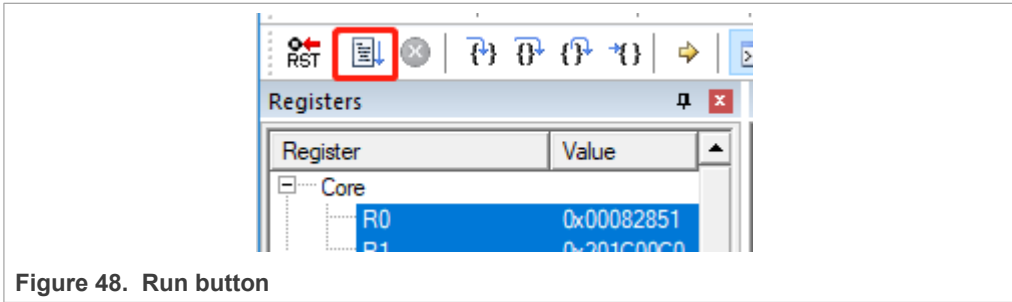


Figure 48. Run button

The `hello_world` application is now running and a banner is displayed on the terminal, as shown in [Figure 49](#). If this is not true, check your terminal settings and connections.

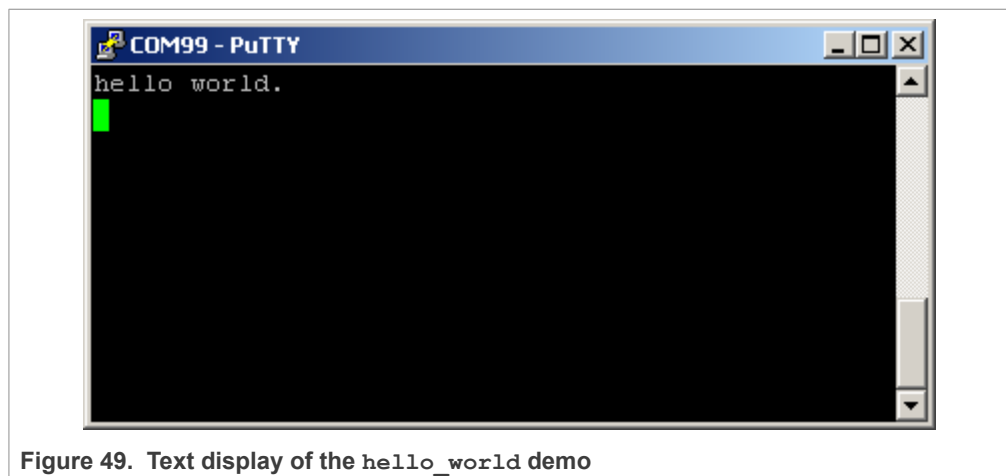


Figure 49. Text display of the `hello_world` demo

6.4 Build a TrustZone example application

This section describes the particular steps that need to be done in order to build and run a TrustZone application. The demo applications workspace files are located in this folder:

```
<install_dir>/boards/<board_name>/trustzone_examples/  
<application_name>/<application_name>_ns/mdk
```

```
<install_dir>/boards/<board_name>/trustzone_examples/  
<application_name>/<application_name>_s/mdk
```

Begin with a simple TrustZone version of the Hello World application. The TrustZone Hello World Keil MSDK/ μ Vision[®] workspaces are located in this folder:

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_ns/mdk/hello_world_ns.uvmpw
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_s/mdk/hello_world_s.uvmpw
```

