Getting Started with MCUXpresso SDK for EVK-MIMX8MP

Contents

Chapter 1
Overview

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides comprehensive software support for microcontrollers. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to demo applications. The MCUXpresso SDK also contains RTOS kernels, and device stack to support rapid development on devices.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for EVK-MIMX8MP* (document MCUXSDKIMX8MPRN).

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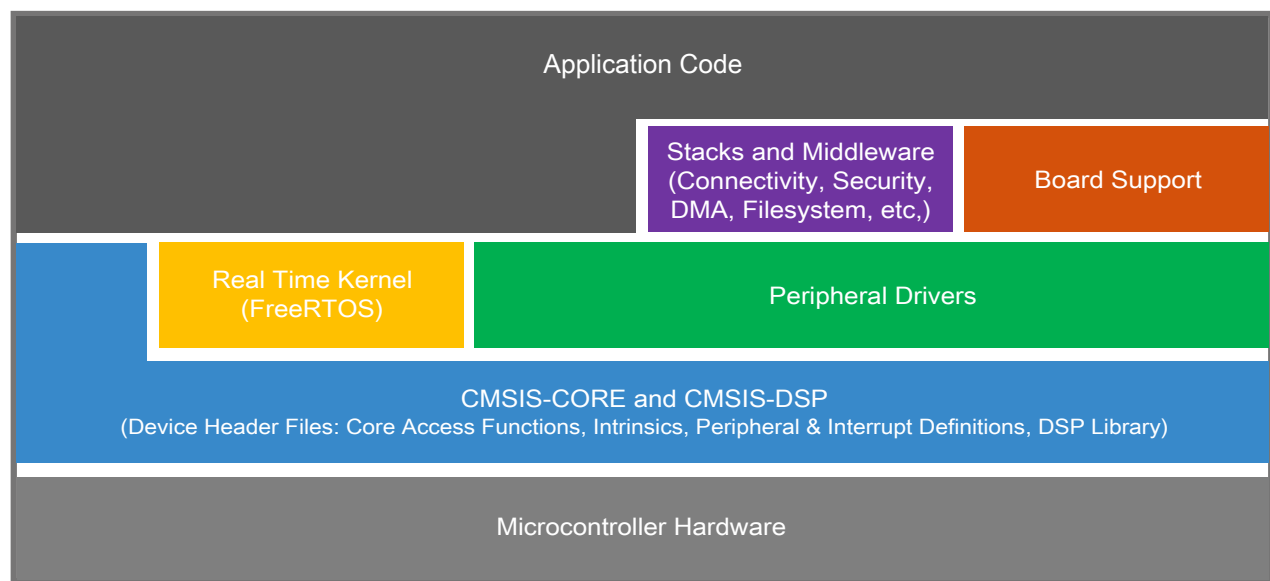

Figure 1. MCUXpresso SDK layers

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.

2.1 Example application structure

This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

Each `<board_name>` folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the `hello_world` example (part of the `demo_apps` folder), the same general rules apply to any type of example in the *<board_name>* folder.

In the `hello_world` application folder you see the following contents:
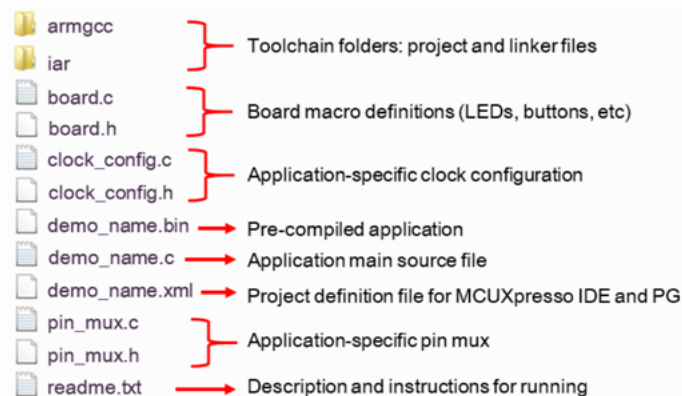

Figure 2. Application folder structure

All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the MCUXpresso SDK.

2.2 Locating example application source files

When opening an example application in any of the supported IDEs, a variety of source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the MCUXpresso SDK tree used in all example applications are:

- *devices/<device_name>*: The device's CMSIS header file, MCUXpresso SDK feature file and a few other files

- *devices/<device_name>/drivers*: All of the peripheral drivers for your specific MCU

- *devices/<device_name>/<tool_name>*: Toolchain-specific startup code, including vector table definitions

- *devices/<device_name>/utilities*: Items such as the debug console that are used by many of the example applications

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

Chapter 3
Toolchain introduction

The MCUXpresso SDK release for i.MX 8M Plus 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 8M Plus 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	developer.arm.com/open-source/gnu-toolchain/gnu-rm www.segger.com
IAR/J-Link	Windows OS	J-Link Plus	www.iar.com www.segger.com

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

NOTE
- To support i.MX 8M Plus, the patch for IAR and Segger J-Link should be installed. The patch named iar_segger_support_patch_imx8mp.zip can be used with MCUXpresso SDK. See the *readme.txt* in the patch for additional information about patch installation.

- For the security reason, only after the U-Boot is boot up, the JTAG can be allowed to connect to M core.

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. The `hello_world` demo application targeted for the i.MX 8MP EVK hardware platform 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_imx8p.zip`. Install the i.MX8MP 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 8MP EVK hardware platform as an example, the `hello_world` workspace is located in:

   ```
   <install_dir>/boards/evkmimx8mp/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.

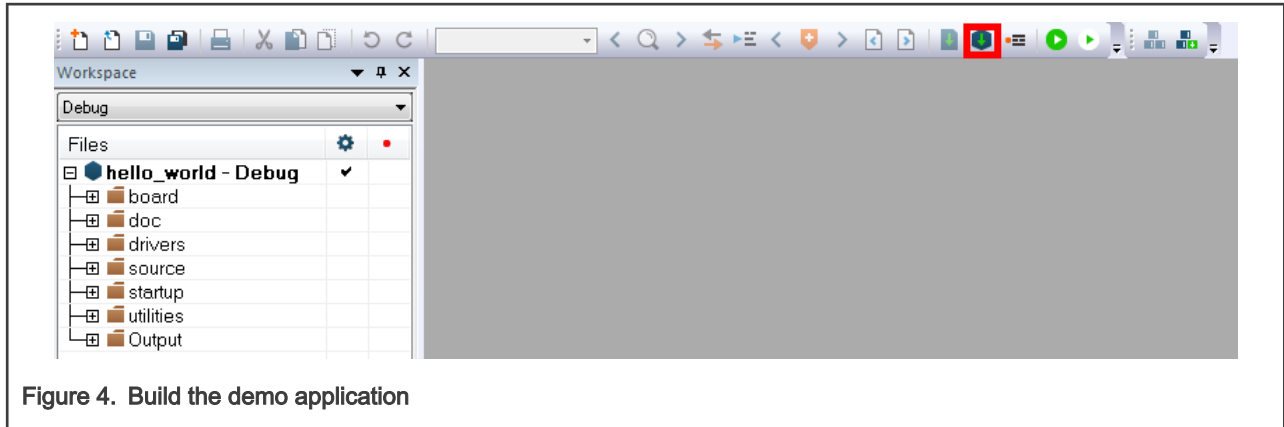   For this example, select **hello_world** – **debug**.
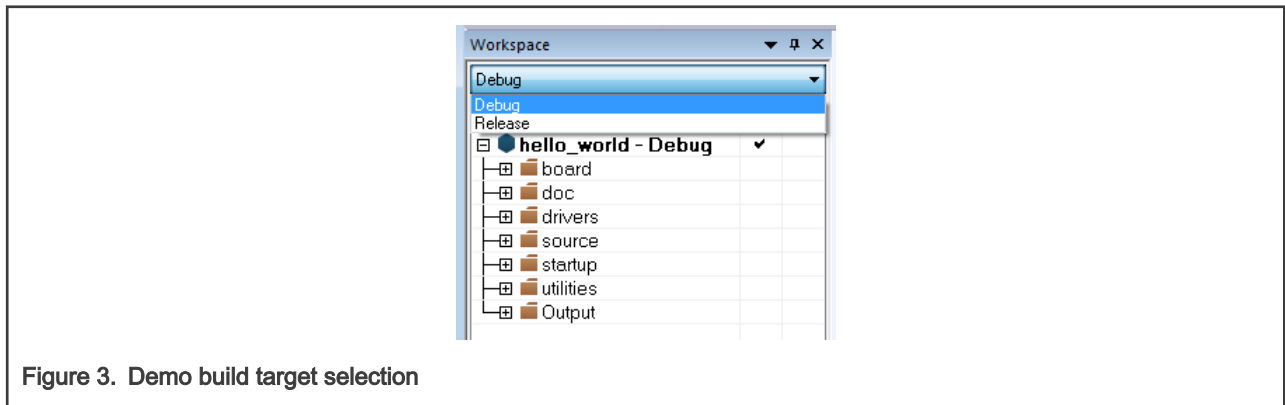
   Figure 3. Demo build target selection

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.

4.2 Run an example application

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 device.

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
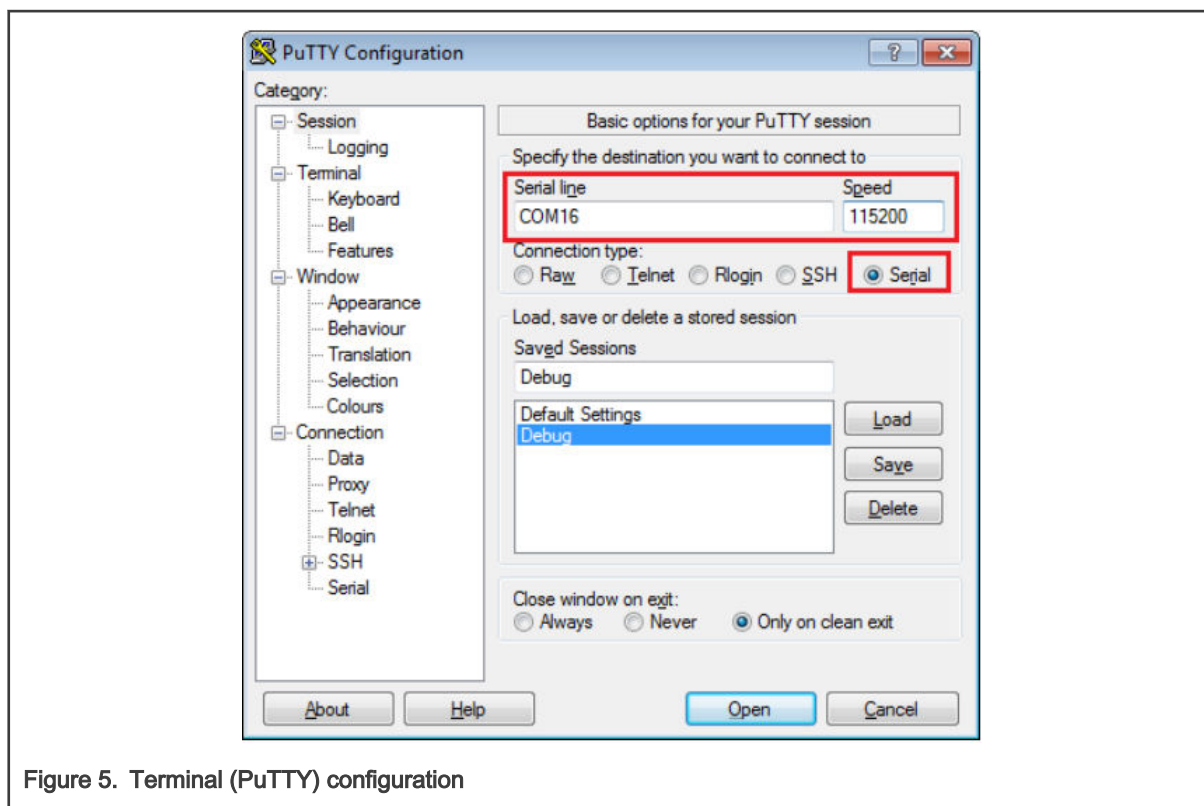
Figure 5. Terminal (PuTTY) configuration

4. In IAR, click **Download and Debug** to download the application to the target.
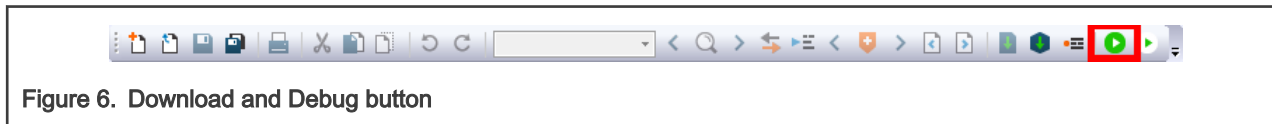
Figure 6. Download and Debug button

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

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

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

NOTE

For downloading the DDR target application, insert one TF card with U-Boot code. This requires both on IAR and GCC.

Chapter 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 targeted for i.MX 8M Plus 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-MIMX8MP* (document MCUXSDKIMX8MPRN).

> **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=/work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major

$ export PATH= $PATH:/work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major

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/evkmimx8mp/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-9-2019-q4-major
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler: /work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major/bin/arm-none-eabi-gcc
-- Configuring done
-- Generating done
```

```
-- Build files have been written to:
/work/platforms/tmp/nxp/SDK_2.9.0_EVK_MIMX8MP/boards/evkmimx8mp/demo_apps/hello_world/armgcc
Scanning dependencies of target hello_world.elf
[ 6%] Building C object CMakeFiles/hello_world.elf.dir/work/platforms/tmp/nxp/
SDK_2.9.0_EVK_MIMX8MP/boards/evkmimx8mp/demo_apps/hello_world/hello_world.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.

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
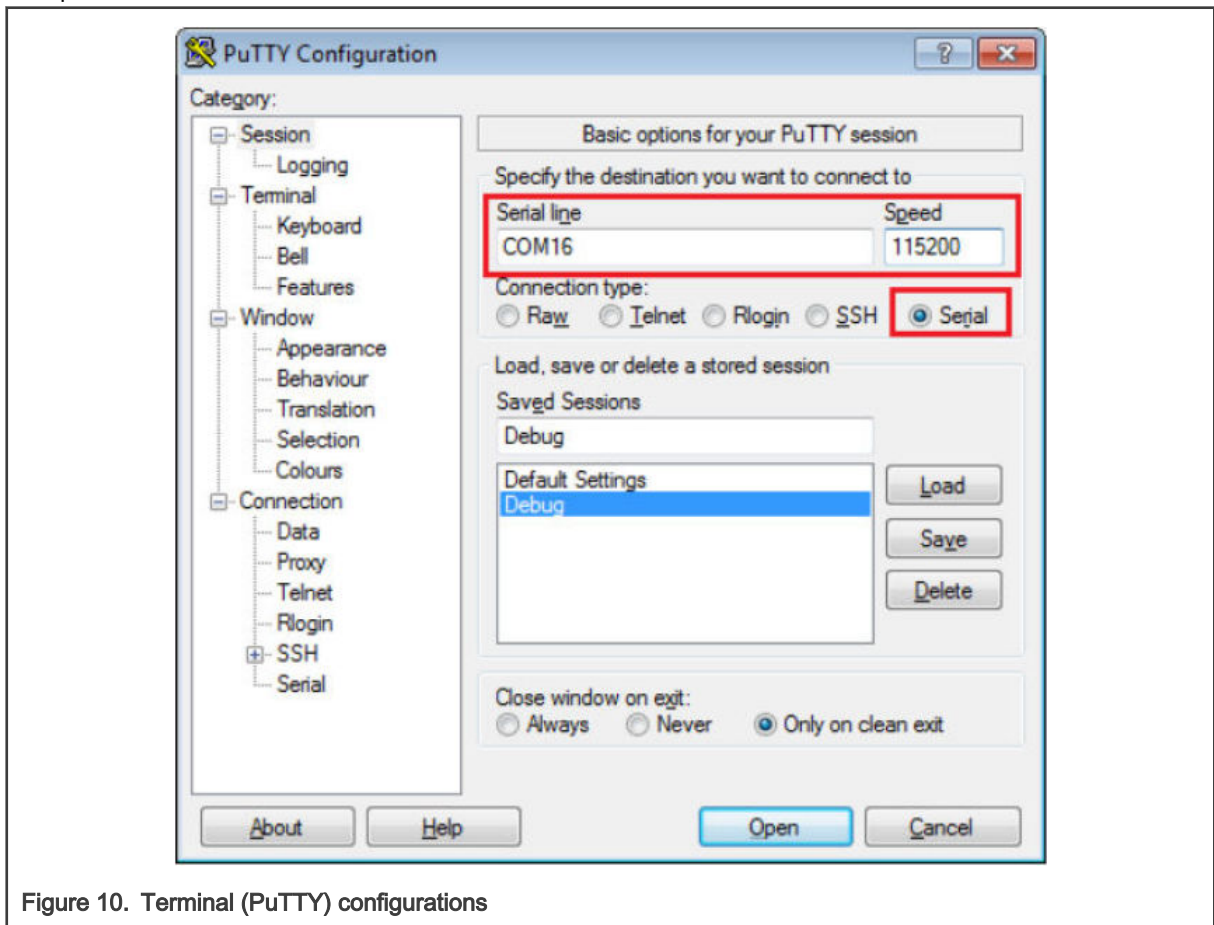
Figure 10. Terminal (PuTTY) configurations

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 *MIMX8ML_M7* device:

```
$ JLinkGDBServer -if JTAG -device
SEGGER J-Link GDB Server Command Line Version
JLinkARM.dll
Command line: -if JTAG -device MIMX8ML8_M7
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port: 2333
Accept remote connection: yes
< -- Skipping lines -- >
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:
Target interface: JTAG
Target interface speed: 1000 kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Feb 2 2020 18:12:40
Hardware: V10.10
S/N: 600109545
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 1.82 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 (Cortex-M7)
Connected to target
Waiting for GDB connection...
```

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/evkmimx8mp/demo_apps/hello_world/armgcc/debug

5. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 9-2019-q4-major) 8.3.0.20190709-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=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
```

```
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...
(gdb)
```

6. Connect to the GDB server and load the binary by running the following commands:

   a. `target remote localhost:2331`

   b. `monitor reset`

   c. `monitor halt`

   d. `load`

```
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000008 in __isr_vector ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load

Loading section .interrupts, size 0x240 lma 0x0
Loading section .text, size 0x3ab8 lma 0x240
Loading section .ARM, size size 0x8 lma 0x3cf8
Loading section .init_array, size 0x4 lma 0x3d00
Loading section .fini_array, size 0x4 lma 0x3d04
Loading section .data, size 0x64 lma 0x3d08
Start address 0x2f4, load size 15724
Transfer rate: 264 KB/sec, 2620 bytes/write.
(gdb)
```

The application is now downloaded and halted at the reset vector. Execute the `monitor go` command to start the demo application.

```
(gdb) monitor go
```

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.
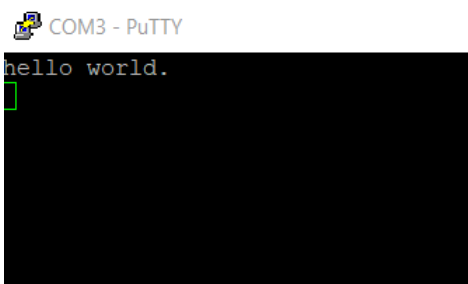
Figure 11. Text display of the hello_world demo

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-MIMX8MP* (document MCUXSDKIMX8MPRN).

