

Getting Started with MCUXpresso SDK for MEK-MIMX8QM

1 Overview

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides bare metal source code to be executed in the i.MX 8QuadMax M4 cores. The MCUXpresso SDK provides comprehensive software support for NXP i.MX 8QuadMax microcontrollers (the cortex 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® 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 i.MX 8QuadMax* (document MCUXSDKIMX8QMRN).

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

Contents

1	Overview.....	1
2	MCUXpresso SDK board support folders.....	2
3	Toolchain introduction.....	3
4	Run a demo using Arm® GCC.....	4
5	Run a demo application using IAR.....	14
6	Run a demo using imx-mkimage.....	17
7	Run a flash target demo by UUU.....	20
8	Appendix A - How to determine COM port.....	24
9	Host setup.....	26
10	Revision history.....	29





Figure 1. MCUXpresso SDK layers

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. These applications typically only use a single peripheral, but there are cases where multiple are used (for example, SPI conversion using DMA).
- `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
- `mmcau_examples`: Simple applications intended to concisely illustrate how to use middleware/mmcau stack.
- `multicore_examples`: Simple applications intended to concisely illustrate how to use middleware/multicore stack.
- `issdk_examples`: Simple applications intended to concisely illustrate how to use middleware/issdk stack.
- `lwip_examples`: Simple applications intended to concisely illustrate how to use middleware/lwip 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 there is one directory for each M core. In the `hello_world/cm4_core0` application folder you see the following contents:

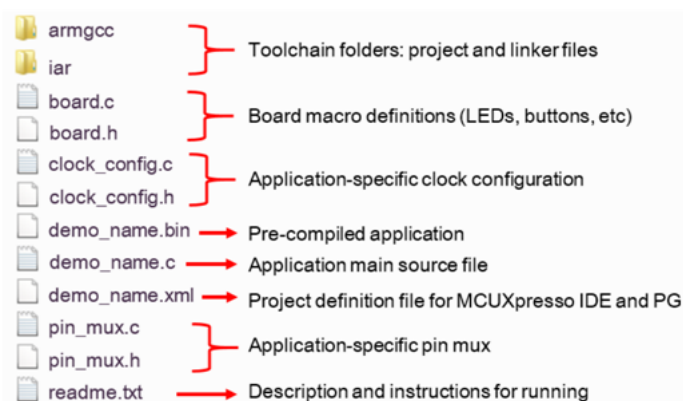


Figure 2. Application folder structure

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
- `devices/<device_name>/project`: Project template used in CMSIS PACK new project creation

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.

3 Toolchain introduction

The MCUXpresso SDK release for i.MX 8QuadMax 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 8QuadMax 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 8QuadMax, the patch for IAR should be installed. The patch named [iar_support_patch_imx8qm.zip](#) can be used with MCUXpresso SDK. See the `readme.txt` in the patch for additional information about patch installation.

3.2 Image creator

The i.MX 8QuadMax 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 8QuadMax device. Currently, the tool can only be executed on Linux OS.

4 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 8QuadMax platform is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

4.1 Linux OS host

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

4.1.1 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.

4.1.1.1 Install GCC Arm embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. 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 the *MCUXpresso SDK Release Notes*. (document MCUXSDKRN).

NOTE

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

4.1.1.2 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>
```

4.1.2 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/mekmimx8qm/demo_apps/hello_world/cm4_core0/armgcc

NOTE

To change directories, use the `cd` command.

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_2.3_MEK-MIMX8QM/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/armgcc
```

```
Scanning dependencies of target hello_world.elf
```

Run a demo using Arm® GCC

```
[ 4%] Building C object CMakeFiles/hello_world.elf.dir/work/platforms/tmp/SDK_2.3_MEK-MIMX8QM/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/board.c.obj
```

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

4.1.3 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 System Controller Firmware (SCFW) 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.

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, 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

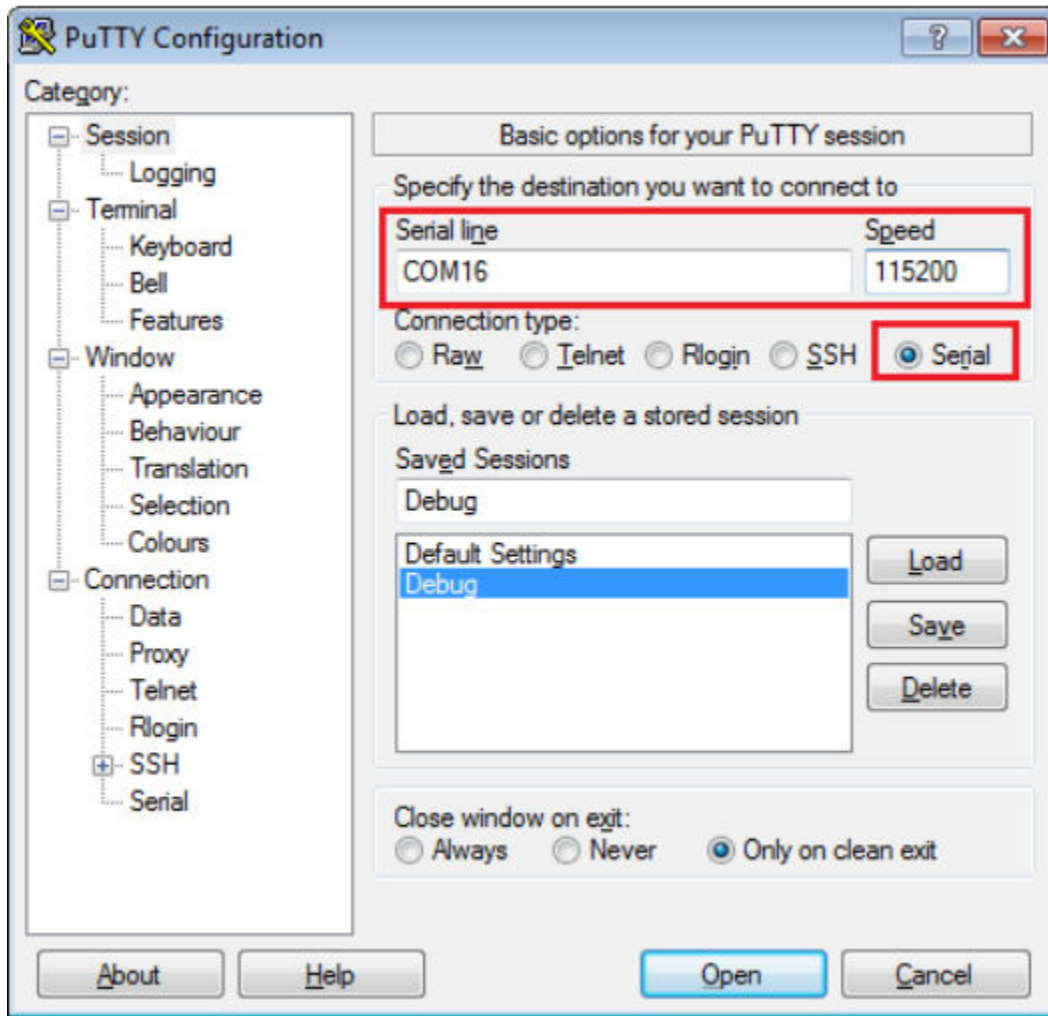


Figure 3. Terminal (PuTTY) configurations

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 MIMX8QM6_M4_0 device:

```
$ JLinkGDBServer -if JTAG -device MIMX8QM6_M4_0
SEGGER J-Link GDB Server V6.20f Command Line Version
JLinkARM.dll V6.20f (DLL compiled Oct 13 2017 17:18:54)
Command line: -if JTAG -device MIMX8QM6_M4_0
-----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: MIMX8QM6_M4_0
Target interface: JTAG
Target interface speed: 1000 kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V9 compiled Oct 6 2017 16:38:28
```

Run a demo using Arm® GCC

```
Hardware: V9.30
S/N: 609302772
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 1.79 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...
```

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>/<core_instance>/
armgcc/debug
```

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

For this example, the path is:

```
<install_dir>/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/armgcc/debug
```

5. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf
GNU gdb (7.10-lubuntu3+9) 7.10
Copyright (C) 2015 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...
(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
0x0000025e in ?? ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0xa00 lma 0x1ffe0000
Loading section .text, size 0x2684 lma 0x1ffe0a00
Loading section .ARM, size 0x8 lma 0x1ffe3084
Loading section .init_array, size 0x4 lma 0x1ffe308c
Loading section .fini_array, size 0x4 lma 0x1ffe3090
Loading section .data, size 0x68 lma 0x1ffe3094
Start address 0x1ffe0ad0, load size 12540
Transfer rate: 84 KB/sec, 1567 bytes/write.
```


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.

