Document identifier: MCUXSDKIMX8ULPGSG
Rev. D, 10 March 2022

Getting Started with MCUXpresso SDK IMX8ULP

Contents

Chapter 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.

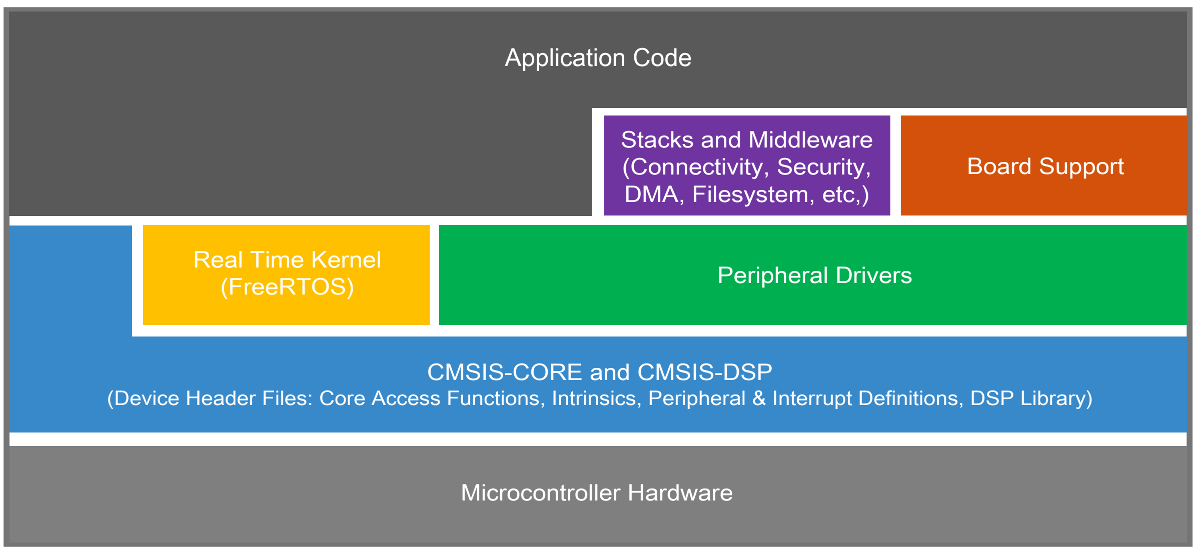

Figure 1. MCUXpresso SDK layers

Chapter 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.

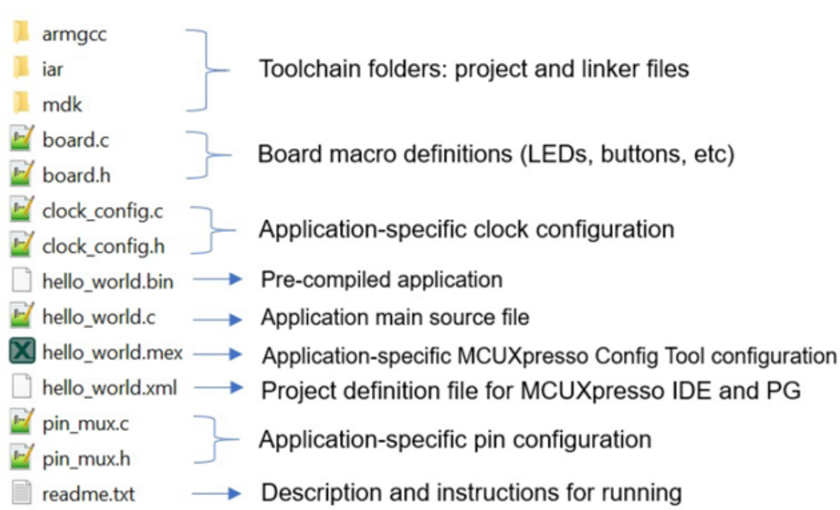

Figure 2. Application folder structure

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).

Chapter 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

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.

Chapter 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 and 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.

