Document Number: MCUXSDKIMX8MMGSUG
Rev. 0, 10/2019

Getting Started with MCUXpresso SDK for i.MX 8M Mini

Contents

1 Overview

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

For supported toolchain versions, see the *MCUXpresso SDK Release Notes Supporting i.MX 8M Mini* (document MCUXSDKIMX8MMRN).

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

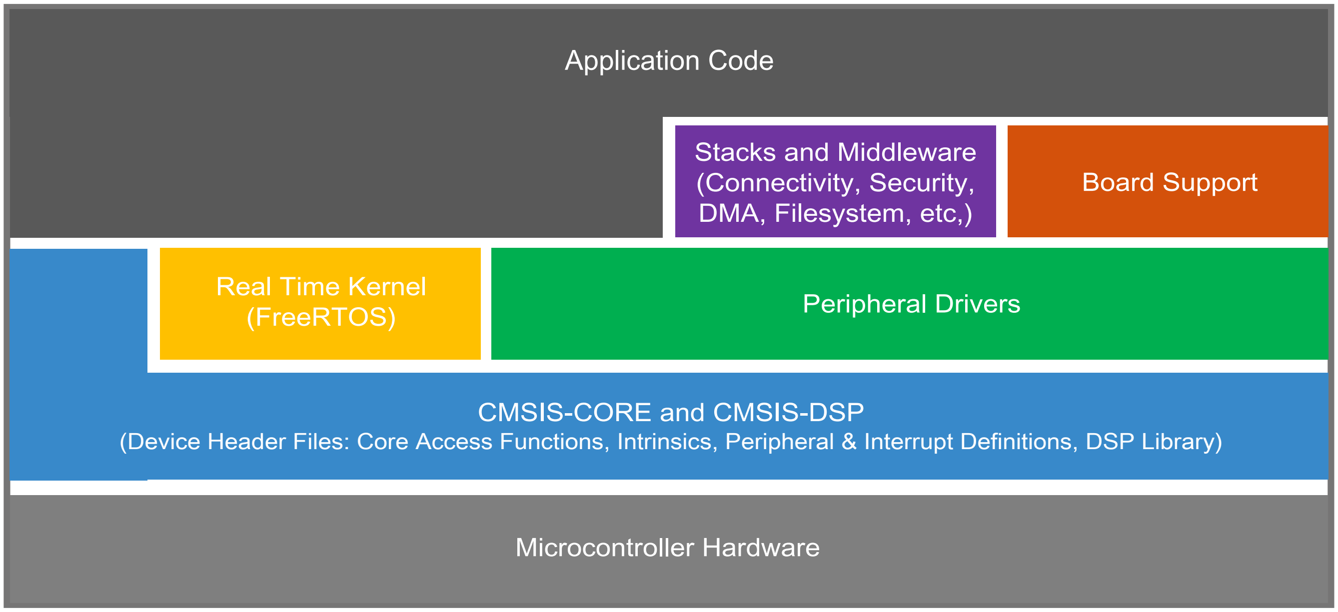


Figure 1. MCUXpresso SDK layers

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.

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* (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:

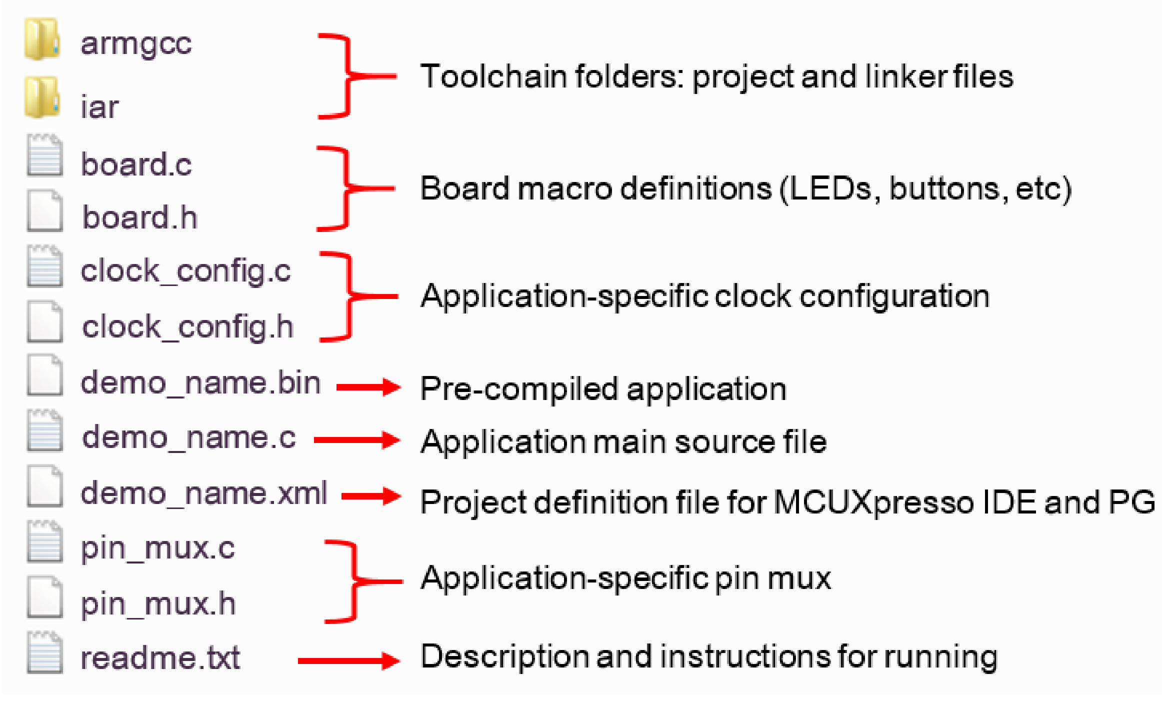


Figure 2. Application folder structure

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 things.
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU.
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code. Vector table definitions are here.
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications.

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the `rtos` folder. Again, 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 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 release supports building and debugging with the toolchains listed below.

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.

4 Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the i.MX 8MMini EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

4.1 Build an example application

The following steps helps you 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 8MMini EVK hardware platform as an example, the `hello_world` workspace is located in

   <install_dir>/boards/evkmimx8mm/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.

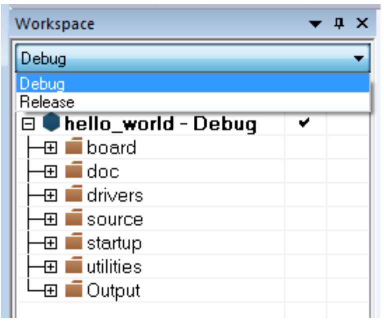


Figure 3. Demo build target selection

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

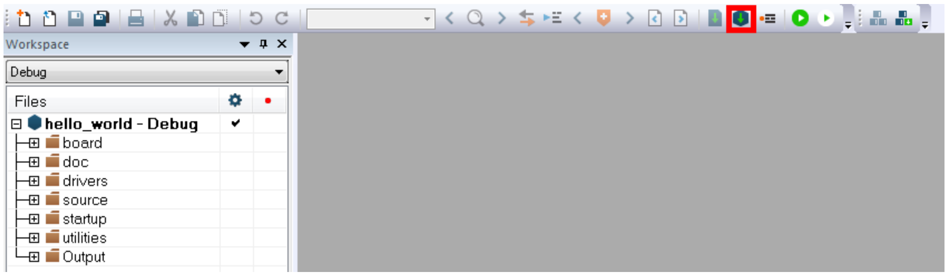


Figure 4. Build the demo application

4. The build completes without errors.

4.2 Run an example application

To download and run the application, perform these steps:

1. This board supports the J-Link PLUS debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from http://www.segger.com/downloads/jlink/.
2. Connect the development platform to your PC via USB cable between the USB-UART MICRO USB connector and the PC USB connector, then connect V power supply and J-Link Plus to the device.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
    a. 115200 baud rate
    b. No parity
    c. 8 data bits
    d. 1 stop bit

