NXP Semiconductors
User's Guide

Document Number: MCUXSDKIMX8QXGSUG
Rev. 0, 01/2020

Getting Started with MCUXpresso SDK for MEK-MIMX8QX

Contents

1	Overview	1
2	MCUXpresso SDK board support folders	2
3	Toolchain introduction	3
4	Run a demo application using IAR	4
5	Run a demo using Arm® GCC	7
6	Run a demo using imx-mkimage	16
7	Run a demo using facility provided by U-Boot	18
8	Run a flash target demo	18
A	Appendix A - How to determine COM port	24
B	Appendix C - Host Setup	26

1 Overview

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides bare metal source code to be executed in the i.MX 8QuadXPlus M4 core. The MCUXpresso SDK provides comprehensive software support for NXP i.MX 8QuadXPlus microcontrollers' M4 core. 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 Supporting i.MX 8QuadXPlus* (document MCUXSDKIMX8QXRN)

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

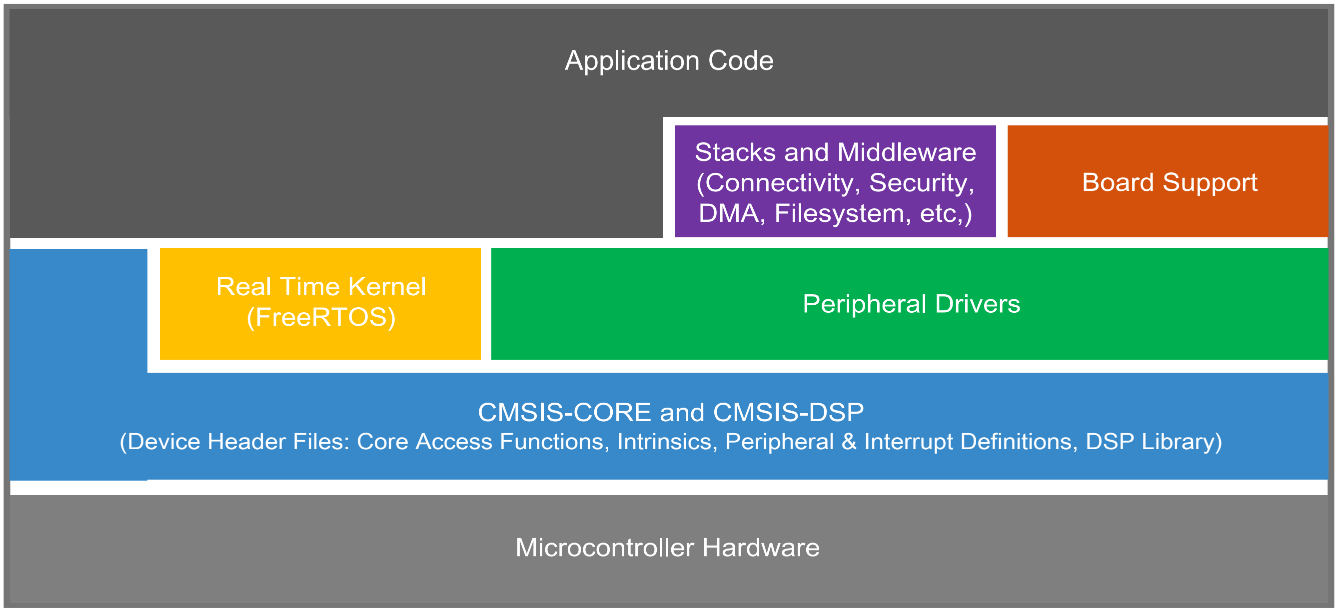

Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK board support folders

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

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

2.1 Example application structure

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

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

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

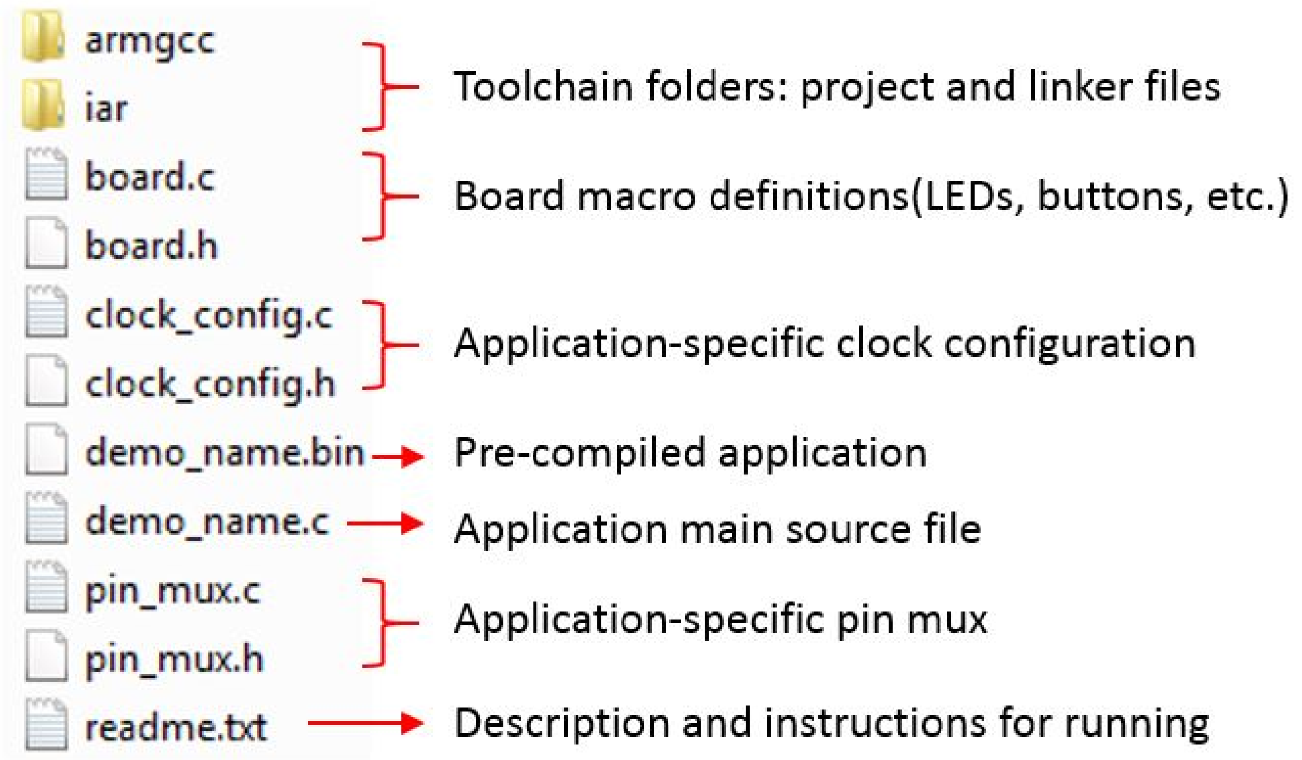

Figure 2. Application folder structure

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

2.2 Locating example application source files

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

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

- `devices/<device_name>`: The device's CMSIS header file, MCUXpresso SDK feature file and a few other files
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<device_name>/scfw_api`: APIs to invoke SCFW calls, let SCU do service on clock, power and resource permission, etc.

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 8QuadXPlus 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 8QuadXPlus 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 8QuadXPlus, the patch for IAR and Segger J-Link should be installed. The patch named `iar_segger_support_patch_imx8.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 8QuadXPlus 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 8QuadXPlus device. Currently, the tool can only be executed on Linux OS.

4 Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK using IAR. The hello_world demo application targeted for the 8QuadXPlus MEK board is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

4.1 Build an example application

Before using IAR, get the IAR and Segger J-Link patch, `iar_segger_support_patch_imx8.zip`. Install the i.MX8QX 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>/iar`

   Using the i.MX 8QuadXPlus MEK board as an example, the `hello_world` workspace is located in:

   `<install_dir>/boards/mekmimx8qx/demo_apps/hello_world/iar/hello_world.eww`

   Other example applications may have additional folders in their path.

2. Select the desired build target from the drop-down menu.

   For this example, select the **hello_world – debug** target.

