MCUXSDKIMX8ULPGSG

Getting Started with MCUXpresso SDK for EVK-MIMX8ULP and EVK9-MIMX8ULP

Rev. 1 — 20 June 2023 User manual

Document Information

Information	Content
Keywords	MCUXSDKIMX8ULPGSG, 8ULP, EVK-MIMX8ULP, EVK9-MIMX8ULP
Abstract	This document describes the steps to get started with MCUXpresso SDK for EVK-MIMX8ULP and EVK9-MIMX8ULP.

1 Overview

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides comprehensive software source code to be executed in the i.MX 8ULP M33 core. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. These drivers can be used standalone or collaboratively with the A35 cores running another Operating System (such as Linux OS 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 RTOS kernels, device stack, and various other middleware to support rapid development.

For supported toolchain versions, see the *MCUXpresso SDK Release Notes for EVK-MIMX8ULP* (document MCUXSDKIMX8ULPRN).

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

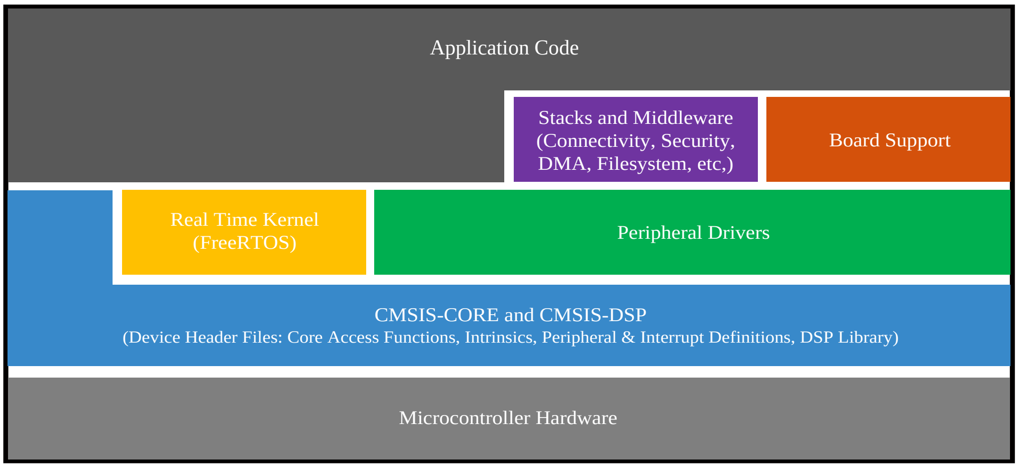

Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support folders

MCUXpresso SDK provides example applications for development and evaluation boards. Board support packages are found inside the top level <board_name> folder, and each supported board has its own folder (an MCUXpresso SDK package can support multiple boards). Within each <board_name> folder, there are various sub-folders for each example they contain. These include (but are not limited to):

- `demo_apps`: 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.
- `rtos_examples`: Basic FreeRTOS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers.
- `cmsis_driver_examples`: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- `multicore_examples`: Simple applications intended to concisely illustrate how to use middleware/multicore stack.

- mmcau_examples: Simple applications intended to concisely illustrate how to use middleware/mmcau 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.

The following figure shows the contents of the hello_world application folder.

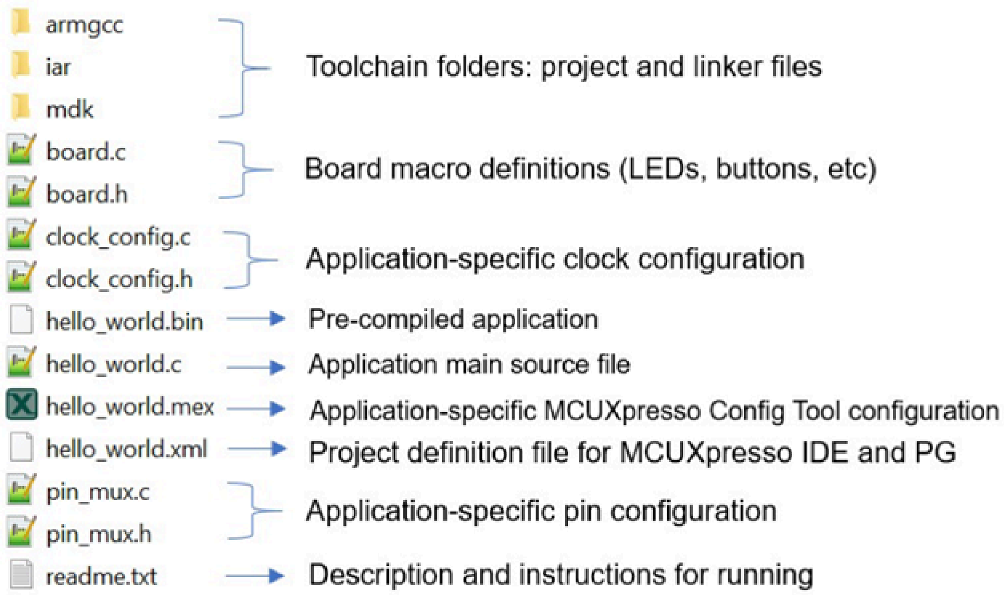

Figure 2. Application folder structure

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, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that the examples reference the same source files and if one of these files is modified, it could potentially impact the behavior of other examples.

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

- devices/<device_name>: The device's CMSIS header file, MCUXpresso SDK feature file, and a few other files
- devices/<device_name>/cmsis_drivers: All the CMSIS drivers for your specific MCU
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU
- devices/<device_name>/<tool_name>: Toolchain-specific startup code, including vector table definitions

- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<devices_name>/project` Project template used in CMSIS PACK new project creation

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

Note: *The `RPMsg-Lite` library is located in the `<install_dir>/middleware/multicore/rpmsg-lite` folder. For detailed information about the `RPMsg-Lite` library, see the RPMsg-Lite User's Guide, open the `index.html` located in the `<install_dir>/middleware/multicore/rpmsg_lite/doc` folder.*

Note: *The package does not include Xplorer IDE and DSP Fusion user guide. If you want to run examples related to DSP Fusion, contact the NXP representative (FAE/SE).*

3 Toolchain introduction

The MCUXpresso SDK release for i.MX 8ULP includes the build system to be used with some toolchains. This chapter lists and explains the supported toolchains.

3.1 Compiler/Debugger

The release supports building and debugging with the toolchains listed in Table 1.

You 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 debug probe and supports the device to attach, debug, and download.