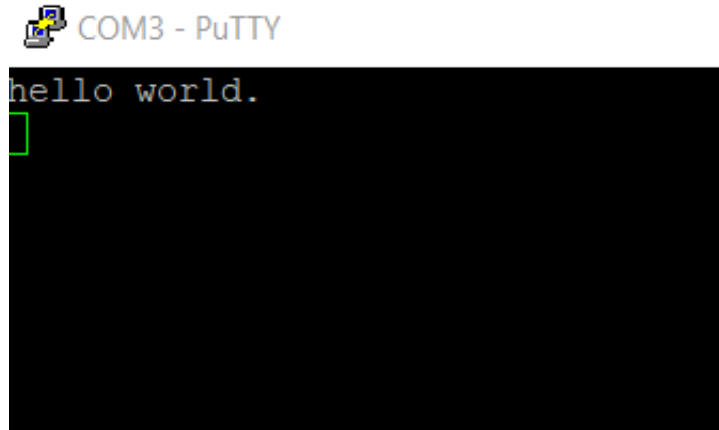


Figure 4. Text display of the `hello_world` demo

4.2 Windows OS host

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

4.2.1 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.

4.2.1.1 Install GCC Arm Embedded tool chain

Download and run the installer from developer.arm.com/open-source/gnu-toolchain/gnu-rm. 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*.

4.2.1.2 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. For this example, the path is:

```
C:\Program Files (x86)\GNU Tools ARM Embedded\6 2017-q2-update
```

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

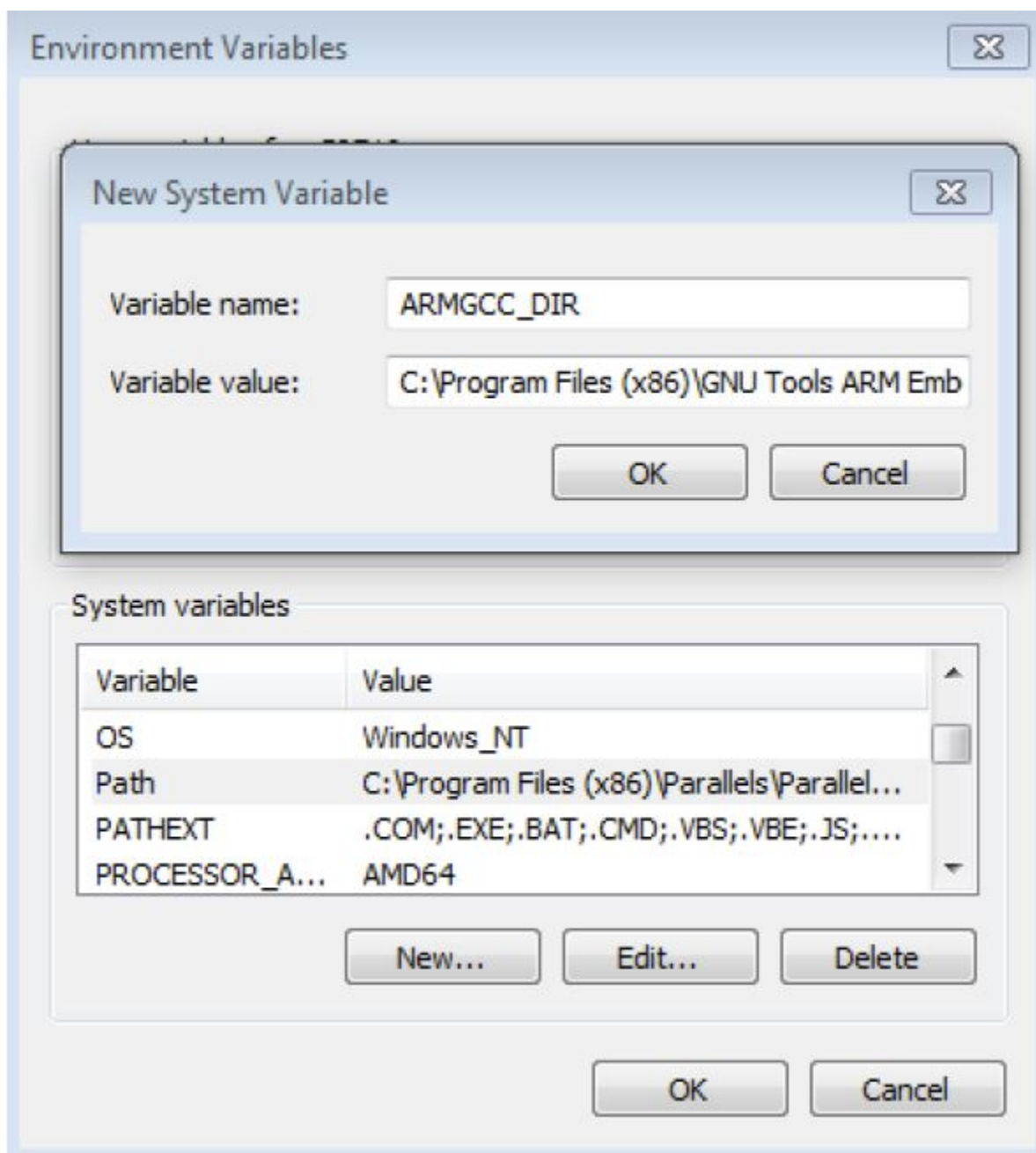


Figure 5. Add ARMGCC_DIR system variable

4.2.2 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**.

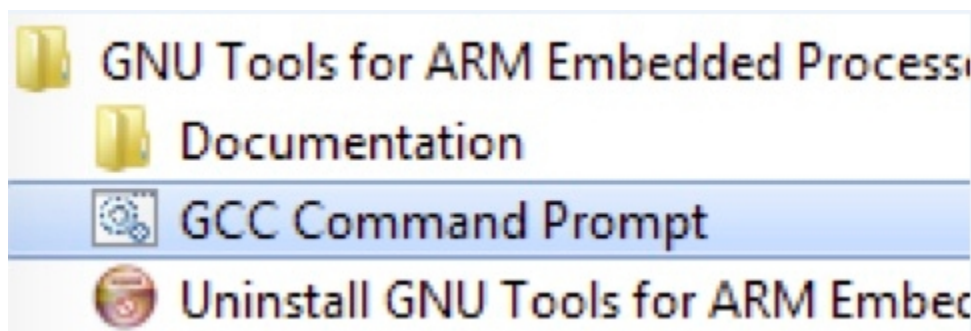


Figure 6. Launch command prompt

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>/<core_instance>/armgcc
```

For this example, the exact path is:

```
<install_dir>/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/armgcc
```

NOTE

To change directories, use the **cd** command.

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 7](#).