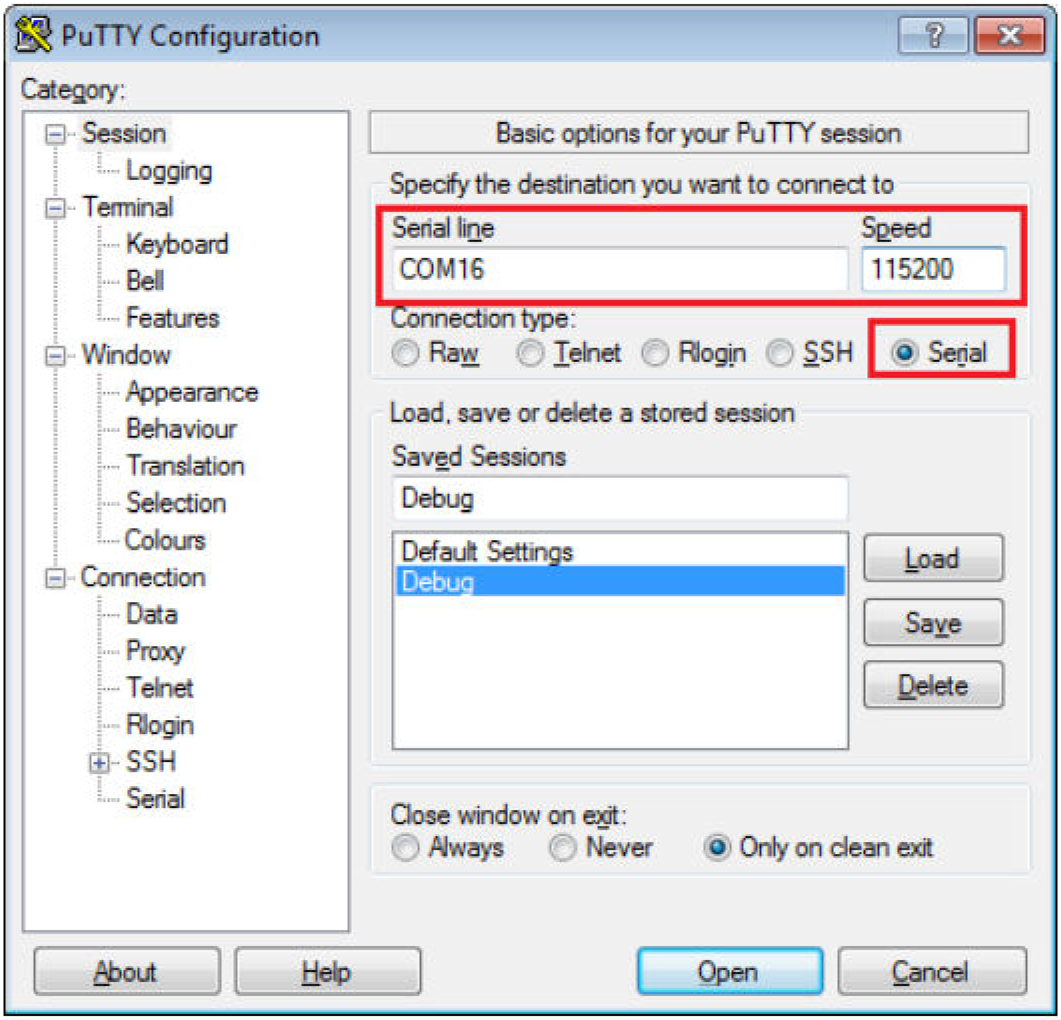


Figure 5. Terminal (PuTTY) configuration

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

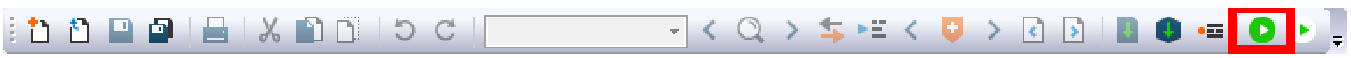


Figure 6. Download and Debug button

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

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

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

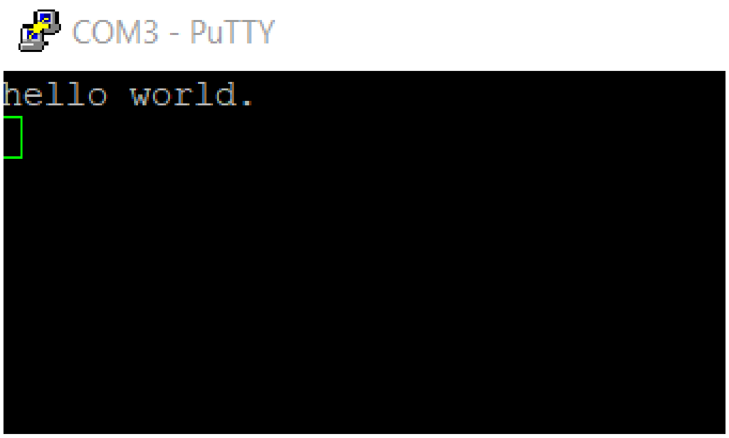


Figure 9. Text display of the `hello_world` demo

NOTE

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

5 Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the MCUXpresso SDK. The hello_world demo application targeted for i.MX 8M Mini is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

5.1 Linux OS host

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

5.1.1 Set up toolchain

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

5.1.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from 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 B for Linux OS before compiling the application.

5.1.1.2 Add a new system environment variable for ARMGCC_DIR

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

```
$ export ARMGCC_DIR=/work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major
```

```
$ export PATH= $PATH:/work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major/bin
```

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/evkmimx8mm/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-7-2017-q4-major
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler: /work/platforms/tmp/gcc-arm-none-eabi-7-2017-q4-major/bin/arm-none-
eabi-gcc
-- Configuring done
-- Generating done
-- Build files have been written to:
```

```
/work/platforms/tmp/nxp/SDK_2.4.0_EVK-MIMX8MM/boards/evkmimx8mm/demo_apps/hello_world/
armgcc


Scanning dependencies of target hello_world.elf




[ 6%] Building C object CMakeFiles/hello_world.elf.dir/work/platforms/tmp/nxp/
SDK_2.4.0_EVK-MIMX8MM/boards/evkmimx8mm/demo_apps/hello_world/hello_world.c.obj


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

5.1.3 Run an example application

This section describes steps to run a demo application using J-Link GDB Server application.