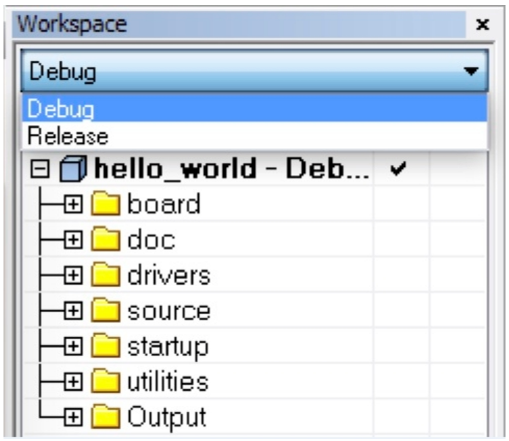

Figure 3. Demo build target selection

3. To build the demo application, click **Make**, highlighted in red, as shown in Figure 4.

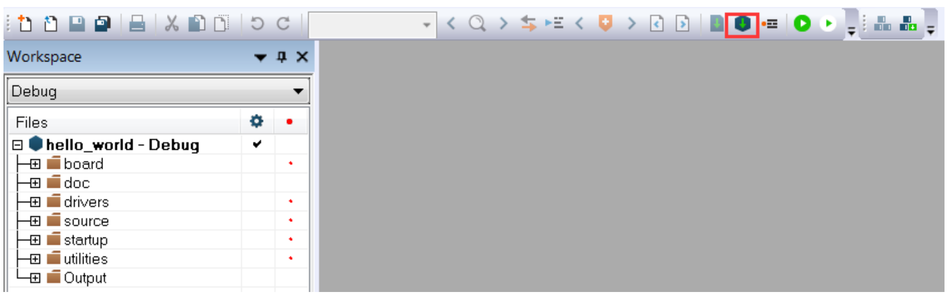

Figure 4. Build the demo application

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

4.2 Run an example application

Before running an example, a bootable SD card with the SCFW (System Controller Firmware) image is needed. See *Section 6.3, "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 Appendix A). Configure the terminal with these settings:
   a. 115200 baud rate
   b. No parity
   c. 8 data bits
   d. 1 stop bit

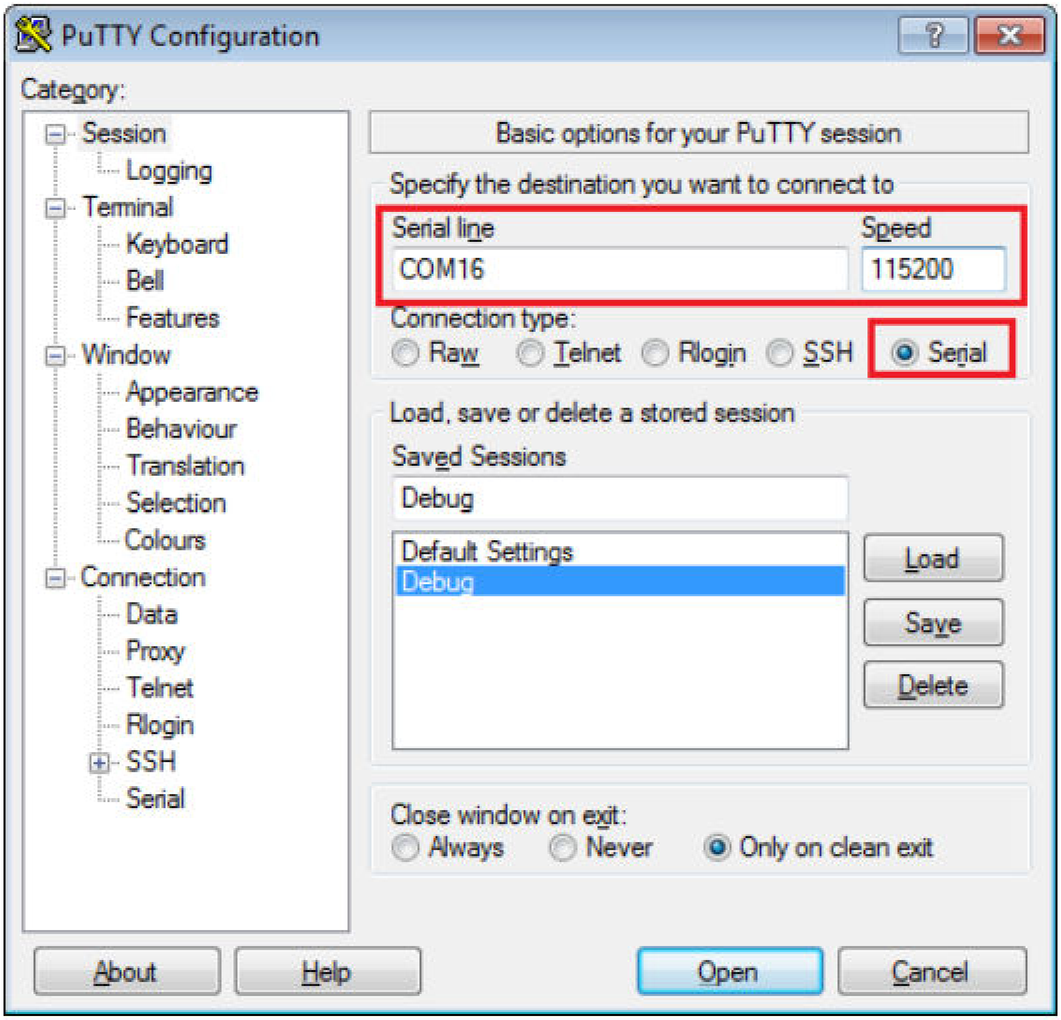

Figure 5. Terminal (PuTTY) configuration

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

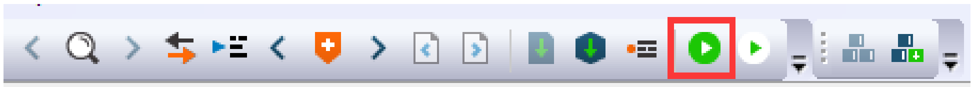

Figure 6. Download and Debug button

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

Figure 7. Stop at main() when running debugging

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

Figure 8. Go button

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

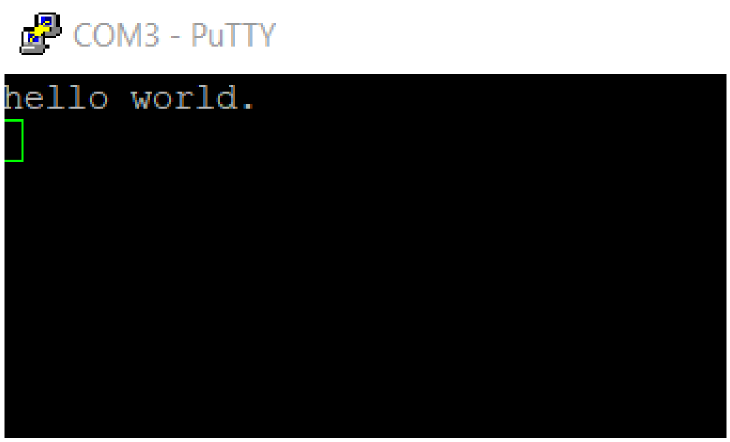

Figure 9. Text display of the hello_world demo

5 Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The hello_world demo application targeted for i.MX8QX platform is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

NOTE

Before running a demo, make sure the SEGGER patch is installed. See Appendix B to know how to install the patch.

5.1 Linux OS host

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

5.1.1 Set up toolchain

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

5.1.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from 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 the *Host Setup* Section in Appendix C for Linux OS before compiling the application.

5.1.1.2 Add a new system environment variable for ARMGCC_DIR

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

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

5.1.2 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/mekmimx8qx/demo_apps/hello_world/armgcc`
2. Run the **build_debug.sh** script on the command line to perform the build. The output is shown in this figure:

```
$ ./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_2.2_MEK_MIMX8QX/boards/mekmimx8qx/demo_apps/hello_world/armgcc




Scanning dependencies of target hello_world.elf

[ 4%] Building C object CMakeFiles/hello_world.elf.dir/work/platforms/tmp/
SDK_2.2_MEK_MIMX8QX/boards/mekmimx8qx/demo_apps/hello_world/board.c.obj




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

5.1.3 Run an example application

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

- Make a bootable SD card with the SCFW (System Controller Firmware) image. See *Section 6.3, "Make a bootable SD card with System Controller Firmware (SCFW)"*
- A standalone J-Link pod that is connected to the debug interface of your board.

NOTE

The Segger J-Link software has to be patched with the JLink_<jlink_version>_8QX_Patch.zip patch for i.MX8QX from iar_segger_support_patch_8qx.zip.

After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the development platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the 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 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

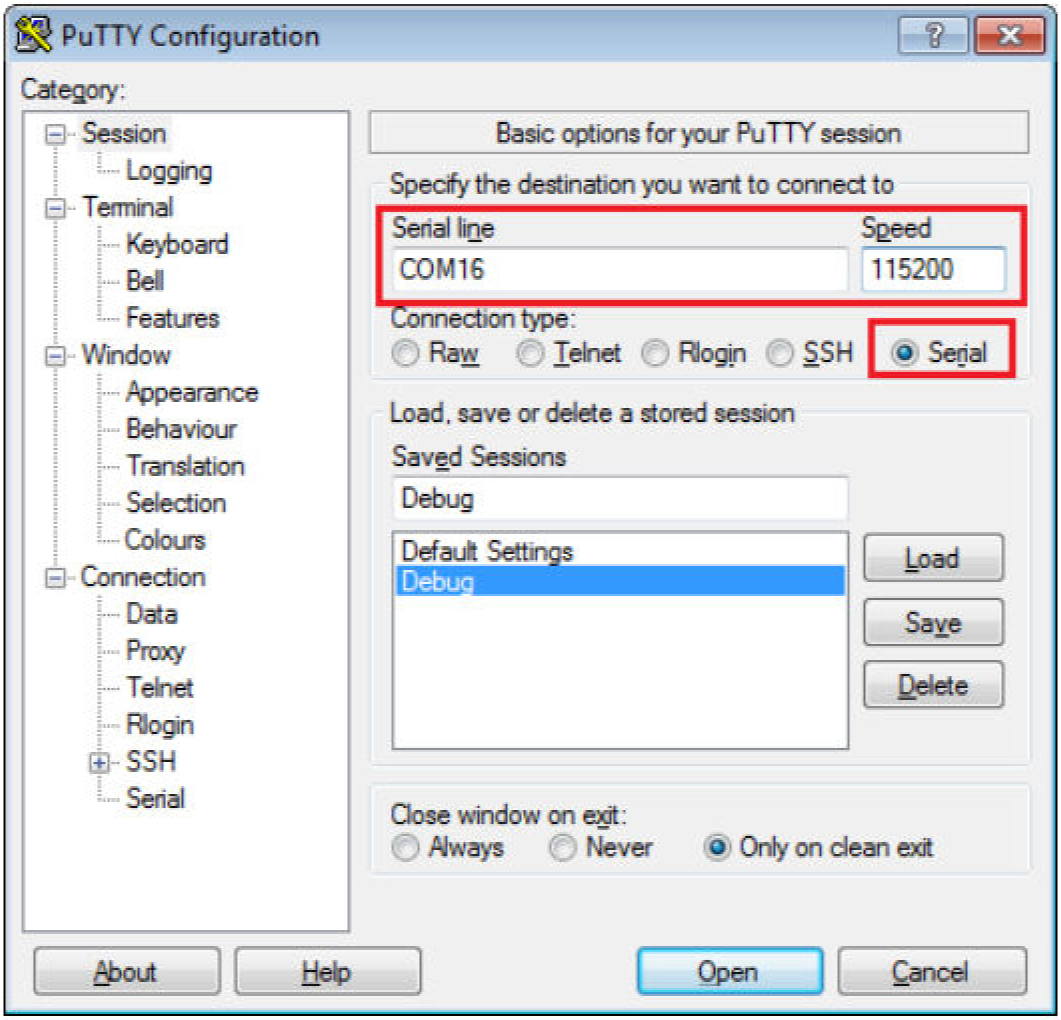

Figure 10. Terminal (PuTTY) configurations

3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched from a new terminal for the MIMX8QX6_CM4 device:

```
$ JLinkGDBServer -if JTAG -device MIMX8QX6_CM4
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 MIMX8QX6_CM4
-----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: MIMX8QX6_CM4
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
```

```
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>/armgcc/debug

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

   For this example, the path is:

   <install_dir>/boards/mekmimx8qx/demo_apps/hello_world/armgcc/debug

5. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf
GNU gdb (7.10-1ubuntu3+9) 7.10
Copyright (C) 2015 Free Softwar.exee 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.

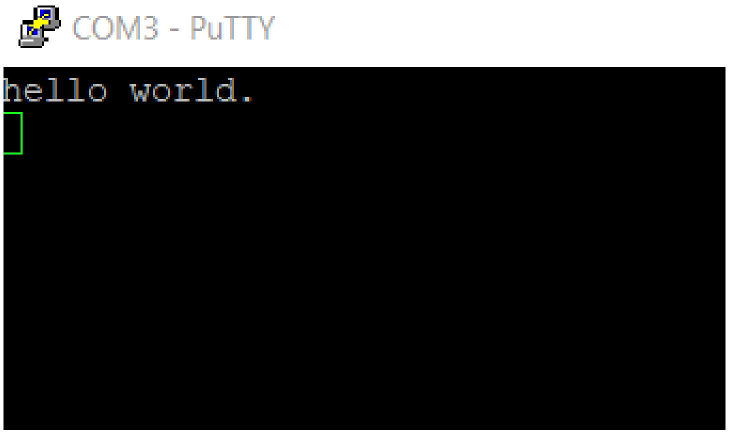

Figure 11. Text display of the hello_world demo

5.2 Windows OS host

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

5.2.1 Set up toolchain

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

5.2.1.1 Install GCC Arm Embedded tool chain

Download and run the installer from 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 the *MCUXpresso SDK Release Notes* (document ID: MCUXSDKRN).

NOTE

See the *Host Setup* Section in Appendix C for Windows OS before compiling the application.

5.2.1.2 Add a new system environment variable for ARMGCC_DIR

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

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

5.2.2 Build an example application

To build an example application, follow these steps.

1. Open a GCC Arm Embedded tool chain command window. To launch the window, from the Windows operating system Start menu, go to “Programs -> GNU Tools ARM Embedded <version>” and select “GCC Command Prompt”.

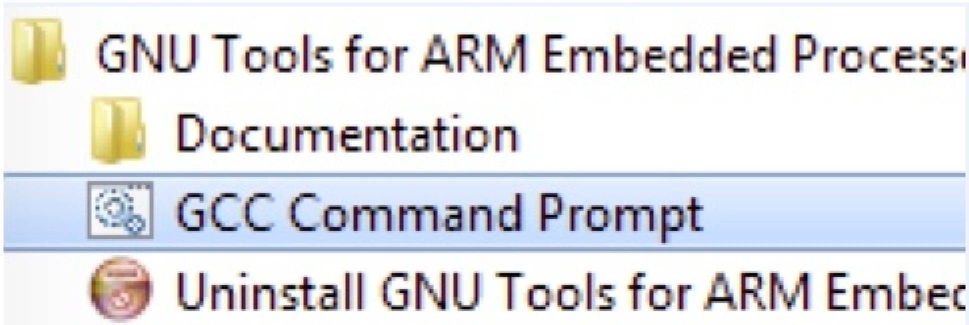

Figure 12. Launch command prompt

2. Change the directory to the example application project directory, which has a path similar to the following:

   <install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc

   For this example, the exact path is: *<install_dir>/boards/mekmimx8qx/demo_apps/hello_world/armgcc/debug*
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 shown in this figure:

5.2.3 Run an example application

Running the Arm GCC built demo also requires J-Link support. Get the IAR and Segger J-Link patch, iar_segger_support_patch_8qx.zip. Install the i.MX8QX support patch following the guides in readme.txt located in the archive.

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

- Make a bootable SD card with the SCFW (System Controller Firmware) image. See *Section 6.2, "Make a bootable SD card with System Controller Firmware (SCFW)"*. You have a standalone J-Link pod that is connected to the debug interface of your board. Make sure the Segger J-Link software i.MX8QX supporting patch, iar_segger_support_patch_imx8.zip is installed.

After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the development platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 baud rate
   b. No parity
   c. 8 data bits
   d. 1 stop bit

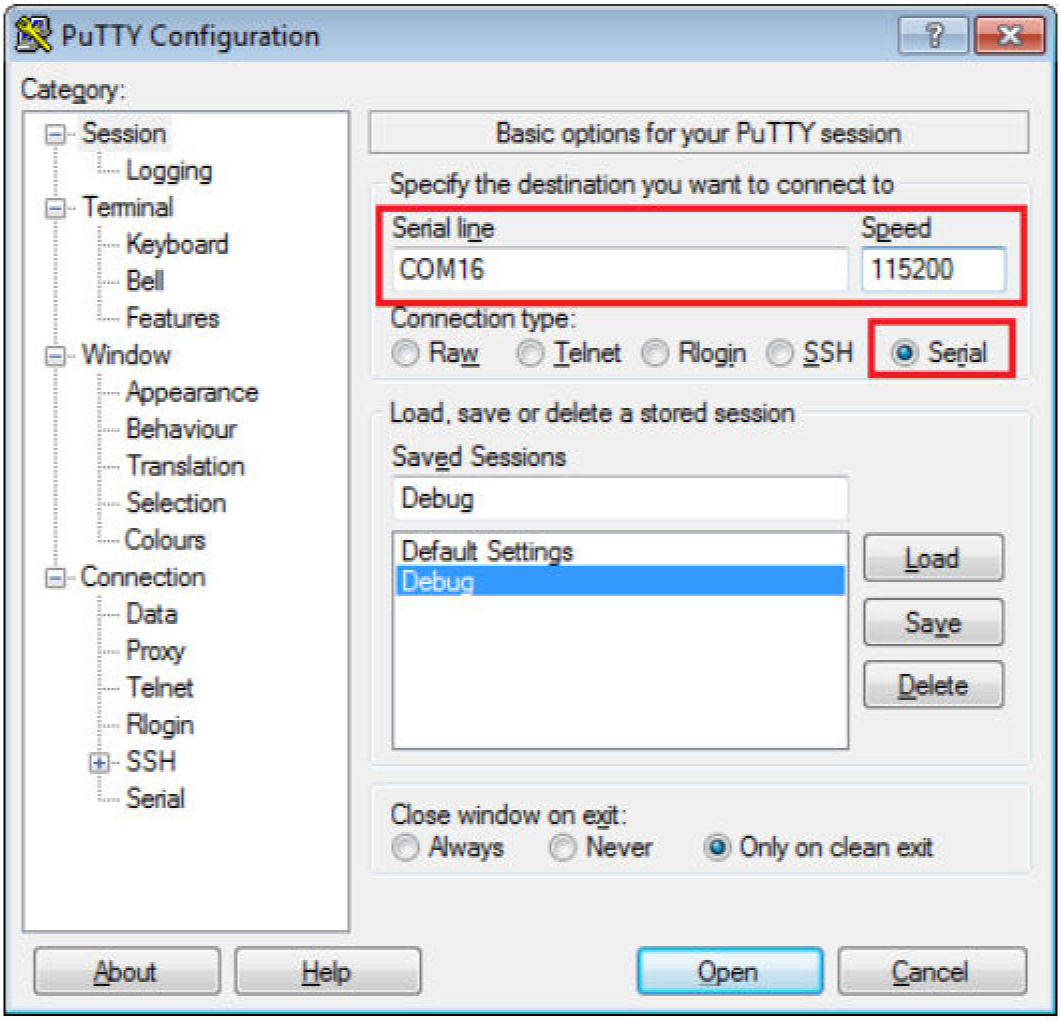

Figure 13. Terminal (PuTTY) configurations

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 below. The target device selection chosen for this example is the MIMX8QX6_CM4.
5. After it is connected, the screen should resemble this figure:

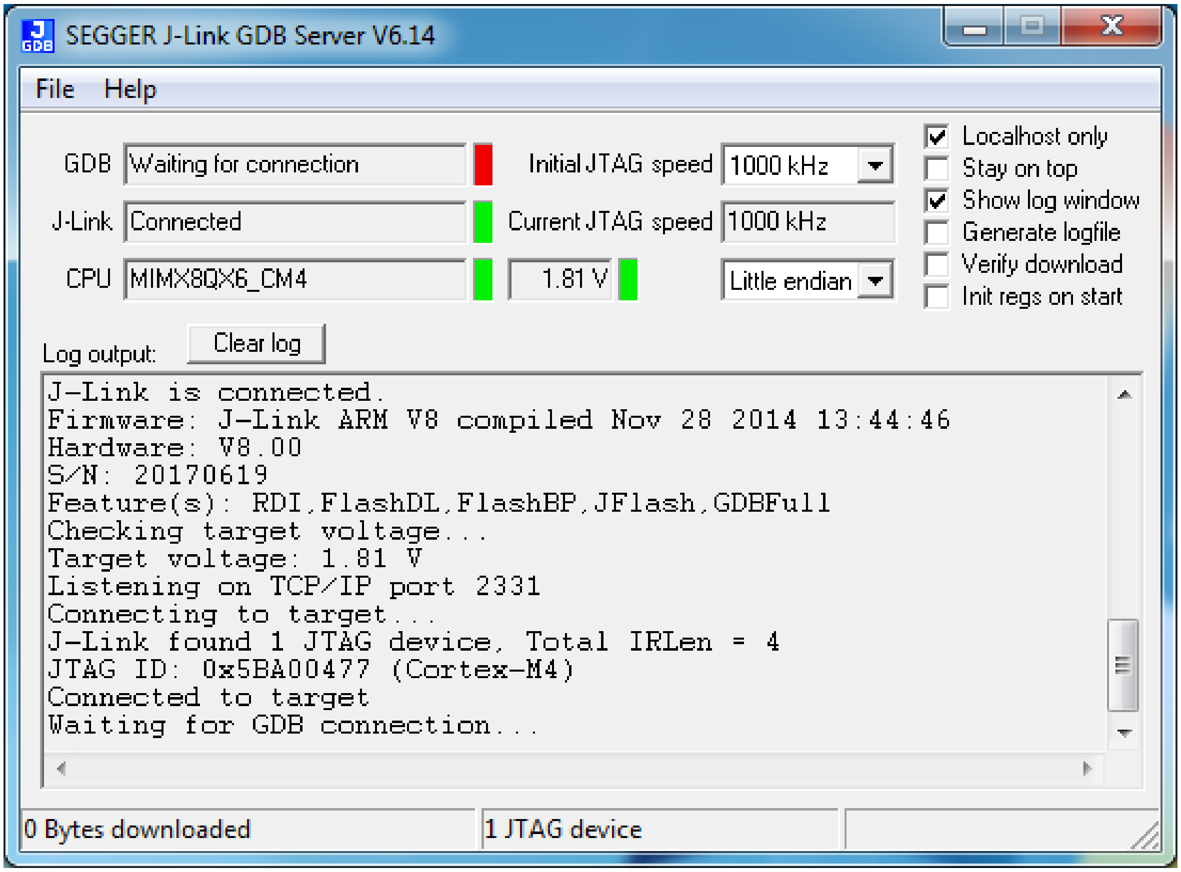

Figure 14. SEGGER J-Link GDB server screen after successful connection

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”.

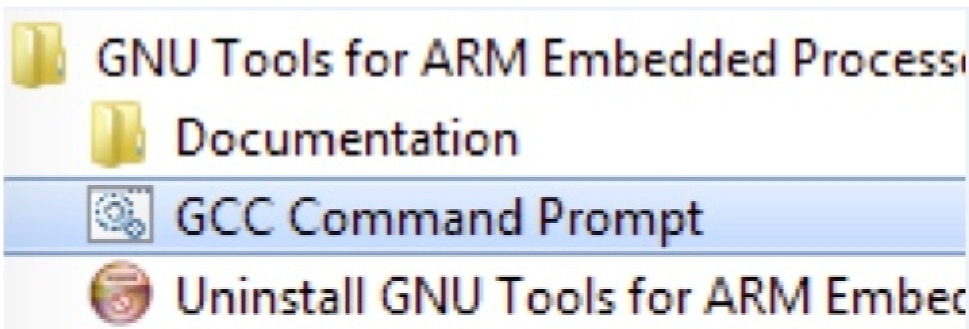

Figure 15. Launch command prompt

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/mekmimx8qx/demo_apps/hello_world/armgcc/debug

8. Run the command “arm-none-eabi-gdb.exe <application_name>.elf”. For this example, it is “arm-none-eabi-gdb.exe hello_world.elf”.

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.

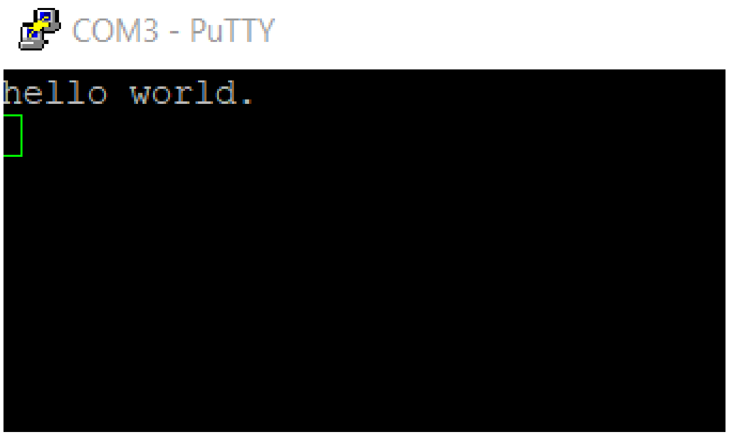

Figure 16. Text display of the hello_world demo

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 version.

   ```
   $ git checkout imx_4.14.98_2.3.0
   ```
3. Get the SCU firmware for i.MX8QX from the NXP website. Then, execute the following command:

   ```
   $ chmod a+x ./imx-sc-firmware-1.3.0 bin

   $ sh ./imx-sc-firmware-1.3.0 bin
   ```

   This extracts the SCU firmware. Rename `mx8qx-mek-scfw-tcm.bin` to `scfw_tcm.bin` and copy the file to `imx-mkimage/iMX8QX`.

4. Get the i.MX seco firmware package from NXP website. Execute the following command:

   ```
   chmod a+x ./imx-seco-2.5.4.bin
   sh ./imx-seco-2.5.4.bin
   ```

This extracts the i.MX firmware. Copy `firmware/seco/mx8qxb0-ahab-container` and `firmware/seco/mx8qxc0-ahab-container` to `imx-mkimage/iMX8QX`. **b0** and **c0** indicate that the firmware is used for B0 silicon and C0 silicon respectively.

5. Generate the `u-boot.bin` and `u-boot-spl.bin` from Linux release package and copy it to `imx-mkimage/iMX8QX`.
6. Generate the Arm Trusted Firmware `bl31.bin` from the Linux release package and copy it to `imx-mkimage/iMX8QX`.

6.1 Run an example application on the M4 core

1. Build the M4 demo application. Rename the generated binary file (`.bin` file) to `m4_image.bin`, and copy to this file to the `imx-mkimage/iMX8QX` folder.
2. In Linux OS, bash `cd` into the *imx-mkimage* installed directory, and run the following command to generate bootable image:

```
$ make clean
```

If the M4 image built is for TCM:

```
$ make SOC=iMX8QX REV=C0 flash_cm4
```

If the M4 image built is for DDR:

```
$ make SOC=iMX8QX REV=C0 flash_cm4_ddr
```

This generates the bootable image `flash.bin` under the **iMX8QX** folder.

NOTE

The `REV=C0` indicates the image is built for the **C0** silicon. For the **B0** silicon, use `REV=B0` instead.

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=./iMX8QX/flash.bin of=/dev/<SD Device> bs=1k seek=32
```

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

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

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

When debugging or running MCUXpresso SDK with IAR and J-Link GDB Server, the bootable SD card with SCU firmware (SCFW) is required. The SCU handles setting the power, clock, pinmux, and so on for other cores, so the SCFW is a 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=iMX8QX REV=C0 flash_scfw
```

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

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

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

1. Copy `u-boot.bin` and `u-boot-spl.bin` into `mx-mkimage/iMX8QX`.
2. Rename the M4 image to `m4_image.bin` and copy it into `imx-mkimage/iMX8QX`.
3. In Linux OS, bash cd into the `imx-mkimage` directory, and run the following command to generate bootable image:

   ```
   $ make clean
   ```

   If the M4 image is built for TCM:

   ```
   make SOC=iMX8QX REV=C0 flash_linux_m4
   ```

   If the M4 image is built for DDR:

   ```
   make SOC=iMX8QX REV=C0 flash_linux_m4_ddr
   ```

   This generates the bootable image `flash.bin` under the **iMX8QX** folder.