```
[ 95%] [100%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.2_UAL-MIMX8QM/boards/valmimx8qm/demo_apps/hello_world/cm4_core0/hello_world.c.obj
Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.2_UAL-MIMX8QM/boards/valmimx8qm/demo_apps/hello_world/cm4_core0/pin_mux.c.obj
Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\nxp\SDK_2.2_UAL-MIMX8QM\boards\valmimx8qm\demo_apps\hello_world\cm4_core0\armgcc>IF "" == "" <pause>
Press any key to continue . . .
```

Figure 7. hello_world demo build successful

4.2.3 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:

- 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.

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

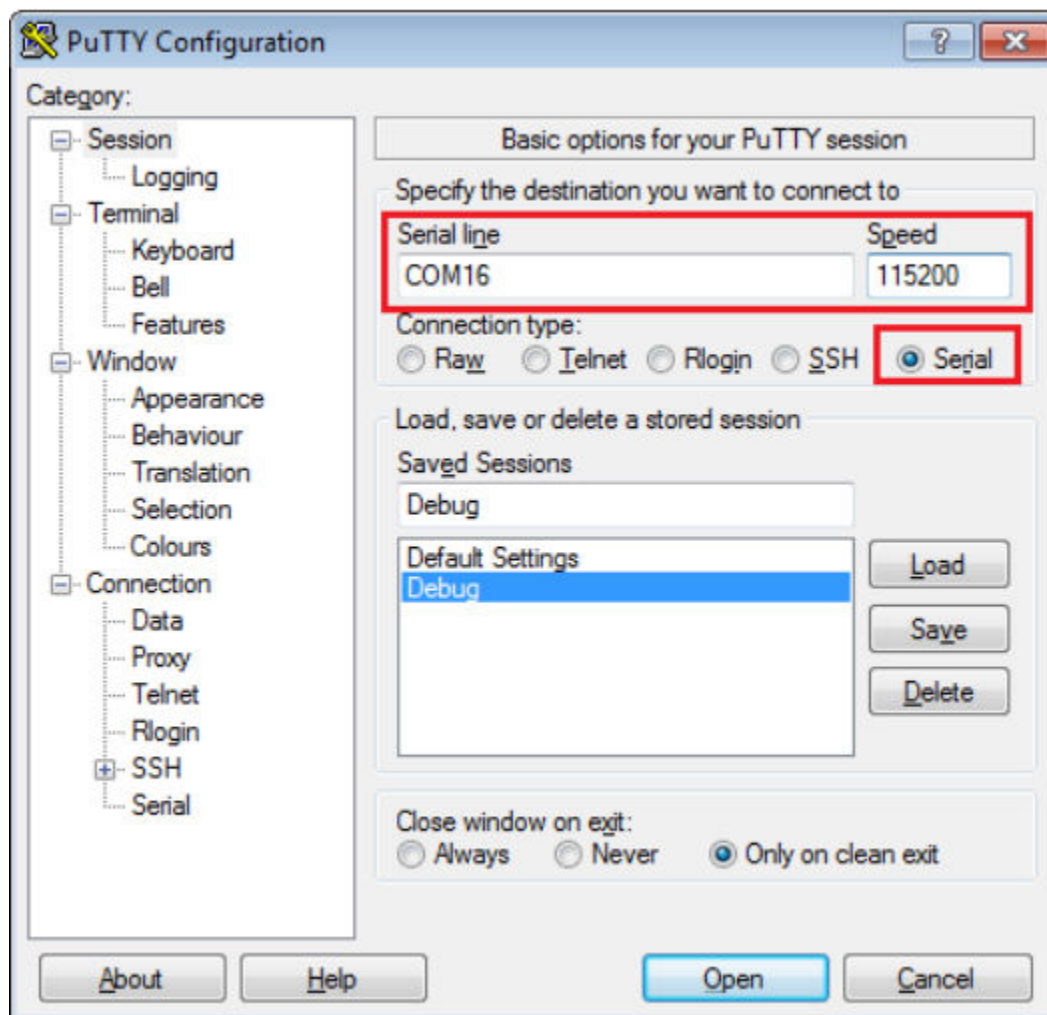


Figure 8. Terminal (PuTTY) configurations

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 . The target device selection chosen for this example is MIMX8QM6_M4_0 .
5. After GDB server is running, the screen should resemble [Figure 9](#) :

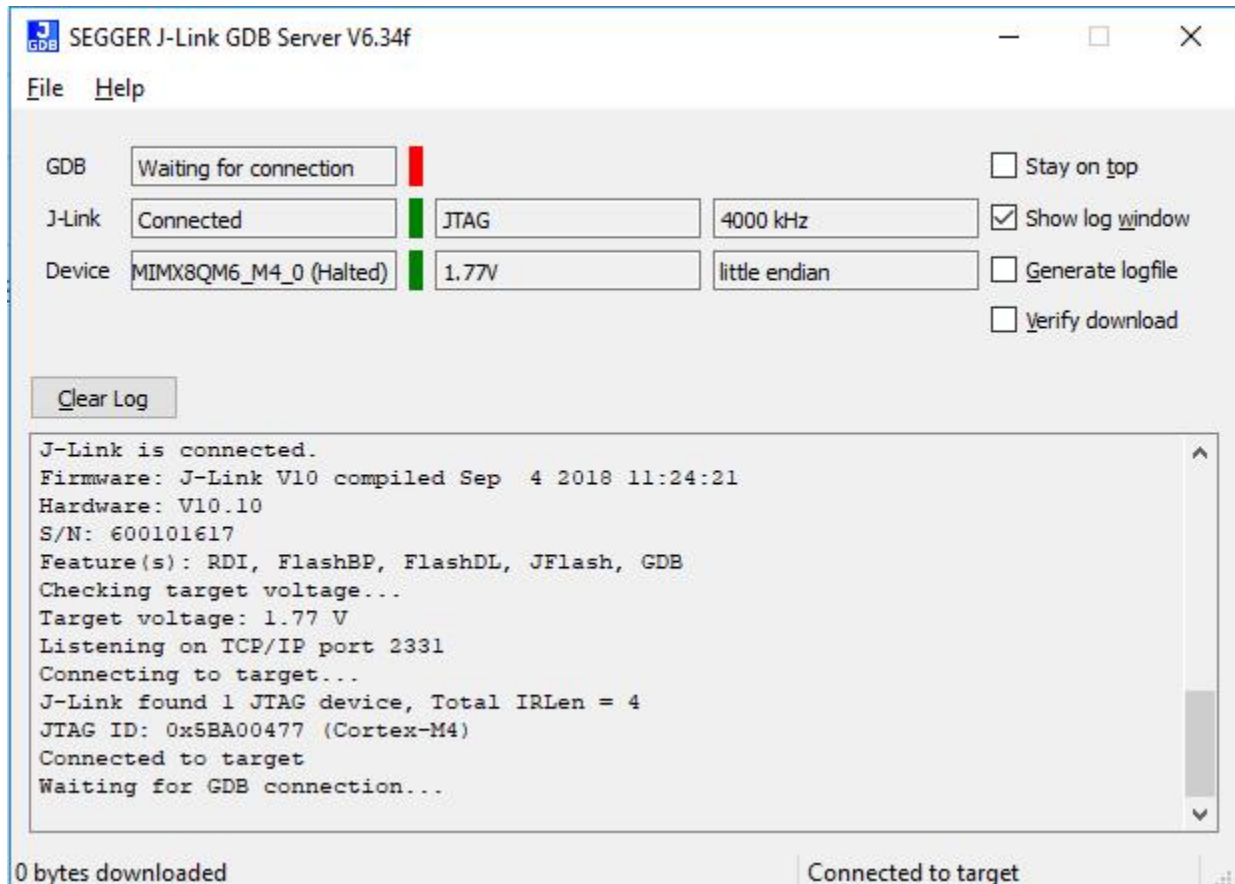


Figure 9. SEGGER J-Link GDB server screen after successful connection

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**.

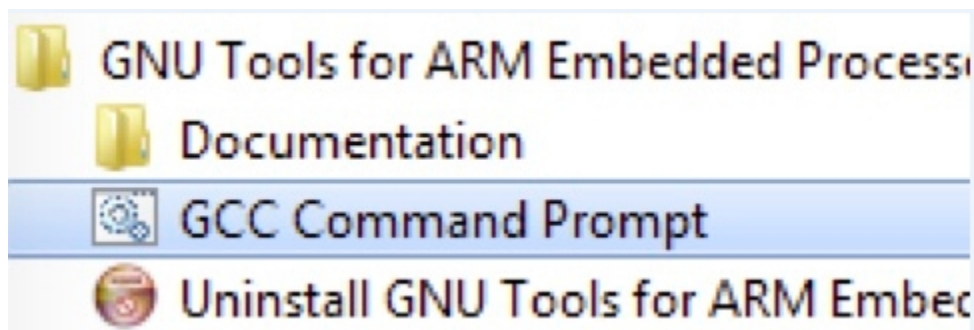


Figure 10. Launch command prompt

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/mekimx8qm/demo_apps/hello_world/cm4_core0/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`.

Run a demo application using IAR

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.

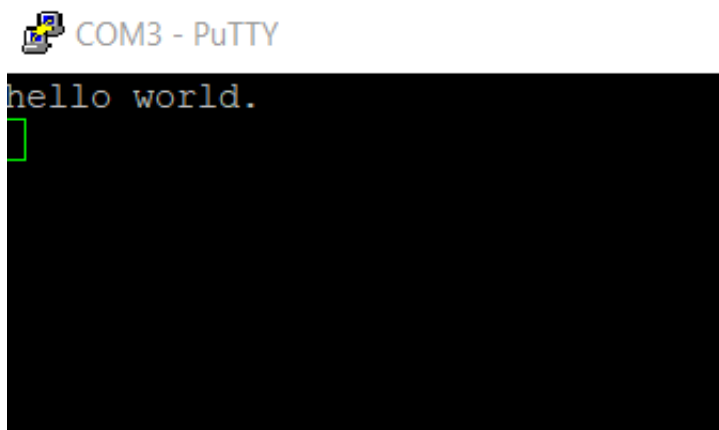


Figure 11. Text display of the `hello_world` demo

5 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 i.MX 8QuadMax MEK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

5.1 Build an example application

Before using IAR, get the IAR patch, [iar_support_patch_imx8qm.zip](#). Install the i.MX8QM support patch following the guides in `readme.txt` located in the archive.

Do the following steps to build the `hello_world` example application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

