

# Getting Started with MCUXpresso SDK for EVK-MIMX8DXL



# Contents

<b>Chapter 1 Overview.....</b>	<b>3</b>
<b>Chapter 2 MCUXpresso SDK board support folders.....</b>	<b>4</b>
2.1 Example application structure.....	4
2.2 Locating example application source files.....	5
<b>Chapter 3 Toolchain introduction.....</b>	<b>6</b>
3.1 Compiler/Debugger.....	6
3.2 Image creator.....	6
<b>Chapter 4 Run a demo application using IAR.....</b>	<b>7</b>
4.1 Build an example application.....	7
4.2 Run an example application.....	8
<b>Chapter 5 Run a demo using Arm<sup>®</sup> GCC.....</b>	<b>12</b>
5.1 Linux OS host.....	12
5.2 Windows OS host.....	16
5.3 GDBServer config file.....	21
<b>Chapter 6 Run a demo using imx-mkimage.....</b>	<b>22</b>
6.1 Run an example application on the M4 core.....	22
6.2 Make a bootable SD card with System Controller Firmware (SCFW).....	23
6.3 Run an example application on the M4 core together with U-Boot.....	23
<b>Chapter 7 Run a demo using facility provided by U-Boot.....</b>	<b>24</b>
<b>Chapter 8 Run a flash target demo by UUU.....</b>	<b>25</b>
8.1 Set up environment.....	25
8.2 Build an example application.....	26
8.3 Run an example application.....	28
<b>Appendix A How to determine COM port.....</b>	<b>30</b>
<b>Appendix B Host setup.....</b>	<b>32</b>

# Chapter 1

## Overview

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides bare metal source code to be executed in the i.MX 8DualXLite M4 core. The MCUXpresso SDK provides comprehensive software support for NXP i.MX 8DualXLite microcontrollers' M4 cores. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications which can be used standalone or collaboratively with the A cores running another Operating System (such as Linux<sup>®</sup> Kernel). 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 demo applications. The MCUXpresso SDK also contains FreeRTOS, and various other middleware to support rapid development.

For supported toolchain versions, see the *MCUXpresso SDK Release Notes for EVK-MIMX8DXL* (document MCUXSDKIMX8DXLRN).

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

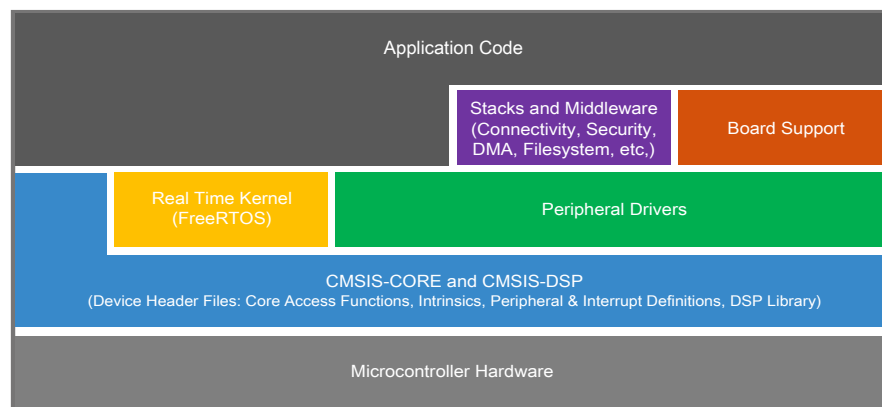


Figure 1. MCUXpresso SDK layers

## Chapter 2

# MCUXpresso SDK board support folders

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

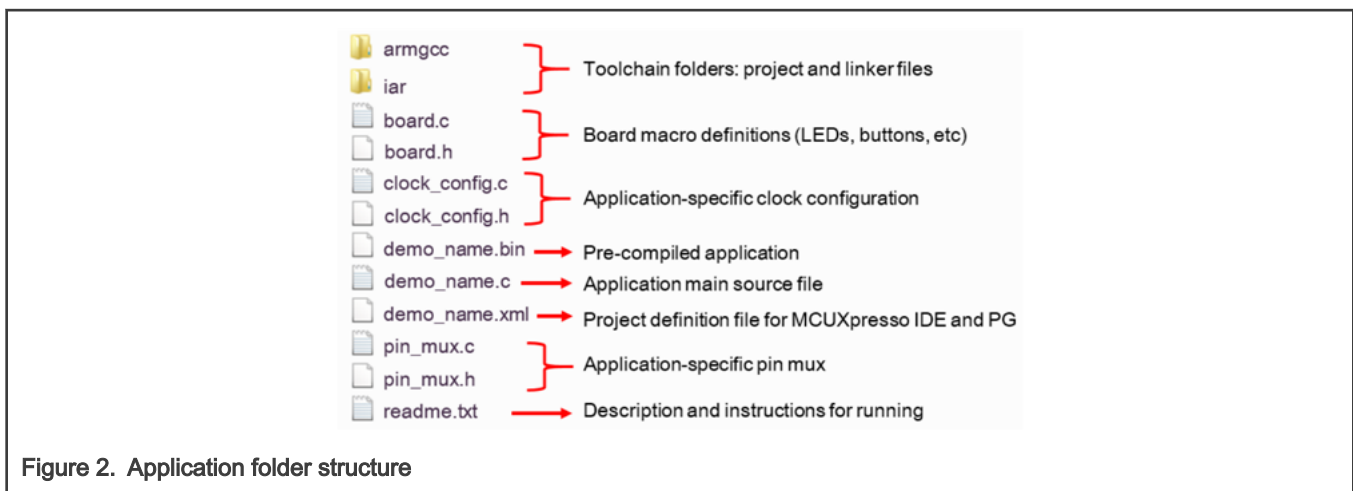
- *cmsis\_driver\_examples*: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- *demo\_apps*: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- *driver\_examples*: Simple applications intended to concisely illustrate how to use the MCUXpresso SDK's peripheral drivers for a single use case.
- *rtos\_examples*: Basic FreeRTOS™ OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers
- *multicore\_examples*: Simple applications intended to concisely illustrate how to use middleware/multicore stack.
- *lwip\_examples*: Demos to demonstrate support for LWIP TCP/IP stack.

### 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>/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>/scfw\_api*: APIs to invoke SCFW calls, let SCU do service on clock, power, and resource permission

For examples containing middleware/stacks or an RTOS, there are references to the appropriate source code. Middleware source files are located in the *middleware* folder and 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.

# Chapter 3

## Toolchain introduction

The MCUXpresso SDK release for i.MX 8DualX Lite includes the build system to be used with some toolchains. In this chapter, the toolchain support is presented and detailed.

### 3.1 Compiler/Debugger

The MCUXpresso SDK i.MX 8DualX Lite release supports building and debugging with the toolchains listed in [Table 1](#).

The user can choose the appropriate one for development.

- Arm GCC + SEGGER J-Link GDB Server. This is a command line tool option and it supports both Windows® OS and Linux® OS.
- IAR Embedded Workbench® for Arm and SEGGER J-Link software. The IAR Embedded Workbench is an IDE integrated with editor, compiler, debugger, and other components. The SEGGER J-Link software provides the driver for the J-Link Plus debugger probe and supports the device to attach, debug, and download.

Table 1. Toolchain information

Compiler/Debugger	Supported host OS	Debug probe	Tool website
ArmGCC/J-Link GDB server	Windows OS/Linux OS	J-Link Plus	<a href="https://developer.arm.com/open-source/gnu-toolchain/gnu-rm">developer.arm.com/open-source/gnu-toolchain/gnu-rm</a> <a href="http://www.segger.com">www.segger.com</a>
IAR/J-Link	Windows OS	J-Link Plus	<a href="http://www.iar.com">www.iar.com</a> <a href="http://www.segger.com">www.segger.com</a>

Download the corresponding tools for the specific host OS from the website.

#### NOTE

To support i.MX 8DualX Lite, the patch for IAR and Segger J-Link should be installed. The patch named *iar\_segger\_support\_patch\_imx8dxl.zip* can be downloaded from [NXP page](#). See the *readme.txt* in the patch for additional information about patch installation.

### 3.2 Image creator

The i.MX 8DualX Lite hardware is developed to only allow the boot if the SCFW firmware is properly installed. The `imx-mkimage` tool is used to combine the SCFW firmware with SDK images or U-Boot and to generate a binary to be used for i.MX 8DualX Lite device. Currently, the tool can only be executed on Linux OS.

## Chapter 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 using IAR. The `hello_world` demo application targeted for the 8DualXLite EVK board is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK

### 4.1 Build an example application

Before using IAR, get the IAR and Segger J-Link patch, *iar\_segger\_support\_patch\_imx8dxl.zip*. Install the .MX8DXL support patch following the guides in `readme.txt` located in the archive.

Perform 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 i.MX 8DualXLite EVK board as an example, the `hello_world` workspace is located in:

`<install_dir>/boards/evkmimx8dxl/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**.

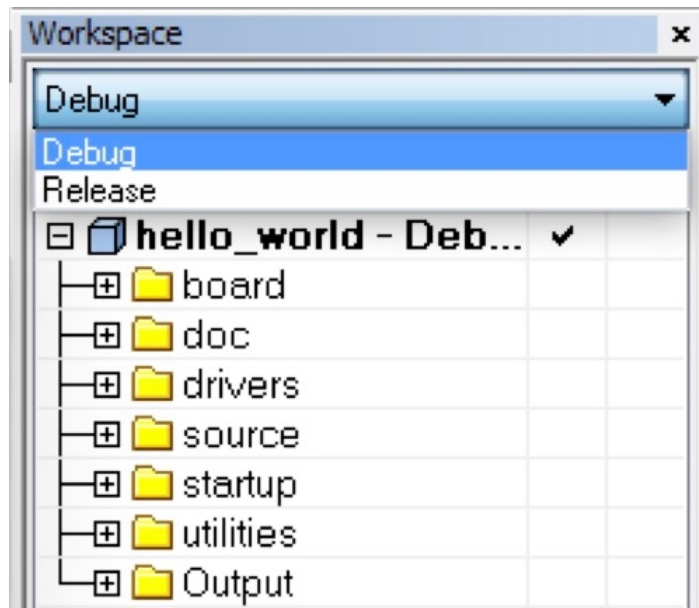


Figure 3. Demo build target selection

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

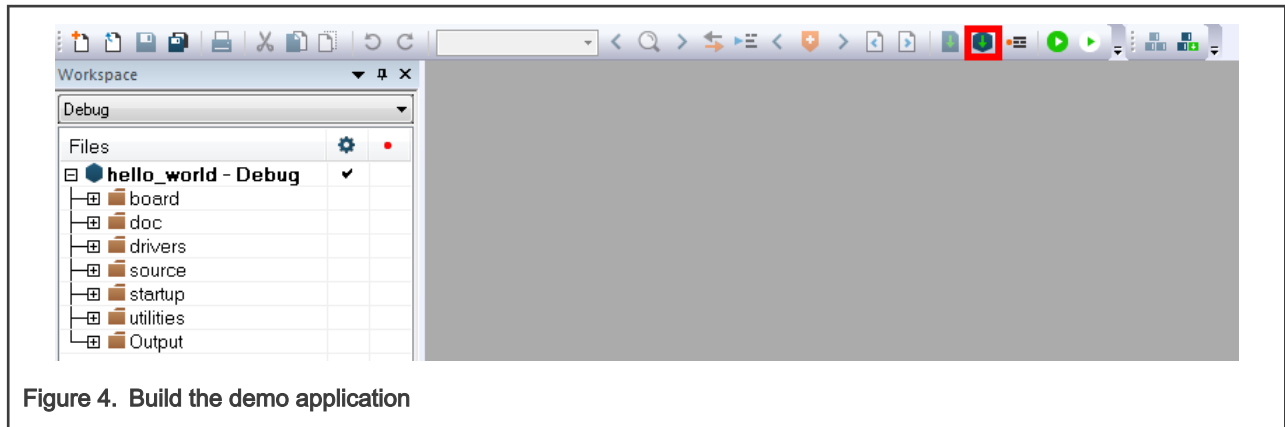


Figure 4. Build the demo application

4. The build completes without errors. There will be an *elf* file with *out* extension and a binary file with *bin* generated in the target directory.