<div align="center">

NOTE

See Host setup for Windows OS before compiling the application.

</div>

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.
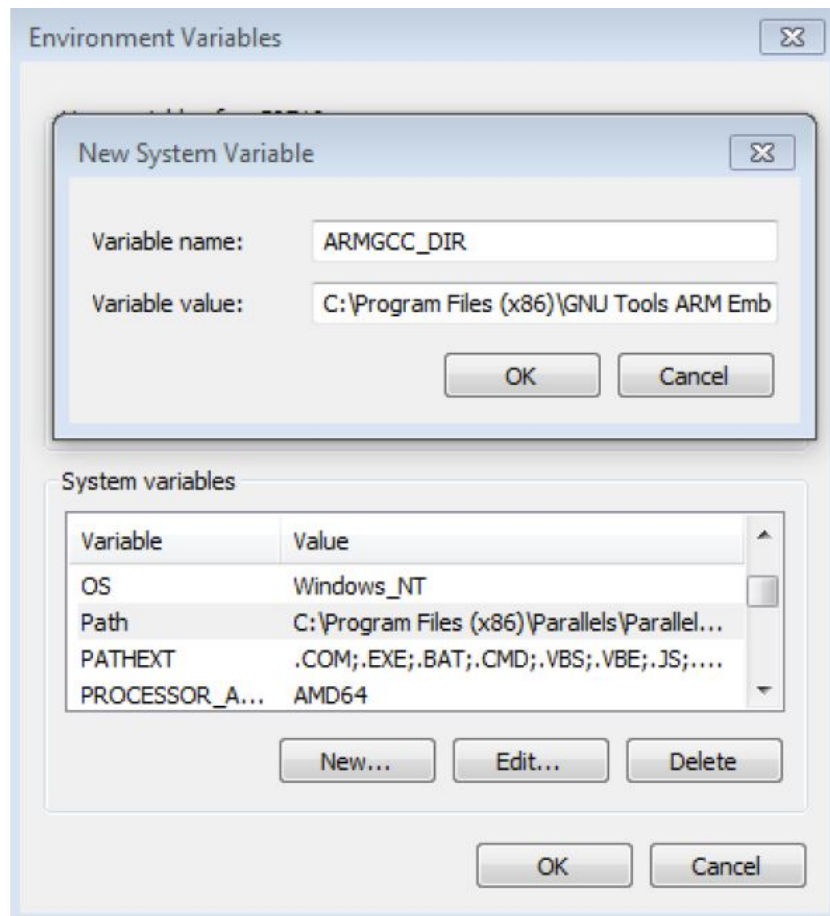
Figure 12. Add ARMGCC_DIR system variable

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**.
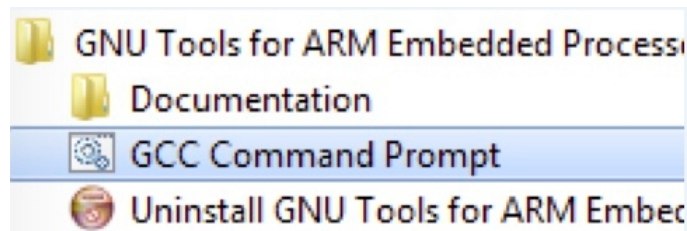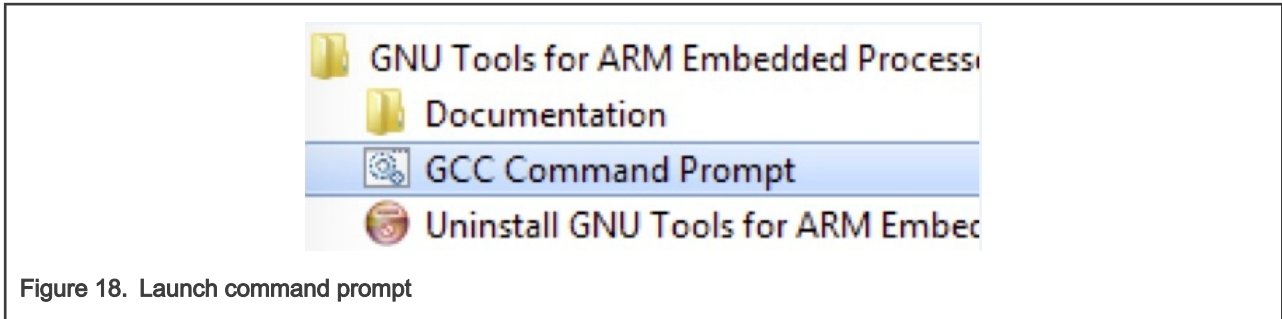

Figure 13. Launch command prompt

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/evkmimx8mp/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 14.

Figure 14. hello_world demo build successful

5.2.3 Run an example application

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:

• You have a standalone J-Link pod that is connected to the debug interface of your board.

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

   b. No parity

   c. 8 data bits

   d. 1 stop bit
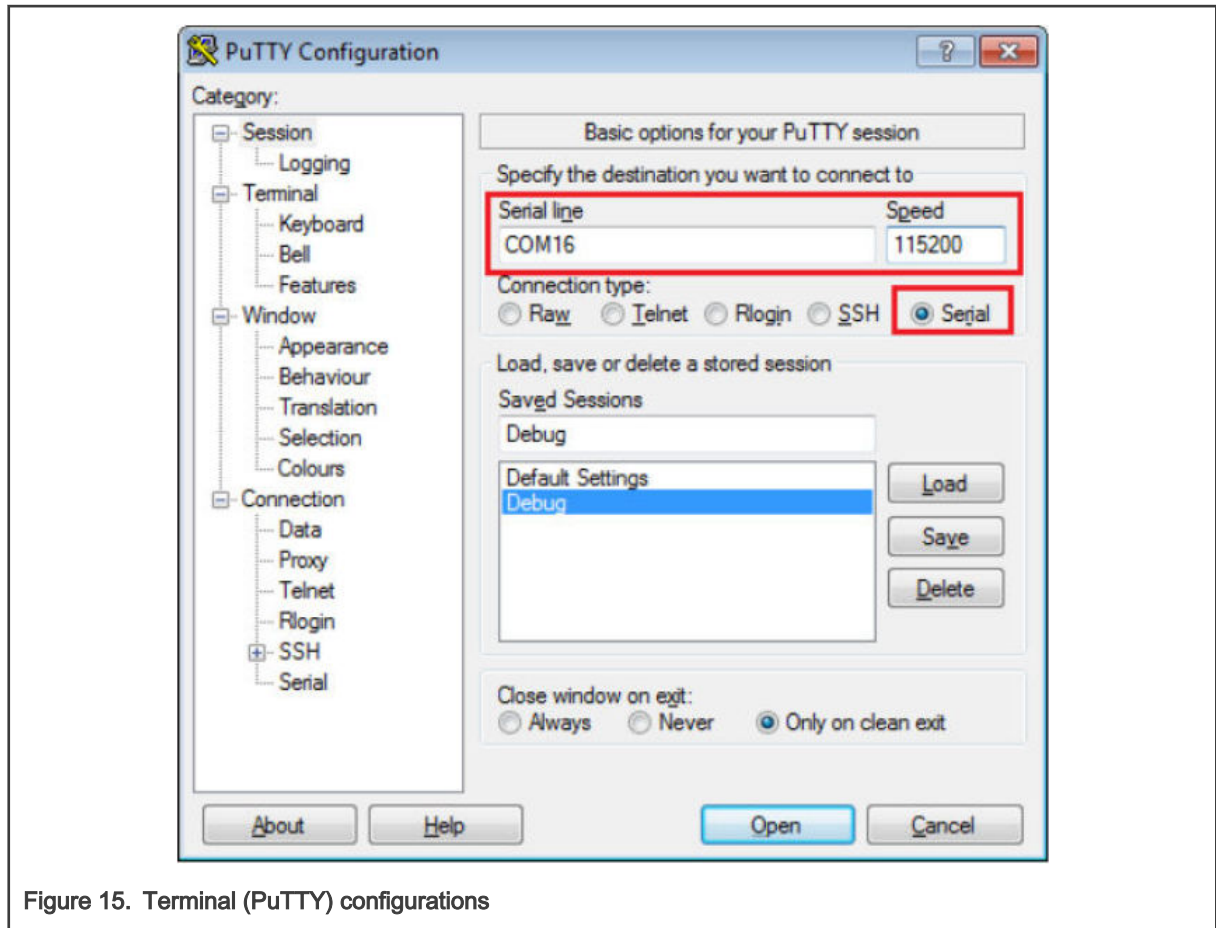
Figure 15. Terminal (PuTTY) configurations

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 16. The target device selection chosen for this example is **MIMX8ML8_M7**.