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 How to set up Windows/Linux host system 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
```

4.1.3 Run an example application

This section describes steps to run a demo application using the 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 Plus debug probe, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or Tera Term, 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

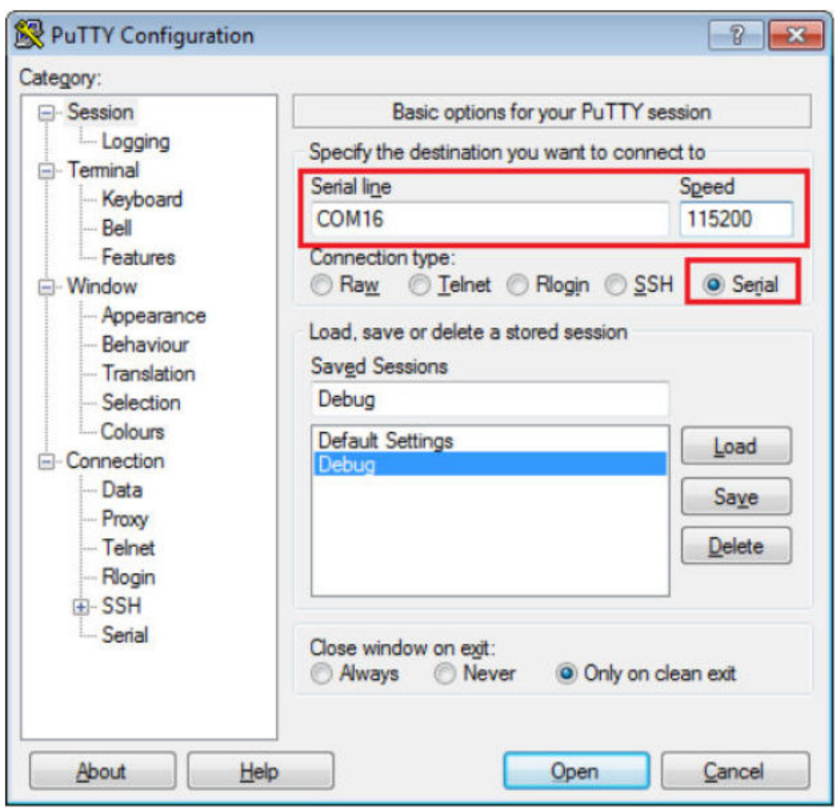

Figure 3. Terminal (PuTTY) configurations

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 MIMX8UD7_M33 device:

```
$ JLinkGDBServer.exe -jlinkscriptfile
Devices\NXP\iMX8ULP\NXP_iMX8ULP_Connect_CortexM33.JLinkScript -device MIMX8UD7_M33 -if SWD
SEGGER J-Link GDB Server V6.98a Command Line Version
JLinkARM.dll V6.98a (DLL compiled Mar  5 2021 17:01:02)

-----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:     localhost only
Generate logfile:             off
Verify download:              off
Init regs on start:           off
Silent mode:                  off
Single run mode:              off
Target connection timeout:    5000 ms
------J-Link related settings------
J-Link Host interface:        USB
J-Link script:                Devices\NXP\iMX8ULP\NXP_iMX8ULP_Connect_CortexM33.JLinkScript
J-Link settings file:         none
------Target related settings------
Target device:                MIMX8UD7_M33
Target interface:             SWD
Target interface speed:       4000kHz
Target endian:                little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Feb  4 2021 12:58:41
Hardware: V10.10
S/N: 600109561
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.32 V
Listening on TCP/IP port 2331
Connecting to target...
Connected to target
Waiting for GDB connection..."
```

4. Change to the directory that contains the example application output. The output can be found in 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/evkmimx8ulp/demo_apps/hello_world/armgcc/debug`

5. Start the GDB client.
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
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
```

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

```
(gdb) continue
```

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.

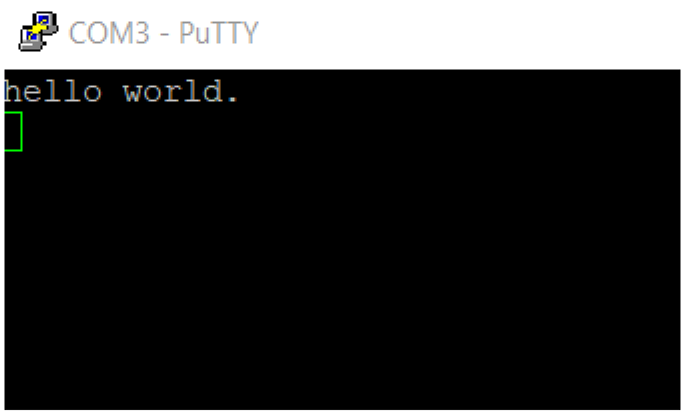

Figure 4. Text display of the hello_world demo

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 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 How to set up Windows/Linux host system 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**.

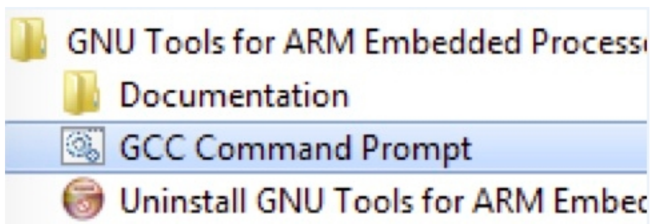

Figure 5. Launch GCC command prompt

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 6.

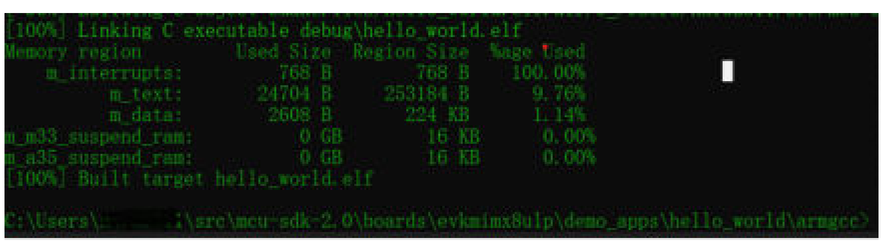

Figure 6. hello_world demo build successful

4.2.3 Run an example application

This section describes steps to run a demo application using the J-Link GDB Server application. To perform these steps, you should have:

- A standalone J-Link Plus debug 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 Plus debug probe, also connect it to the SWD/JTAG connector of the board.