## 4.2 Run an example application

Before running an example, a bootable SD card with the System Controller FirmWare (SCFW) image is needed. See [Make a bootable SD card with System Controller Firmware \(SCFW\)](#).

To download and run the application, perform these steps:

1. This board supports the J-Link PLUS debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from [Jlink](#).
2. Connect the development platform to your PC via USB cable between the USB-UART Micro USB connector and the PC USB connector, then connect 12 V power supply and J-Link Plus to the hardware platform.
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 [How to determine COM port](#)). Configure the terminal with these settings:
  - a. 115200 baud rate
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit



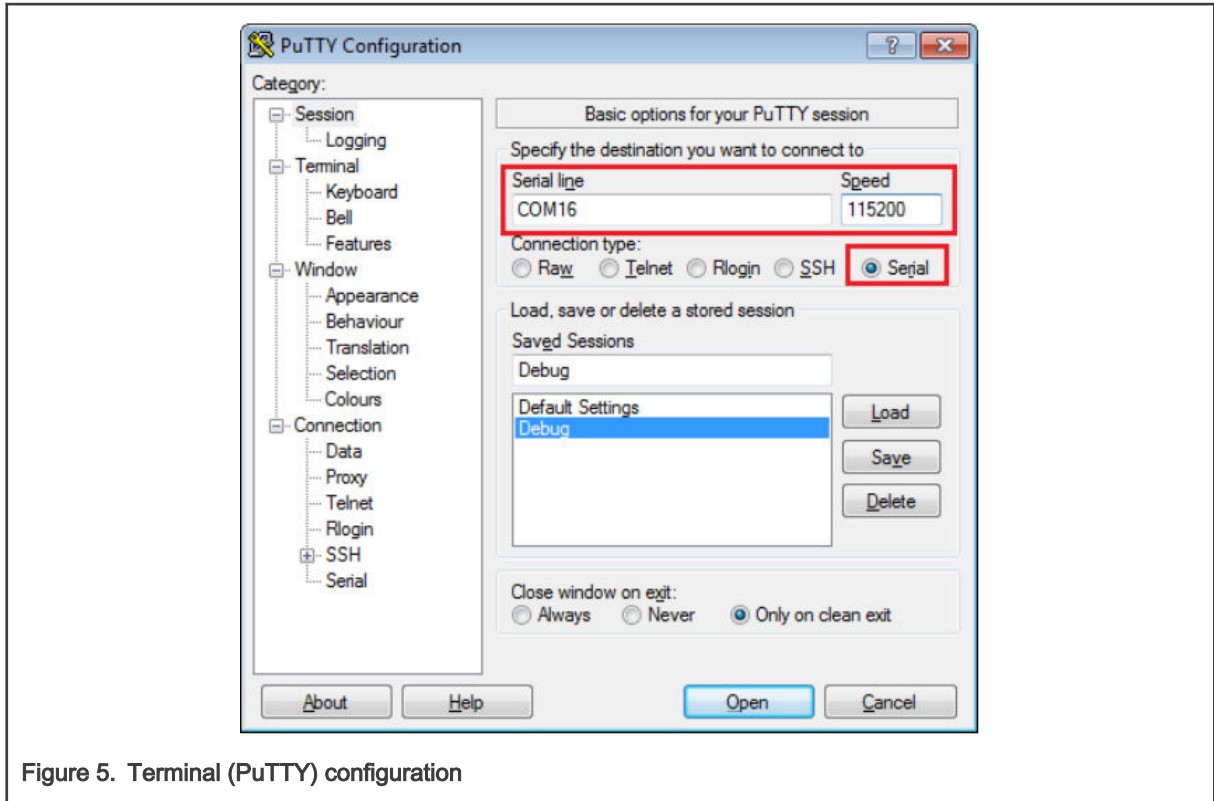


Figure 5. Terminal (PuTTY) configuration

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

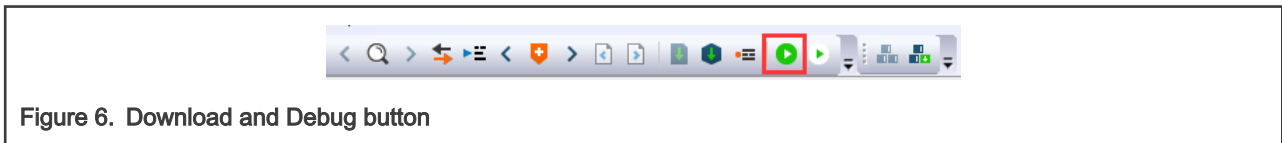


Figure 6. Download and Debug button

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

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

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



Figure 8. Go button

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

Figure 9. Text display of the `hello_world` demo

#### 4.2.1 C-Spy macros for DDR Target

For `ddr_debug` and `ddr_release` target, the link address for `text` section is aliased from DDR memory map to XIP memory map. The purpose is to better utilize the Harvard bus architecture: use code bus to fetch instruction, meanwhile use system bus to fetch data, achieving a better overall performance.

The alias is achieved by setting **MCM\_PID** to **0x7E**. The M4 startup code handles this so it is transparent when running the SDK demo from bootloader, see [Run a demo using imx-mkimage](#).

But when using IAR to download and run the demo, IAR should be responsible to set the **MCM\_PID**. It is achieved by C-Spy macros which is executed automatically before IAR download the image.

The C-Spy macros file is `boards\evkmimx8dxl\evkmimx8dxl_ddr_xip_init.mac`. The IAR project option will specify it in debugger setup settings, as shown in [Figure 10](#).

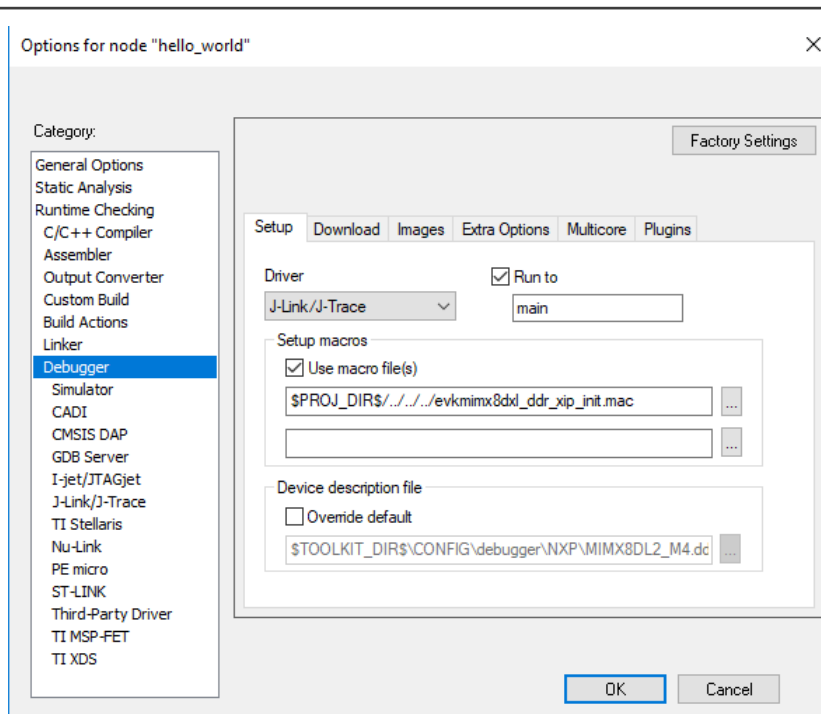


Figure 10. C-Spy macros are specified in IAR project settings

# Chapter 5

## Run a demo using Arm<sup>®</sup> GCC

This section describes the steps to configure the command line Arm<sup>®</sup> GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The `hello_world` demo application targeted for i.MX8DXL platform is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

### 5.1 Linux OS host

The following sections provide steps to run a demo compiled with Arm GCC on Linux host.

#### 5.1.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK.

##### 5.1.1.1 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 for EVK-MIMX8DXL* (document MCUXSDKIMX8DXLRN).

#### NOTE

See [Host setup](#) for Linux OS before compiling the application.

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

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

```
$ export ARMGCC_DIR=<path_to_GNUARM_GCC_installation_dir>
```

##### 5.1.1.3 Download and install JLink software and documentation pack for Linux

It provides `JLinkGDBServer` for future use.

#### 5.1.2 Build an example application

To build an example application, follow these steps.

1. 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/evkmimx8dxl/demo_apps/hello_world/armgcc
```

2. Run the `build_debug.sh` script on the command line to perform the build. The output is shown as below:

```
$ ./build_debug.sh
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-5_4-2016q3
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-5_4-2016q3
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
```

```
-- Found assembler: /work/platforms/tmp/gcc-arm-none-eabi-5_4-2016q3/bin/arm-none-eabi-gcc
-- Configuring done
-- Generating done
-- Build files have been written to:
/work/platforms/tmp/SDK_2.7.0_EVK_MIMX8DXL/boards/evkmimx8dxl/demo_apps/hello_world/armgcc
Scanning dependencies of target hello_world.elf
[ 4%] Building C object CMakeFiles/hello_world.elf.dir/work/platforms/tmp/SDK_2.7.0_EVK_MIMX8DXL/boards/evkmimx8dxl/demo_apps/hello_world/board.c.obj