Figure 16. SEGGER J-Link GDB server configuration
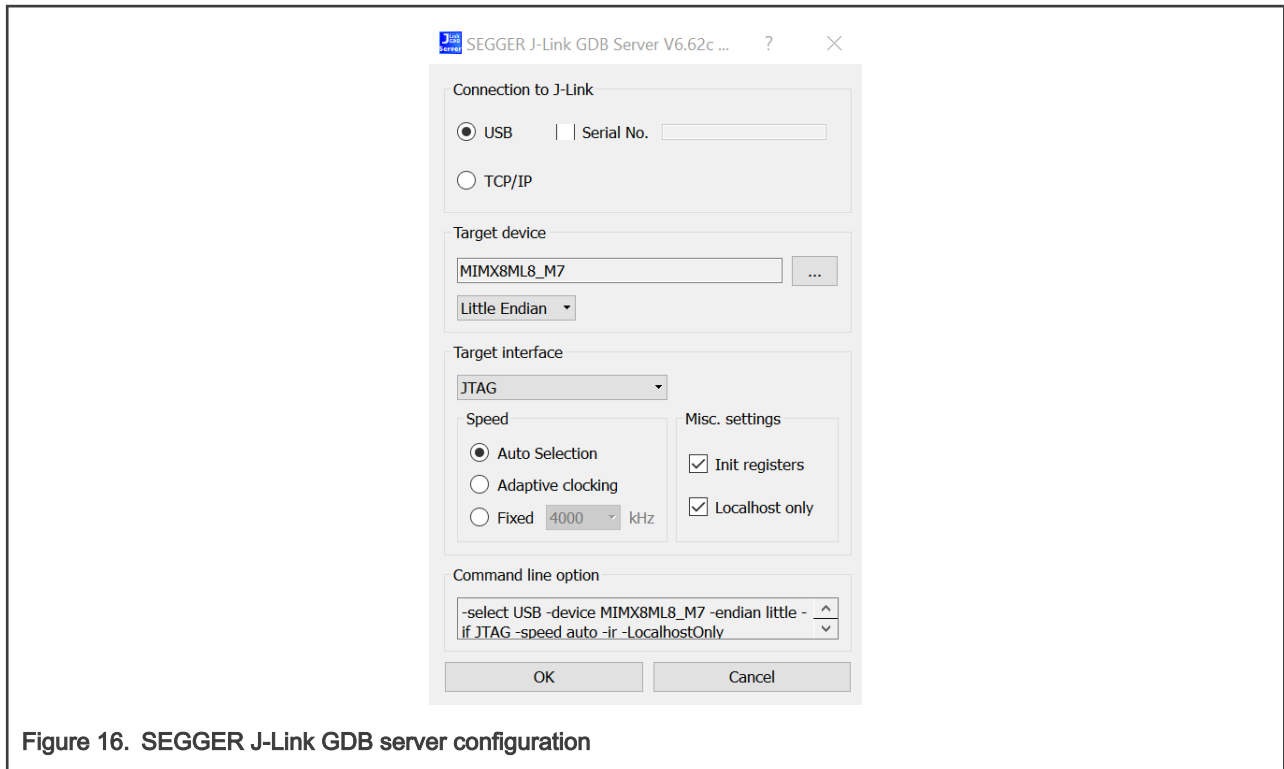
5. After GDB server is running, the screen should resemble Figure 17.

Figure 17. SEGGER J-Link GDB server screen after successful connection
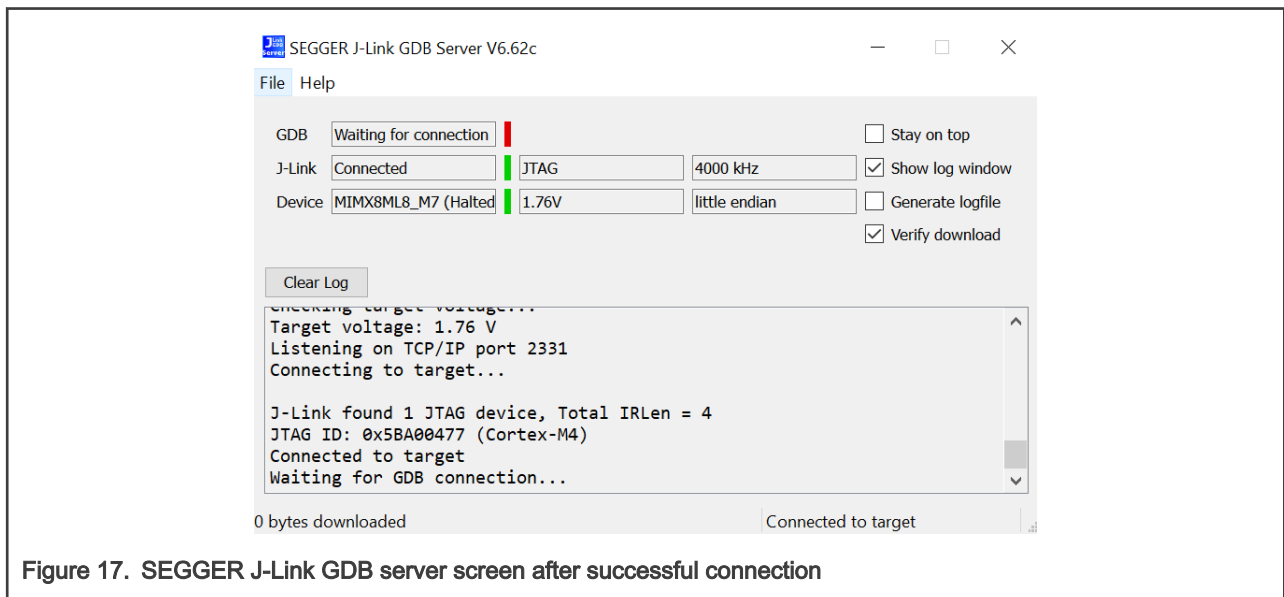
6. If not already running, open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system **Start** menu, go to **Programs** -> **GNU Tools ARM Embedded <version>** and select **GCC Command Prompt**.

Figure 18. 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/evkmimx8mp/demo_apps/hello_world/armgcc/debug

8. Run the command of `arm-none-eabi-gdb.exe <application_name>.elf`. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.
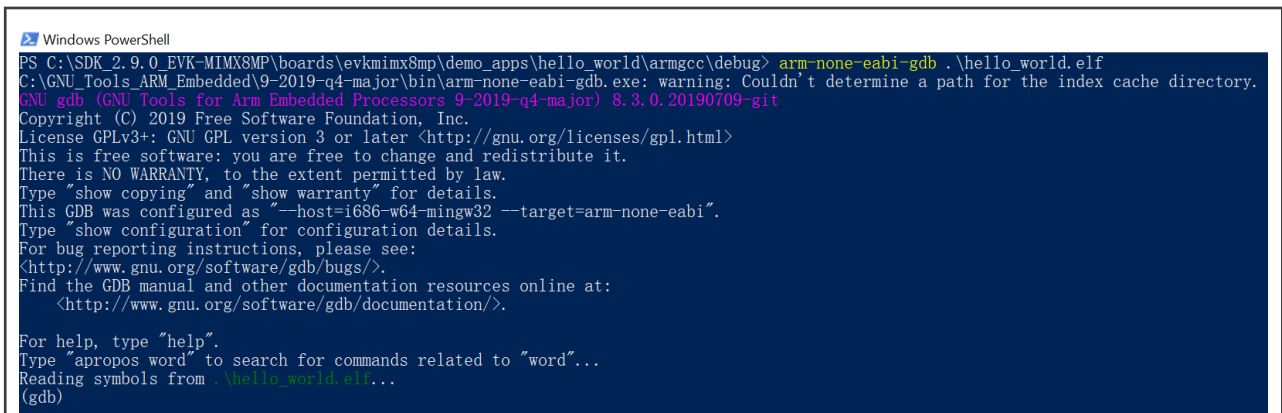
Figure 19. Run arm-none-eabi-gdb

9. Run these commands:

   a. `target remote localhost:2331`

   b. `monitor reset`

   c. `monitor halt`

   d. `load`

10. The application is now downloaded and halted at the reset vector. Execute the `monitor go` command to start the demo application.

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

Figure 20. Text display of the hello_world demo

Chapter 6
Run an application by U-Boot

This section describes the steps to write a bootable SDK bin file to TCM, DRAM and QSPI/XIP with the prebuilt U-Boot image for the i.MX processor. The following steps describe how to use the U-Boot:

1. Connect the **DEBUG UART** slot on the board to your PC through the USB cable. The Windows® OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.