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

7 Run a demo using facility provided by U-Boot

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

8 Run a flash target demo

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

8.1 Set up environment

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

8.1.1 Download the MfgTool

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

The tool can be downloaded from github. Use version 1.3.96 or higher for full support for the M4 image. Download 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.

8.1.2 Switch to SERIAL mode

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

1. Set the board boot mode to SERIAL[b'0001].
2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J10 USB Type-C on the CPU board.
3. The PC recognizes the i.MX8QX device as (VID:PID)=(1FC9:012F), which is shown in the figure below.

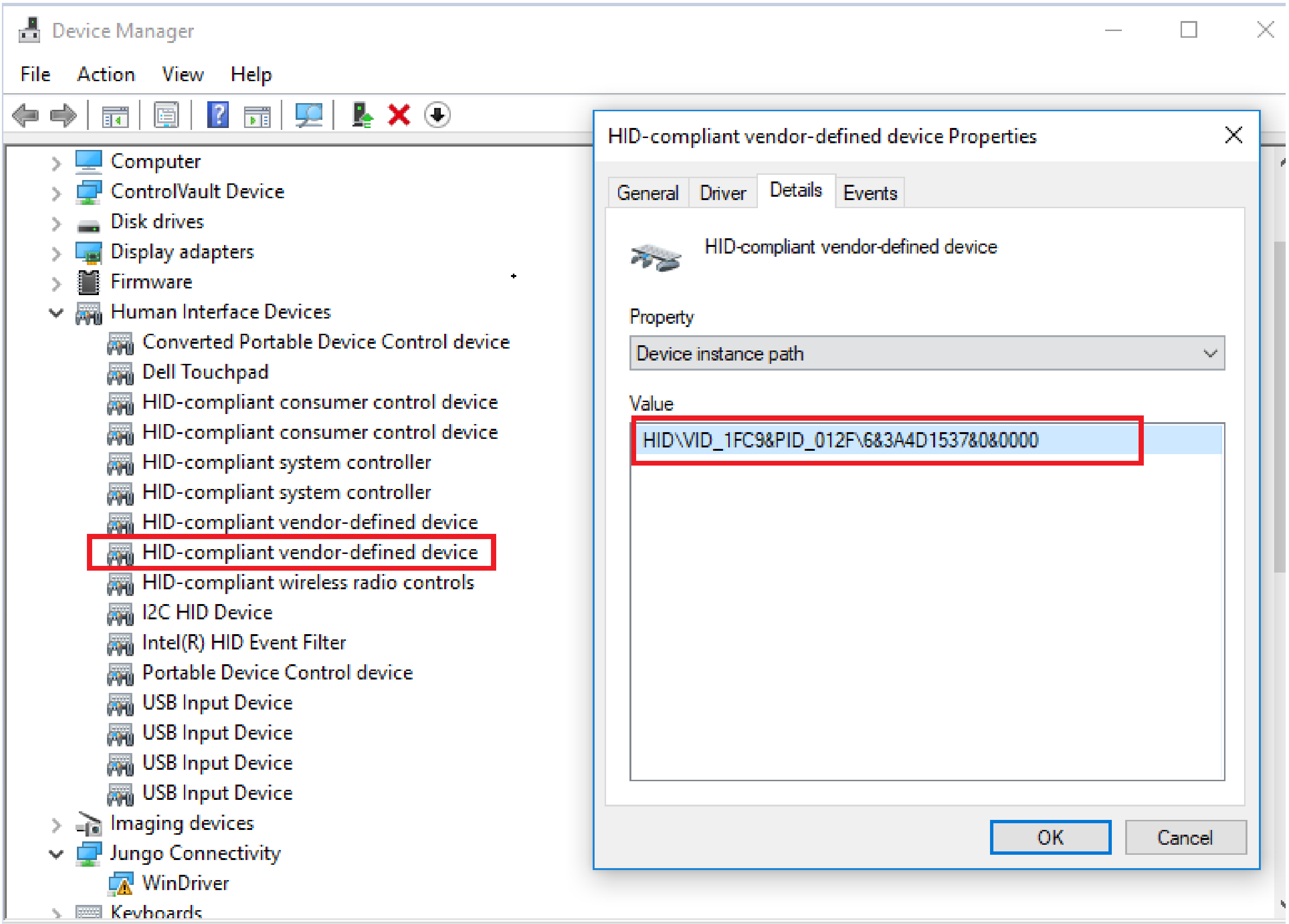

Figure 17. Device as shown in Device Manager

8.2 Build an example application

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

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

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

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

   ```
   <install_dir>/boards/mekmimx8qx/multicore_examples/rpmsg_lite_pingpong_rtos/
   linux_remote/iar/rpmsg_lite_pingpong_rtos_linux_remote.eww
   ```
2. Select the desired build target from the drop-down. For this example, select the **rpmsg_lite_pingpong_rtos_linux_remote - flash_debug** target.