2. Open the terminal application on the PC, such as PuTTY or Tera Term, 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

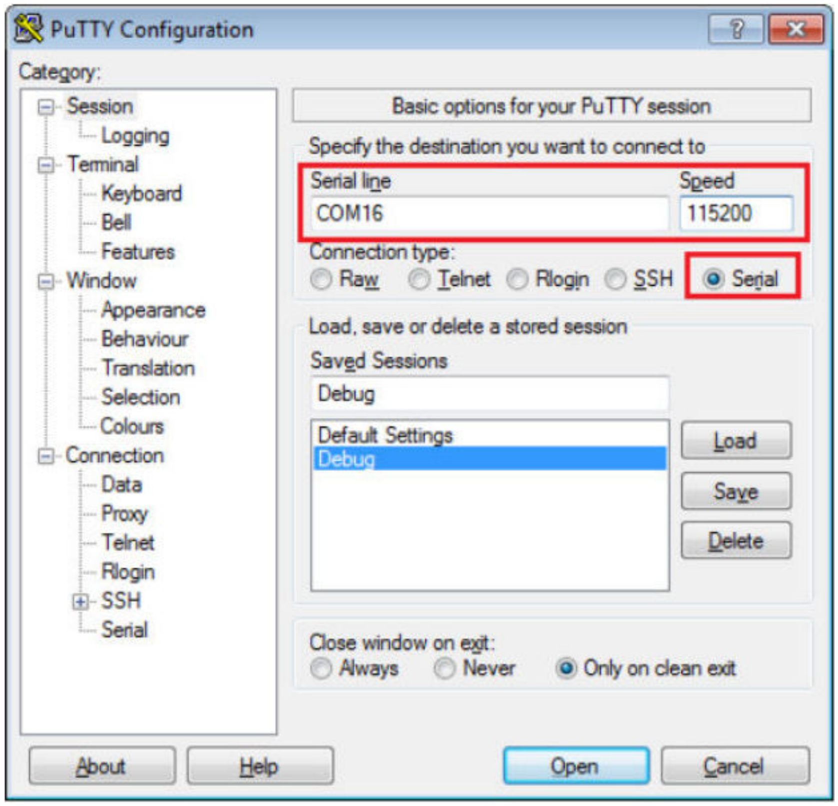

Figure 7. Terminal (PuTTY) configurations

3. After GDB server is running, the screen should resemble Figure 8.

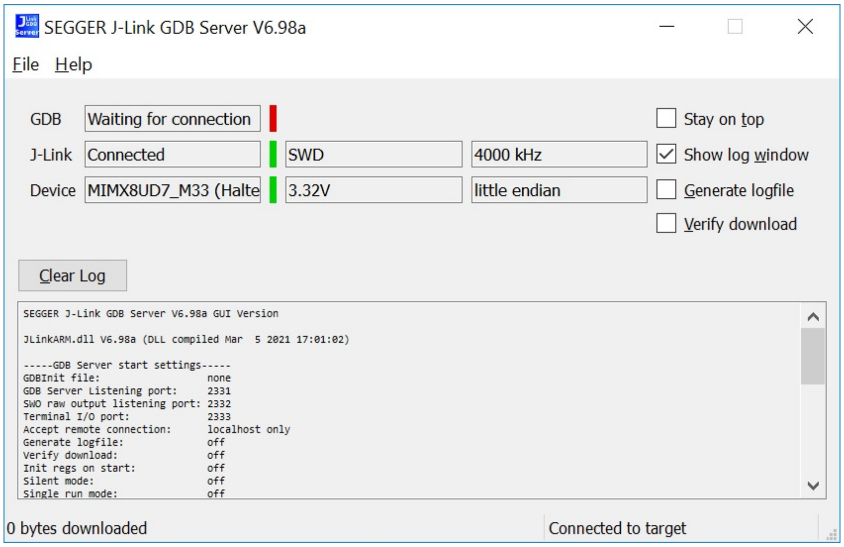

Figure 8. SEGGER J-Link GDB Server screen after successful connection

4. If not already running, open a GCC Arm embedded toolchain command window. To launch the window, from the Windows operating system, select **Start -> Programs -> GNU Tools ARM Embedded <version> -> GCC Command Prompt**.