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

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

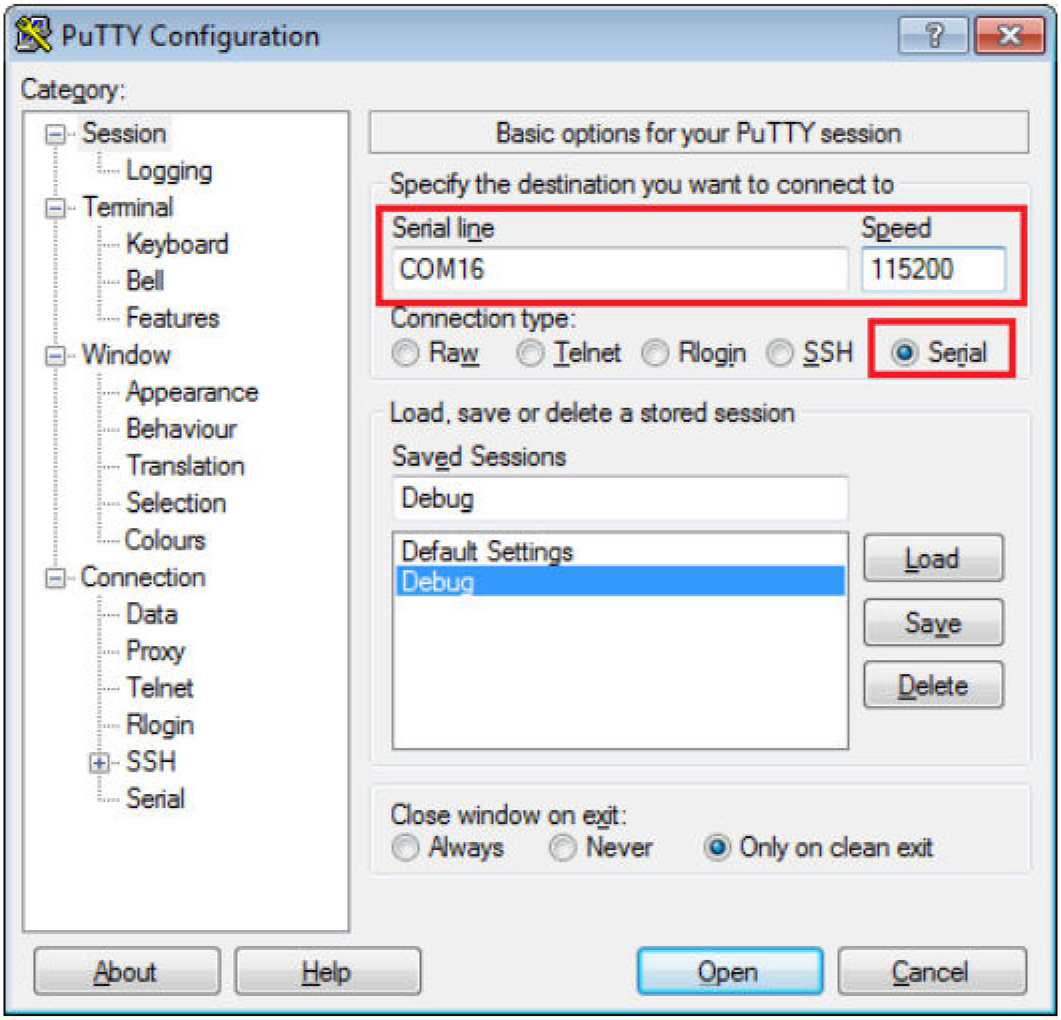


Figure 10. Terminal (PuTTY) configurations

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 MIMX8MM6_M4 device:

```
$ JLinkGDBServer -if JTAG -device
SEGGER J-Link GDB Server  Command Line Version
JLinkARM.dll
Command line: -if JTAG -device MIMX8MM6_M4
-----GDB Server start settings-----
GDBInit file: none
GDB Server Listening port: 2331
SWO raw output listening port: 2332
Terminal I/O port: 2333
Accept remote connection: yes
< -- Skipping lines -- >
Target connection timeout: 0 ms
------J-Link related settings------
J-Link Host interface: USB
J-Link script: none
J-Link settings file: none
------Target related settings------
Target device:
Target interface: JTAG
Target interface speed: 1000 kHz
Target endian: little
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Feb 2 2018 18:12:40
```

```
Hardware: V10.10
S/N: 600109545
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 1.82 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 (Cortex-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/evkmimx8mm/demo_apps/hello_world/armgcc/debug

5. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello_world.elf...
(gdb)
```

6. Connect to the GDB server and load the binary by running the following commands:
   a. "target remote localhost:2331"
   b. "monitor reset"
   c. "monitor halt"
   d. "load"

```
(gdb) target remote localhost:2331
Remote debugging using localhost:2331

(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load

Loading section .interrupts, size 0x240 lma 0x1ffe0000
Loading section .text, size 0x3858 lma 0x1ffe0240
Loading section .ARM, size 0x8 lma 0x1ffe3a98
Loading section .init_array, size 0x4 lma 0x1ffe3aa0
Loading section .fini_array, size 0x4 lma 0x1ffe3aa4
Loading section .data, size 0x64 lma 0x1ffe3aa8
Start address 0x1ffe02f4, load size 15116
Transfer rate: 81 KB/sec, 2519 bytes/write.
(gdb)
```