```
<install_dir>/boards/<board_name>/<example_type>/<application_name>/<core_instance>/iar
```

Using the i.MX 8QuadMax MEK hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/iar/hello_world_m40.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**.

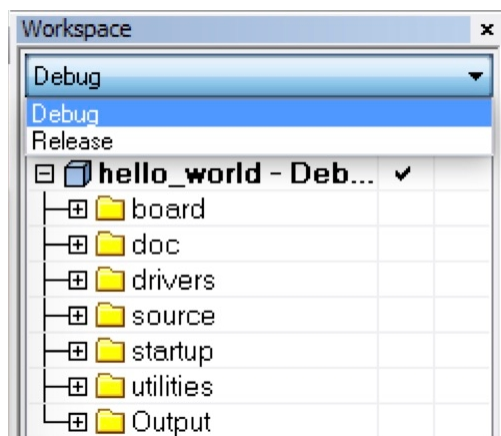


Figure 12. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 13.

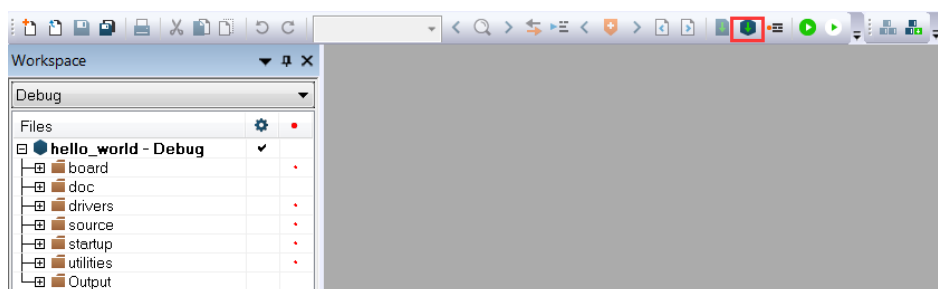


Figure 13. Build the demo application

4. The build completes without errors.

5.2 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 debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from www.segger.com/jlink-software.html.
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

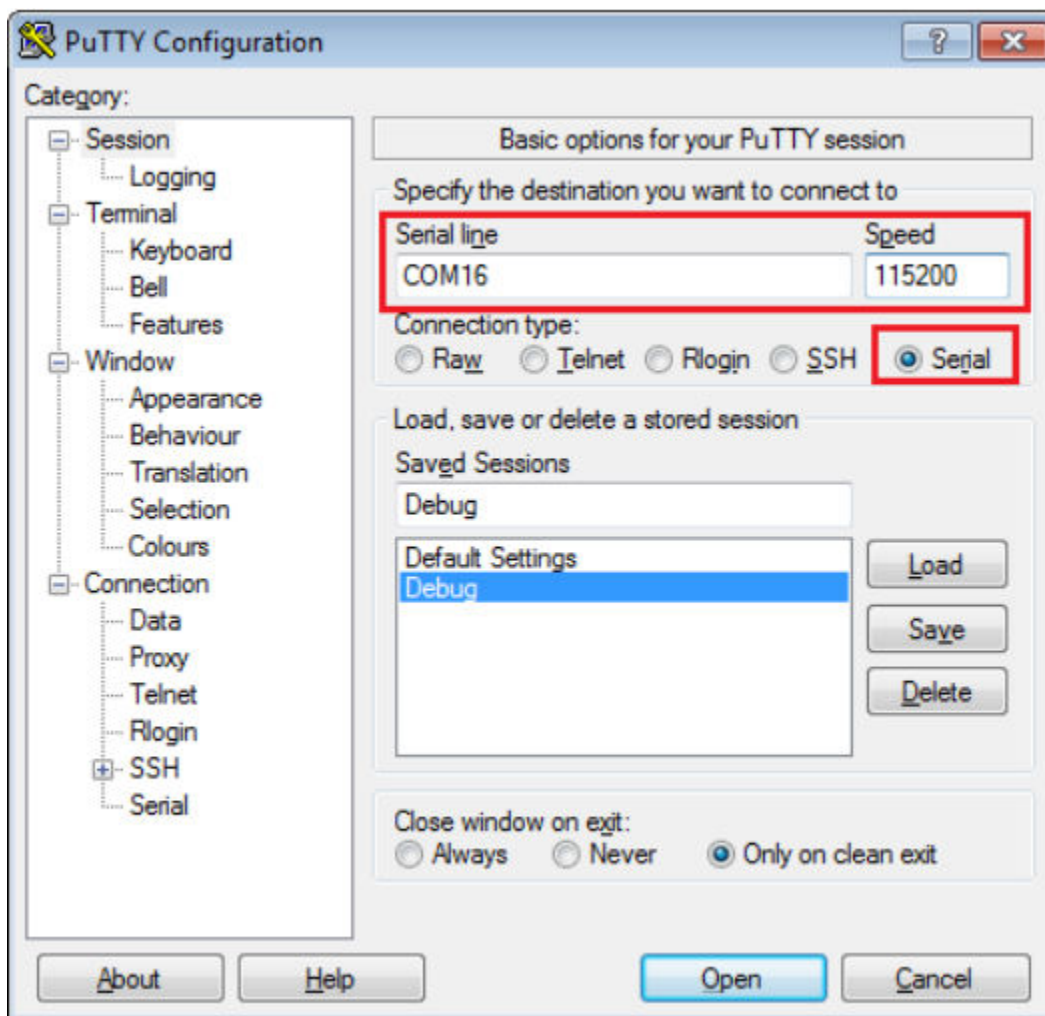


Figure 14. Terminal (PuTTY) configuration

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



Figure 15. Download and Debug button

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

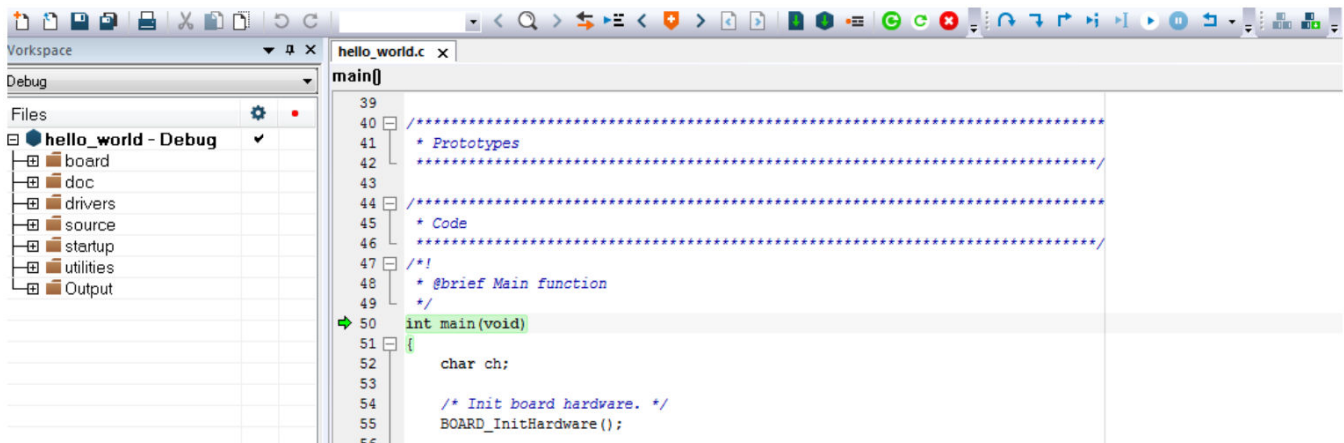


Figure 16. Stop at main() when running debugging

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



Figure 17. 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.

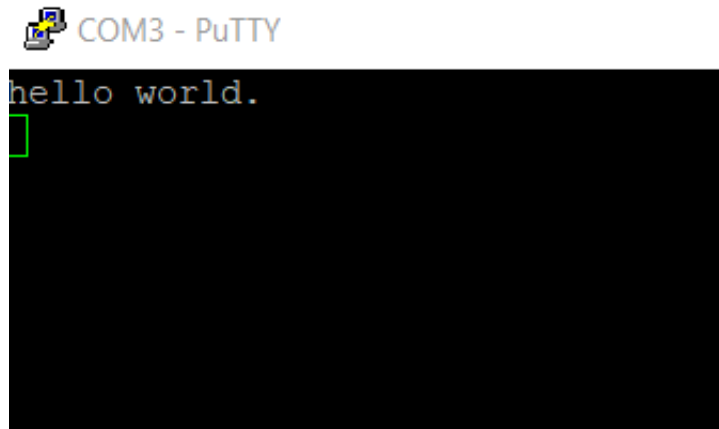


Figure 18. Text display of the hello_world demo

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 IVT (Image Vector Table) 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 provided in corresponding Linux Release Notes document.

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

Run a demo using imx-mkimage

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

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

This extracts the SCU firmware. Rename `mx8qm-mek-scfw-tcm.bin` to `scfw_tcm.bin` and copy the file to `imx-mkimage/iMX8QM`.

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

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

This extracts the i.MX SECO firmware. Copy `firmware/seco/mx8qm-ahab-container.img` to `imx-mkimage/iMX8QM`.

6. Generate the `u-boot.bin` from Linux release package and copy it to `imx-mkimage/iMX8QM`.
7. Generate the Arm Trusted Firmware `bl31.bin` from the Linux release package and copy it to `imx-mkimage/iMX8QM`.

6.1 Run an example application on one of the two M4 cores

1. Build the M4 demo application (for example, `hello_world`) with the RAM (TCM) linker file. Rename the generated binary file (.bin file) to `m4_image.bin`, and copy to this file to the `<imx-mkimage install_dir>/iMX8QM` folder.
2. In Linux OS, bash cd into the `imx-mkimage` installed directory, and run the following command to generate bootable image:

If the binary file is for Cortex-M4 core0:

```
$ make clean
```

```
$ make SOC=iMX8QM flash_cm4
```

If the binary file is for Cortex-M4 core1:

```
$ make clean
```

```
$ make SOC=iMX8QM flash_cm41
```

This generates the bootable image `flash.bin` under the “iMX8QM” 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=./iMX8QM/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. Connect the development platform to your PC via USB cable between the USB-UART connector and the PC USB connector. 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 Appendix A). Configure the terminal with these settings:
 - a. 115200 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in `board.h` file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit
5. Insert the SD card to SD1 card slot and power on the board. The M4 is running.

6.2 Run example applications on both M4 cores

1. Build the M4 demo application.

There are six targets for a demo, which are “debug”, “release”, “ddr_debug”, “ddr_release”, “flash_debug”, and “flash_release”. The “debug” and “release” targets use a RAM (TCM) linker file, while “ddr_debug” and “ddr_release” use a DDR linker file which puts text and data in DDR. Rename the built binary file (.bin file) to *m4_image.bin* for M4 core0, and *m4_1_image.bin* for M4 core1. Copy the renamed files into the *<imx-mkimage install_dir>/iMX8QM* folder.

2. In Linux OS, bash cd into the imx-mkimage installed directory and run the following command to generate a bootable image:

If the binary files are TCM target:

```
$ make clean
```

```
$ make SOC=iMX8QM flash_cm4s
```

If the binary files are DDR target:

```
$ make clean
```

```
$ make SOC=iMX8QM flash_cm4s_ddr
```

This will generate bootable image flash.bin under the “iMX8QM” folder. If U-Boot needs to be run, use the make targets named flash_linux_m4 for TCM targets, or flash_linux_m4_ddr for DDR targets.

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=./iMX8QM/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. Connect the development platform to your PC via USB cable between the USB-UART connector and the PC USB connector. 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 Appendix A). Configure the terminal with these settings:
 - a. 115200 baud rate, depending on your board (reference BOARD_DEBUG_UART_BAUDRATE variable in board.h file)
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit
5. Insert the SD card to SD1 card slot and power on the board. The M4s are running.

6.3 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=iMX8QM flash_scfw
```

Run a flash target demo by UUU

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

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 8QuadMax MEK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

The flash target (`flash_debug`, `flash_release`) which is built with flash XIP linker files cannot be debugged using IAR or GDB. Those targets can only be run through `imx-mkimage` and UUU.

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 [github](#). Use version 1.2.0 or higher for full support for the M4 image. Download `libusb-1.0.dll` and `uuu.exe` for Windows OS, or download UUU for Linux. Configure the path so that the executable can later be called anywhere in the command line.

7.1.2 Switch to SERIAL mode

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

1. Set SW2 on the CPU board to set the board boot mode to 'SERIAL BOOT' [b'000100] .
2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J17 USB Type-C on the CPU board.
3. The PC recognizes the i.MX 8QM device as (VID:PID)=(1FC9:0129), as shown in [Figure 19](#).

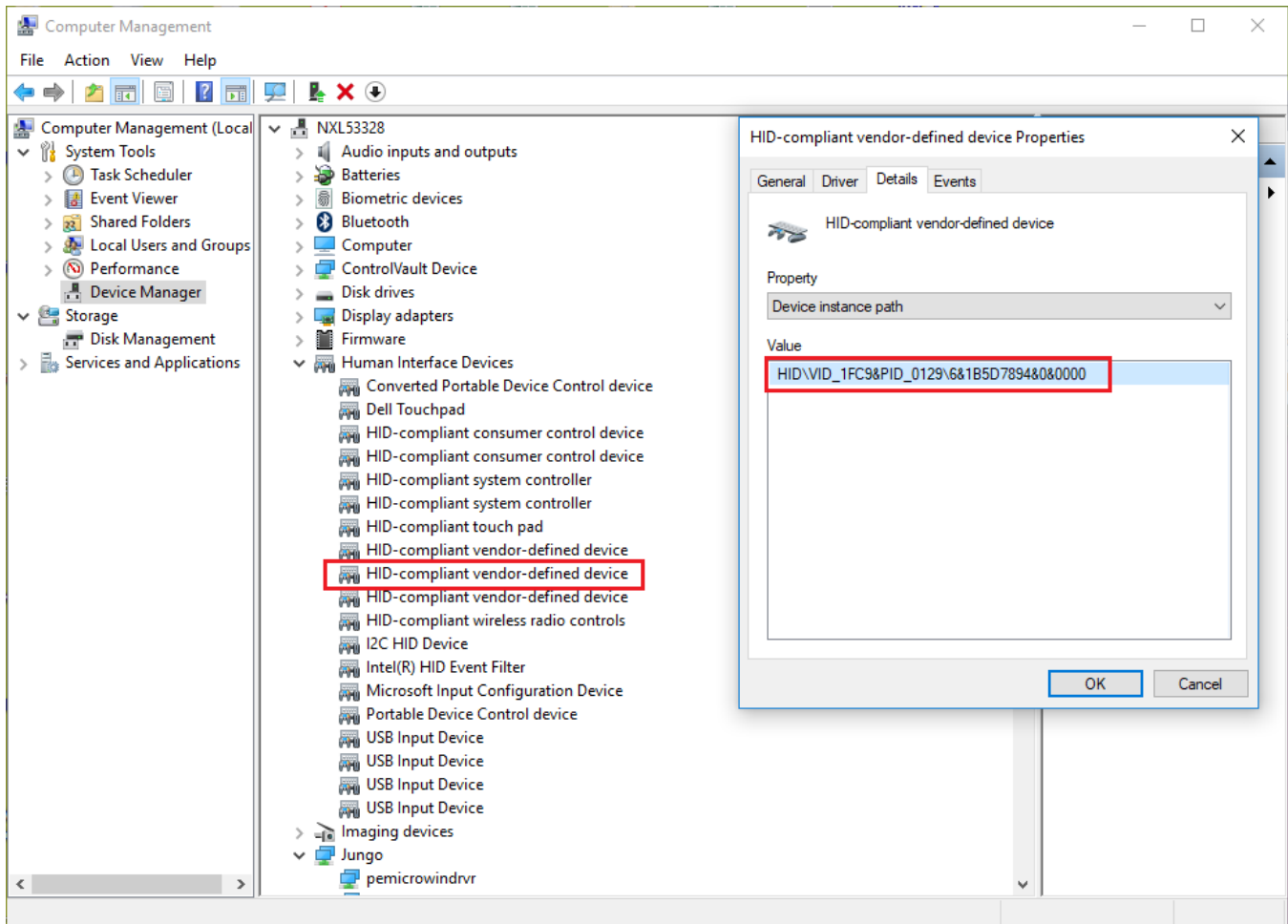


Figure 19. Device as shown in Device Manager

7.2 Build an example application

The following steps guide you through building 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 path.