Table 1. Toolchain information

Compiler/Debugger	Supported host OS	Debug probe	Tool website
Arm GCC/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 8ULP, the patch for IAR and SEGGER J-Link should be installed. The patch named iar_segger_support_patch_imx8ulp.zip can be used with the MCUXpresso SDK. See `readme.txt` in the patch for additional information about patch installation.*

4 Running a Demo Application Using Arm GCC

This section describes the steps to configure the command-line Arm GCC tools to build, run, and debug demo applications. This section also lists the necessary driver libraries provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the MIMX8ULP hardware platform is used as an example, though these steps can be applied to any board, demo, or example application in the MCUXpresso SDK.

Note: *Run an application using imx-mkimage. Generate and download flash.bin to emmc or flexspi nor flash when DBD_EN (Deny By Default) is fused.*

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 toolchain

Download and run the installer from the GNU Arm Embedded Toolchain Downloads page. The GNU Arm embedded toolchain contains the GCC compiler, libraries, and other tools required for bare-metal software development. The GCC toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes for EVK-MIMX8ULP* (document MCUXSDKIMX8ULPRN).

Note: *See Section 9 for setting up Linux host 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 toolchain 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/evkmimx8ulp/demo_apps/hello_world/armgcc`
2. Run the `build_debug.sh` script at the command-line to perform the build. The output is shown as below:

```
$ ./build_debug.sh
-- TOOLCHAIN_DIR:
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR:
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler:
-- Configuring done
-- Generating done
-- Build files have been written to:
Scanning dependencies of target hello_world.elf
 < -- skipping lines -- >
[100%] Linking C executable debug/hello_world.elf
[100%] Built target hello_world.elf
```

Note: *To run the application, see the Section 6.*

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 toolchain

Download and run the installer from the GNU Arm Embedded Toolchain Downloads page. The GNU Arm embedded toolchain contains the GCC compiler, libraries, and other tools required for bare-metal software development. The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes for EVK-MIMX8ULP* (document MCUXSDKIMX8ULPRN).

Note: *See Section 9 for setting up Windows host before compiling the application.*

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 toolchain installation path. For this example, the path is:

```
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update
```

Reference the installation folder of the GNU Arm GCC embedded tools for the exact pathname.

4.2.2 Build an example application

To build an example application, follow these steps.

1. Open the GCC Arm embedded toolchain command window. To launch the window on the Windows operating system, select **Start** -> **Programs** -> **GNU Tools ARM Embedded <version>** -> **GCC Command Prompt**.

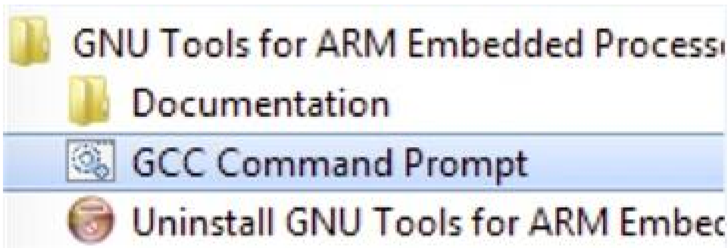

Figure 3. Launch GCC command prompt

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/evkmimx8ulp/demo_apps/hello_world/armgcc
```

3. Type `build_debug.bat` at the command-line or double-click the `build_debug.bat` file in Windows Explorer to perform the build. The output is as shown in Figure 4.

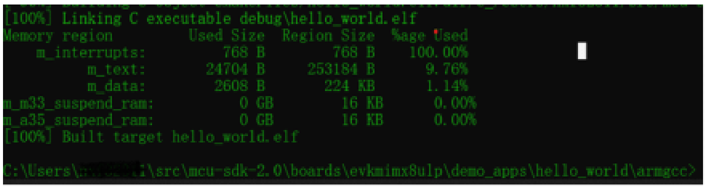

Figure 4. hello_world demo build successful

Note: *To run the application, see the Section 6.*

5 Running 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 MIMX8ULP hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

Note:

- *Newer versions of the IAR are compatible with older versions of the project format. However, using an older version of the IAR to load the SDK project that uses the newer format generates an error. To use the SDK, it is recommended to upgrade the IAR version to 9.30.1.*
- *Run an application using imx-mkimage. Generate and download flash.bin to emmc or flexspi nor flash when DBD_EN (Deny By Default) is fused.*

5.1 Build an example application

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
   ```

   For using MIMX8ULP-EVK hardware platform as an example, the `hello_world` workspace is located at:

   ```
   <install_dir>/boards/evkmimx8ulp/demo_apps/hello_world/iar/hello_world.eww
   ```

   Other example applications may have additional folders in the respective paths.
2. Select the desired build target from the drop-down menu.
   For this example, select **hello_world** – **Debug**.

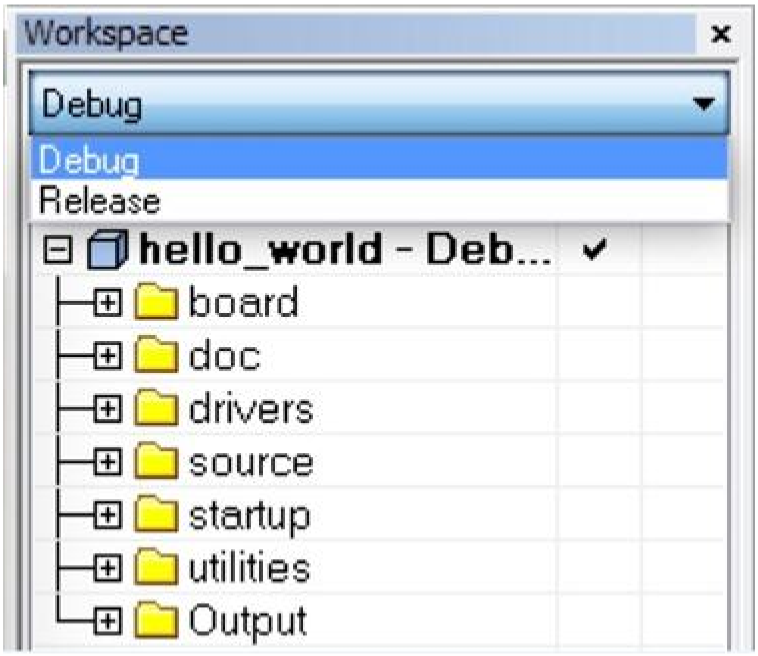

Figure 5. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 6.