< -- skipping lines -- >
[100%] Linking C executable debug/hello_world.elf
[100%] Built target hello_world.elf
```

### 5.1.3 Run an example application

This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, follow these steps:

- Make a bootable SD card with the SCFW (System Controller Firmware) image. See [Make a bootable SD card with System Controller Firmware \(SCFW\)](#).
- A standalone J-Link probe that is connected to the debug interface of your board.

#### NOTE

The Segger J-Link software has to be patched with the JLink.zip patch for i.MX8DXL from *iar\_segger\_support\_patch\_imx8dxl.zip*.

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 USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the 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 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit

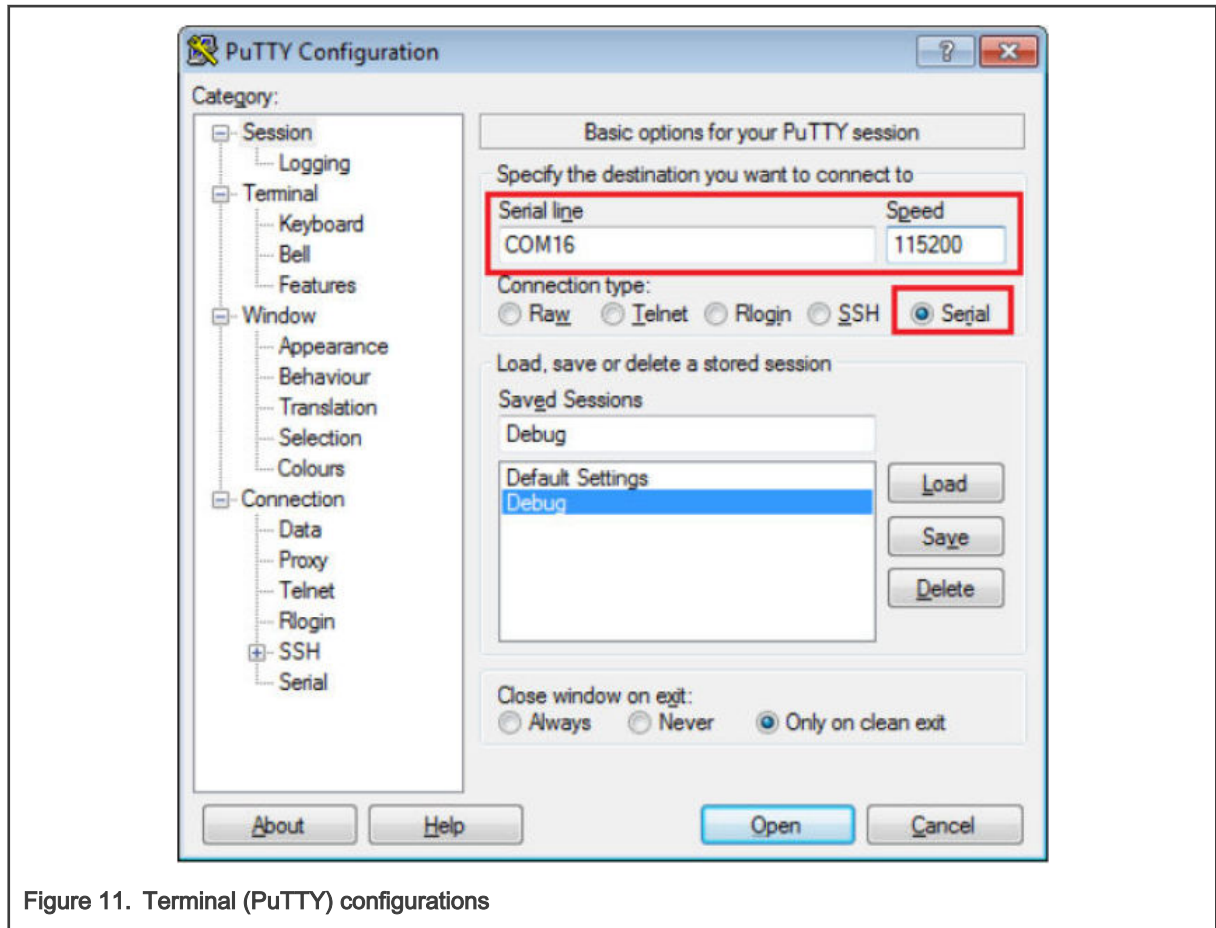


Figure 11. Terminal (PuTTY) configurations

3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched from a new terminal for the *MIMX8DL1\_M4* device:

```
$ JLinkGDBServer -if JTAG -device MIMX8DL1_M4 -vd -xc evkmimx8dxl_gdbsrv.cfg
```

*vd* configures *gdbserver* to verify data after every download. *-xc evkmimx8dxl\_gdbsrv.cfg* specifies a *gdbserver* config file, and the command sequence inside it will be executed every time a debugging session is created. The config file is located in *boards/evkmimx8dxl/evkmimx8dxl\_gdbsrv.cfg*. For details, see [GDBServer config file](#).

```

zodiace@zodiace-ubuntu:~/SDK_2.9.0_EVK-MIMX8DXL/boards/evkmimx8dxl/demo_apps/hello_world/armgcc$ JLinkGDBServer -ir -halt -stayontop
-if JTAG -device MIMX8DL1_M4 -vd -xc ./evkmimx8dxl_gdbsrv.cfg
SEGGER J-Link GDB Server V6.88a Command Line Version

JLinkARM.dll V6.88a (DLL compiled Nov 18 2020 16:08:10)

Command line: -ir -halt -stayontop -if JTAG -device MIMX8DL1_M4 -vd -xc ./evkmimx8dxl_gdbsrv.cfg
-----GDB Server start settings-----
GDBInit file:                ./evkmimx8dxl_gdbsrv.cfg
GDB Server Listening port:    2331
SWO raw output listening port: 2332
Terminal I/O port:          2333
Accept remote connection:    yes
Generate logfile:             off
Verify download:              on
Init regs on start:           on
Silent mode:                  off
Single run mode:              off
Target connection timeout:    0 ms
-----J-Link related settings-----
J-Link Host interface:       USB
J-Link script:               none
J-Link settings file:        none
-----Target related settings-----
Target device:               MIMX8DL1_M4
Target interface:             JTAG
Target interface speed:       4000kHz
Target endian:                little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Nov 12 2020 10:06:35
Hardware: V10.10
S/N: 600109559
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 1.80 V
Listening on TCP/IP port 2331
Connecting to target...

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 (Cortex-M4)
Connected to target
Waiting for GDB connection...

```

Figure 12. GDB Server is ready for connection in Linux OS

4. 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/evkmimx8dxl/demo_apps/hello_world/armgcc/debug
```

5. Start the GDB client:

```
$ arm-none-eabi-gdb --command=evkmimx8dxl_gdb_cmd_seq hello_world.elf
```

`evkmimx8dxl_gdb_cmd_seq` specifies the initial gdb commands to be executed. The file is located in `boards/evkmimx8dxl/evkmimx8dxl_gdb_cmd_seq`. The following command is executed one by one as specified in this file:

```
target remote localhost:2331
load hello_world.elf
monitor go
```

The `gdb` console output is as shown in [Figure 13](#).

- `target remote localhost:2331` creates a debug session with gdbserver.
- `load hello_world.elf` loads the elf to the target device. Please give appropriate path for the specified `elf`.

- `monitor go` lets the target device running.

You can change the command sequence or execute other commands in *gdb* console after the initial commands get executed.

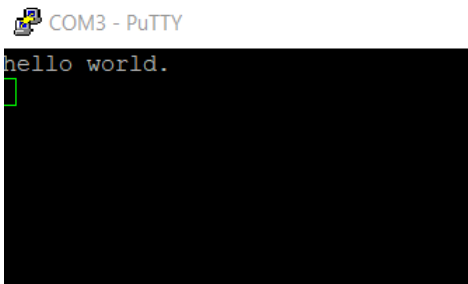
6. You can see the following output for the created *gdb* debug session.

```
zodiac@Zodiac-ProBook:~/SDK_2.8.0_EVK-MIMX8DXL/boards/evkmimx8dxl/demo_apps/hello_world/armgcc/debug$ arm-none-eabi-gdb
--command=../evkmimx8dxl_gdb_cmd_seq hello_world.elf
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...
0x00000000 in ?? ()
Loading section .interrupts, size 0x7b8 lma 0x1ffe0000
Loading section .resource_table, size 0x10 lma 0x1ffe0a00
Loading section .text, size 0x3760 lma 0x1ffe0a10
Loading section .ARM, size 0x8 lma 0x1ffe4170
Loading section .init_array, size 0x4 lma 0x1ffe4178
Loading section .fini_array, size 0x4 lma 0x1ffe417c
Loading section .data, size 0x68 lma 0x1ffe4180
Start address 0x1ffe0ac4, load size 16288
Transfer rate: 148 KB/sec, 2326 bytes/write.
(gdb) █
```

Figure 13. GDB session connected and program get running in Linux OS

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.



```
COM3 - PuTTY
hello world.
█
```

Figure 14. Text display of the `hello_world` demo

## 5.2 Windows OS host

The following sections provide steps to run a demo compiled with Arm GCC on Windows OS host.

### 5.2.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application with the Arm GCC toolchain on Windows OS, as supported by the MCUXpresso SDK.



### 5.2.1.1 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 for EVK-MIMX8DXL* (document MCUXSDKIMX8DXLRN).

#### NOTE

See [Host setup](#) for Windows OS before compiling the application.

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

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

*C:\Program Files (x86)\GNU Tools ARM Embedded\9-2019-q4-major*

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

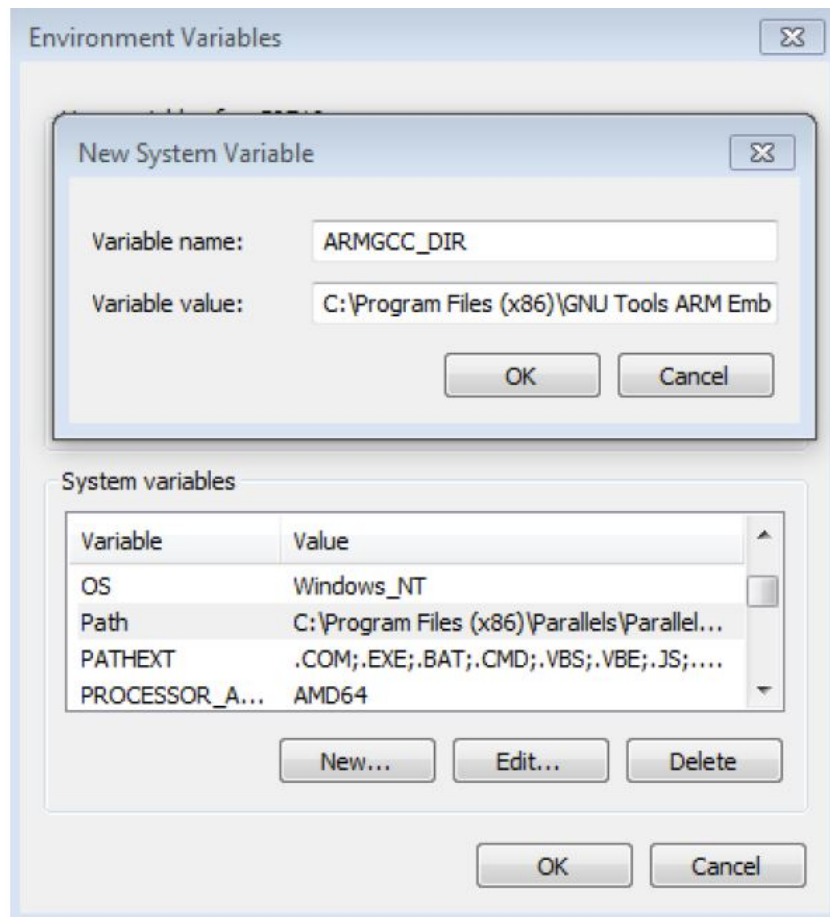


Figure 15. Add ARMGCC\_DIR system variable

### 5.2.1.3 Download and install JLink software and documentation pack for Linux

It provides `JLinkGDBServer` for future use.