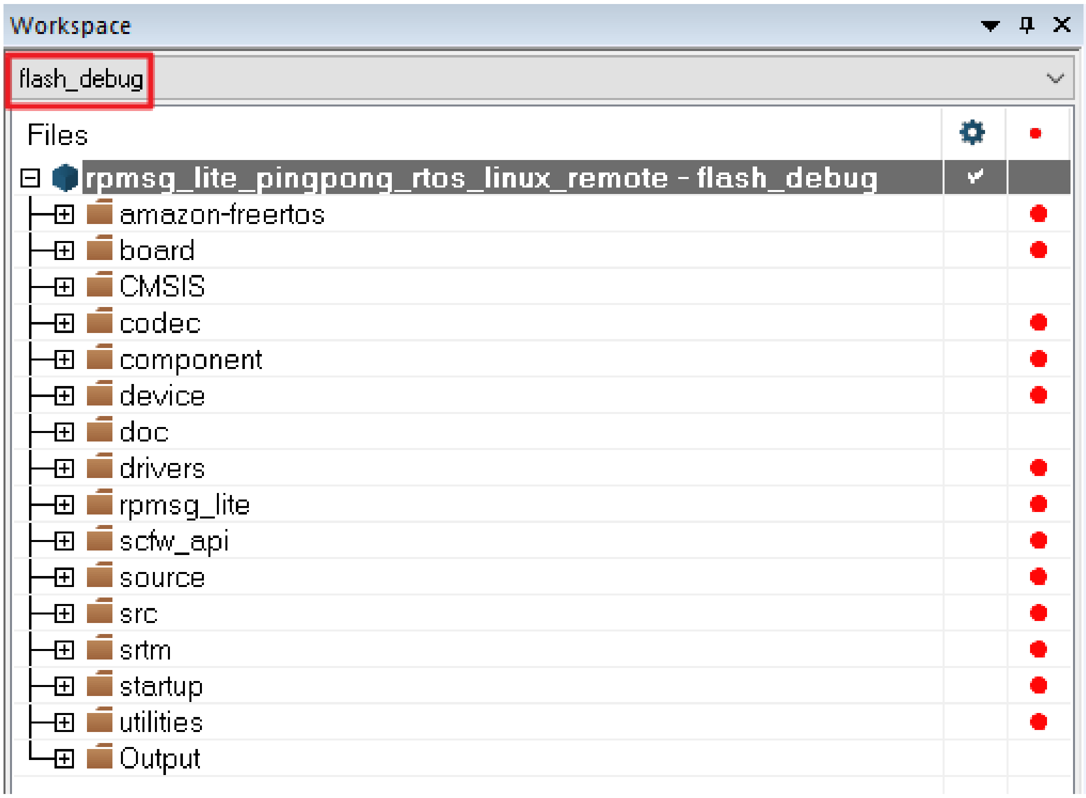

Figure 18. Demo build target selection

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

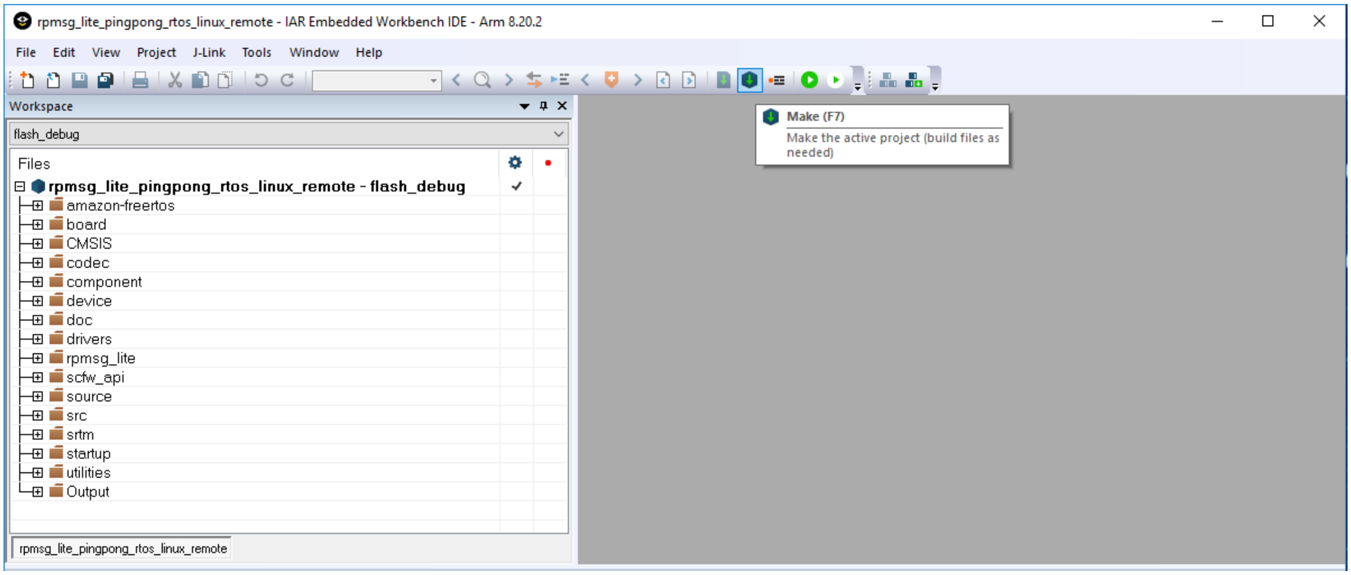

Figure 19. Building the demo application

4. The build completes without errors.
5. Step 1-4 are used for IAR toolchain to build the flash target M4 demo. For ARMGCC toolchain, simply run the `build_flash_debug` or `build_flash_release` script to build the flash target M4 demo.
6. Rename the generated `rpmsg_lite_pingpong_rtos_linux_remote.bin` to `m4_image.bin`, then copy it to the `mkimage` tool under `imx-mkimg/iMX8QX`.
7. There are two targets to generate `flash.bin` which contains the XIP M4 target in `imx-mkimage`:
   - `flash_m4_xip`: To generate a `flash.bin` which only contains the M4 XIP image.
   - `flash_linux_m4_xip`: To generate a `flash.bin` which contains both M4 XIP and U-Boot.

   Use `make SOC=iMX8QX REV=C0 flash_m4_xip` or `make SOC=iMX8QX REV=C0 flash_linux_m4_xip` to generate the desired `flash.bin`.
8. Use `make SOC=iMX8QX flash_flexspi` to generate a `flash.bin` which contains flexspi U-Boot. Rename this to `flash_uboot.bin` for future use.

8.3 Run an example application

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

1. Connect the development platform to your PC via USB cable between the J11 USB DEBUG connector and the PC. It provides console output while using UUU.
2. Connect the J10 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

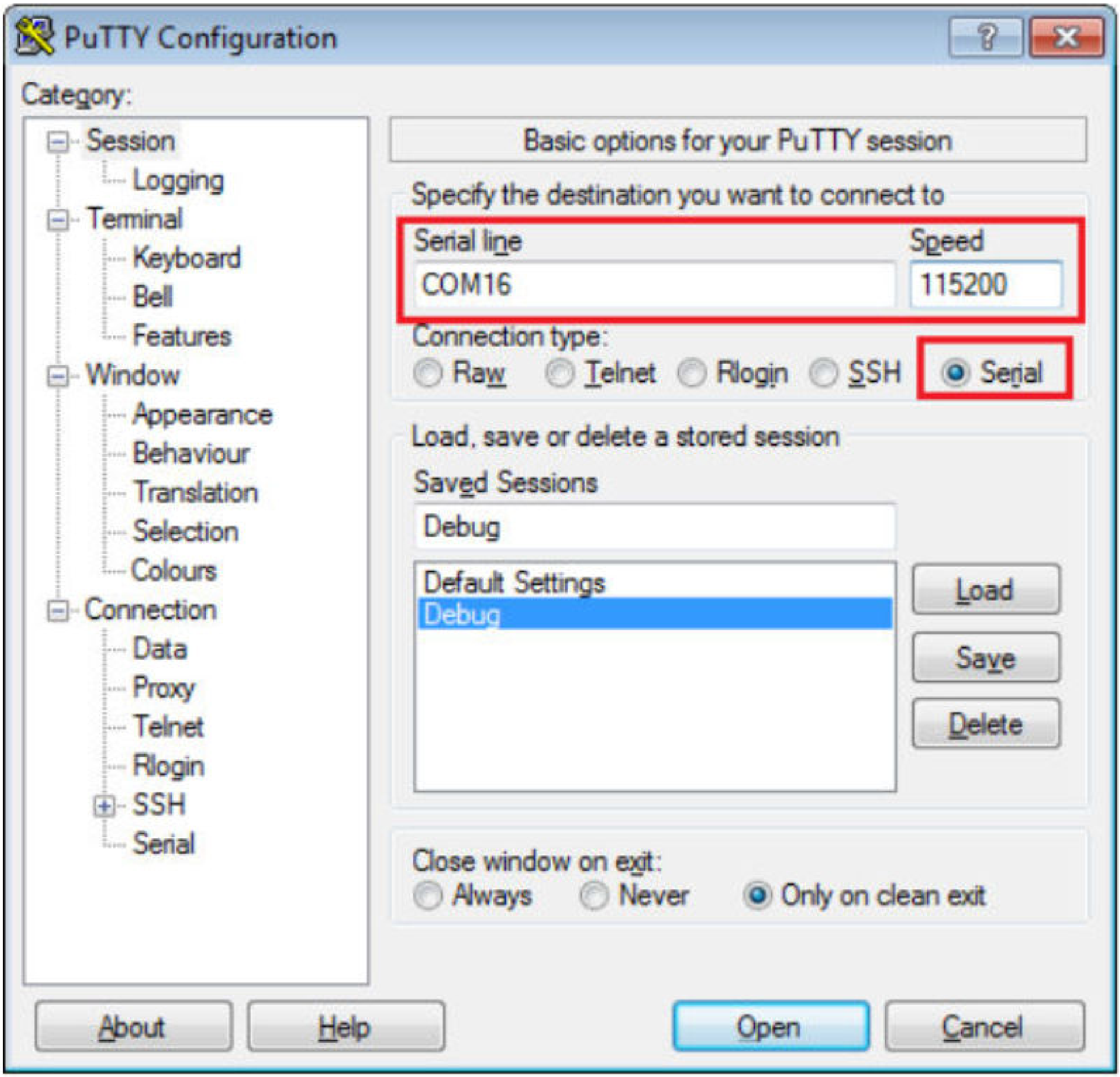

Figure 20. Terminal (PuTTY) configuration

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

   The UUU puts the platform into fast boot mode and automatically flashes the target bootloader to QSPI. The command line and fast boot console is shown on the following figure:

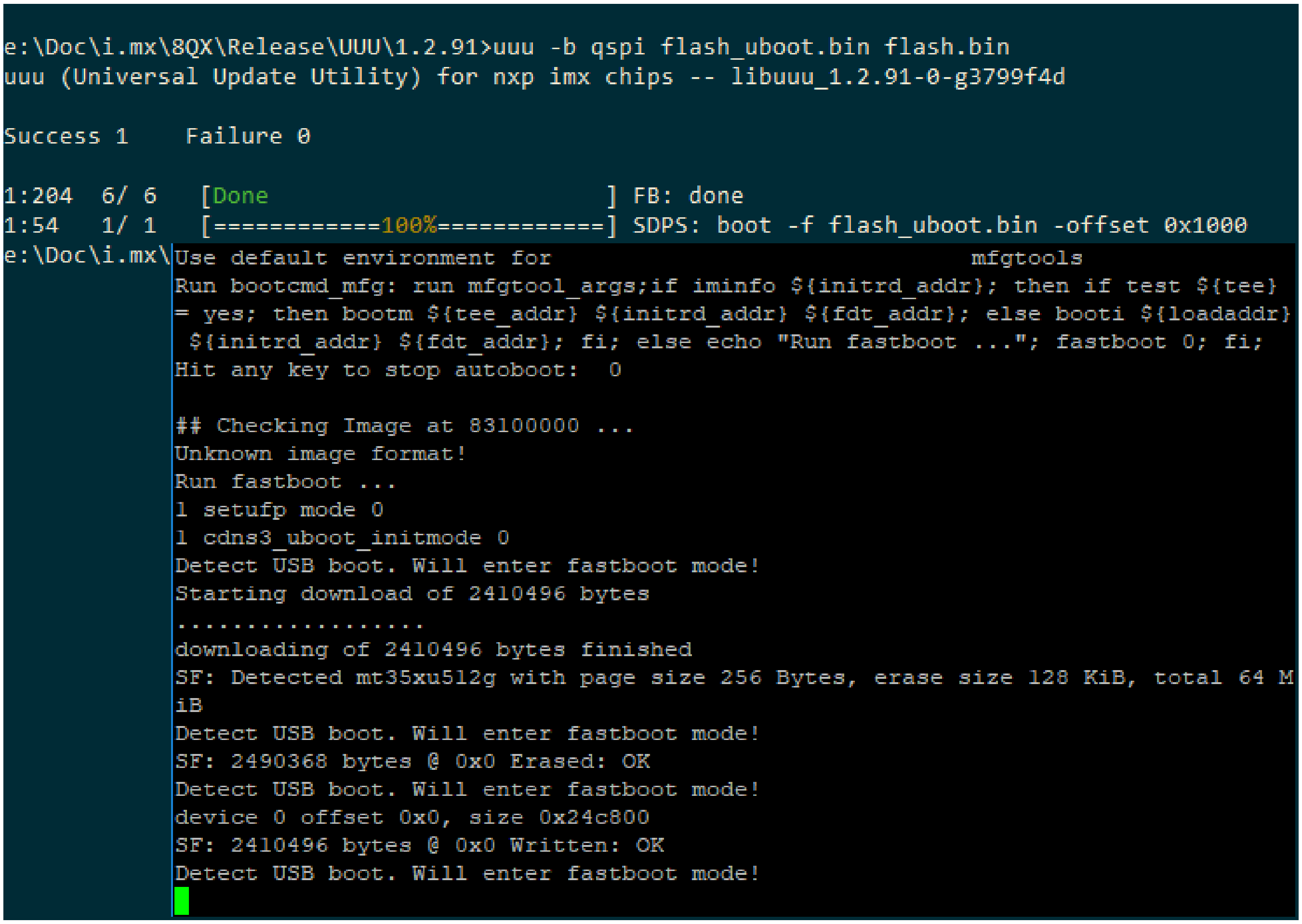

Figure 21. Command line and fast boot console output when executing UUU

In this example, the flash.bin is generated using the `flash_linux_m4_xip` target, which contains both M4 XIP and U-Boot.