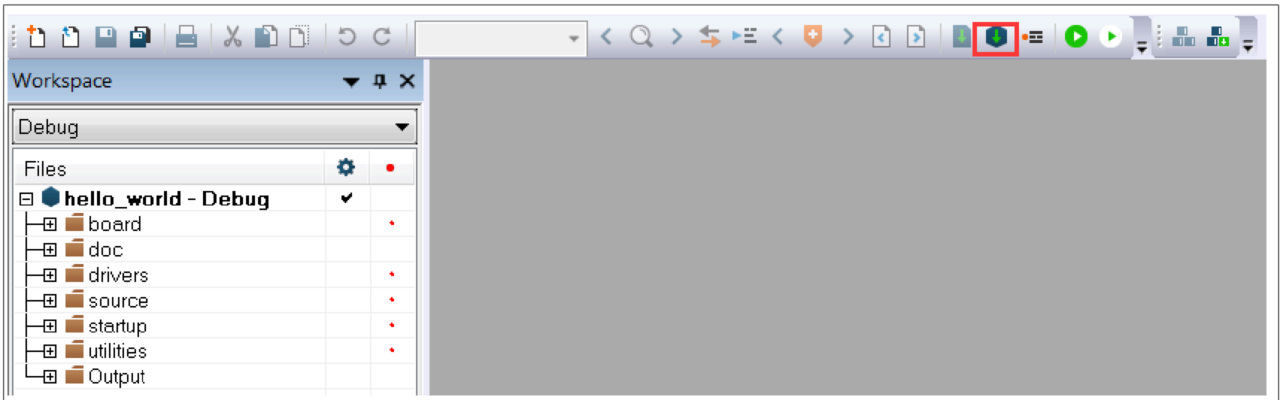

Figure 6. Build the demo application

4. The build completes without errors.

Note: *To run the application, see the Section 6.*

6 Run an application using imx-mkimage

This section describes the steps to write a bootable SDK image to the eMMC/FlexSPI NOR flash for the i.MX processor.

Note: *Attach core to debug code with J-Link probe.*

The following steps describe how to write container image (flash.bin):

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 and LPT)**. Assume that the ports are COM9 and COM10. One port is for the debug message from the Cortex-A35 and the other is for the Cortex-M33. 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.

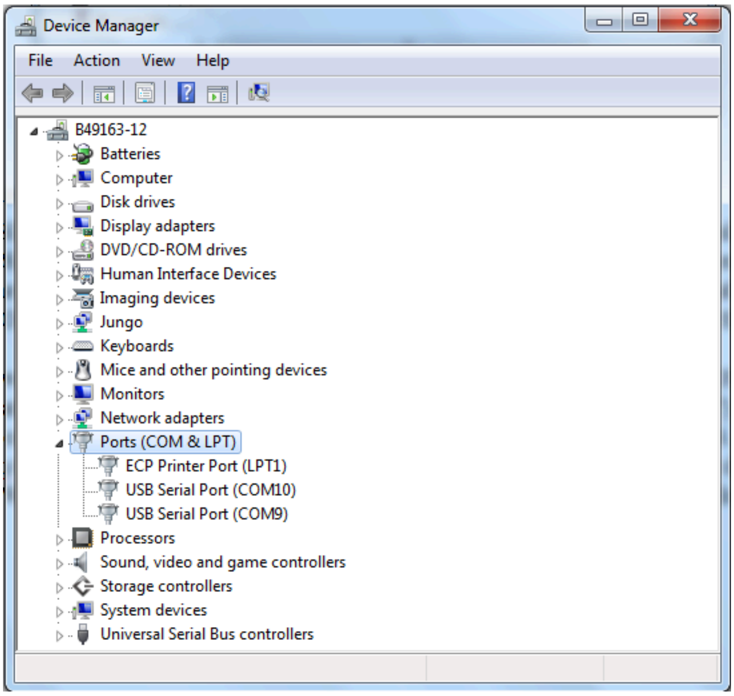

Figure 7. Determining the COM port of target board

3. Generate m33 firmware:
   - **For RAM target**:

     ```
     $ ./build_debug.sh
     ```

     or

     ```
     $ ./build_release.sh
     ```

   - **For FLASH target(XIP)**:

     ```
     $ ./build_flash_debug.sh
     ```

     or

     ```
     $ ./build_flash_release.sh
     ```

4. Get `imx-mkimage`, `s400 firmware(mx8ulpa2-ahab-container.img)`, `OPTEE(tee.bin)`, `upower firmware(upower.bin)`, `uboot-spl(u-boot-spl.bin)`, `uboot(u-boot.bin)`, and `TF-A(bl31.bin)` from the Linux release package.
   a. Clone the `imx-mkimage` from NXP public git.

      ```
      $ git clone https://github.com/nxp-imx/imx-mkimage
      ```

   b. 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.

      ```
      $ cd imx-mkimage
      $ git checkout [branch name]
      ```

c. Get `s400 firmware(mx8ulpa2-ahab-container.img)`.

```
$ cp mx8ulpa2-ahab-container.img iMX8ULP/
```

d. Get `upower firmware(upower.bin)`.

```
$ cp upower.bin iMX8ULP/
```

e. Get `u-boot-spl.bin and u-boot.bin`.

- For EVK-MIMX8ULP:

```
$ cp u-boot-spl.bin-imx8ulpevk-sd iMX8ULP/u-boot-spl.bin
$ cp u-boot-imx8ulpevk.bin-sd iMX8ULP/u-boot.bin
```

- For EVK9-MIMX8ULP:

```
$ cp u-boot-spl.bin-imx8ulp-9x9-lpddr4-evk-sd iMX8ULP/u-boot-spl.bin
$ cp u-boot-imx8ulp-9x9-lpddr4-evk.bin-sd iMX8ULP/u-boot.bin
```

f. Get `bl31.bin`.

```
$ cp bl31-imx8ulp.bin-optee iMX8ULP/bl31.bin
```

5. Generate container image table with `imx-mkimage`:

<table>
<tr><th>boot type</th><th>A35</th><th>M33</th><th>SW5[8:1]</th></tr>
<tr><td>Single Boot</td><td colspan="2">make SOC=iMX8ULP flash_singleboot
For RAM target:
make SOC=iMX8ULP flash_singleboot_m33
Note: Does not support pack Flash target into flash.bin when boot type is single boot type.</td><td>1000_xx00 Single Boot-e MMC</td></tr>
<tr><td></td><td colspan="2">make SOC=iMX8ULP flash_singleboot_flexspi
For RAM target:
make SOC=iMX8ULP flash_singleboot_m33_flexspi
Note: Does not support pack Flash target into flash.bin when boot type is single boot type.</td><td>1010_xx00 Single Boot-Nor</td></tr>
<tr><td>Dual Boot</td><td>make SOC=iMX8ULP flash_dualboot</td><td rowspan="4">For RAM target:
make SOC=iMX8ULP flash_dualboot_m33
For Flash target:
make SOC=iMX8ULP flash_dualboot_m33_xip</td><td>1000_0010 A35-eMMC/M33-Nor</td></tr>
<tr><td></td><td>make SOC=iMX8ULP flash_dualboot_flexspi</td><td>1010_0010 A35-Nor/M33-Nor</td></tr>
<tr><td>Low Power Boot</td><td>make SOC=iMX8ULP flash_dualboot</td><td>1000_00x1 A35-eMMC/M33-Nor</td></tr>
<tr><td></td><td>make SOC=iMX8ULP flash_dualboot_flexspi</td><td>1010_00x1 A35-Nor/M33-Nor</td></tr>
</table>