## 5.2.2 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**.

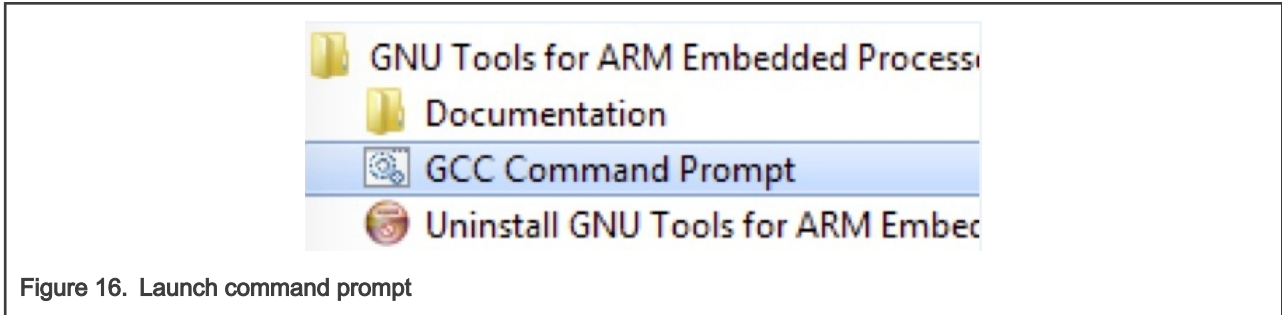


Figure 16. Launch command prompt

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/evkmimx8dxl/demo_apps/hello_world/armgcc
```

3. Type **build\_debug.bat** on the command line or double click on the *build\_debug.bat* file in Windows Explorer to perform the build. The output is as shown in Figure 17.

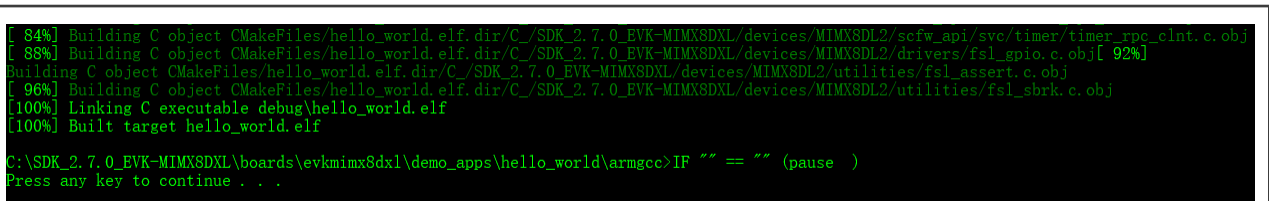


Figure 17. hello\_world demo build successful

## 5.2.3 Run an example application

Running the Arm GCC built demo also requires J-Link support. Get the IAR and Segger J-Link patch, *iar\_segger\_support\_patch\_imx8dxl.zip*. Install the i.MX8DXL support patch following the guides in *readme.txt* located in the archive.

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

- Make a bootable SD card with the System Controller FirmWare (SCFW) image. See [Make a bootable SD card with System Controller Firmware \(SCFW\)](#). You have a standalone J-Link pod that is connected to the debug interface of your board. Make sure the Segger J-Link software i.MX8DXL supporting patch, *iar\_segger\_support\_patch\_imx8dxl.zip*, is installed.

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 USB-UART 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 [How to determine COM port](#)). Configure the terminal with these settings:
  - a. 115200 baud rate

- b. No parity
- c. 8 data bits
- d. 1 stop bit

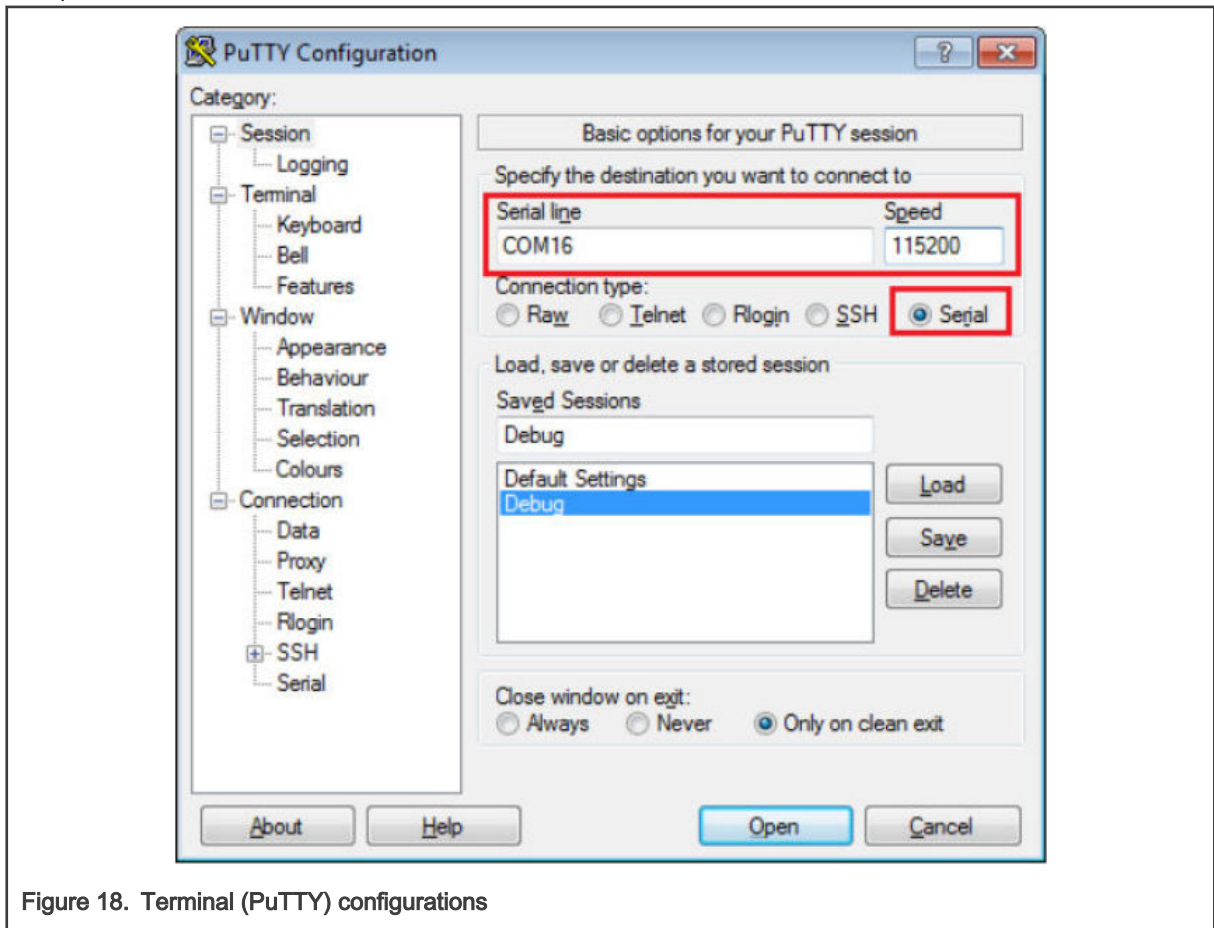


Figure 18. Terminal (PuTTY) configurations

3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched from a new cmd window for the MIMX8DL1\_M4 device:

```
JLinkGDBServer -ir -halt -stayontop -if JTAG -device MIMX8DL1_M4 -vd -xc evkmimx8dxl_gdbsrv.cfg
```

*vd* configures *gdbserver* to verify data after every download. *-xc evkmimx8dxl\_gdbsrv.cfg* specifies a *gdbserver* config file. The command sequence inside it will be executed every time a debugging session is created. The config file is located in *boards/evkmimx8dxl/evkmimx8dxl\_gdbsrv.cfg*. For details, see [GDBServer config file](#).

4. After GDB server is running, the screen should resemble [Figure 19](#).

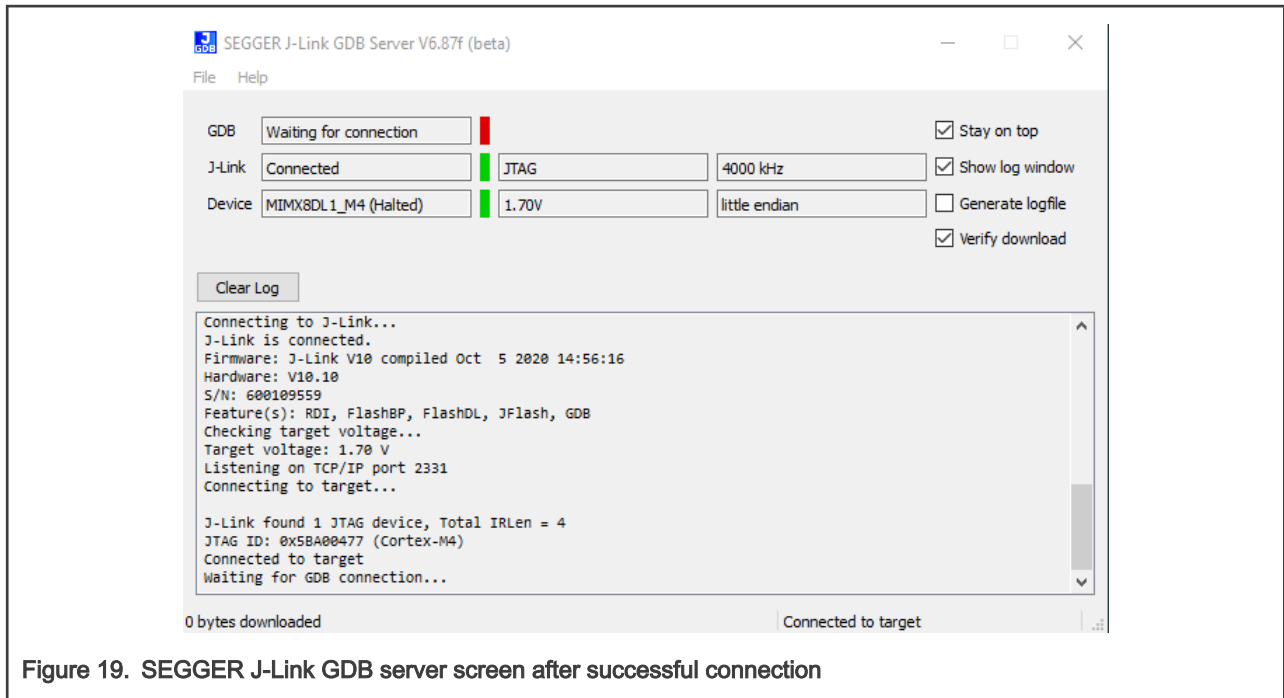


Figure 19. SEGGER J-Link GDB server screen after successful connection

5. 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**.

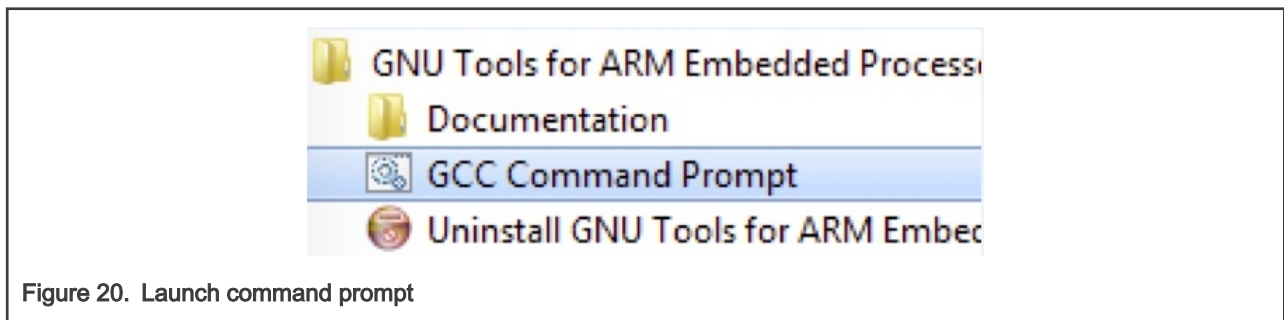


Figure 20. Launch command prompt

6. 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/evkmimx8dxl/demo_apps/hello_world/armgcc/debug`

7. Run the following commands:

```
arm-none-eabi-gdb --command=evkmimx8dxl_gdb_cmd_seq
```

`evkmimx8dxl_gdb_cmd_seq` specifies the initial `gdb` commands to be executed. The file is located in `boards/evkmimx8dxl/evkmimx8dxl_gdb_cmd_seq`. The following command is executed one by one as specified in this file:

```
target remote localhost:2331
load hello_world.elf
monitor go
```

- `target remote localhost:2331` creates a debug session with *gdbserver*.
- `load hello_world.elf` loads the *elf* to the target device. Please give appropriate path for the specified *elf*.
- `monitor go` lets the target device running.

You can change the command sequence or execute other commands in *gdb* console after the initial commands get executed.

The *gdb* console output is as shown in Figure 21.

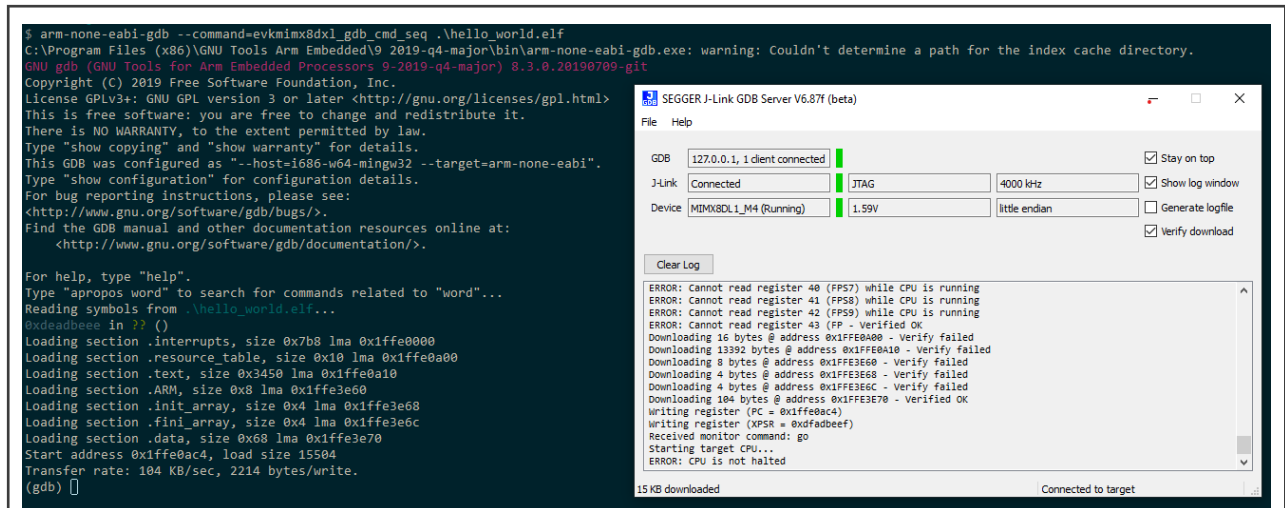


Figure 21. GDB session connected and program get running in Windows OS

The *hello\_world* application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.

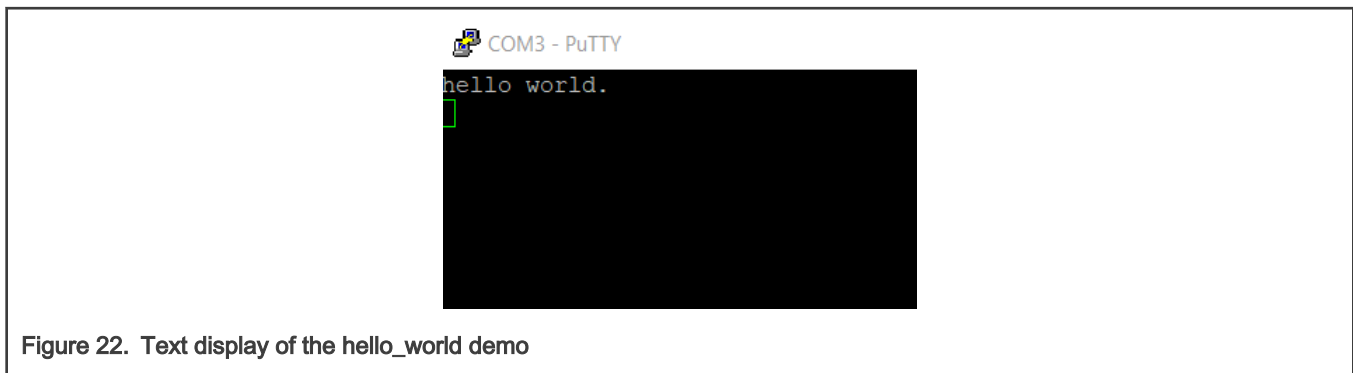


Figure 22. Text display of the *hello\_world* demo

### 5.3 GDBServer config file

As mentioned earlier, *evkmimx8dxl* specifies a *gdb* server config file which will get executed everytime a debugging session is created. It does target initialization in preparation for debugging session creation. In our example, two commands are specified.

```
// Reset the chip to get to a known state
halt
// Enable The Following Line to Support DDR XIP Alias Feature
// MemU32 0xE0080030 = 0x7E
```

As described in the comment, the first command simply put the M4 core to halt state. The second command is disabled by default. It should be enabled for DDR target to turn on DDR to XIP memory alias. For details, see [C-Spy macros for DDR Target](#).

# Chapter 6

## Run a demo using imx-mkimage

The *imx-mkimage* is used to combine various input images and generate the all-in-one boot image with the appropriate Image Vector Table (IVT) set. It can be directly flashed to boot medium, such as an SD card to boot various cores in the SOC. This includes SCU firmware, U-Boot for A core, and the M4 image for M core. Currently the *imx-mkimage* can only work on Linux OS. Use the following steps to prepare for working with *imx-mkimage*:

1. Clone the *imx-mkimage* from NXP public git.

```
$ git clone https://source.codeaurora.org/external/imx/imx-mkimage
```

2. Check out the correct branch. The branch name is named after linux release version which is compatible with the SDK. You can get the version information from corresponding Linux Release Notes document.

```
$ git checkout [branch name]
```

3. Get the SCU firmware package with the link provided in corresponding Linux Release Notes document. Then execute the following command:

```
$ chmod a+x [bin package]
$ sh [bin package]
```

This extracts the SCU firmware. Rename *mx8dxl-evk-scfw-tcm.bin* to *scfw\_tcm.bin* and copy the file to *imxmimage/iMX8DXL*.

4. Get the i.MX SECO firmware package with the link provided in corresponding Linux Release Notes document. Execute the following command:

```
$ chmod a+x [bin package]
$ sh [bin package]
```

This extracts the i.MX SECO firmware. Copy *firmware/seco/mx8dxla1-ahab-container* to *imx-mkimage/iMX8DXL*. *a1* indicates that the firmware is used for A1 silicon.

5. Generate the *u-boot.bin* and *u-boot-spl.bin* from Linux release package and copy it to *imx-mkimage/iMX8DXL*.
6. Generate the Arm Trusted Firmware *bl31.bin* from the Linux release package and copy it to *imx-mkimage/iMX8DXL*.

### 6.1 Run an example application on the M4 core

1. Build the M4 demo application. Rename the generated binary file (.bin file) to *m4\_image.bin*, and copy to this file to the *imx-mkimage/iMX8DXL* folder.
2. In Linux OS, bash *cd* into the *imx-mkimage* installed directory, and run the following command to generate bootable image:

```
$ make clean
```

If the M4 image built is for TCM:

```
$ make SOC=iMX8DXL flash_m4
```

If the M4 image built is for DDR:

```
$ make SOC=iMX8DXL flash_m4_ddr
```

This generates the bootable image *flash.bin* under the *iMX8DXL* folder.

3. Write the image into the SD card. Insert the SD card into the Linux PC, and run the following command in Linux bash with ROOT permission:

```
dd if=./iMX8DXL/flash.bin of=/dev/<SD Device> bs=1k seek=32
```

The `<SD Device>` is the device node of your SD card such as `sdb`.

4. Insert the SD card to SD1 card slot and power on the board. See [Run an example application](#) for steps to connect the board with PC and configure debugging terminals. It can be observed that the M4 demo is running.

## 6.2 Make a bootable SD card with System Controller Firmware (SCFW)

When debugging or running MCUXpresso SDK with IAR and J-Link GDB Server, the bootable SD card with SCU firmware (SCFW) is required. The SCU handles setting the power, clock, pinmux, and so on for other cores, so the SCFW is needed to run MCUXpresso SDK. To keep the peripherals in the chip at reset status, do not put the CM4 image in the booting image (*flash.bin*) when debugging or running CM4 cores with IAR and the J-Link GDB Server.

To make a bootable SD card with only SCFW, use the following command to generate a bootable image in *imx-mkimage.tool*.