5. Then, power off the board, change the boot mode to QSPI[b'0110], and power on the board again. The 2 UART consoles display the U-Boot and M4 demo output respectively.

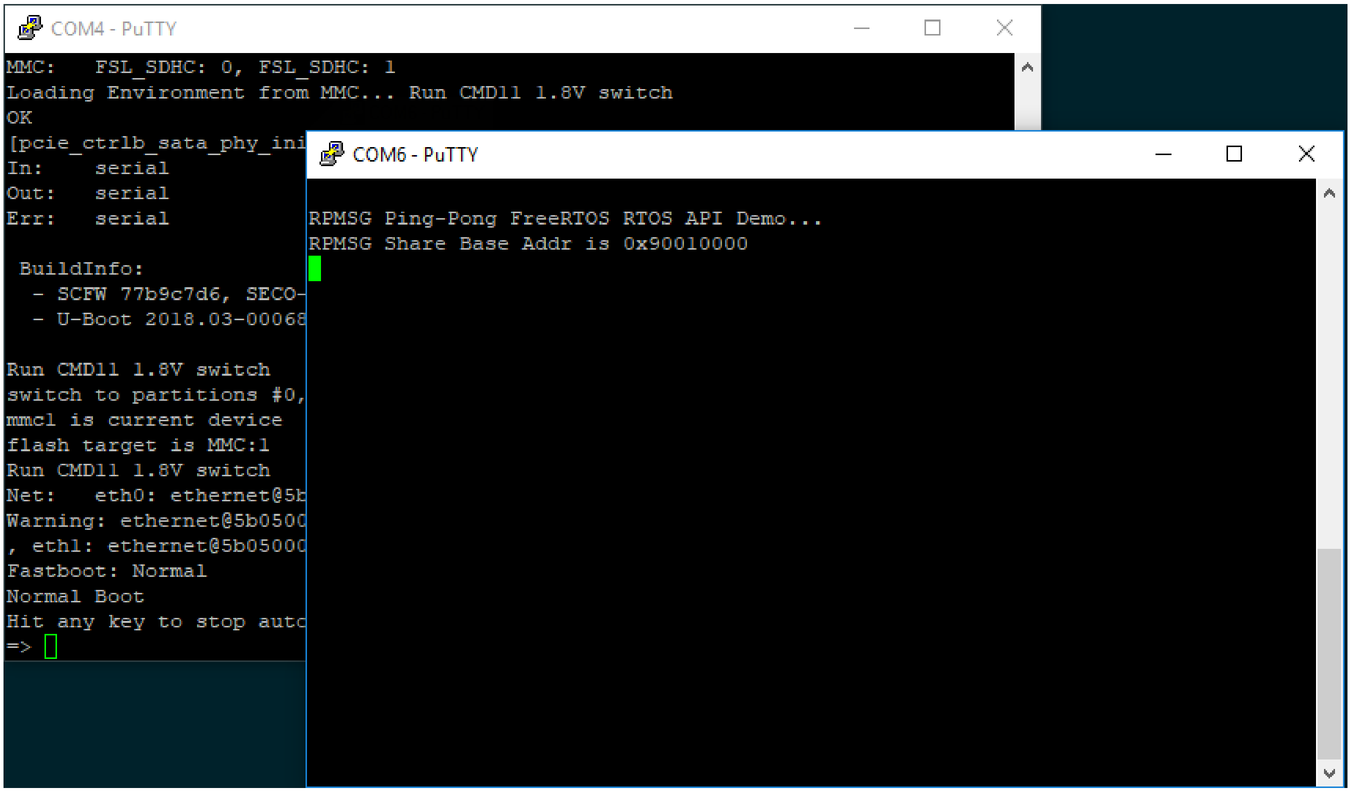

Figure 22. Console output from QSPI Boot

Appendix A Appendix A - How to determine COM port

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

Linux:

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

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

There are 2 Ports. The first is the Cortex-A debug console, and the second is for the CM4 debug console.

Windows:

1. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system Start menu and typing “Device Manager” in the search bar, as shown below:

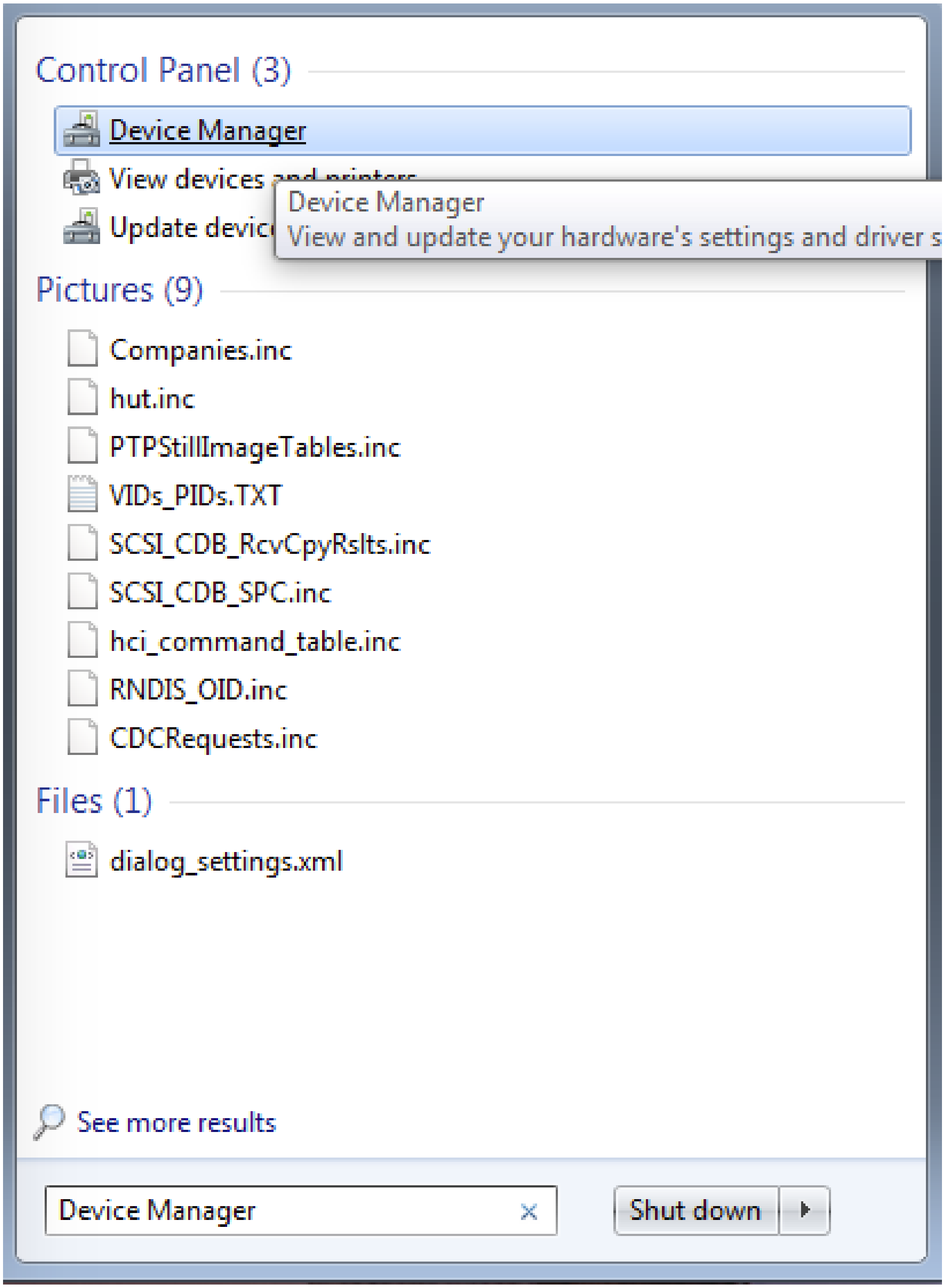

Figure A-1. Device manager

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

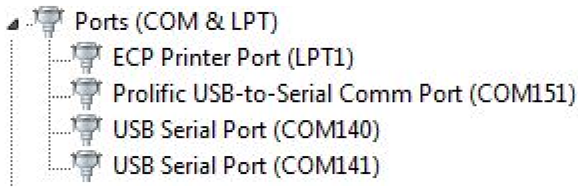

Figure A-2. USB-UART interface

There will be 4 Ports. The first is the Cortex-A debug console, and the second is for the CM4 debug console.