Note:

- *For details, see `imx-mkimage/iMX8ULP/README`.*
- *Does not support pack Flash target firmware to flash.bin when boot type is single boot type.*
- *RAM target: debug/release.*
- *Flash target: `flash_debug/flash_release`.*
- *Need generate two flash.bin and download to emmc/flexspi2 nor flash of a35 and flexspi0 nor flash of m33, one for A35, another one for M33 when boot type is dual boot type or low power boot type.*

6. Build the application (for example, `hello_world`), get binary image `sdk20-app.bin`, copy to `imx-mkimage` project folder `iMX8ULP/` and rename to `m33_image.bin`.

```
cp sdk20-app.bin <imx-mkimage path>/iMX8ULP/m33_image.bin
```

7. Under `imx-mkimage` project folder, execute the following command to generate m33 container image.
   a. When boot type is dual boot/low power boot type:
      For RAM (TCM) target:

      ```
      make SOC=iMX8ULP flash_dualboot_m33 (write flash.bin to flexspi0 nor flash of m33;
      ```

      For Flash target:

      ```
      make SOC=iMX8ULP flash_dualboot_m33_xip (write flash.bin to flexspi0 nor flash of m33);
      ```

   b. When boot type is single boot type:

      ```
      for RAM (TCM) target and sw5[8:1] = 1000_xx00 Single Boot-eMMC:
      make SOC=iMX8ULP flash_singleboot_m33 (write flash.bin to emmc);
      ```

      ```
      for RAM (TCM) target and sw5[8:1] = 1010_xx00 Single Boot-Nor:
      make SOC=iMX8ULP flash_singleboot_m33_flexspi (write flash.bin to flexspi2 nor flash of
      a35);
      ```

8. Copy the `flash.bin` image to your tftpboot server.
9. Write `flash.bin to flexspi0 nor flash`. There are two ways:
   a. Write `flash.bin to flexspi0 nor flash with uboot`.
      i. Switch to single boot type (`sw[8:1]=1000 0000`) and boot the board, assuming your board can boot to U-Boot.
      ii. At the U-Boot console, execute following commands to download image (from network) and flash to FlexSPI0 NOR flash.

      ```
      setenv serverip <tftpboot server ip>
      dhcp
      tftpboot 0xa0000000 flash.bin
      setenv erase_unit 1000
      setexpr erase_size ${filesize} + ${erase_unit}
      setexpr erase_size ${erase_size} / ${erase_unit}
      setexpr erase_size ${erase_size} * ${erase_unit}
      sf probe 0:0
      sf erase ${erase_size}
      sf write  0xa0000000 0 ${filesize}
      ```

   b. Write `flash.bin to flexspi0 nor flash with JLink`:

      ```
      J-Link>connect
      Device>
      TIF>s (Choose target interface as SWD, unless failed to do anything)
      Speed>
      J-Link>r
      J-Link>h
      J-Link>loadbin flash.bin 0x4000000
      ```

10. Write `flash.bin to emmc with uuu` (only for the RAM target):
    a. Start `uuu`.

       ```
       uuu -b emmc workable-flash.bin flash.bin (workable-flash.bin: uboot and m33 image are
        workable)
       ```

    b. Enter serial download mode.
       i. Change SW5[8:1] to 01xx_xxxx Serial Downloader.

ii. Enter serial download mode with uboot.

```
=> fastboot 0
```

11. Open another terminal application on the PC, such as PuTTY and connect to the debug COM port (to determine the COM port number, see Section 8). Configure the terminal with these settings:
    - 115200
    - No parity
    - 8 data bits
    - 1 stop bit
12. Power off and switch to low-power boot mode (`sw5[8:1]=1000 0001`), then repower the board.
13. The `hello_world` application is now executed and a banner is displayed at the terminal. If this is not true, check your terminal settings and connections.