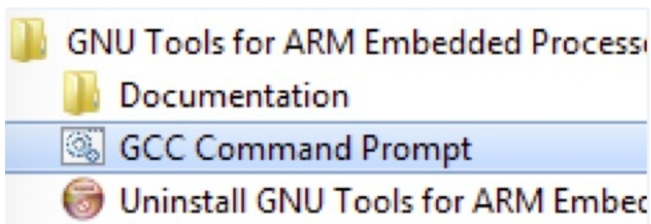

Figure 9. Launch command prompt

5. Change to the directory that contains the example application output. The output can be found in 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/evkmimx8ulp/demo_apps/hello_world/armgcc/debug
```

6. Run the command `arm-none-eabi-gdb.exe <application_name>.elf`. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.
7. Run the following commands:
   a. `target remote localhost:2331`
   b. `monitor reset`
   c. `monitor halt`
   d. `load`

8. The application is now downloaded and halted at the reset vector. Execute the `continue` 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.

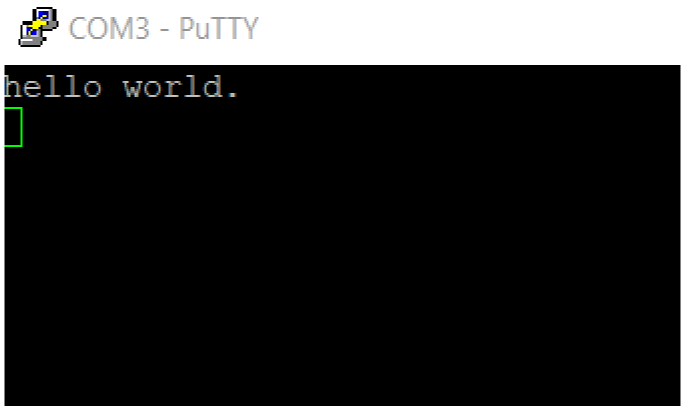

Figure 10. Text display of the hello_world demo

Chapter 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.

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**.

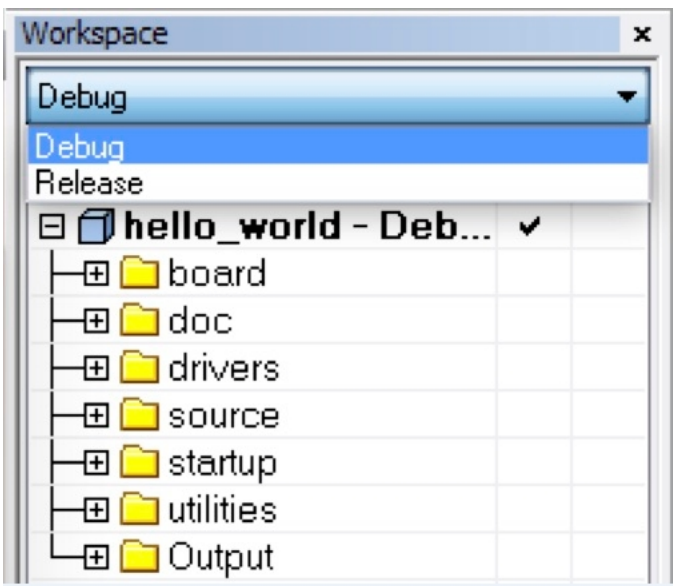

Figure 11. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red in Figure 12.

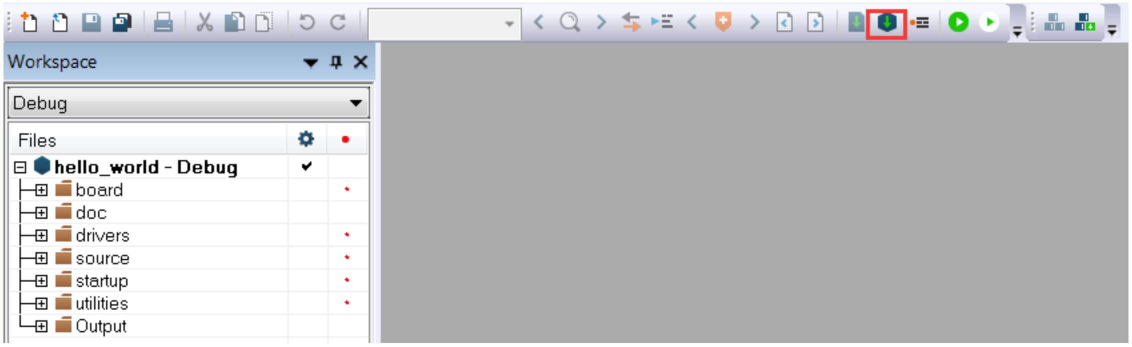

Figure 12. Build the demo application

4. The build completes without errors.

5.2 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 www.segger.com.
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 5 V power supply and J-Link Plus debug probe to the device.
3. Open the terminal application on the PC, such as PuTTY or Tera Term, 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

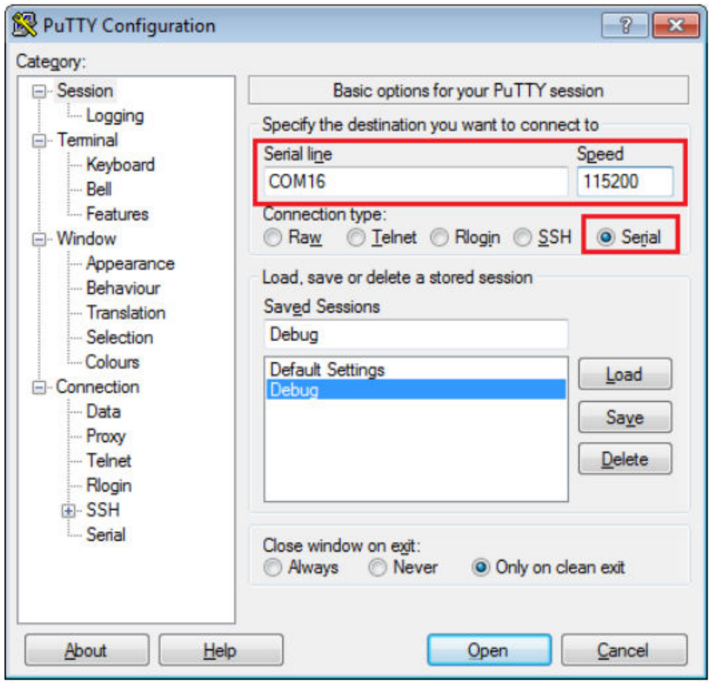

Figure 13. Terminal (PuTTY) configuration

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