2. On Windows OS, open the device manager, find **USB serial Port** in **Ports (COM & LPT)**. Assume that the ports are COM71 - COM74. COM73 is for the debug message from the Cortex®-A53 and COM74 is for the Cortex®-M7. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name `/dev/ttyUSB*` to determine your debug port. Similar to Windows OS, opening both is beneficial for development.
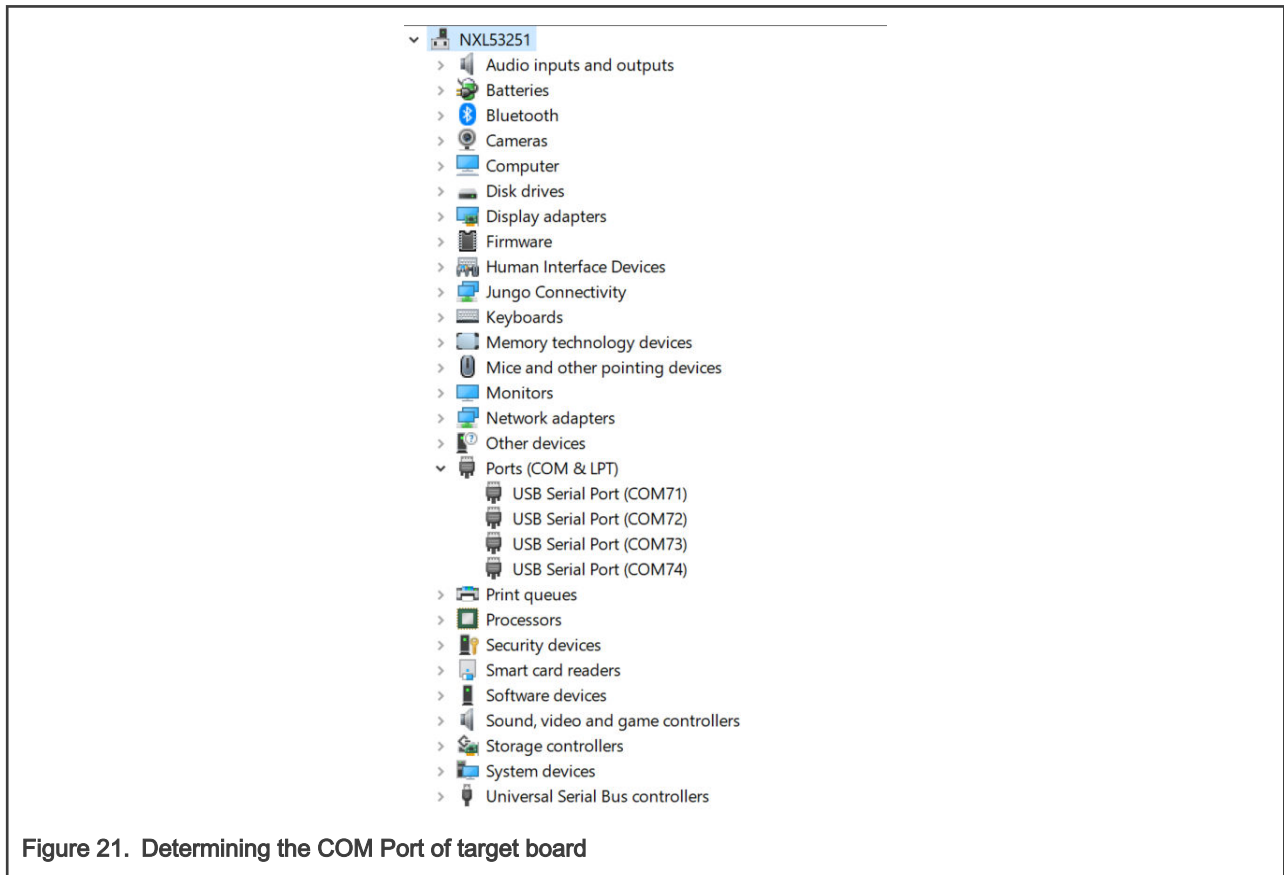
Figure 21. Determining the COM Port of target board

3. Build the application (for example, `hello_world`) to get the bin file (*hello_world.bin*).

4. Prepare an SD card with the prebuilt U-Boot image and copy bin file (*hello_world.bin*) into the SD card. Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dipswitch configuration.

5. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.

6. Power on the board and hit any key to stop autoboot in the terminals, then enter to U-Boot command line mode. You can then write the image and run it from TCM, DRAM and QSPI/XIP with the following commands:

   a. If the *hello_world.bin* is made from the debug/release target, which means the binary file will run at TCM, use the following commands to boot:

   - `fatload mmc 1:1 0x48000000 hello_world.bin;cp.b 0x48000000 0x7e0000 20000;`

- `bootaux 0x7e0000`

b. If the *hello_world.bin* is made from the `ddr_debug`/`ddr_release` target, which means the binary file runs at DRAM, use the following commands:

- `fatload mmc 1:1 0x80000000 hello_world.bin`

- `dcache flush`

- `bootaux 0x80000000`

c. If the *hello_world.bin* is made from the `flash_debug`/`flash_release` target, which means the binary file runs at QSPI/XIP, use the following commands:

- `fatload mmc 1:1 0x48000000 hello_world.bin`

- `dcache flush`

- `sf probe;sf erase 0 0x100000`

- `sf write 0x48000000 0 0x100000`

- `bootaux 0x8000000`

NOTE

If the Linux OS kernel runs together with M7, make sure the correct *dtb* file is used. This *dtb* file reserves resources used by M7 and avoids the Linux kernel from configuring them. Use the following command in U-Boot before running the kernel:

```
setenv fdt_file 'imx8mp-evk-rpmsg.dtb'
```

```
COM73 - PuTTY

U-Boot SPL 2019.04-5.4.3-1.1.0+g8beac7ef22 (Feb 06 2020 - 10:33:05 +0000)
power_pca9450b_init
DDRINFO: start DRAM init
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from ROMAPI
image offset 0x8000, pagesize 0x200, ivt offset 0x0


U-Boot 2019.04-5.4.3-1.1.0+g8beac7ef22 (Feb 06 2020 - 10:33:05 +0000)

CPU:   Freescale i.MX8MP rev1.0 2300 MHz (running at 1200 MHz)
CPU:   Commercial temperature grade (0C to 95C) at 43C
Reset cause: POR
Model: NXP i.MX8MPlus LPDDR4 EVK board
DRAM:  6 GiB
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C2 0x50]
SNK.Power3.0 on CC1
PDO 0: type 0, 5000 mV, 3000 mA [E]
PDO 1: type 0, 9000 mV, 3000 mA []
PDO 2: type 0, 15000 mV, 3000 mA []
PDO 3: type 0, 20000 mV, 2250 mA []
Requesting PDO 3: 20000 mV, 2250 mA
Source accept request
PD source ready!
tcpc_pd_receive_message: Polling ALERT register, TCPC_ALERT_RX_STATUS bit failed, ret = -62
Power supply on USB2
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C1 0x50]
MMC:   FSL_SDHC: 1, FSL_SDHC: 2
Loading Environment from MMC... Run CMD11 1.8V switch
OK
No panel detected: default to MIPI2HDMI
adv7535_init: Can't find device id=0x3d, on bus 1
Display: MIPI2HDMI (1920x1080)
Video: 1920x1080x24
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF a7b31cf
  - U-Boot 2019.04-5.4.3-1.1.0+g8beac7ef22

Run CMD11 1.8V switch
switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Run CMD11 1.8V switch
Net:
Error: ethernet@30be0000 address not set.

Error: ethernet@30be0000 address not set.

Error: ethernet@30bf0000 address not set.
eth-1: ethernet@30be0000
Error: ethernet@30bf0000 address not set.
, eth-1: ethernet@30bf0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
u-boot=> fatload mmc 1:1 0x48000000 hello_world.bin;cp.b 0x48000000 0x7e0000 20000;
16812 bytes read in 20 ms (820.3 KiB/s)
u-boot=> bootaux 0x7e0000
## Starting auxiliary core at 0x007E0000 ...
u-boot=>
```

Figure 22. U-Boot command to run application on TCM

Figure 23. U-Boot command to run application on DRAM
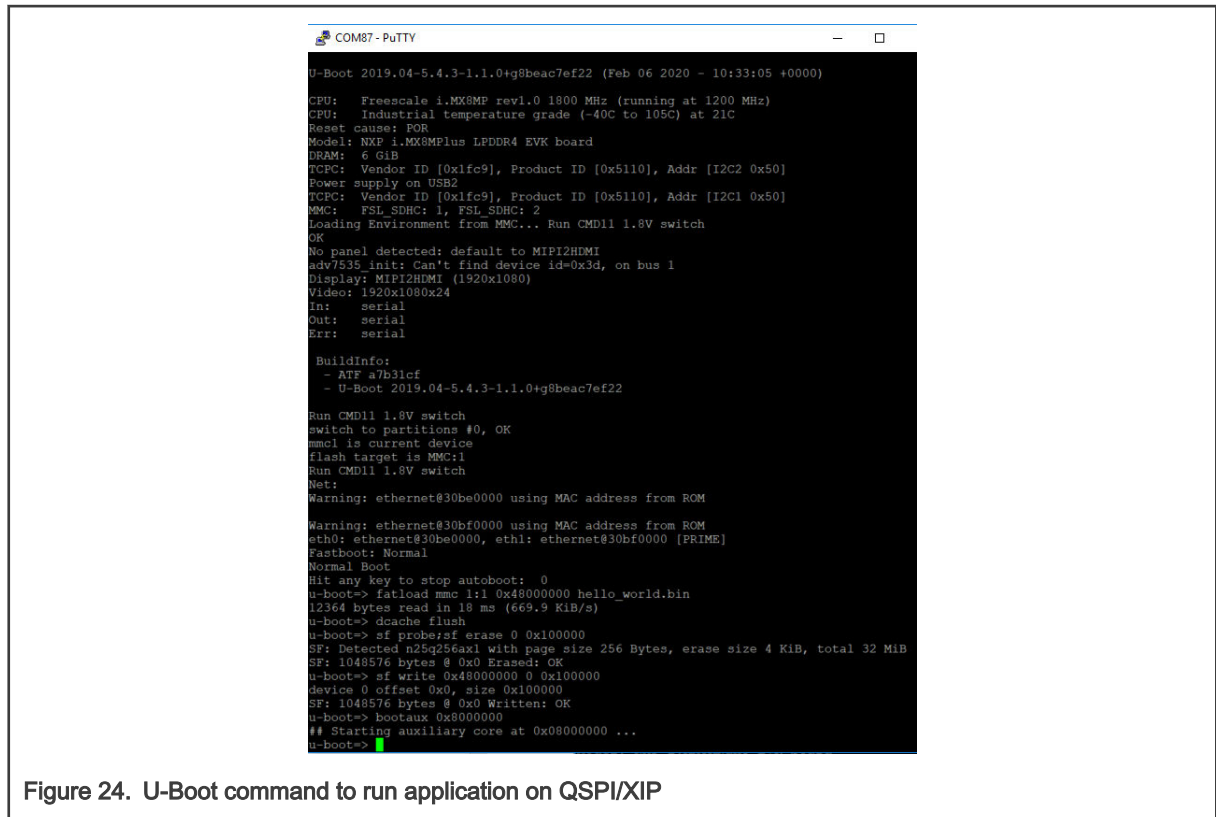
Figure 24. U-Boot command to run application on QSPI/XIP