7.2.1 Build an example application using IAR

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>/<core_instance>/iar`

Using the i.MX 8QuadMax MEK board as an example, the `hello_world` workspace is located in `<install_dir>/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/iar/hello_world.eww`

2. Select the desired build target from the drop-down. For this example, select the “`hello_world – flash_debug`” target. “`flash_debug`” and “`flash_release`” targets are using flash linker file and can be run in flash.

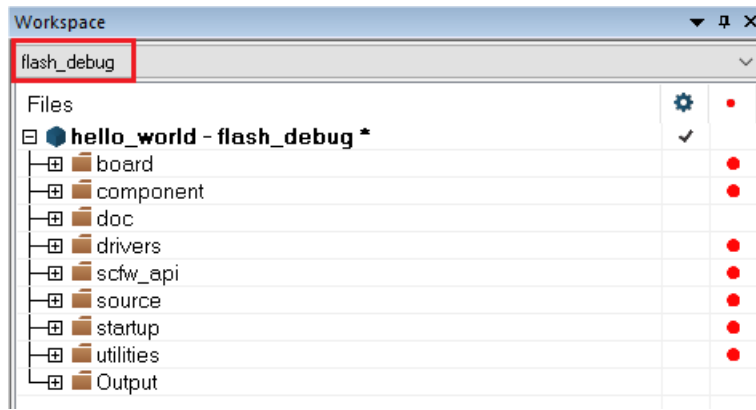


Figure 20. Demo build target selection

3. To build the demo application, click the "Make" button, highlighted in red below.

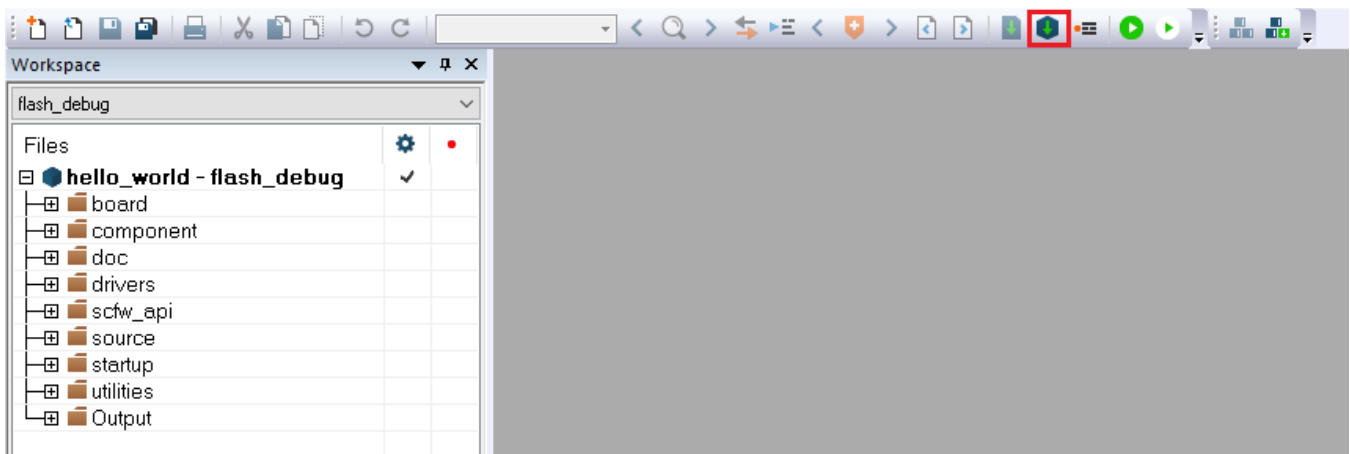


Figure 21. Build the demo application

4. The build completes without errors. Add the generated "m4_image.bin" in the `<iar_project_rootdir>/flash_debug` folder.

7.2.2 Build an example application using Arm® GCC

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>/<core_instance>/armgcc`

For this example, the exact path is: `<install_dir>/boards/mekmimx8qm/demo_apps/hello_world/cm4_core0/armgcc`

2. If using Linux Host, run the `build_flash_debug.sh` script on the command line to perform the build.

If using Windows Host, run the `build_flash_debug.bat` script to perform the build.

3. The build completes without errors. And generated "m4_image.bin" in `flash_debug` folder.

7.2.3 Build an bootable image using imx-mkimage tool

There are 3 pre-defined targets to generate flash.bin, which contains the XIP M4 target in imx-mkimage: flash_cm40flexspi, flash_cm4sflexspi, and flash_flexspi_all. The first is used to generate a flash.bin that only contains the CM4_0 XIP image. The second is to generate a flash.bin that contains both CM4 cores XIP image, and the last generates a flash.bin with two CM4 cores XIP images and U-Boot. Copy the generated M4 image into mkimage tool under the *imx-mkimage/iMX8QM* folder and rename it to “m4_image.bin” for CM4_0 or “m4_1_image.bin” for CM4_1 according to the mkimage target you are using.

1. Use the make SOC=iMX8QM flash_flexspi to generate a flash.bin which contains flexspi U-Boot. Rename this to flash_uboot_flash.bin for future use.
2. Use make SOC=iMX8QM flash_cm4_xip or make SOC=iMX8QM flash_cm41_xip or make SOC=iMX8QM flash_linux_m4_xip to generate the desired flash.bin.

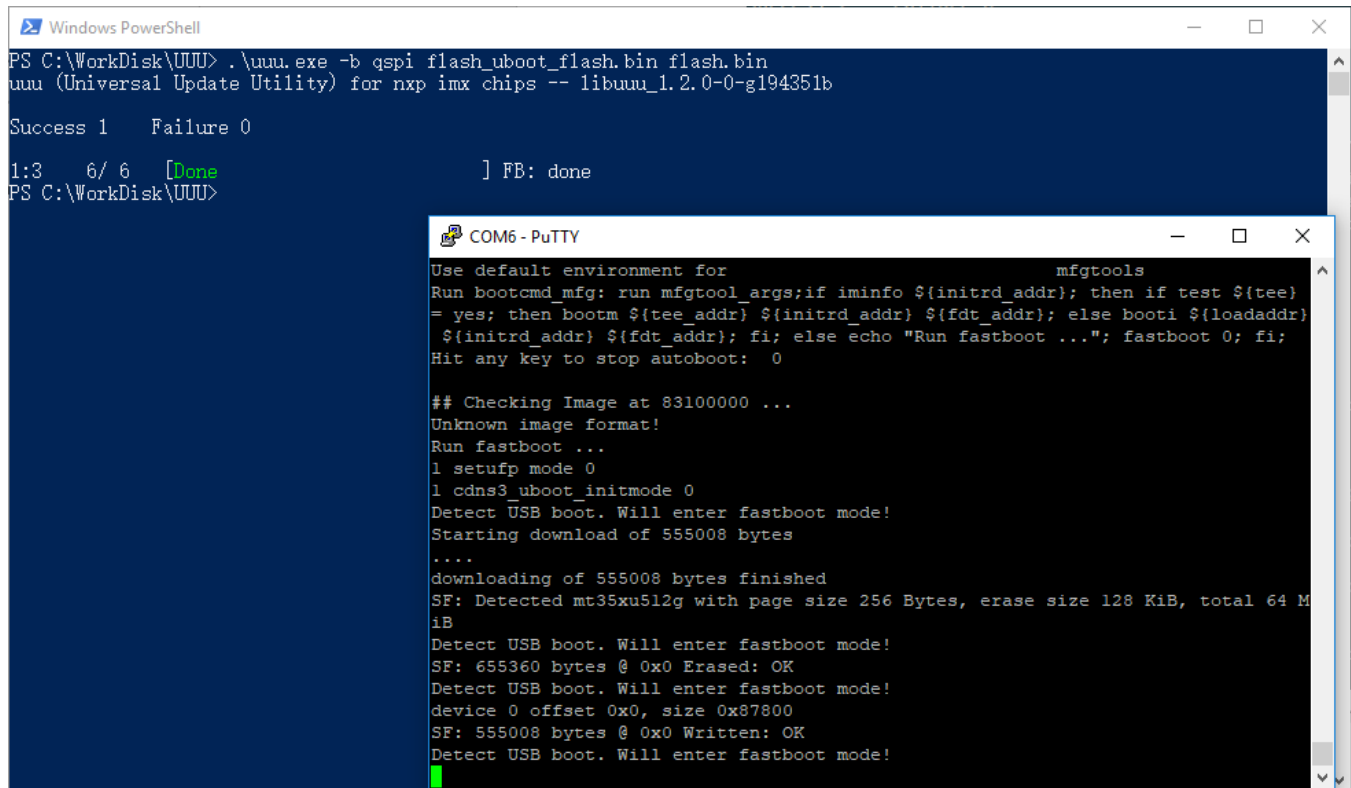
7.3 Run an example application

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

1. Connect the development platform to your PC via USB cable between the J18 USB DEBUG connector and the PC. It provides console output while using UUU.
2. Connect the J17 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 Appendix A). Configure the terminal with these settings:
 - a. 115200 baud rate
 - b. No parity
 - c. 8 data bits
 - d. 1 stop bit
4. In the command line, execute uuu with -b qspi parameter

```
uuu -b qspi flash_uboot_flash.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 shown on the following picture:



```

PS C:\WorkDisk\UUU> .\uuu.exe -b qspi flash_uboot_flash.bin flash.bin
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.0-0-g194351b

Success 1   Failure 0

1:3    6/ 6    [Done]    ] FB: done
PS C:\WorkDisk\UUU>
  
```

```

COM6 - PuTTY
Use default environment for                                mfgtools
Run bootcmd_mfg: run mfgtool_args;if iminfo ${initrd_addr}; then if test ${tee}
= yes; then bootm ${tee_addr} ${initrd_addr} ${fdt_addr}; else booti ${loadaddr}
${initrd_addr} ${fdt_addr}; fi; else echo "Run fastboot ..."; fastboot 0; fi;
Hit any key to stop autoboot:  0

## Checking Image at 83100000 ...
Unknown image format!
Run fastboot ...
l setupf mode 0
l cdns3 uboot initmode 0
Detect USB boot. Will enter fastboot mode!
Starting download of 555008 bytes
....
downloading of 555008 bytes finished
SF: Detected mt35xu5l2g with page size 256 Bytes, erase size 128 KiB, total 64 M
iB
Detect USB boot. Will enter fastboot mode!
SF: 655360 bytes @ 0x0 Erased: OK
Detect USB boot. Will enter fastboot mode!
device 0 offset 0x0, size 0x87800
SF: 555008 bytes @ 0x0 Written: OK
Detect USB boot. Will enter fastboot mode!
  
```

Figure 22. Command line and fast boot console output when executing UUU

5. Then, power off the board, change the boot mode to QSPI[b'011000], and power on the board again. The debug consoles display the M4 demo output respectively.

8 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. The USB Debug port (J18) on CPU board only provides COM ports for the Cortex-A and Cortex-M4 core0. To make it easy to evaluate the SDK, the RS232 port(J37) on the base board is used as the Cortex-M4 core1 debug console. A USB to RS232 converter is needed if there is no RS232 port on your host PC.

Linux:

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