Figure 14. Download and Debug button

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

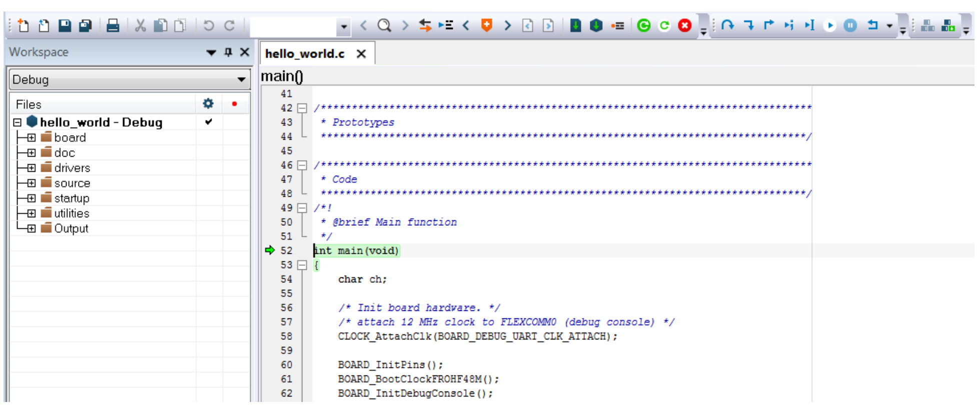

Figure 15. Stop at main() when running debugging

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

Figure 16. 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.

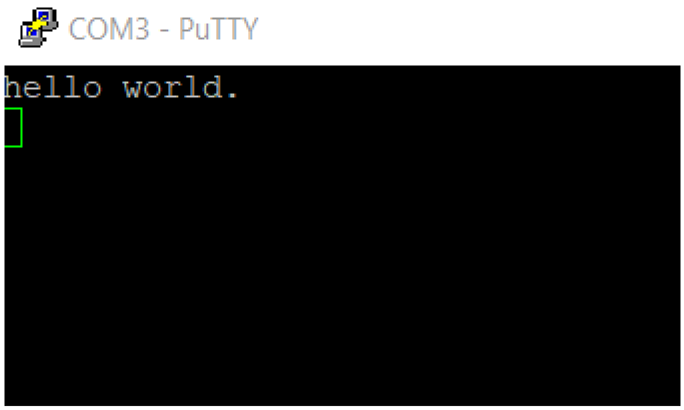

Figure 17. Text display of the hello_world demo

Chapter 6
Running 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. 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.

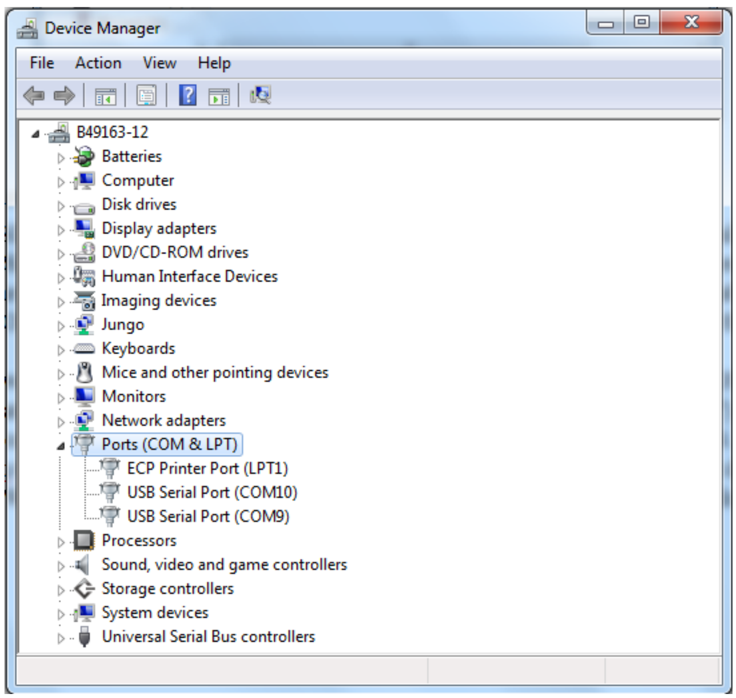

Figure 18. Determining the COM port of target board

3. Get `imx-mkimage`, `s400 firmware(mx8ulpa0-ahab-container.img)`, `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://source.codeaurora.org/external/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(mx8ulpa0-ahab-container.img)`.

   ```
   $ cp mx8ulpa0-ahab-container.img iMX8ULP/
   ```

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

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

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

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

f. Get `bl31.bin`.

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

4. 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</td><td>1000_xx00 Single Boot-eMMC</td></tr>
<tr><td></td><td colspan="2">make SOC=iMX8ULP flash_singleboot_flexspi
for RAM target: make SOC=iMX8ULP flash_singleboot_m33_flexspi</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`.

5. 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
```