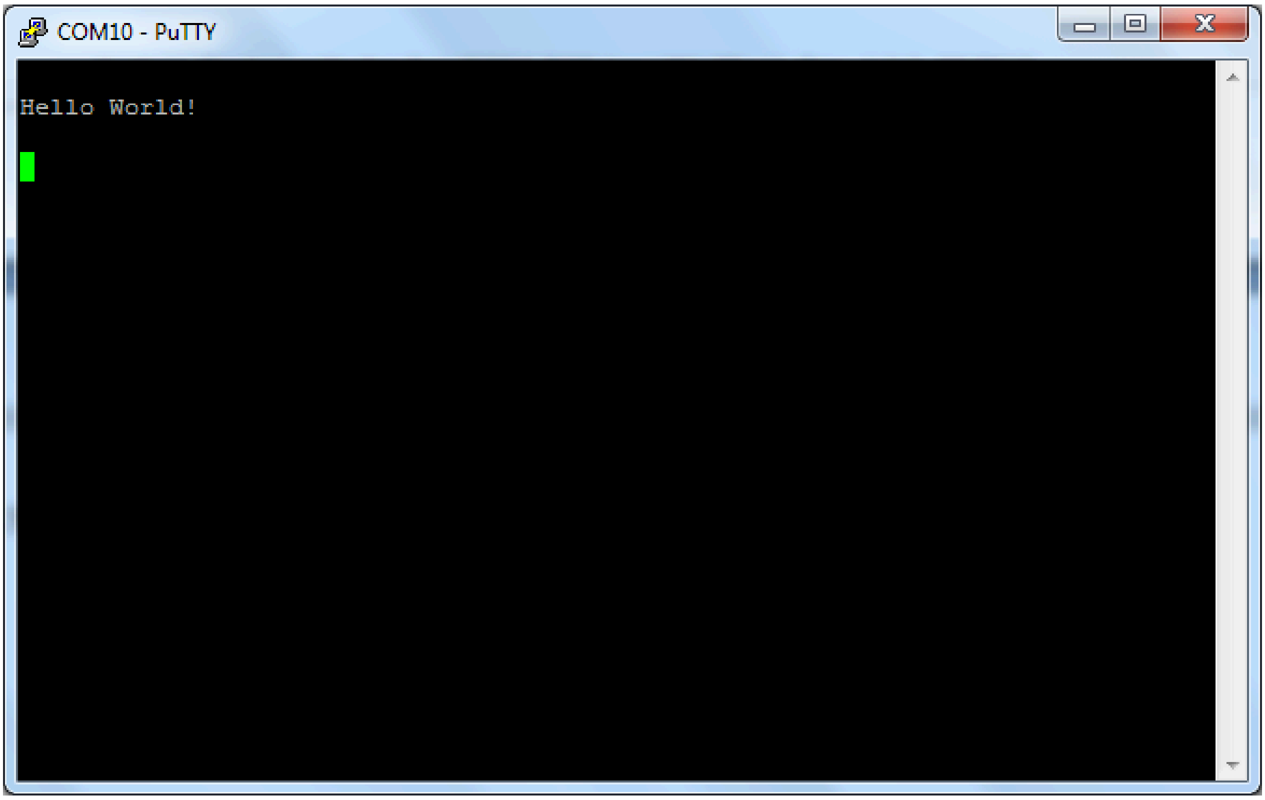

Figure 8. Hello world demo running on Cortex-M33 core

7 Memory attribution map after doing handshake

The memory attribution map settings after the handshake procedure is successful between Cortex-M33 and Cortex-A35.

Table 2. Memory attribution map for domain 0 in M33 domain

Name	Memory block checker/ Memory region checker (MBC/MRC)	Resulting access level	
FLEXSPI1 (alias)	Non Secure	Non Secure	0x5FFF_FFFF
			0x5000_0000

Table 2. Memory attribution map for domain 0 in M33 domain*...continued*

Name	Memory block checker/ Memory region checker (MBC/MRC)	Resulting access level	
FLEXSPI1	Non Secure	Non Secure	0x4FFF_FFFF
			0x4000_0000
PBridge1 FlexCAN0 (alias)	Non Secure	Non Secure	0x380A_BFFF
			0x380A_8000
PBridge1 SAI0 (alias)	Non Secure	Non Secure	0x3809_C0FF
			0x3809_C000
PBridge1 LPUART1 (alias)	Non Secure	Non Secure	0x3809_B02F
			0x3809_B000
PBridge1 LPI2C0 (alias)	Non Secure	Non Secure	0x3809_8173
			0x3809_8000
PBridge1 FlexSPI1 (alias)	Non Secure	Non Secure	0x3809_22FF
			0x3809_2000
PBridge0 LPSPI1 (alias)	Non Secure	Non Secure	0x3803_F7FF
			0x3803_F000
PBridge0 FlexIO0 (alias)	Non Secure	Non Secure	0x3803_C91F
			0x3803_C000
PBridge0 FlexSPI0 (alias)	Non Secure	Non Secure	0x3803_92FF
			0x3803_9000
PBridge1 FlexCAN0	Non Secure	Non Secure	0x280A_BFFF
			0x280A_8000
PBridge1 SAI0	Non Secure	Non Secure	0x2809_C0FF
			0x2809_C000
PBridge1 LPUART1	Non Secure	Non Secure	0x2809_B02F
			0x2809_B000
PBridge1 LPI2C0	Non Secure	Non Secure	0x2809_8173
			0x2809_8000
PBridge1 FlexSPI1	Non Secure	Non Secure	0x2809_22FF
			0x2809_2000
PBridge0 LPSPI1	Non Secure	Non Secure	0x2803_F7FF
			0x2803_F000
PBridge0 FlexIO0	Non Secure	Non Secure	0x2803_C91F
			0x2803_C000
PBridge0 FlexSPI0	Non Secure	Non Secure	0x2803_92FF
			0x2803_9000
SSRAM P6 (alias)	Non Secure	Non Secure	0x3007_FFFF

Table 2. Memory attribution map for domain 0 in M33 domain*...continued*

Name	Memory block checker/ Memory region checker (MBC/MRC)	Resulting access level	
			0x3006_0000
SSRAM P5 (alias)	Non Secure	Non Secure	0x3005_FFFF
			0x3004_0000
SSRAM P4 (alias)	Non Secure	Non Secure	0x3003_FFFF
			0x3003_0000
SSRAM P3 (alias)	Non Secure	Non Secure	0x3002_FFFF
			0x3002_0000
SSRAM P2 (alias)	Non Secure	Non Secure	0x3001_FFFF
			0x3001_0000
SSRAM P1 (alias)	Non Secure	Non Secure	0x3000_FFFF
			0x3000_8000
SSRAM P0 (alias)	Non Secure	Non Secure	0x3000_7FFF
			0x3000_0000
SSRAM P6	Non Secure	Non Secure	0x2007_FFFF
			0x2006_0000
SSRAM P5	Non Secure	Non Secure	0x2005_FFFF
			0x2004_0000
SSRAM P4	Non Secure	Non Secure	0x2003_FFFF
			0x2003_0000
SSRAM P3	Non Secure	Non Secure	0x2002_FFFF
			0x2002_0000
SSRAM P2	Non Secure	Non Secure	0x2001_FFFF
			0x2001_0000
SSRAM P1	Non Secure	Non Secure	0x2000_FFFF
			0x2000_8000
SSRAM P0	Non Secure	Non Secure	0x2000_7FFF
			0x2000_0000
SSRAM P7 (alias)	Non Secure	Non Secure	0x1FFF_FFFF
			0x1FFC_0000
FlexSPI0 (alias)	Non Secure	Non Secure	0x1BFF_FFFF
			0x1400_0000
SSRAM P7	Non Secure	Non Secure	0x0FFF_FFFF
			0x0FFC_0000
FlexSPI0	Non Secure	Non Secure	0x0BFF_FFFF
			0x0400_0000

Note:

1. *Assign Domain 1 for DMA1, USB0, USB1, ENET, USDHC0, USDHC1, USDHC2, and CAAM Master.*
2. *The bus attribute for DMA1, USB0, USB1, ENET, USDHC0, USDHC1, and USDHC2 is Non Secure.*
3. *The bus attribute for CAAM Master is Secure.*
4. *Security level of MBC/MRC settings of other memory space that are not be shown in the table for Domain 1 are Secure, so master cannot access resources that are controlled by MBC/MRC in other memory spaces when master is in Non Secure state.*