```
<install_dir>/boards/evkmimxrt595/trustzone_examples/  
hello_world/hello_world_s/mdk/hello_world.uvmpw
```

This project `hello_world.uvmpw` contains both secure and non-secure projects in one workspace and it allows the user to easily transition from one project to another.

Build both applications separately by clicking **Rebuild**. It is requested to build the application for the secure project first, because the non-secure project needs to know the secure project since CMSE library is running the linker. It is not possible to finish the non-secure project linker with the secure project because CMSE library is not ready.

6.5 Run a TrustZone example application

The secure project is configured to download both secure and non-secure output files so debugging can be fully managed from the secure project.

To download and run the TrustZone application, switch to the secure application project and perform steps as described in [Section 6.5](#). These steps are common for single core, dual-core, and TrustZone applications in μ Vision. After clicking **Download and Debug**, both the secure and non-secure image are loaded into the device flash memory, and the secure application is executed. It stops at the `main()` function.

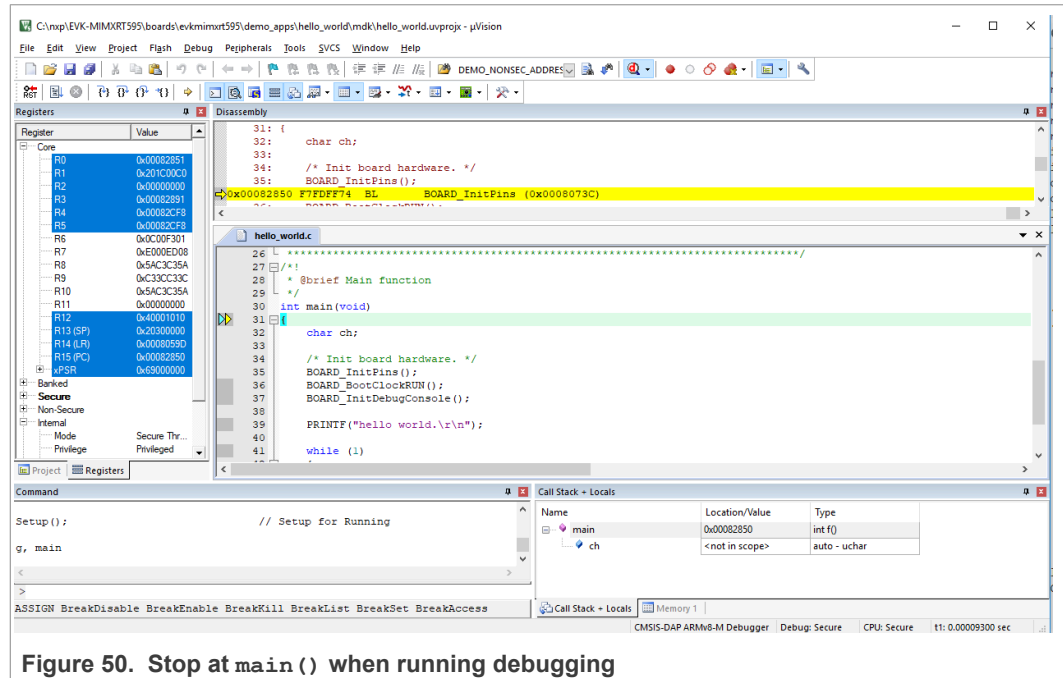


Figure 50. Stop at `main()` when running debugging

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

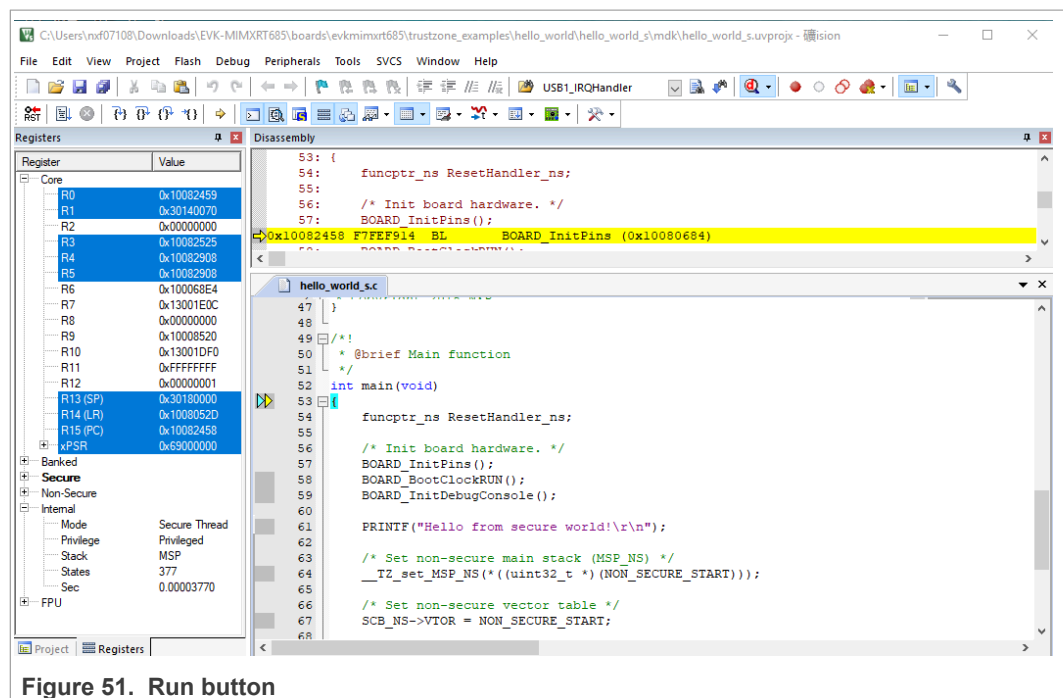


Figure 51. Run button

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.

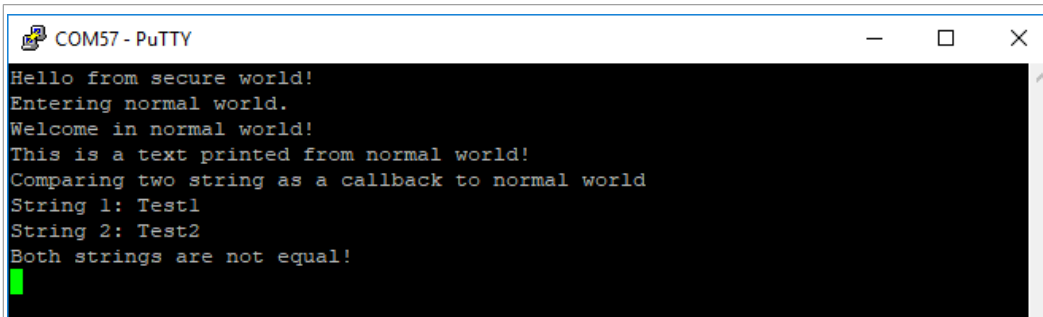


Figure 52. Text display of the trustzone `hello_world` application

7 Run a demo using the prebuilt binary

This section describes the steps to write a prebuilt binary in the SDK package to the external Flash and boot from the external Flash. The `hello_world` demo application is used as an example.

Note: The prebuilt binaries are provided for quick evaluation of the SDK and the board. Only parts of the examples are provided with prebuilt binary such as examples under `boards/<board_name>/demo_apps` and `boards/<board_name>/usb_examples`.

7.1 Identify the load address of the binary

The MCUXpresso SDK supports several toolchains. The prebuilt binaries are built from the `armgcc flash_release` targets using Arm GCC. If there is no `flash_release` target, then the `release` target is used.

For example, the `hello_world` demo application located in `<install_dir>/boards/<board_name>/<demo_apps>/hello_world/`.

The example was built from `<install_dir>/boards/<board_name>/<demo_apps>/hello_world/armgcc/build_flash_release.sh`.

The linker file in the same folder identifies the load address. For example, `<device_name>_flash.ld`.