6. 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)
```

7. Copy the `flash.bin` image to your tftpboot server.
8. Write `flash.bin to flexspi0 nor flash`. There are two ways:
   a. 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
```

   b. 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
sf probe 0:0
sf erase 0 80000
sf write  0xa0000000 0 ${filesize}
```

9. 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
```

10. 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

11. Power off and switch to low-power boot mode (`sw5[8:1]=1000 0001`), then repower the board.
12. 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.

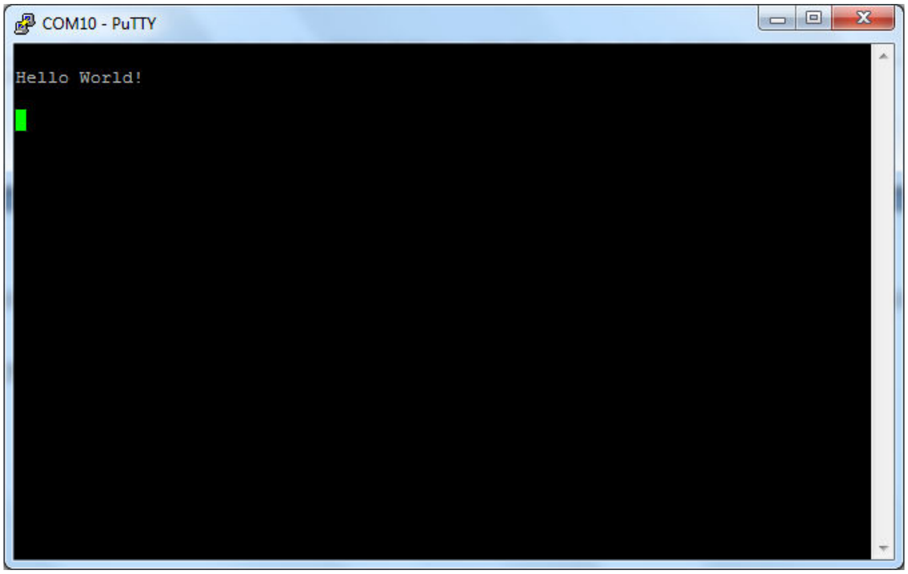

Figure 19. Hello world demo running on Cortex-M33 core

Appendix A
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.