7. Open another terminal application on the PC, such as PuTTY 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:

- 115200

- No parity

- 8 data bits

- 1 stop bit

8. 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.
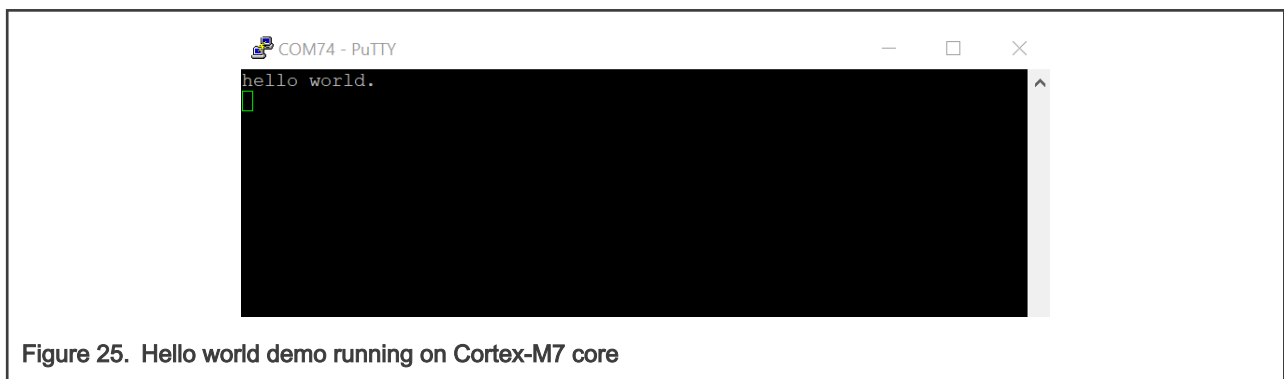
Figure 25. Hello world demo running on Cortex-M7 core

Chapter 7
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 8M Plus EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

7.1 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.

7.1.1 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 1.3.134 or higher for full support for the M7 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.

7.1.2 Switch to download mode

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

1. Set the board boot mode to Download Mode [SW0001[1-4]].

2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J301 on the base board.

3. The PC recognizes the i.MX 8M Plus device as (VID:PID)=(1FC9:0146), as shown in Figure 26.
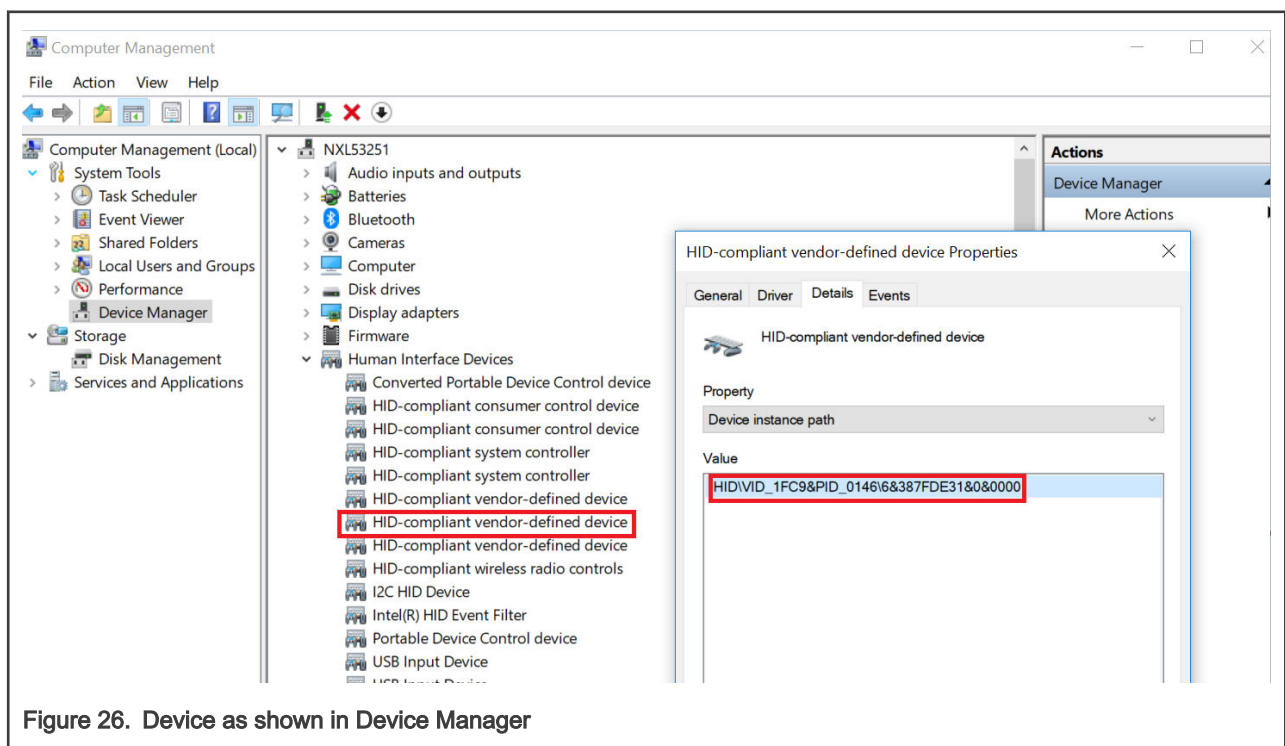
Figure 26. Device as shown in Device Manager

7.2 Build an example application

The following steps guide you through opening the `hello_world` example application. These steps may change slightly for other example applications, as some of these applications may have additional layers of folders in their 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 8M Plus EVK board as an example, the `hello_world` workspace is located in:

   <install_dir>/boards/evkmimx8mp/demo_apps/hello_world/iar/hello_world.eww

2. Select the desired build target from the drop-down. For this example, select **hello_world – flash_debug**.
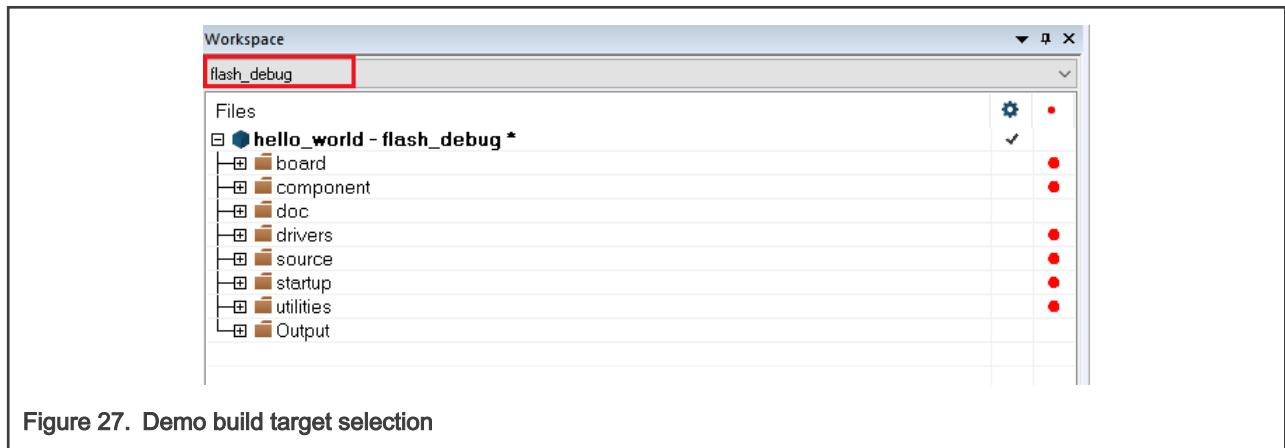
Figure 27. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 28.
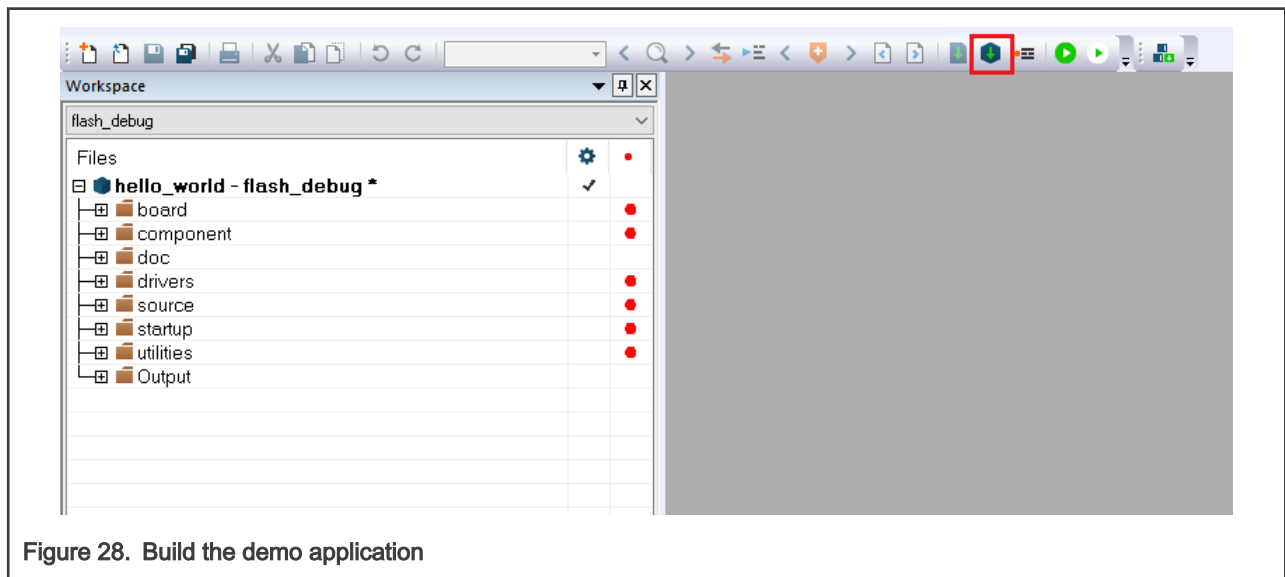
Figure 28. Build the demo application