```
$ make clean
```

```
$ make SOC=iMX8DXL flash_scfw
```

Follow the steps described in [Run an example application on the M4 core](#) to write the generated *flash.bin* into the SD card.

## 6.3 Run an example application on the M4 core together with U-Boot

When the **A** core and **M** core are running together, they need to run in two different partitions. This is achieved by the special target provided by `<em>mkimage</em>` facility.

1. Copy *u-boot.bin* and *u-boot-spl.bin* into *mx-mkimage/iMX8DXL*.
2. Rename the M4 image to *m4\_image.bin* and copy it into *imx-mkimage/iMX8DXL*.
3. In Linux OS, bash cd into the *imx-mkimage* directory, and run the following command to generate bootable image:

```
$ make clean
```

If the M4 image is built for TCM:

```
make SOC=iMX8DXL flash_regression_linux_m4
```

If the M4 image is built for DDR:

```
make SOC=iMX8DXL flash_regression_linux_m4_ddr
```

This generates the bootable image *flash.bin* under the **iMX8DXL** folder.

Follow the steps described in [Make a bootable SD card with System Controller Firmware \(SCFW\)](#) to write the generated *flash.bin* into the *emmc*.

# Chapter 7

## Run a demo using facility provided by U-Boot

The `bootaux` command on U-Boot is obsolete because the A and M core must run on different partitions. We can no longer kick off M4 demo from U-Boot.



# Chapter 8

## Run a flash target demo by UUU

This section describes the steps to use the UUU to build and run example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the i.MX 8DualXLite EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

### 8.1 Set up environment

This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application, as supported by the MCUXpresso SDK.

#### 8.1.1 Download the MfgTool

The Universal Upgrade Utility (UUU) is an upgraded version of MfgTool. It is a command line tool that aims at installing the bootloader to various storage including SD, QSPI, and so on, for i.MX series devices with ease.

The tool can be downloaded from [MfgTool page](#). Use version 1.3.191 or higher for full support for the M4 image. Download the *uuu.exe* file for Windows OS or UUU for Linux OS. Configure the path so that the executable can later be called anywhere in the command line.

#### 8.1.2 Switch to SERIAL mode

The board needs to be in SERIAL mode for UUU to download images:

1. Set the board boot mode to `SERIAL[b'0001]`.
2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J46 USB Type-C on the CPU board.
3. The PC recognizes the i.MX8DXL device as (VID:PID)=(1FC9:0147), as shown in [Figure 23](#).

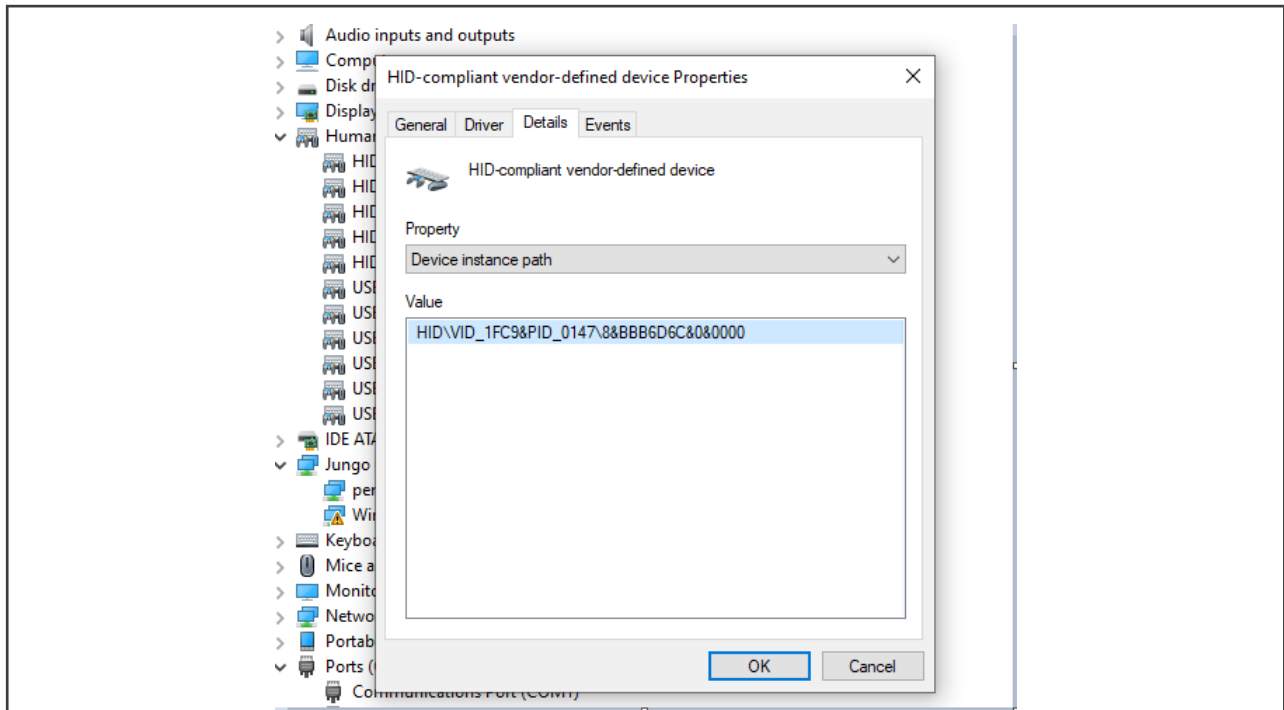


Figure 23. Device as shown in Device Manager

## 8.2 Build an example application

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

1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:

*<install\_dir>/boards/<board\_name>/<example\_type>/<application\_name>/iar*

Using the i.MX 8DualXLite EVK board as an example, the `rpmsg_pingpong` workspace is located in:

*<install\_dir>/boards/evkmimx8dxl/multicore\_examples/rpmsg\_lite\_pingpong\_rtos/  
linux\_remote/iar/rpmsg\_lite\_pingpong\_rtos\_linux\_remote.eww*

2. Select the desired build target from the drop-down. For this example, select **rpmsg\_lite\_pingpong\_rtos\_linux\_remote – flash\_debug**.

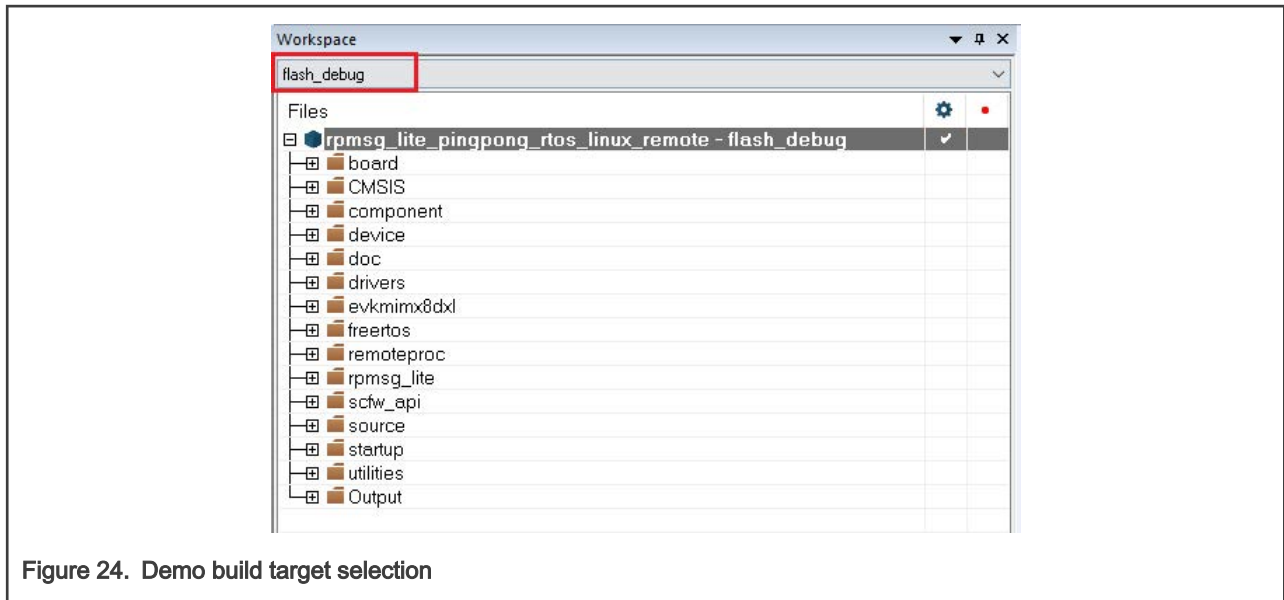


Figure 24. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 25.

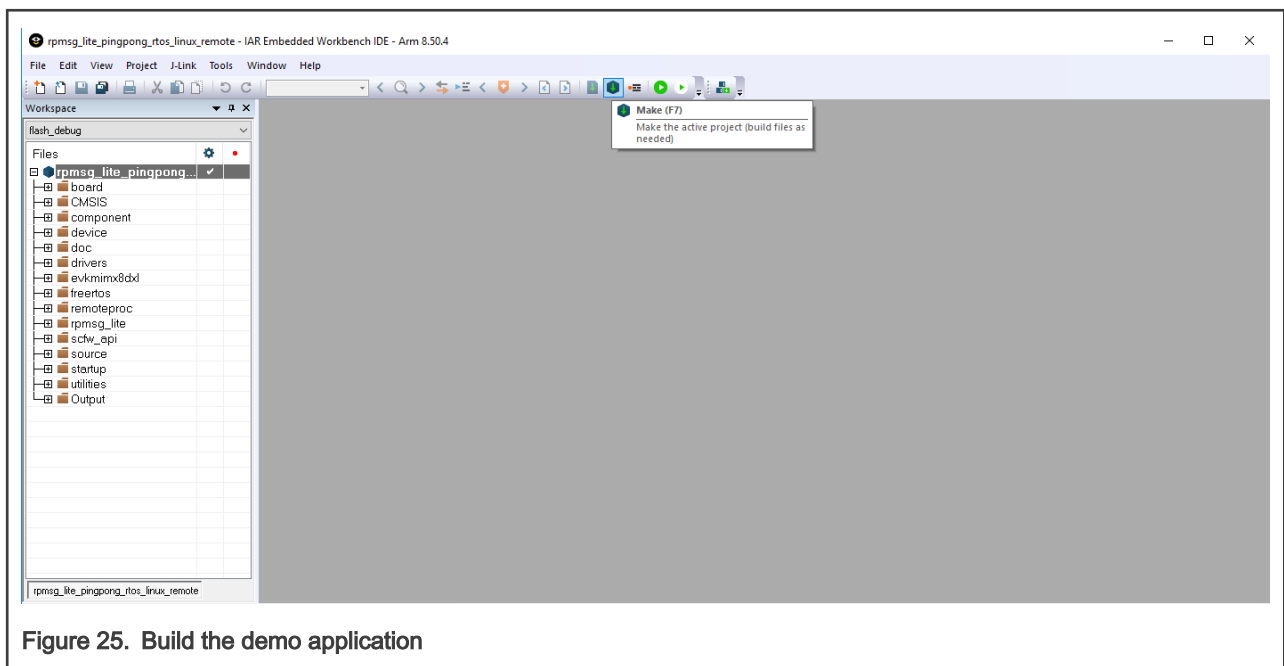


Figure 25. Build the demo application

4. The build completes without errors.
5. [Step 1](#) to [Step 4](#) are used for IAR toolchain to build the flash target M4 demo. For ARMGCC toolchain, simply run the `build_flash_debug` or `build_flash_release` script to build the flash target **M4** demo.
6. Rename the generated `rpmsg_lite_pingpong_rtos_linux_remote.bin` to `m4_image.bin`, then copy it to the `mkimage` tool under `imx-mkimng/iMX8DXL`.
7. There are two targets to generate `flash.bin` which contains the XIP M4 target in `imx-mkimng`:
  - `flash_m4_xip` generates a `flash.bin` which only contains the M4 XIP image.
  - `flash_regression_linux_m4_xip` generates a `flash.bin` which contains both M4 XIP and U-Boot.

Use `make SOC=iMX8DXL flash_m4_xip` or `make SOC=iMX8DXL flash_regression_linux_m4_xip` to generate the desired `flash.bin`.

8. Use `make SOC=iMX8DXL flash_flexspi` to generate a *flash.bin* which contains flexspi U-Boot. Rename this to **flash\_uboot.bin** for future use.

### 8.3 Run an example application

To download and run the application via UUU, perform these steps:

1. Connect the development platform to your PC via USB cable between the J19 USB DEBUG connector and the PC. It provides console output while using UUU.
2. Connect the J46 USB Type-C connector and the PC. It provides the data path for UUU.
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 [How to determine COM port](#)). Configure the terminal with these settings:
  - a. 115200 baud rate
  - b. No parity
  - c. 8 data bits
  - d. 1 stop bit

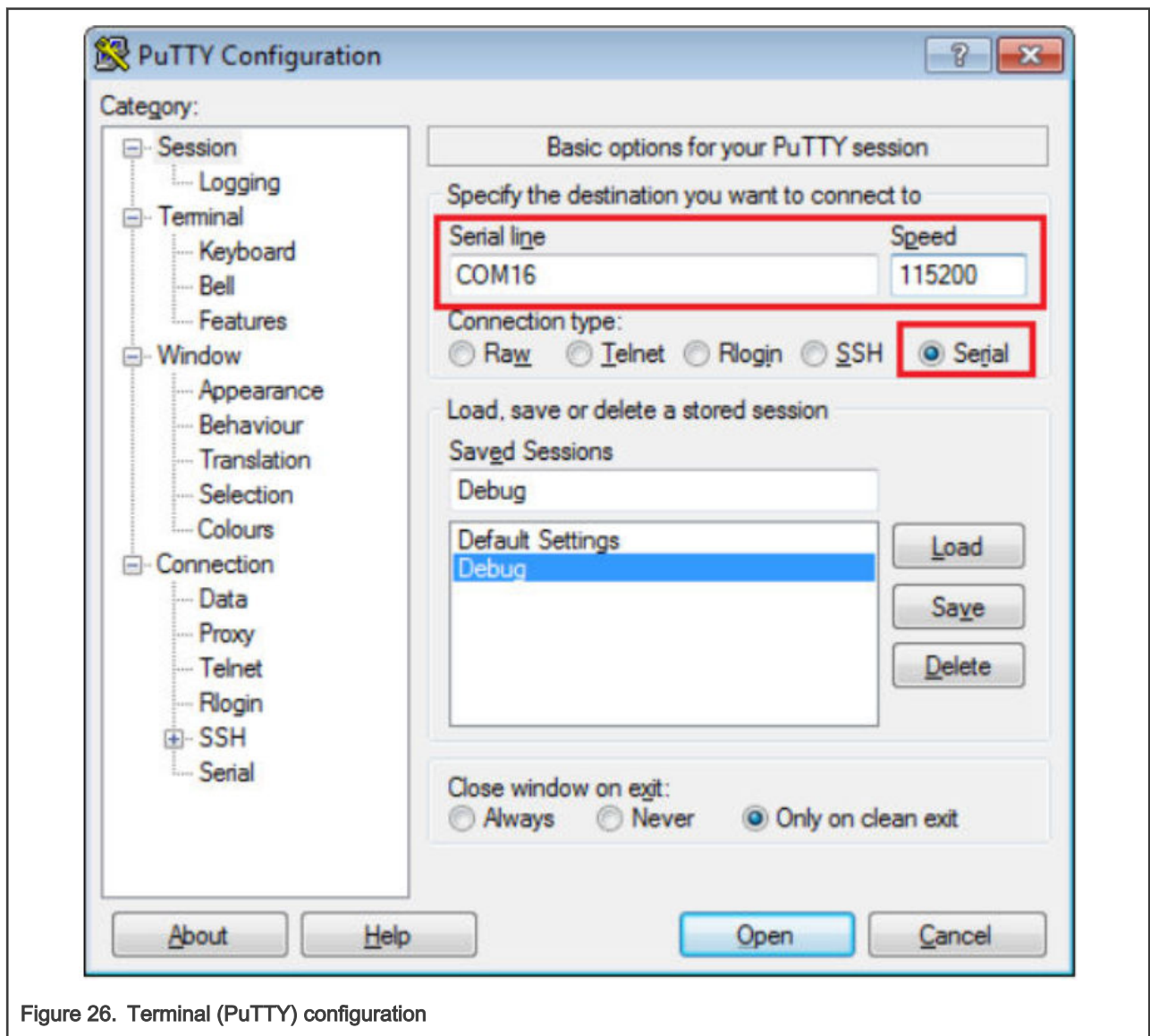


Figure 26. Terminal (PuTTY) configuration

4. In the command line, execute uuu with the **-b qspi** switch: `-b qspi flash_uboot.bin flash.bin`.

The UUU puts the platform into fast boot mode and automatically flashes the target bootloader to QSPI. The command line and fast boot console is as shown in [Figure 27](#).