Table 3. Memory attribution map for domain 1 in M33 domain

Name	MBC/MRC	Resulting access level	
PBridge1	Non Secure	Non Secure	0x280F_FFFF
			0x2808_0000
SSRAM P2	Secure	No Access	0x2001FFFF
			0x20018000
	Non Secure	Non Secure	0x20017FFF
			0x20010000

Note:

1. *Assign Domain 1 for DMA1, USB0, USB1, ENET, UDSHC0, USDHC1, UDSHC2, and CAAM Master.*
2. *Security level of MBC/MRC settings of other memory space that are not shown in the table for Domain 1 are Secure, so the master cannot access resources that are controlled by MBC/MRC in other memory spaces.*

Table 4. Memory attribution map for domain 6 in M33 domain

Name	SAU	IDAU	MBC/ MRC	Resulting access level	
GPIOC_REGS (alias)	Secure	Secure	Secure	Secure	0x3882_FFFF
					0x3882_0000
GPIOB_REGS (alias)	Secure	Secure	Secure	Secure	0x3881_FFFF
					0x3881_0000
GPIOA_REGS (alias)	Secure	Secure	Secure	Secure	0x3880_FFFF
					0x3880_0000
MICFIL (alias)	Secure	Secure	Secure	Secure	0x3811_10AB
					0x3811_1000
SAI3 (alias)	Secure	Secure	Secure	Secure	0x3811_00FF
					0x3811_0000
SAI2 (alias)	Secure	Secure	Secure	Secure	0x3810_F0FF
					0x3810_F000
LPSPI3 (alias)	Secure	Secure	Secure	Secure	0x3810_E7FF
					0x3810_E000
LPSPI2 (alias)	Secure	Secure	Secure	Secure	0x3810_D7FF
					0x3810_D000
LPUART3 (alias)	Secure	Secure	Secure	Secure	0x3810_C02F

Table 4. Memory attribution map for domain 6 in M33 domain*...continued*

Name	SAU	IDAU	MBC/ MRC	Resulting access level	
					0x3810_C000
LPUART2 (alias)	Secure	Secure	Secure	Secure	0x3810_B02F
					0x3810_B000
I3C1 (alias)	Secure	Secure	Secure	Secure	0x3810_AFFF
					0x3810_A000
LPI2C3 (alias)	Secure	Secure	Secure	Secure	0x3810_9173
					0x3810_9000
LPI2C2 (alias)	Secure	Secure	Secure	Secure	0x3810_8173
					0x3810_8000
MRT (alias)	Secure	Secure	Secure	Secure	0x3810_70FF
					0x3810_7000
TPM3 (alias)	Secure	Secure	Secure	Secure	0x3810_6087
					0x3810_6000
TPM2 (alias)	Secure	Secure	Secure	Secure	0x3810_5087
					0x3810_5000
PCC2 (alias)	Secure	Secure	Secure	Secure	0x3810_2047
					0x3810_2000
WDOG2 (alias)	Secure	Secure	Secure	Secure	0x3810_100F
					0x3810_1000
MU1_B (alias)	Secure	Secure	Secure	Secure	0x3810_028F
					0x3810_0000
FlexCAN0 (alias)	Secure	Secure	Secure	Secure	0x380A_BFFF
					0x380A_8000
ADC1 (alias)	Secure	Secure	Secure	Secure	0x380A_2303
					0x380A_2000
IOMUXC0 (alias)	Secure	Secure	Secure	Secure	0x380A_1AEB
					0x380A_1000
SAI1 (alias)	Secure	Secure	Secure	Secure	0x3809_D0FF
					0x3809_D000

Note:

1. *SAU is disabled.*
2. *Cortex-M33 can access all of the secure resources. All of the resources (the security level of these resources that are controlled by MBC/MRC) are secure when Cortex-M33 is in secure state.*
3. *Assign domain 6 for Cortex-M33.*

Table 5. Memory attribution map for domain 7 in M33

Name	MBC/MRC	Resulting access level	
PBridge1 IOMUXC0	Non Secure	Non Secure	0x280A_1AEB
			0x280A_1000
PBridge1 LPI2C0	Non Secure	Non Secure	0x2809_8173
			0x2809_8000
PBridge1 TPM0	Non Secure	Non Secure	0x2809_5087
			0x2809_5000
PBridge1 PCC1	Non Secure	Non Secure	0x2809_10BF
			0x2809_1000
PBridge0 FlexSPI0	Non Secure	Non Secure	0x2803_92FF
			0x2803_9000
PBridge0 SEMA42_0	Non Secure	Non Secure	0x2803_7043
			0x2803_7000
PBridge0 CGC0	Non Secure	Non Secure	0x2802_FFFF
			0x2802_F000
PBridge0 SIM0-S	Non Secure	Non Secure	0x2802_B3FF
			0x2802_B000
S400 MU-AP of EdgeLock secure enclave	Non Secure	Non Secure	0x2702_028C
			0x2702_0000
FSB of EdgeLock secure enclave	Non Secure	Non Secure	0x2701_0BFC
			0x2701_0000
SSRAM P7	Non Secure	Non Secure	0x1FFF_FFFF
			0x1FFF_8000
	Secure	Secure	0x1FFF_7FFF
			0x1FFC_0000
FlexSPI0	Non Secure	Non Secure	0x0BFF_FFFF
			0x0400_0000

Note:

1. *Security level of MBC/MRC settings of Other memory space that are not shown in the table for Domain 7 are Secure. The master can access resources that are controlled by MBC/MRC in other memory spaces when the master is in secure state.*
2. *Assign domain 7 for Cortex-A35.*

8 How to determine COM port

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

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

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

   There are two ports, one is Cortex-A core debug console and the other is for Cortex M33.
2. **Windows**: To determine the COM port, open **Device Manager**. Click the **Start** menu and type **Device Manager** in the search bar.