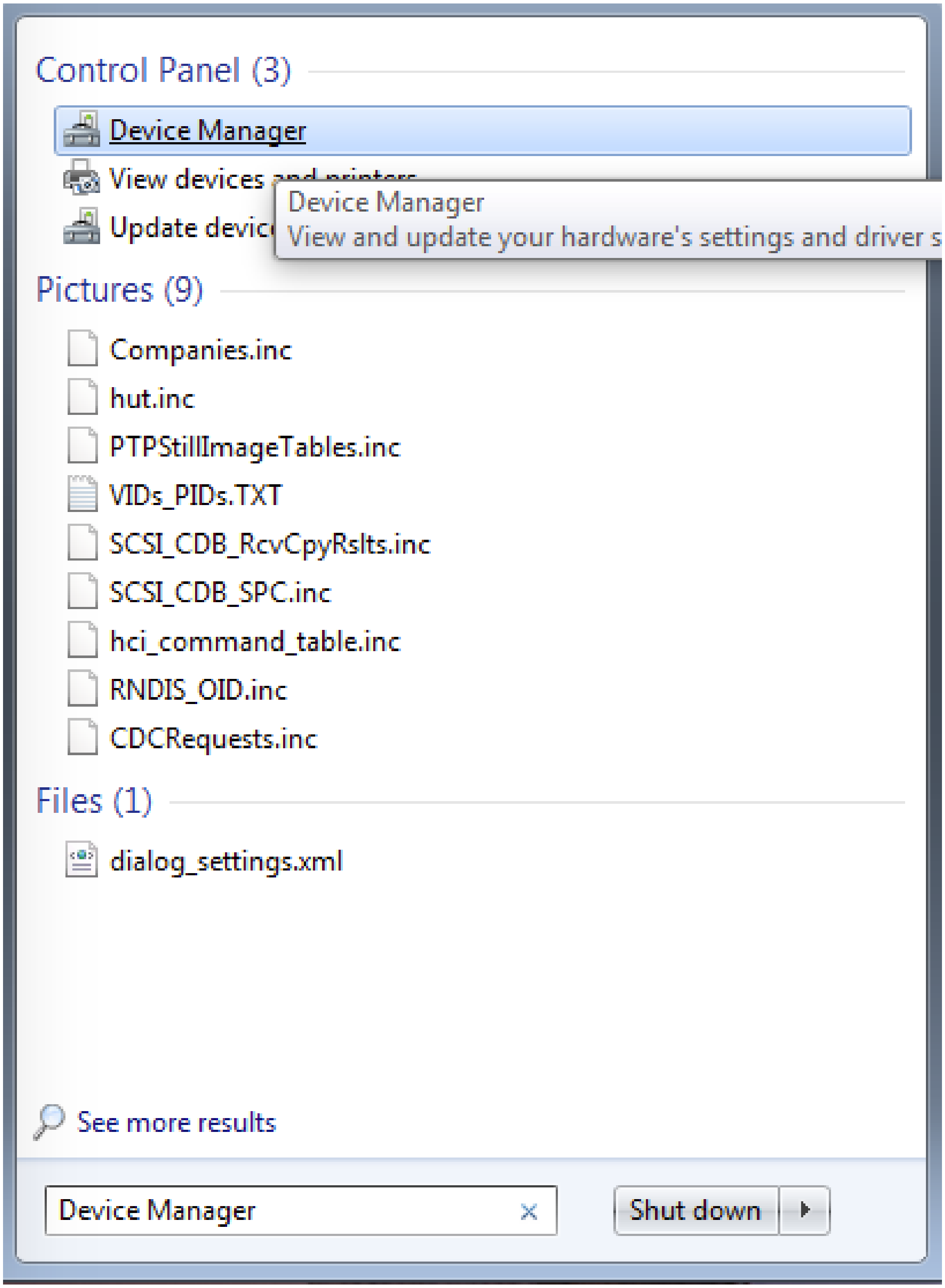

Figure 20. Device Manager

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

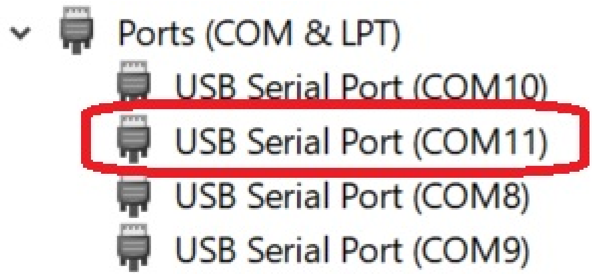

Figure 21. USB-UART interface

Appendix B
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**.

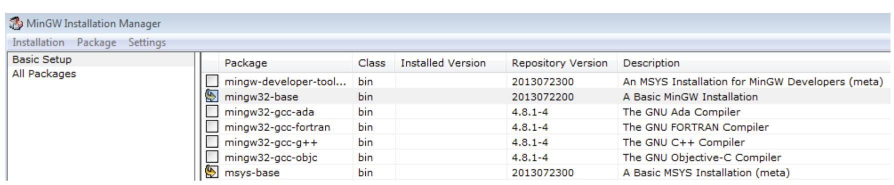

Figure 22. Setup MinGW and MSYS

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

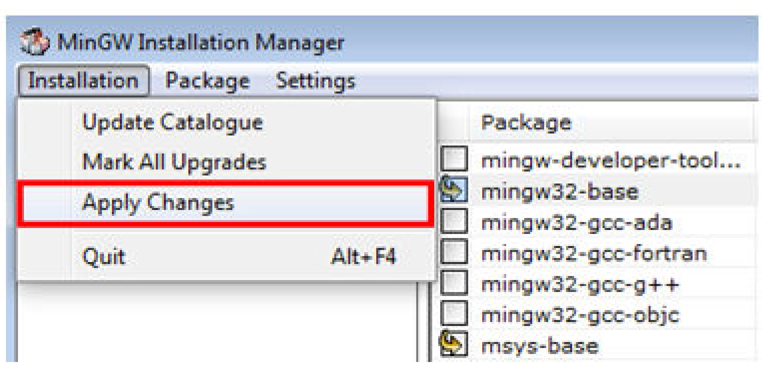

Figure 23. Complete MinGW and MSYS installation

  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 24. 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.