```
Run bootcmd mfg: run mfgtool_args; if iminfo ${initrd_addr}; then if test ${tee} = yes; then bootm ${tee_addr} ${i
nitrd_addr} ${fdt_addr}; else booti ${loadaddr} ${initrd_addr} ${fdt_addr}; fi; else echo "Run fastboot ..."; fas
tboot auto; fi;
Hit any key to stop autoboot: 0

## Checking Image at 83100000 ...
Unknown image format!
Run fastboot ...
auto usb 0
Detect USB boot. Will enter fastboot mode!
Starting download of 2412544 bytes
.....
downloading of 2412544 bytes finished
Unknown command 'qspihdr' - try 'help'
Detect USB boot. Will enter fastboot mode!
Detect USB boot. Will enter fastboot mode!
SF: Detected mt35x5l2aba with page size 256 Bytes, erase size 128 KiB, total 64 MiB
Detect USB boot. Will enter fastboot mode!
SF: 2490368 bytes @ 0x0 Erased: OK
Detect USB boot. Will enter fastboot mode!
device 0 offset 0x0, size 0x24d000
SF: 2412544 bytes @ 0x0 Written: OK
Detect USB boot. Will enter fastboot mode!

C:\uuu>uuu.exe -b qspi flash_uboot.bin flash.bin
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.3.191-0-g4fe24b9

Success 1   Failure 0

3:14      8/ 8 [Done] FB: done
```

Figure 27. Command line and fast boot console output when executing UUU

In this example, the *flash.bin* is generated using the `flash_regression_linux_m4_xip` target, which contains both M4 XIP and U-Boot.

5. Then, power off the board and change the boot mode to QSPI[b'0110], and power on the board again. The two UART consoles display the U-Boot and M4 demo output respectively.

```
COM17 - PuTTY
Model: NXP i.MX8DXL EVK Board
Board: iMX8DXL EVK
Boot: FLEXSPI
DRAM: 8GiB
MMC: FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... Run CMD11 1.8V switch
*** Warning - bad CRC, using default environment

In: serial
Out: serial
Err: serial

BuildInfo:
- SCFW 2274ac84, SECO-FW b3c3cb07, IMX-MKIMAGE 98e
- U-Boot 2020.04-5.4.47-2.2.0+g168a662de3
- V2X-FW 2c8f793d version 0.0.4

Run CMD11 1.8V switch
switch to partitions #0, OK
mmc1 is current device
Net:
Warning: ethernet@5b050000 (eth1) using random MAC a
eth1: ethernet@5b050000 [PRIME]
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot: 0
=>

COM16 - PuTTY
RPMSP Ping-Pong FreeRTOS RTOS API Demo...
RPMSP Share Base Addr is 0x90010000
```

Figure 28. U-Boot and M4 demo output

# Appendix A

## How to determine COM port

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

- **Linux**

```
$ dmesg | grep ttyUSB1  
[434269.853961] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0  
[434269.857824] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB1
```

There are two Ports. The first is the Cortex-A debug console, and the second is for the CM4 debug console.

- **Windows**

1. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system **Start** menu and typing **Device Manager** in the search bar, as shown in [Figure 29](#).

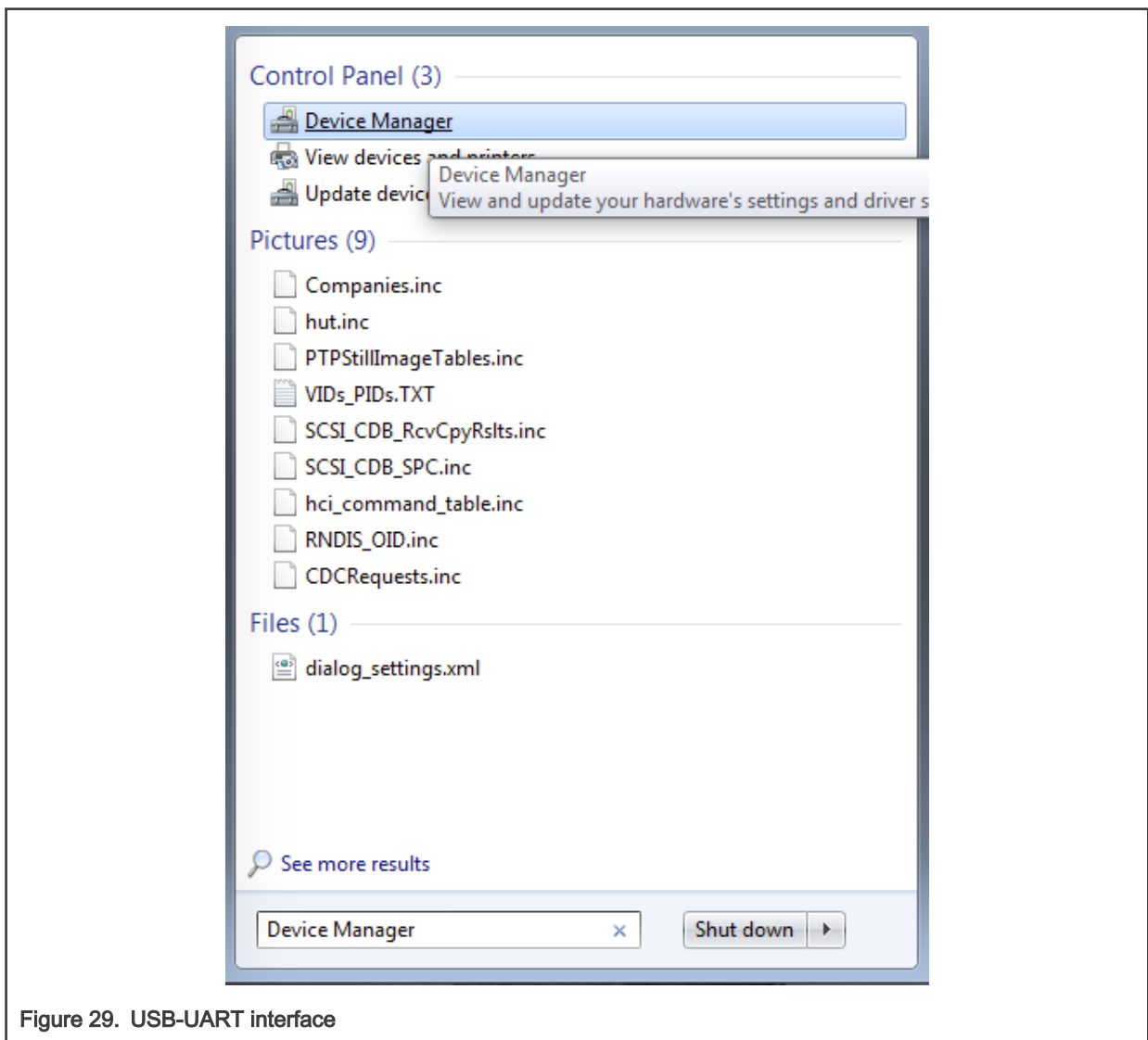
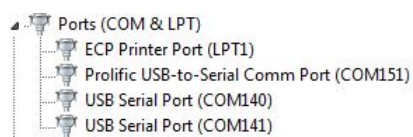


Figure 29. USB-UART interface

2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports.

## a. USB-UART interface

**Figure 30. USB-UART interface**

There will be four ports. The first is the Cortex-A debug console, and the second is for the CM4 debug console.

# Appendix B

## Host setup

An MCU SDK build requires that some packages are installed on the Host. Depending on the used Host Operating System, the following tools should be installed.

- **Linux:**

- **Cmake**