4. The build completes without errors.

5. Copy it to the UUU tool directory.

7.3 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 J23 USB DEBUG connector and the PC. It provides console output while using UUU.

2. Connect the J6 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
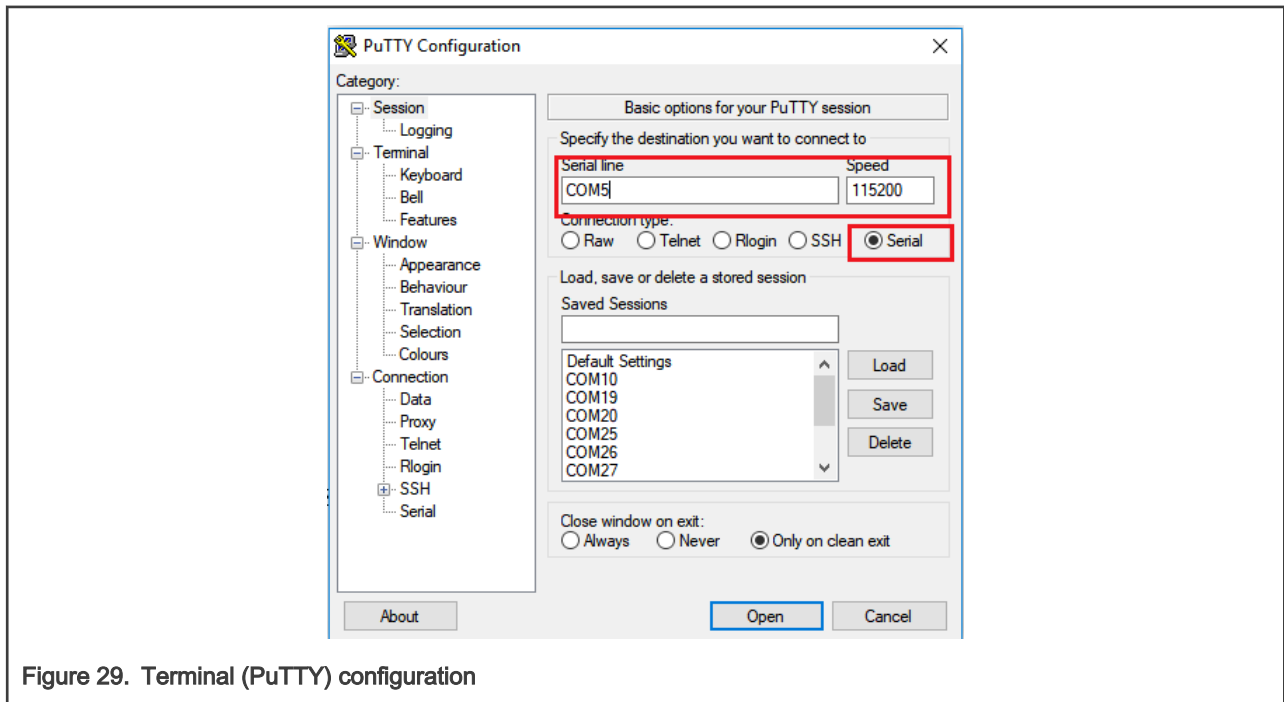
Figure 29. Terminal (PuTTY) configuration

4. Get the *fspi* version U-Boot image named **imx-boot-imx8mpevk-fspi.bin-flash_evk_flexspi** from the linux release package.

5. In the command line, execute uuu with the *-b qspi* switch: `uuu -b qspi imx-boot-imx8mpevk-fspi.bin-flash_evk_flexspi hello_world.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 30.

Figure 30. Command line and fast boot console output when executing UUU

6. Then, power off the board and change the boot mode to `MicroSD Mode [SW0011[1-4]]`, and power on the board again.

7. Use following command in U-Boot to kickoff m7:

```
sf probe
bootaux 0x8000000
```


Figure 31. Console output from QSPI Boot

Appendix A
How to determine COM port

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

1. 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 32.
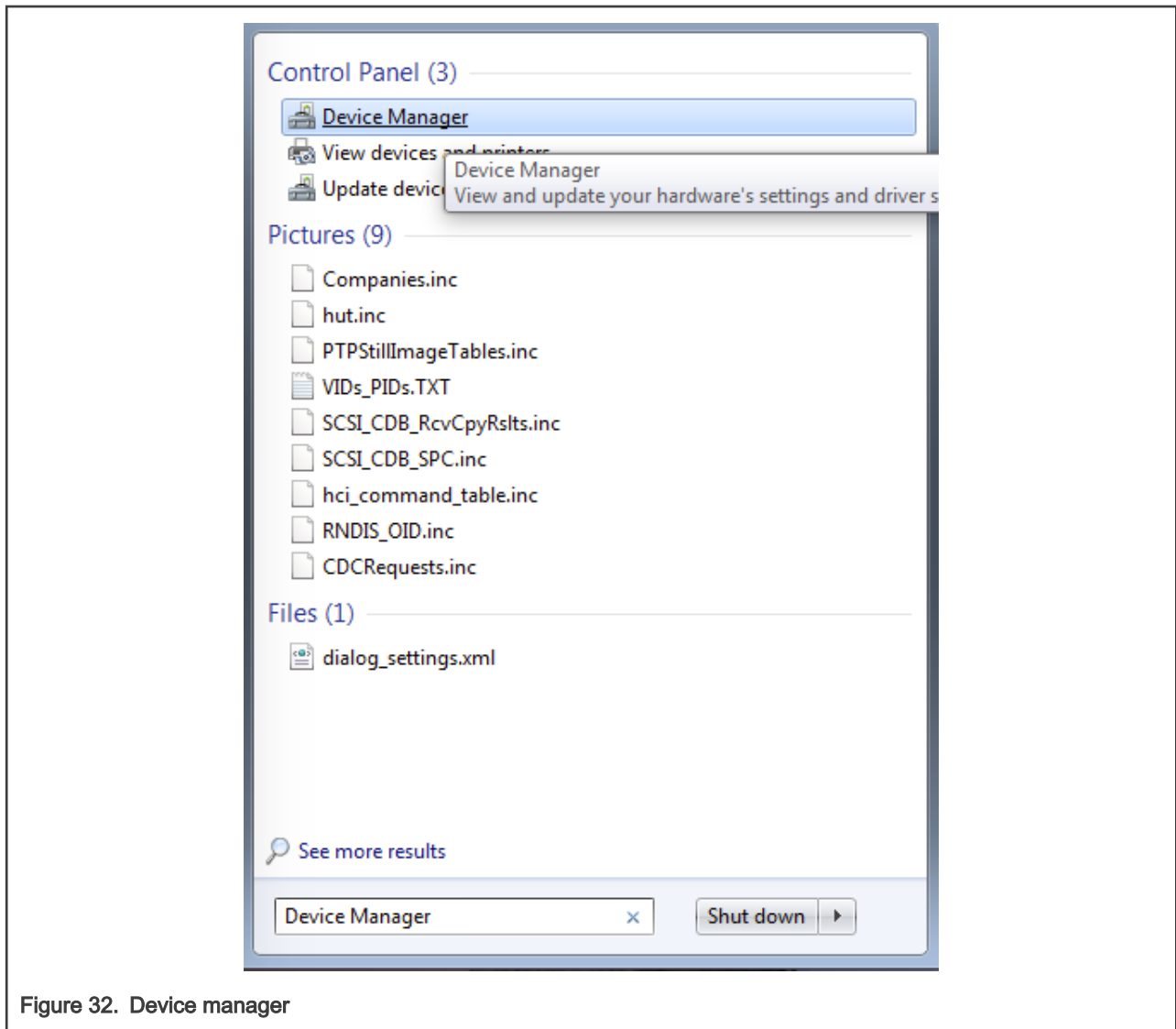
Figure 32. Device manager

2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. Depending on the NXP board you're using, the COM port can be named differently.