Appendix B Appendix C - Host Setup

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

Linux:

- Cmake

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

Windows:

- MinGW

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

1. Download the latest MinGW mingw-get-setup installer from 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 cannot contain any spaces.

3. Ensure that the “mingw32-base” and “msys-base” are selected under Basic Setup.

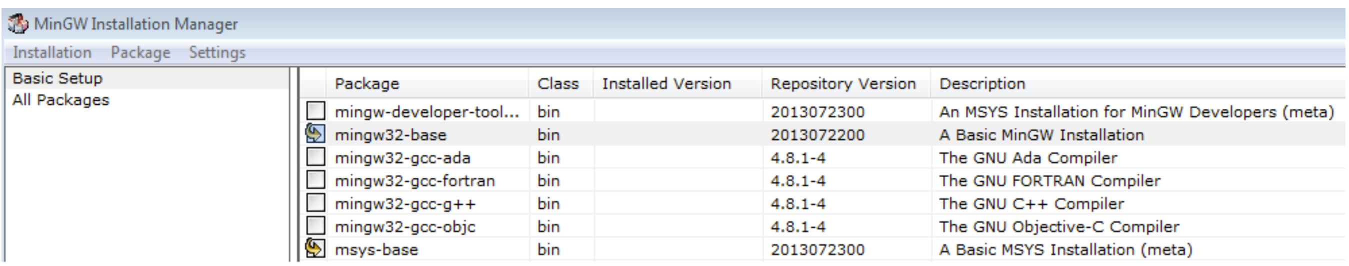

Figure B-1. Setup MinGW and MSYS

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

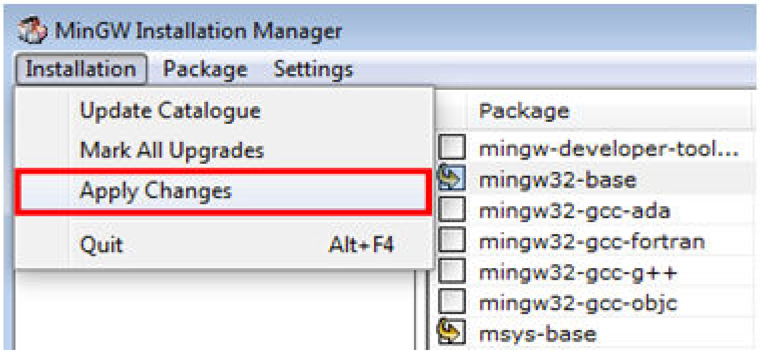

Figure B-2. Complete MinGW and MSYS installation

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 shown below. 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.

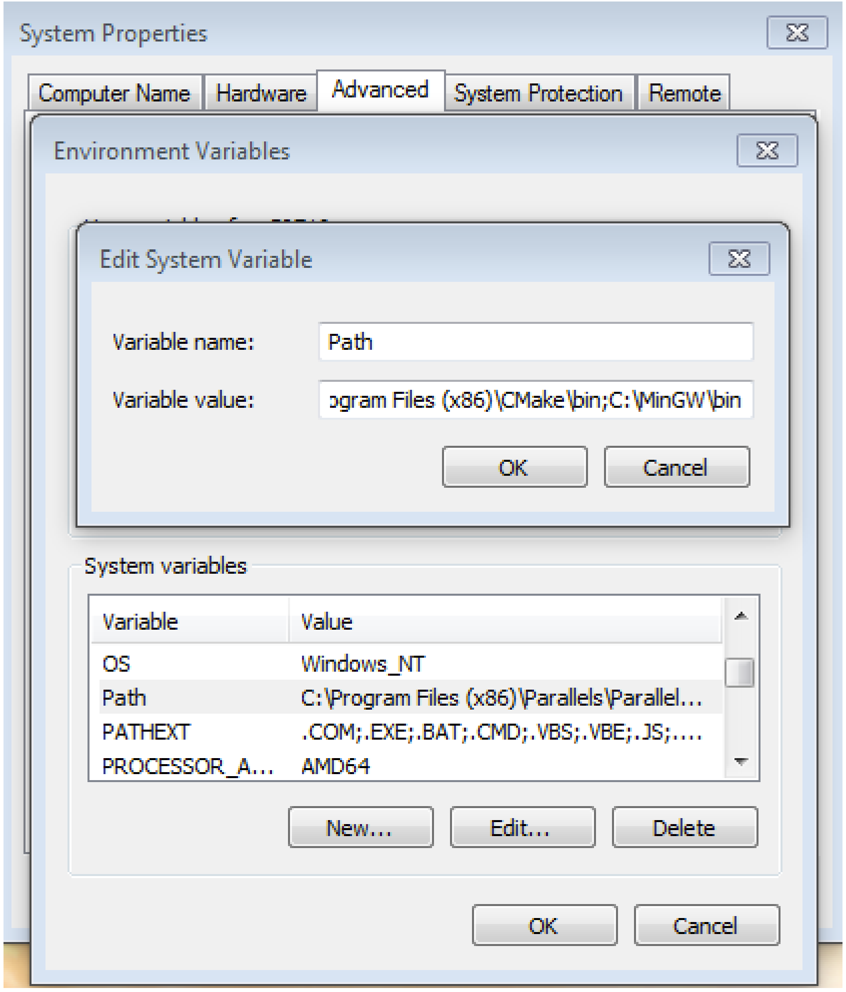

Figure B-3. Add Path to systems environment

- Cmake

1. Download CMake 3.0.x from www.cmake.org/cmake/resources/software.html.
2. Install CMake, ensuring that the option "Add CMake to system PATH" is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.

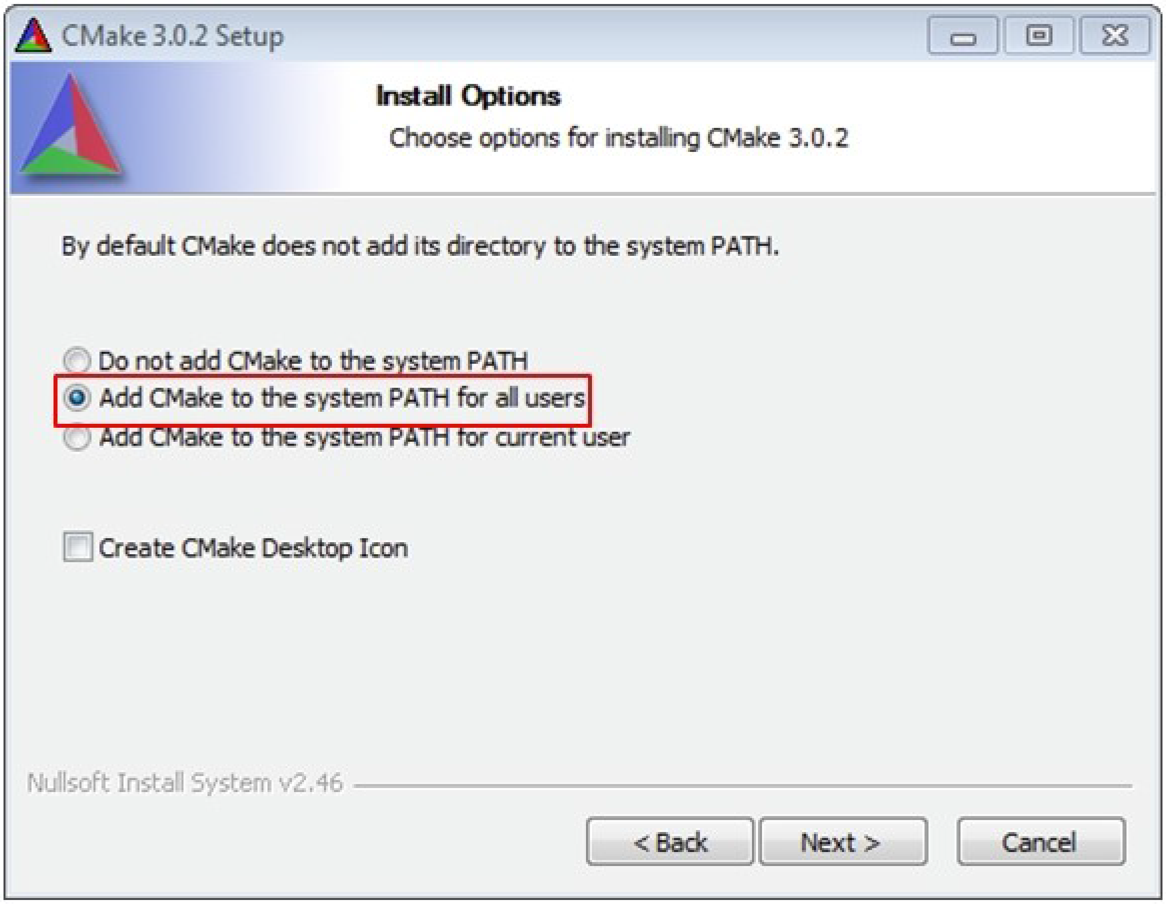

Figure B-4. Install Cmake

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

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MCUXSDKIMX8QXGSUG
Revision 0, 01/2020