```
$ sudo apt-get install cmake
$ # Check the version >= 3.0.x
$ cmake --version
```

- **Windows:**

- **MinGW**

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

1. Download the latest MinGW mingw-get-setup installer from [MinGW - Minimalist GNU for Windows](#).
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 **mingw32-base** and **msys-base** are selected under **Basic Setup**.

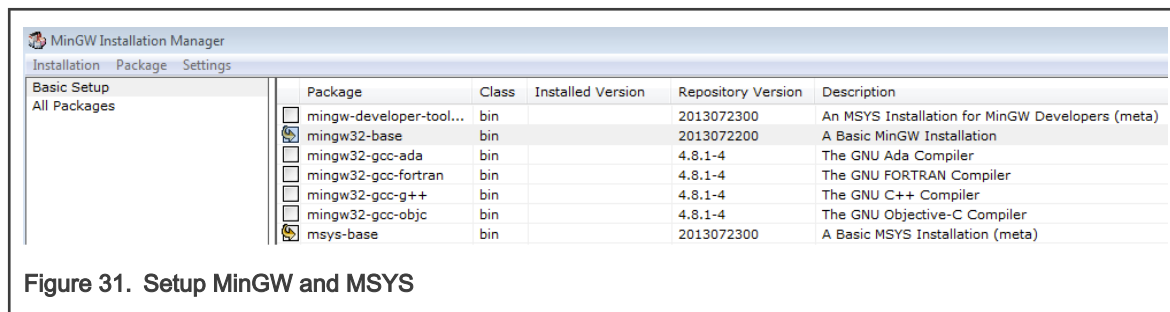


Figure 31. Setup MinGW and MSYS

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

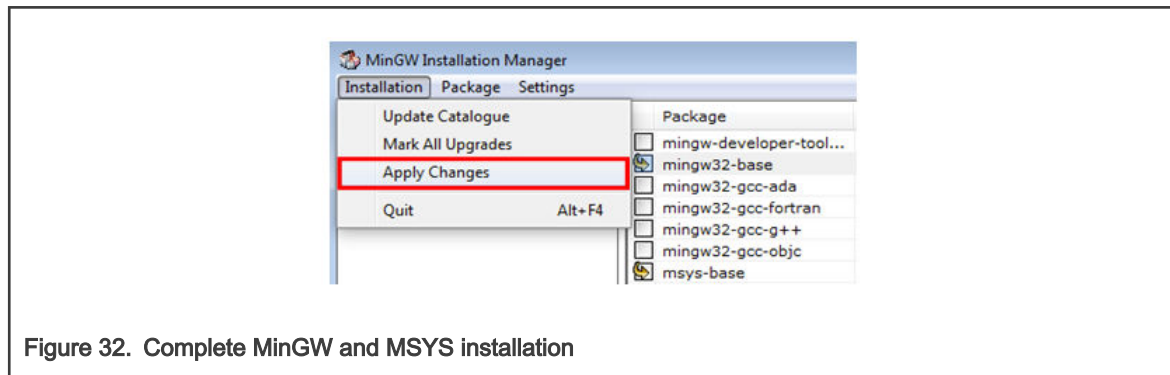


Figure 32. Complete MinGW and MSYS installation



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 as shown in [Figure 31](#). If the path is not set correctly, the toolchain does not work.

#### NOTE

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

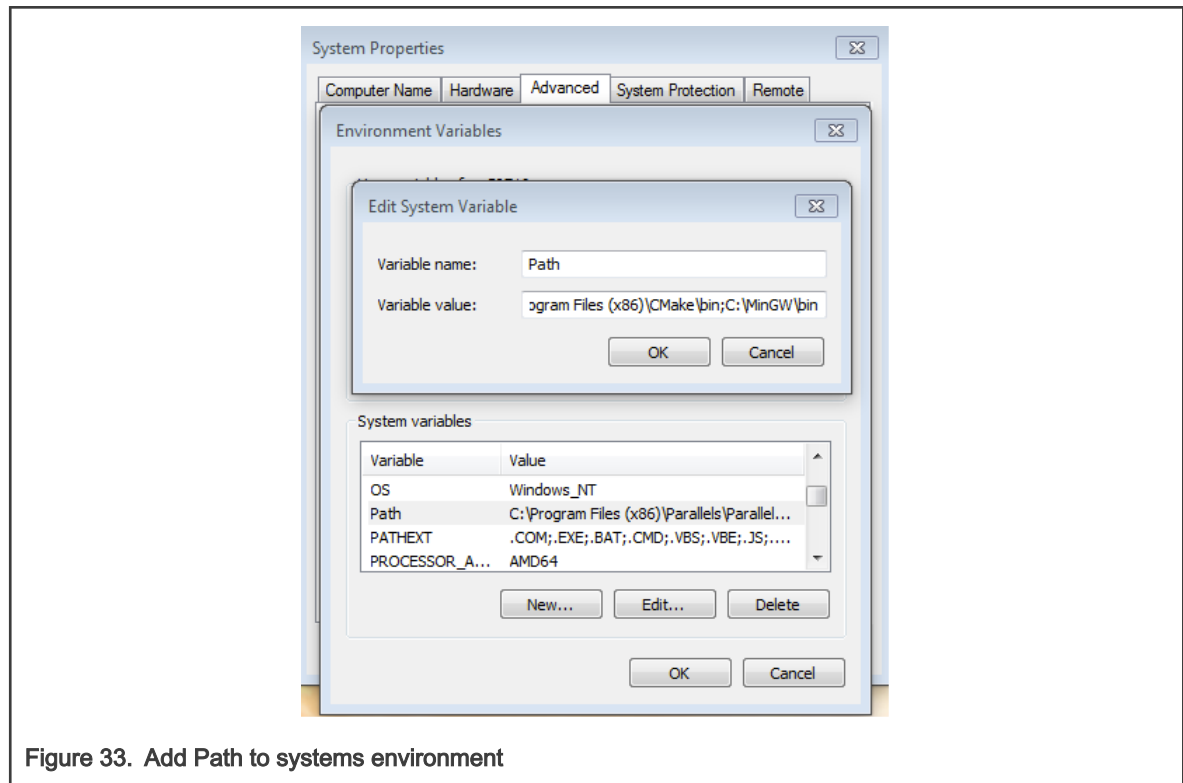


Figure 33. Add Path to systems environment

#### — Cmake

1. Download CMake 3.0.x from [Software download page](#).
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.

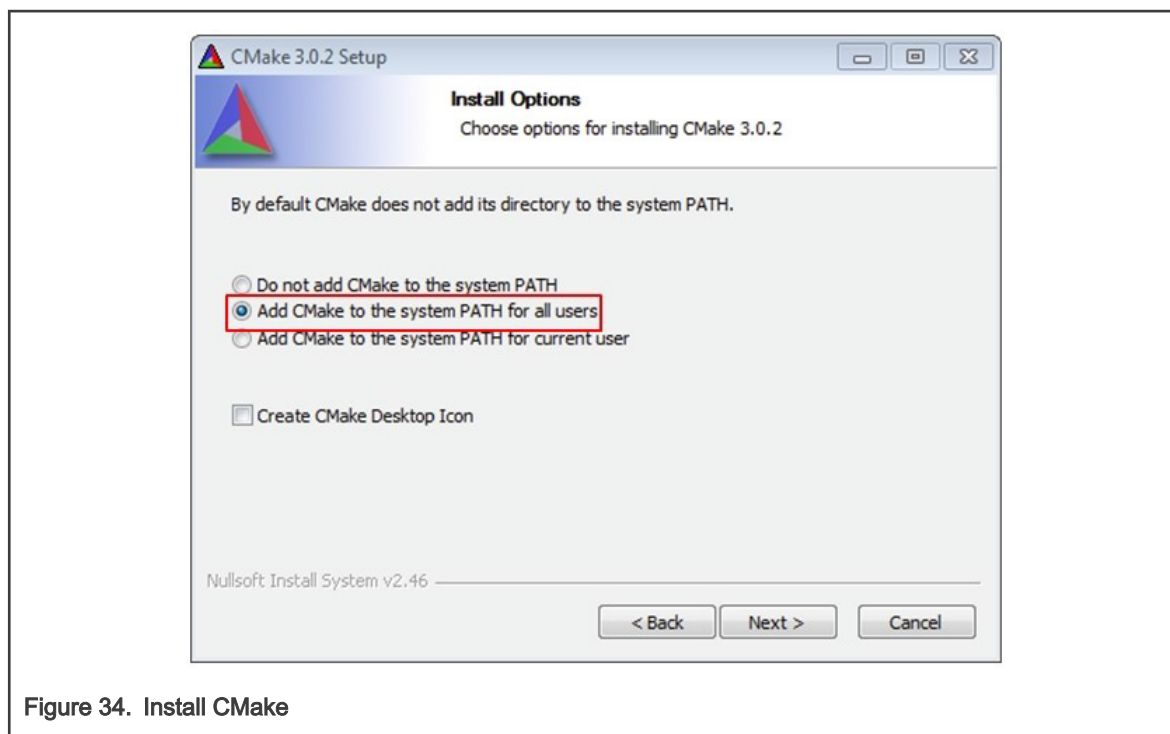


Figure 34. Install CMake

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

## How To Reach Us

### Home Page:

[nxp.com](http://nxp.com)

### Web Support:

[nxp.com/support](http://nxp.com/support)

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

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

**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.

**Security** — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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](mailto:PSIRT@nxp.com)) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© NXP B.V. 2020.

All rights reserved.

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

For sales office addresses, please send an email to: [salesaddresses@nxp.com](mailto:salesaddresses@nxp.com)

Date of release: December 30, 2020

Document identifier: MCUXSDKIMX8DXLGSUG