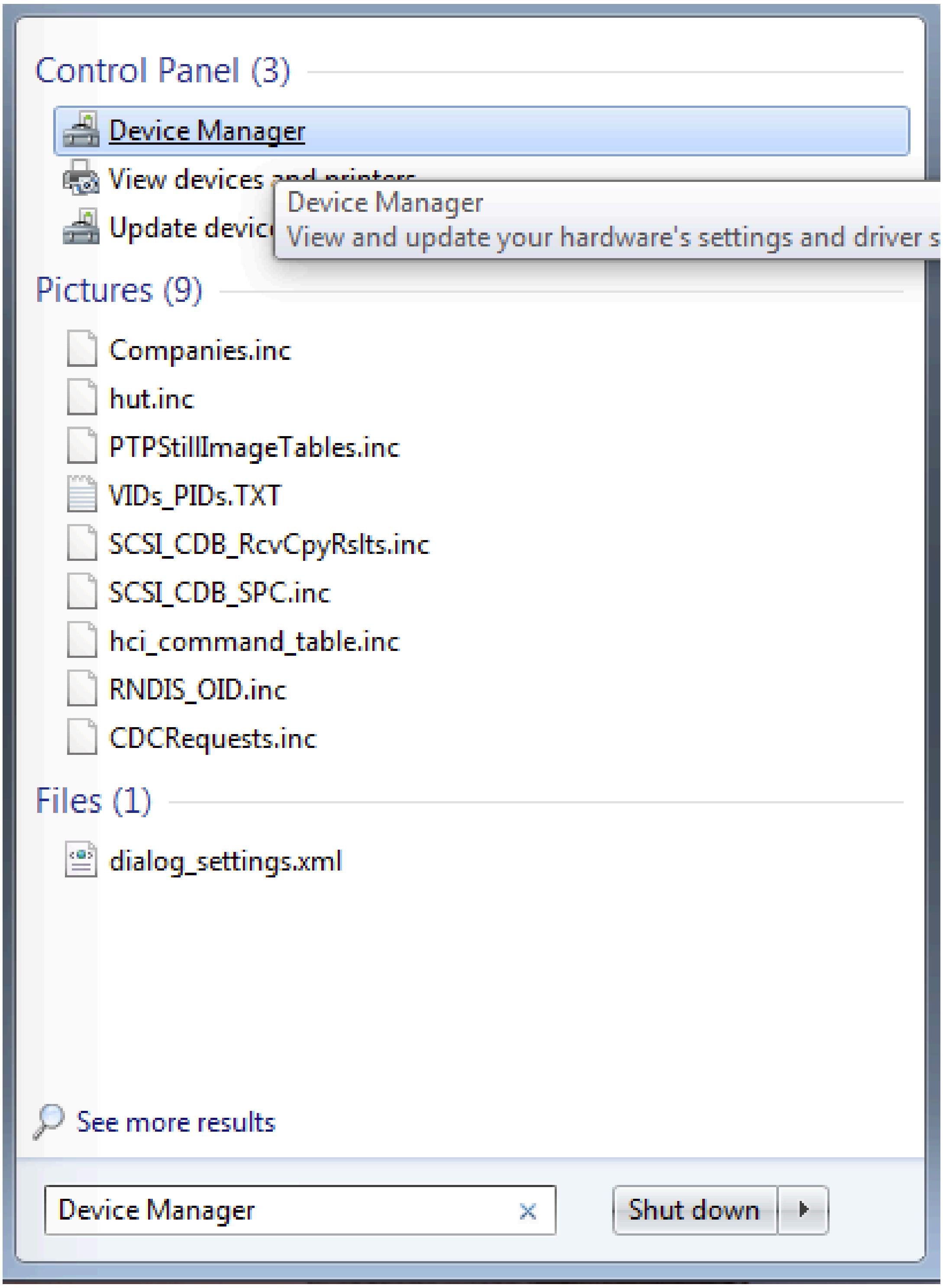

Figure 9. Device Manager

3. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names are different for all the NXP boards.
   a. **USB-UART** interface

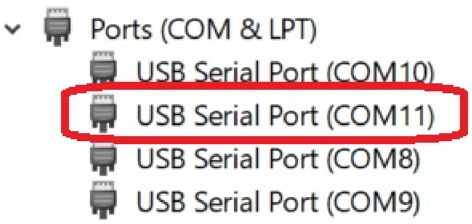

Figure 10. USB-UART interface

9 How to set up Windows/Linux host system

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 sourceforge.net/projects/mingw/files/Installer/.
  2. Run the installer. The recommended installation path is `C:\MinGW`, however, you may install to any location.
     Note: *The installation path should not contain any spaces.*
  3. Ensure that **mingw32-base** and **msys-base** are selected under **Basic Setup**.

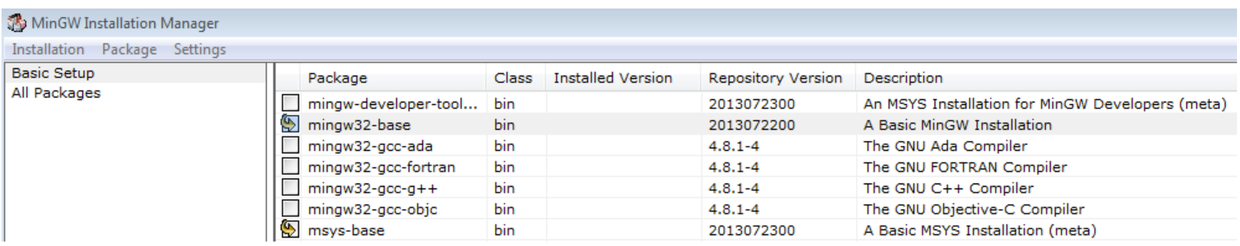

Figure 11. Setup MinGW and MSYS

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

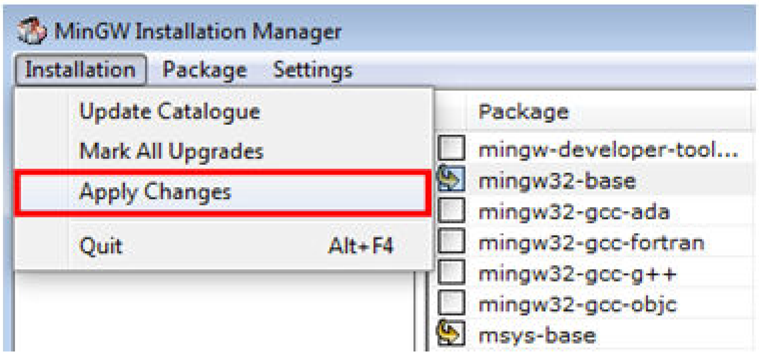

Figure 12. Complete MinGW and MSYS installation

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 13. 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 Kinetis SDK v2.10.0), remove it to ensure that the new GCC build system works correctly.*