```
MEMORY
{
  m_flash_config      (RX) : ORIGIN = 0x08000400, LENGTH = 0x00000200
  m_interrupts        (RX) : ORIGIN = 0x08001000, LENGTH = 0x00000130
  m_text              (RX) : ORIGIN = 0x08001130, LENGTH = 0x001FEED0
  m_data              (RW) : ORIGIN = 0x20080000, LENGTH = 0x00180000
  m_usb_sram          (RW) : ORIGIN = 0x40140000, LENGTH = 0x00004000
}
```

Figure 53. Memory layout in the linker

Therefore, the load address for `hello_world.bin` is `0x08000400`.

7.2 Write the binary to external flash

To write the binary to external flash:

1. To write the hello_world.bin to external flash, take J-Link Commander as an example.
2. Run the J-Link Commander and to connect the core run the following command.

```
J-Link>connect
J-Link>?
```

3. Select the correct device from the pop-up window **Target device settings**.
4. Specify targets interface as SWD and specify target interface speed per the guide in the command window.

The "Cortex-M33 identified" appears when it is connected successfully.

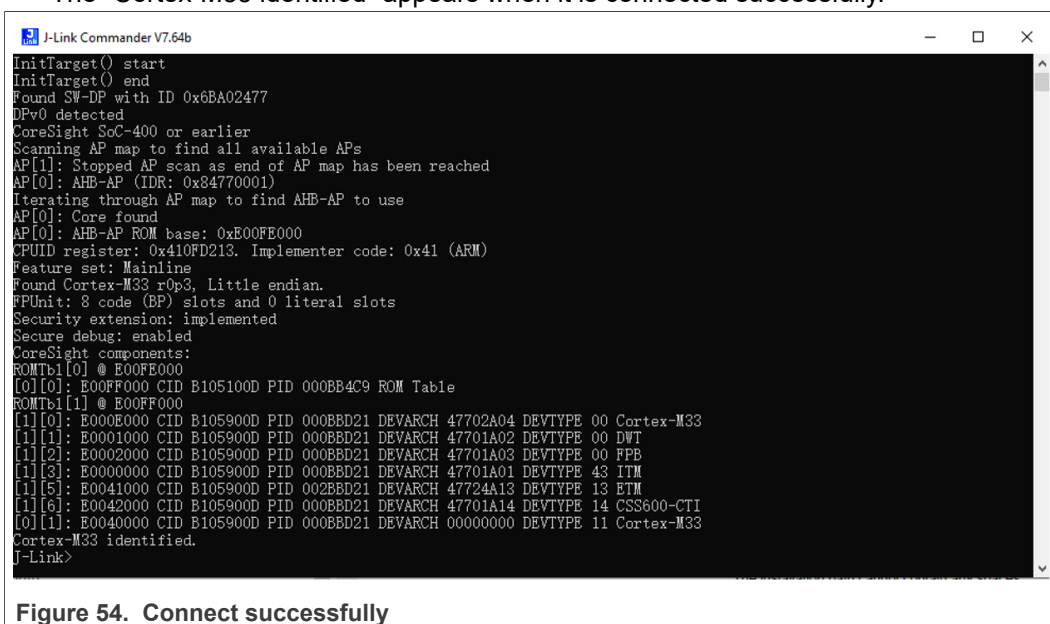


Figure 54. Connect successfully

5. Use below command to write the hello_world.bin to the load address 0x08000400.