   a. USB-UART interface
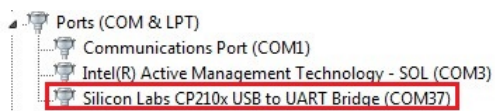
Figure 33. USB-UART interface

Appendix B
Host setup

An MCUXpresso 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**.
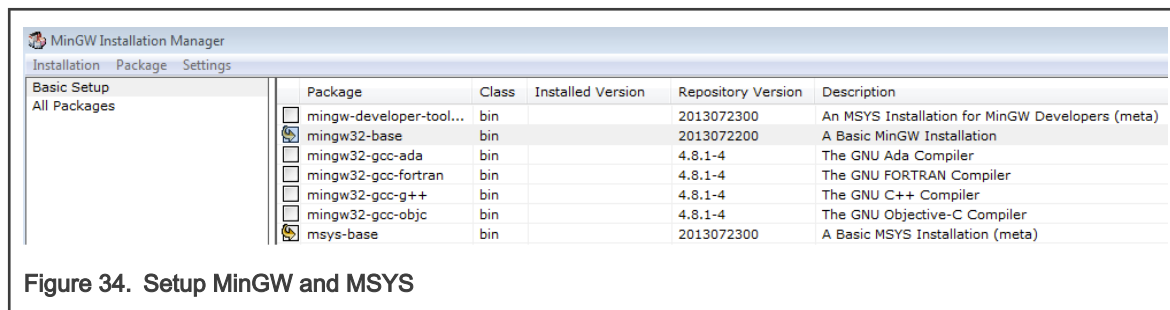
Figure 34. Setup MinGW and MSYS

    4. Click **Apply Changes** in the **Installation** menu and follow the remaining instructions to complete the installation.
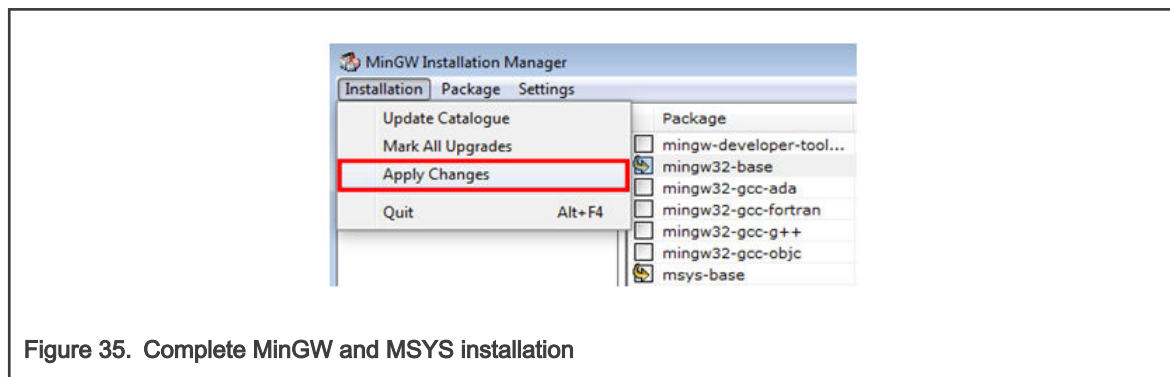
Figure 35. Complete MinGW and MSYS installation

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 36. If the path is not set correctly, the toolchain does not work.

NOTE

If you have *C:\MinGW\msys\x.x\bin* in your PATH variable (as required by KSDK 1.0.0), remove it to ensure that the new GCC build system works correctly.
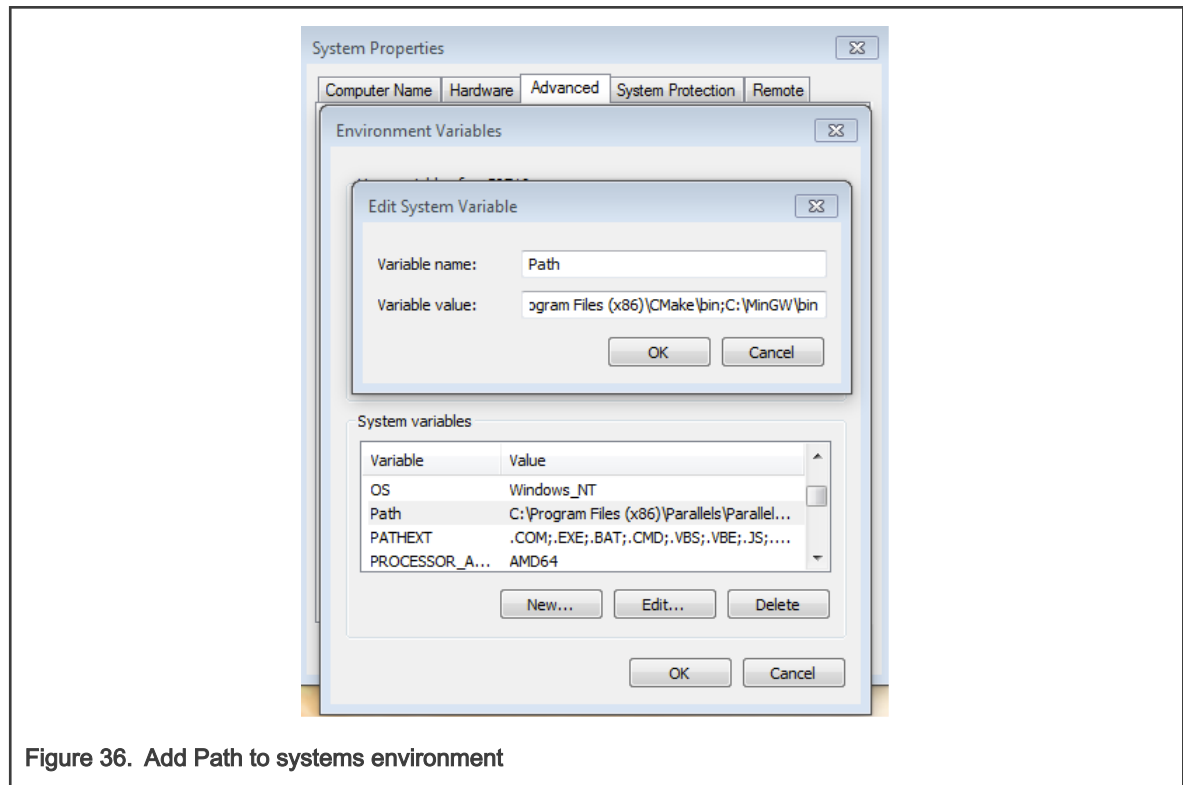
Figure 36. Add Path to systems environment

— 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.
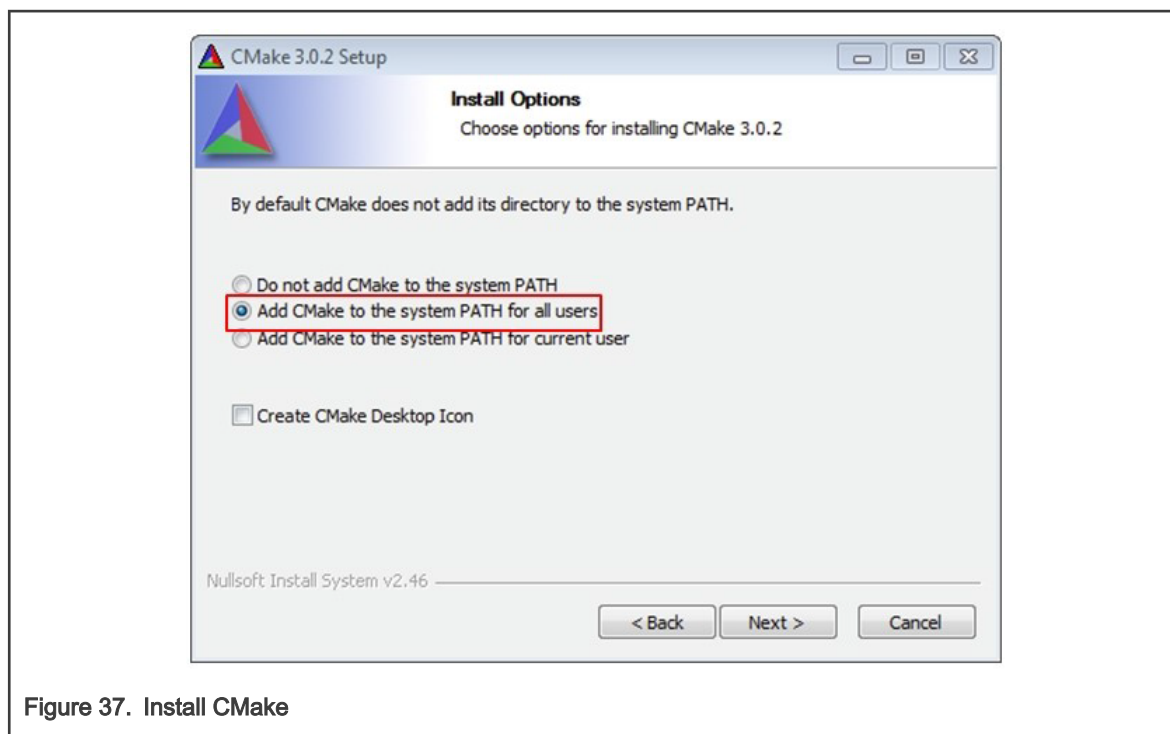
Figure 37. Install CMake

3. Follow the remaining instructions of the installer.

4. You may need to reboot your system for the PATH changes to take effect.

arm