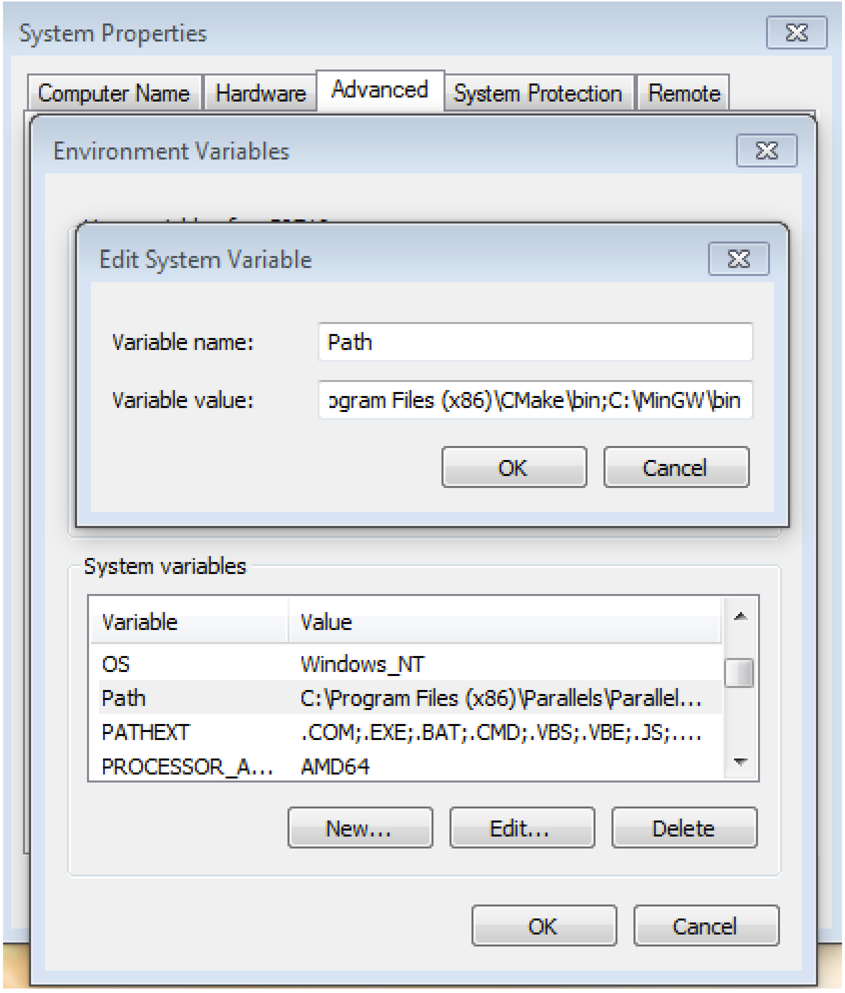

Figure 13. Add Path to systems environment

- CMake
  1. Download CMake 3.0.x from www.cmake.org/cmake/resources/software.html.
  2. While installing, ensure that the option **Add CMake to system PATH for all users** is selected. You can select install CMake into the path for all users or just the current user. In this example, it is installed for all users.

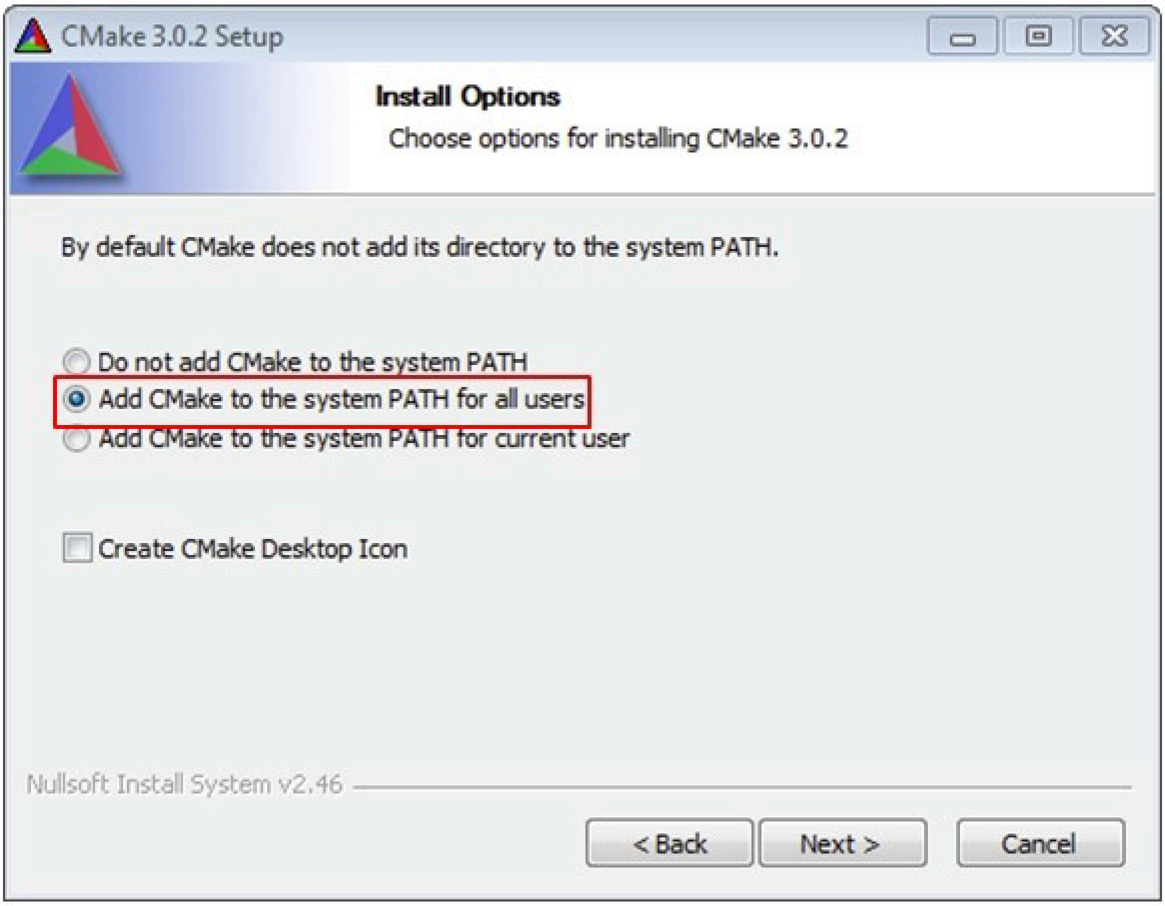

Figure 14. Install CMake

3. Follow the remaining instructions of the installer.
4. Reboot your system for the path changes to take effect.

10 Revision history

Table 6 below summarizes the revisions to this document.

Table 6. Revision history

Revision number	Date	Substantive changes
1	20 June 2023	Initial public release

11 Legal information

11.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

11.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

11.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 20 June 2023
Document identifier: MCUXSDKIMX8ULPGSG