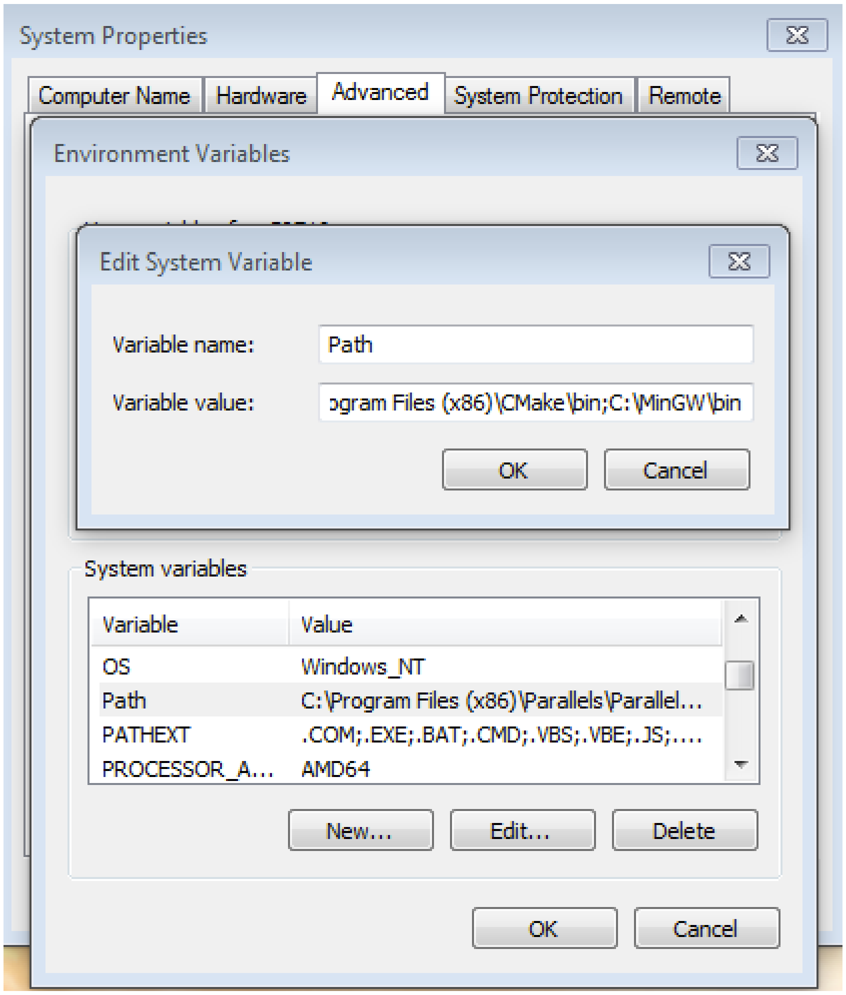

Figure 24. Add Path to systems environment

- 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.

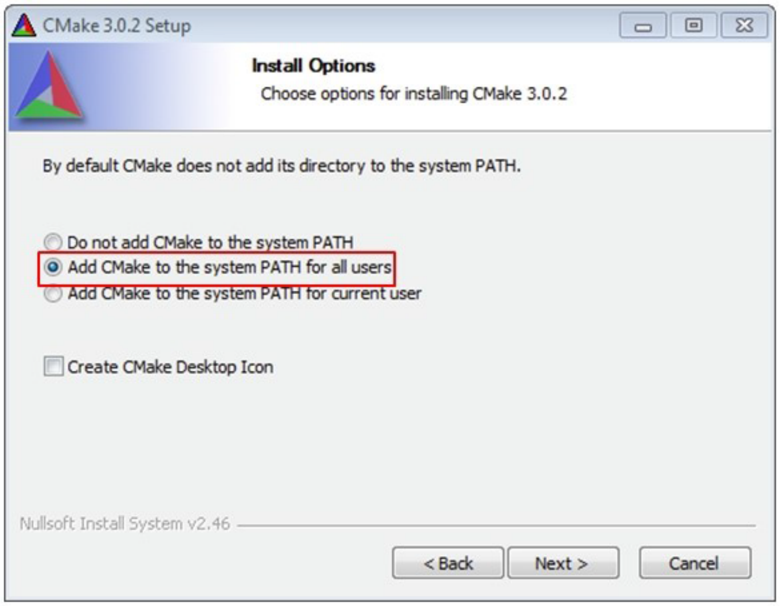

Figure 25. Install CMake

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

Appendix C Revision history

The table below summarizes the revisions to this document.

Table 2. Revision history

Revision	Date	Change description
Rev. D	10 March 2022	Added a note.
Rev. C	09 November 2021	Updated steps in Running an application using imx-mkimage
Rev. B	13 September 2021	Updated Running an application using imx-mkimage
Rev. A	16 June 2021	Initial NDA release

Legal information

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.

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 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.

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.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Airfast — is a trademark of NXP B.V.

Bluetooth — the Bluetooth wordmark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by NXP Semiconductors is under license.

Cadence — the Cadence logo, and the other Cadence marks found at www.cadence.com/go/trademarks are trademarks or registered trademarks of Cadence Design Systems, Inc. All rights reserved worldwide.

CodeWarrior — is a trademark of NXP B.V.

ColdFire — is a trademark of NXP B.V.

ColdFire+ — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

EdgeScale — is a trademark of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

FeliCa — is a trademark of Sony Corporation.

Freescale — is a trademark of NXP B.V.

HITAG — is a trademark of NXP B.V.

ICODE and I-CODE — are trademarks of NXP B.V.

Immersiv3D — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

Mantis — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

MOBILEGT — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

Processor Expert — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

SafeAssure — is a trademark of NXP B.V.

SafeAssure — logo is a trademark of NXP B.V.

StarCore — is a trademark of NXP B.V.

Synopsys — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

Tower — is a trademark of NXP B.V.

UCODE — is a trademark of NXP B.V.

VortiQa — is a trademark of NXP B.V.

arm

Date of release: 10 March 2022

Document identifier: MCUXSDKIMX8ULPGSG