```

$ dmesg | grep "ttyUSB"
[ 6761.714077] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
[ 6761.717904] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB1
[ 6768.710065] usb 2-2.2: pl2303 converter now attached to ttyUSB2
  
```

The FTDI USB Serial Device converter provides debug consoles for Cortex-A and CM4 core 0. It is recommended to open all the ports it has provided. The last port is provided by USB to the RS232 converter used for the CM4 core 1.

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 below:

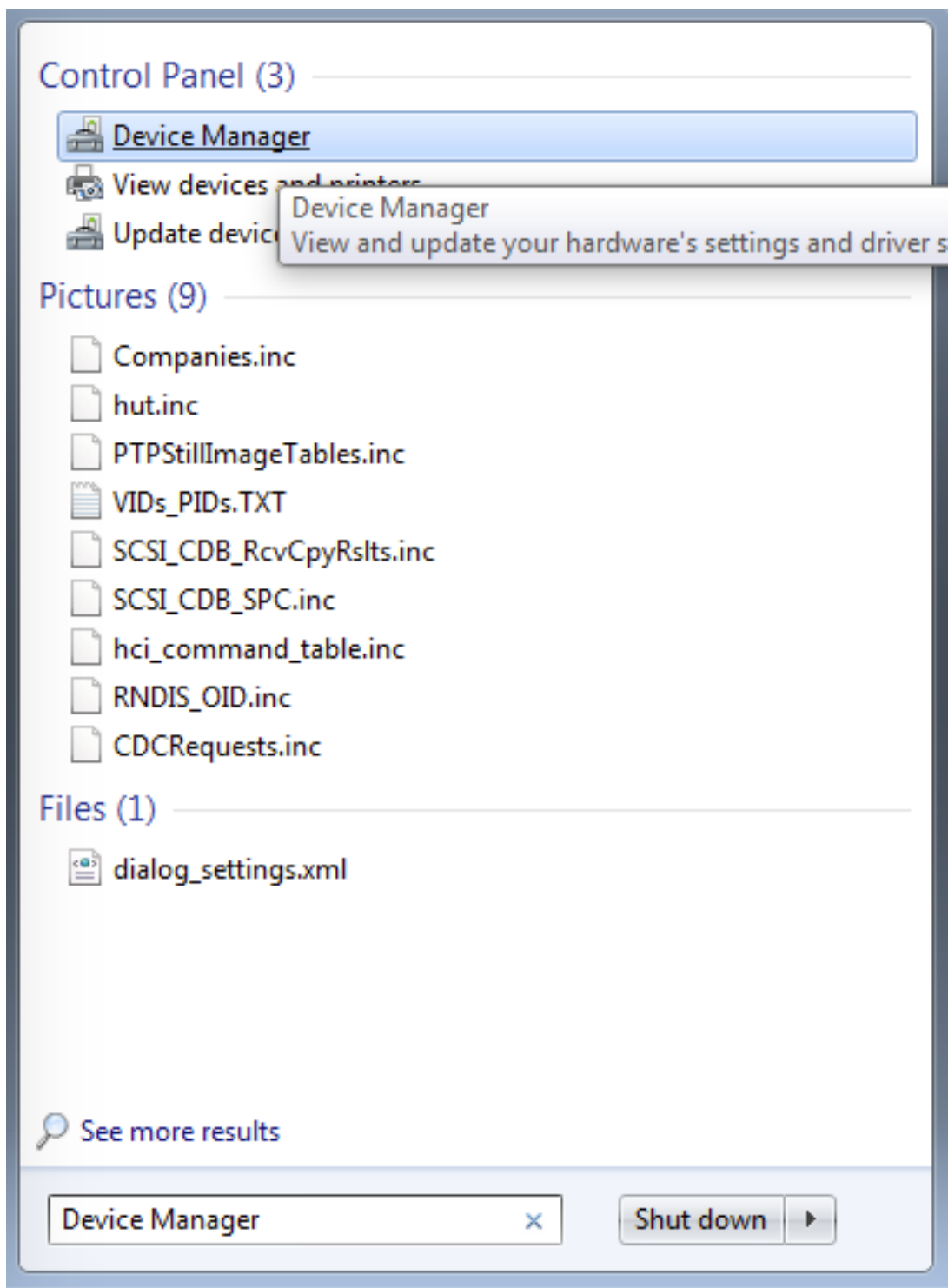


Figure 23. Device manager

2. In the Device Manager, expand the “Ports (COM & LPT)” section to view the available ports.
 - a. USB-UART interface

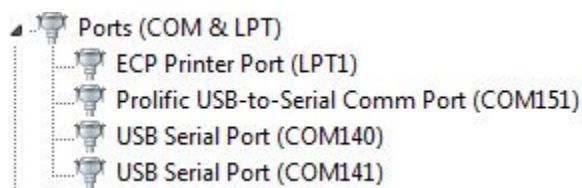


Figure 24. USB-UART interface

There will be three Ports. The ports provided by USB Debug Port (J18) is the Cortex-A and CM4 core0 debug console. The port provided by RS232 port on base board is used as the CM4 core1 debug console. Taking the above figure as example, the COM140 is for Cortex-A, COM141 is for the CM4 ccore0 and COM151 is for the CM4 core1.

9 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.

- Download the latest MinGW mingw-get-setup installer from sourceforge.net/projects/mingw/files/Installer/.
- Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location.

NOTE

The installation path cannot contain any spaces.

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

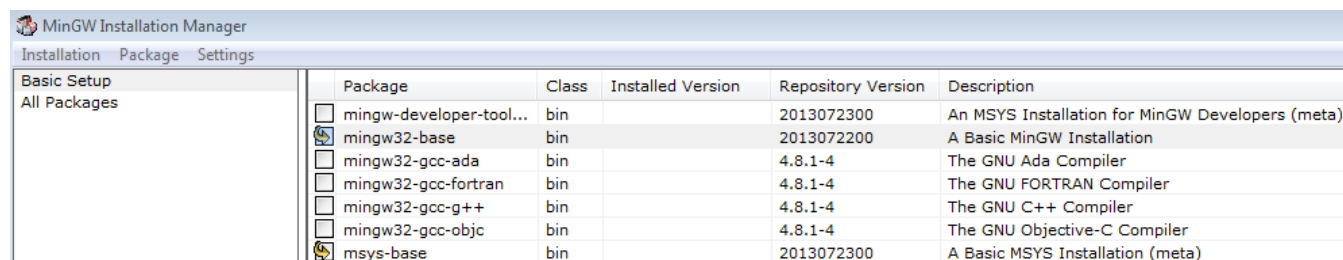


Figure 25. Setup MinGW and MSYS

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

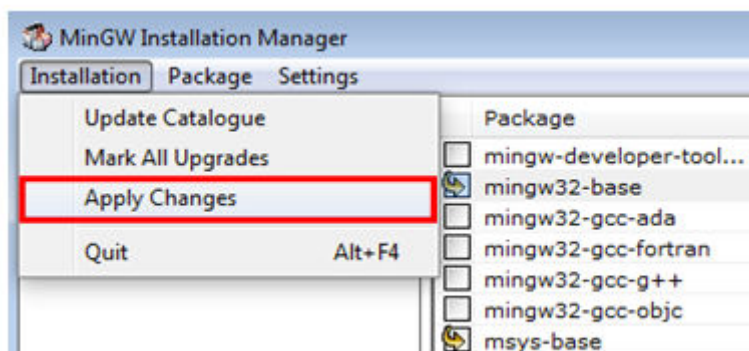


Figure 26. Complete MinGW and MSYS installation

- e. 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 27](#). 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.

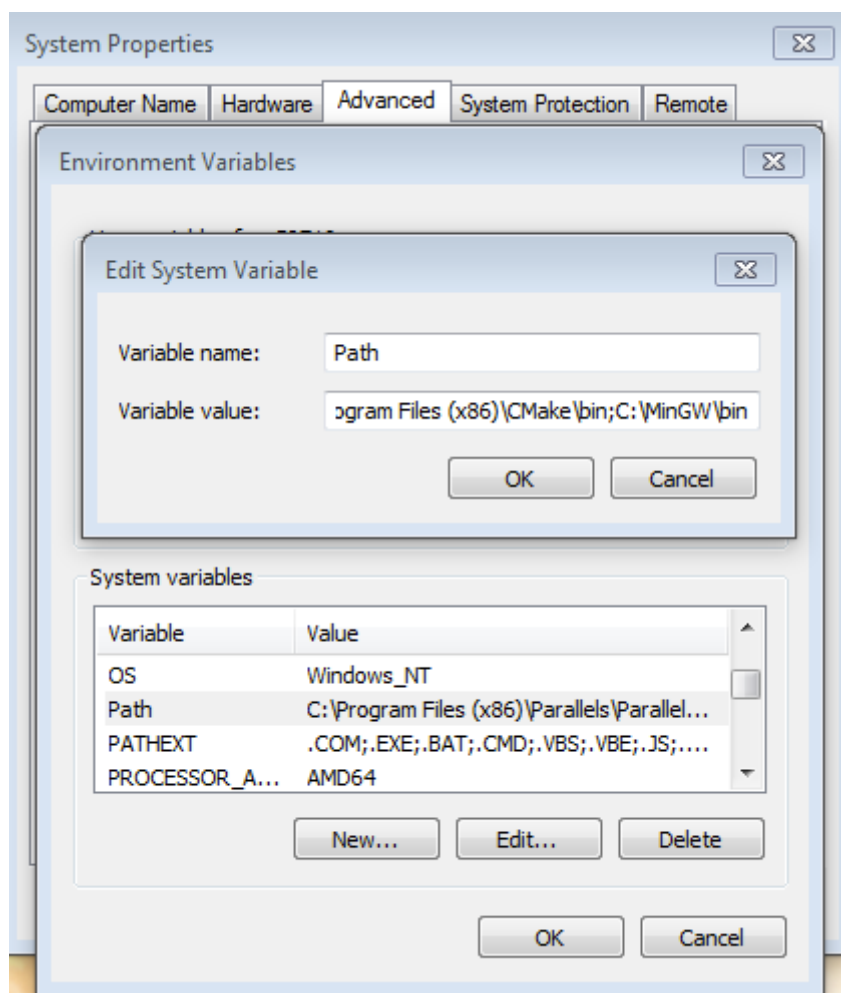


Figure 27. Add Path to systems environment

- Cmake
 - a. Download CMake 3.0.x from www.cmake.org/cmake/resources/software.html.
 - b. 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.

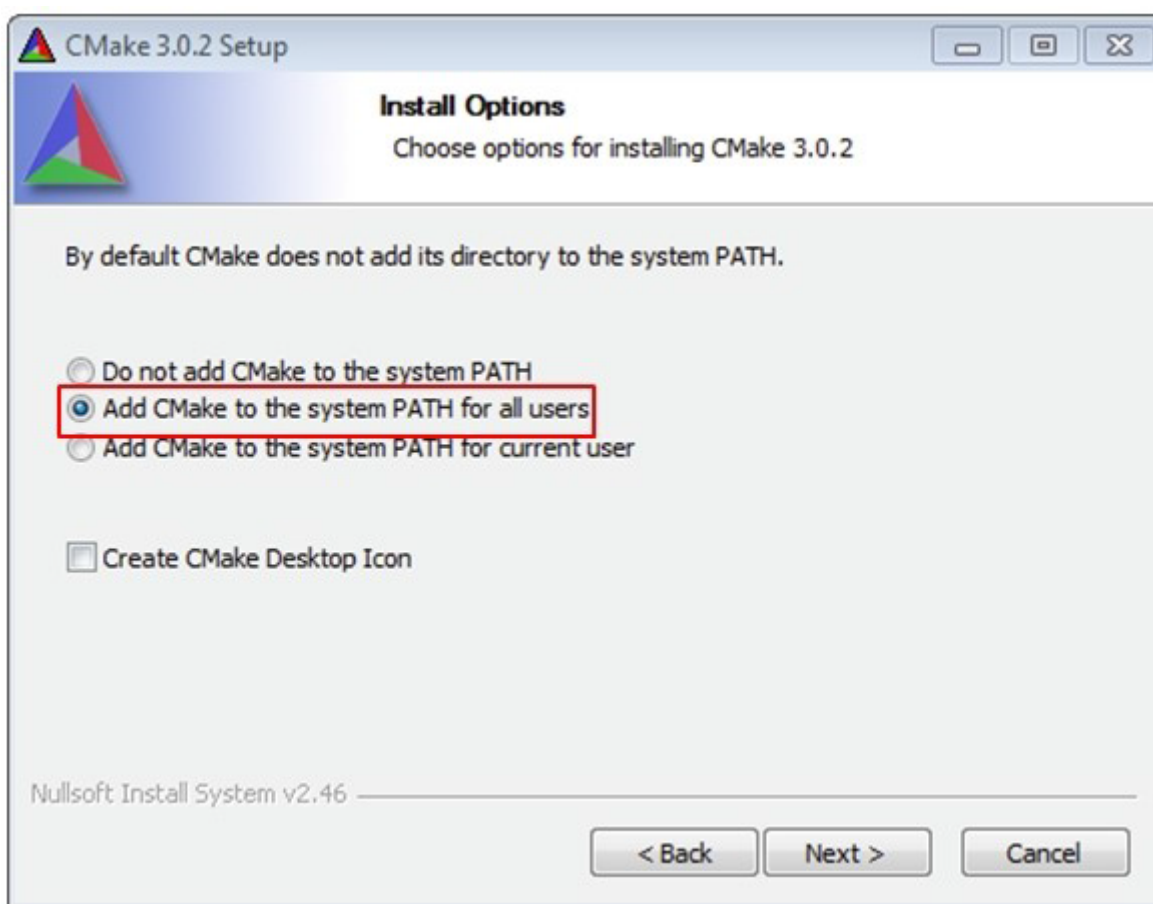


Figure 28. Install CMake

- c. Follow the remaining instructions of the installer.
- d. You may need to reboot your system for the PATH changes to take effect.

10 Revision history

This table summarizes revisions to this document.

Table 2. Revision history

Revision number	Date	Substantive changes
0	06/2018	Initial release RFP
1	08/2019	Updated for v2.7.0
2	06/2020	Updated for MCUXpresso SDK v2.8.0

How to Reach Us:**Home Page:**nxp.com**Web Support:**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.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, 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+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS 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, μ Vision, 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.

© 2019-2020 NXP B.V.

Document Number MCUXSDKIMX8QMGUSUG
Revision 2, 24 June 2020