```
J-Link>loadbin hello_world.bin 0x08000400
```

7.3 Run the application

Either press the reset button on the board or use the following command in J-Link Commander to run the demo.

```
J-Link>rn
```

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 55](#).



Figure 55. 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.

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 implemented 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. [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 MCUXpresso 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/mbd/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/mbd/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/mbd/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

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

Hardware platform	Default interface	OpenSDA details
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/mbd/DAPLink	OpenSDA v2.1
FRDM-KW36	DAPLink	OpenSDA v2.2
FRDM-KW41Z	CMSIS-DAP/DAPLink	OpenSDA v2.1 or greater
Hexiwear	CMSIS-DAP/mbd/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
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
TWR-K21D50M	P&E Micro OSJTAG	N/AOpenSDA v2.0
TWR-K21F120M	P&E Micro OSJTAG	N/A
TWR-K22F120M	P&E Micro OpenSDA	OpenSDA v1.0
TWR-K24F120M	CMSIS-DAP/mbd	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

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

Hardware platform	Default interface	OpenSDA details
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
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

12 Updating debugger firmware

12.1 Updating LPCXpresso board firmware

The LPCXpresso 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 LPCScript. 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 LPCScript utility, which is the recommended tool for programming the latest versions of CMSIS-DAP and J-Link firmware onto LPC-Link2 or LPCXpresso boards. The utility can be downloaded from www.nxp.com/lpcutilities.

These steps show how to update the debugger firmware on your board for Windows operating system. For Linux OS, follow the instructions described in LPCScript user guide (www.nxp.com/lpcutilities, select **LPCScript**, and then the documentation tab).

1. Install the LPCScript utility.
2. Unplug the board's USB cable.
3. Make the DFU link (install the jumper labelled DFULink).
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 LPCScript installation directory (<LPCScript 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.

13 Revision history

This table summarizes the revisions to this document.

Revision history

Revision number	Date	Substantive changes
0	20 December 2020	Initial release
2.10.0	10 July 2021	Updated for MCUXpresso SDK v2.10.0
2.10.1	09 September 2021	Updated for MCUXpresso SDK v2.10.1
2.11.0	11 November 2021	Updated for MCUXpresso SDK v2.11.0
2.11.1	11 March 2022	Updated for MCUXpresso SDK v2.11.1
2.12.0	30 June 2022	Updated for MCUXpresso SDK v2.12.0

14 Legal information

14.1 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.

14.2 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 <http://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 data sheet 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.

14.3 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.

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	4
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	6
3.4	Build a TrustZone example application	11
3.5	Run a TrustZone example application	13
4	Run a demo application using IAR	14
4.1	Build an example application	14
4.2	Run an example application	15
4.3	Build a TrustZone example application	17
4.4	Run a TrustZone example application	18
5	Run a demo using Arm® GCC	19
5.1	Set up toolchain	20
5.1.1	Install GCC Arm Embedded tool chain	20
5.1.2	Install MinGW (only required on Windows OS)	20
5.1.3	Add a new system environment variable for ARMGCC_DIR	22
5.1.4	Install CMake	23
5.2	Build an example application	24
5.3	Run an example application	25
5.4	Build a TrustZone example application	30
5.5	Run a TrustZone example application	31
6	Run a demo using Keil® MDK/μVision	32
6.1	Install CMSIS device pack	33
6.2	Build an example application	33
6.3	Run an example application	33
6.4	Build a TrustZone example application	36
6.5	Run a TrustZone example application	36
7	Run a demo using the prebuilt binary	38
7.1	Identify the load address of the binary	38
7.2	Write the binary to external flash	38
7.3	Run the application	39
8	MCUXpresso IDE New Project Wizard	39
9	How to determine COM port	40
10	How to define IRQ handler in CPP files	40
11	Default debug interfaces	41
12	Updating debugger firmware	43
12.1	Updating LPCXpresso board firmware	43
13	Revision history	44
14	Legal information	45

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

© NXP B.V. 2022.

All rights reserved.

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

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

Date of release: 30 June 2022

Document identifier: MCUXSDKMIMXRT5XXGSUG