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

```
(gdb) monitor go
```

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

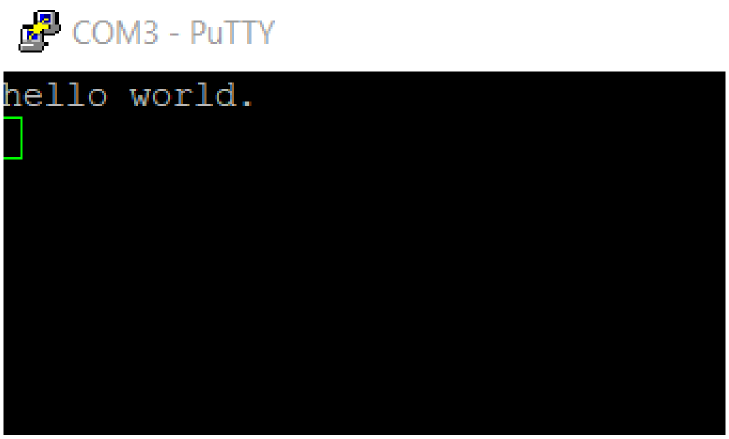


Figure 11. Text display of the hello_world demo

5.2 Windows OS host

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

5.2.1 Set up toolchain

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

5.2.1.1 Install GCC Arm Embedded tool chain

Download and run the installer from 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 MCUXSDKRN).

NOTE

See the *Host Setup* Section in Appendix B 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”.

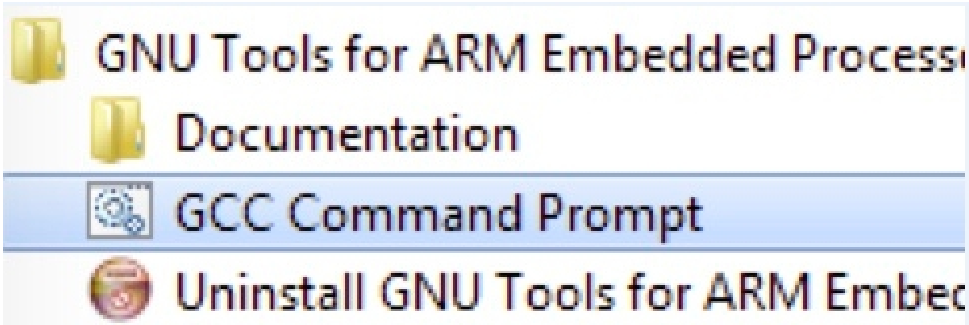
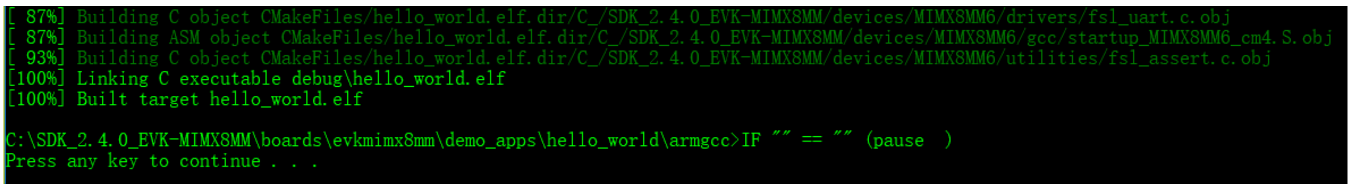


Figure 12. Launch command prompt

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/evkmimx8mm/demo_apps/hello_world/armgcc*
3. Type “build_debug.bat” on the command line or double click on the "build_debug.bat" file in Windows Explorer to perform the build. The output is shown in this figure:



```
[ 87%] Building C object CMakeFiles/hello_world.elf.dir/C_/SDK_2.4.0_EVK-MIMX8MM/devices/MIMX8MM6/drivers/fsl_uart.c.obj
[ 87%] Building ASM object CMakeFiles/hello_world.elf.dir/C_/SDK_2.4.0_EVK-MIMX8MM/devices/MIMX8MM6/gcc/startup_MIMX8MM6_cm4.S.obj
[ 93%] Building C object CMakeFiles/hello_world.elf.dir/C_/SDK_2.4.0_EVK-MIMX8MM/devices/MIMX8MM6/utilities/fsl_assert.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf

C:\SDK_2.4.0_EVK-MIMX8MM\boards\evkmimx8mm\demo_apps\hello_world\armgcc>IF "" == "" (pause  )
Press any key to continue . . .
```

Figure 13. hello_world demo build successful

5.2.3 Run an example application

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

- You have a standalone J-Link pod that is connected to the debug interface of your board. Make sure the Segger J-Link software supporting patch, 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

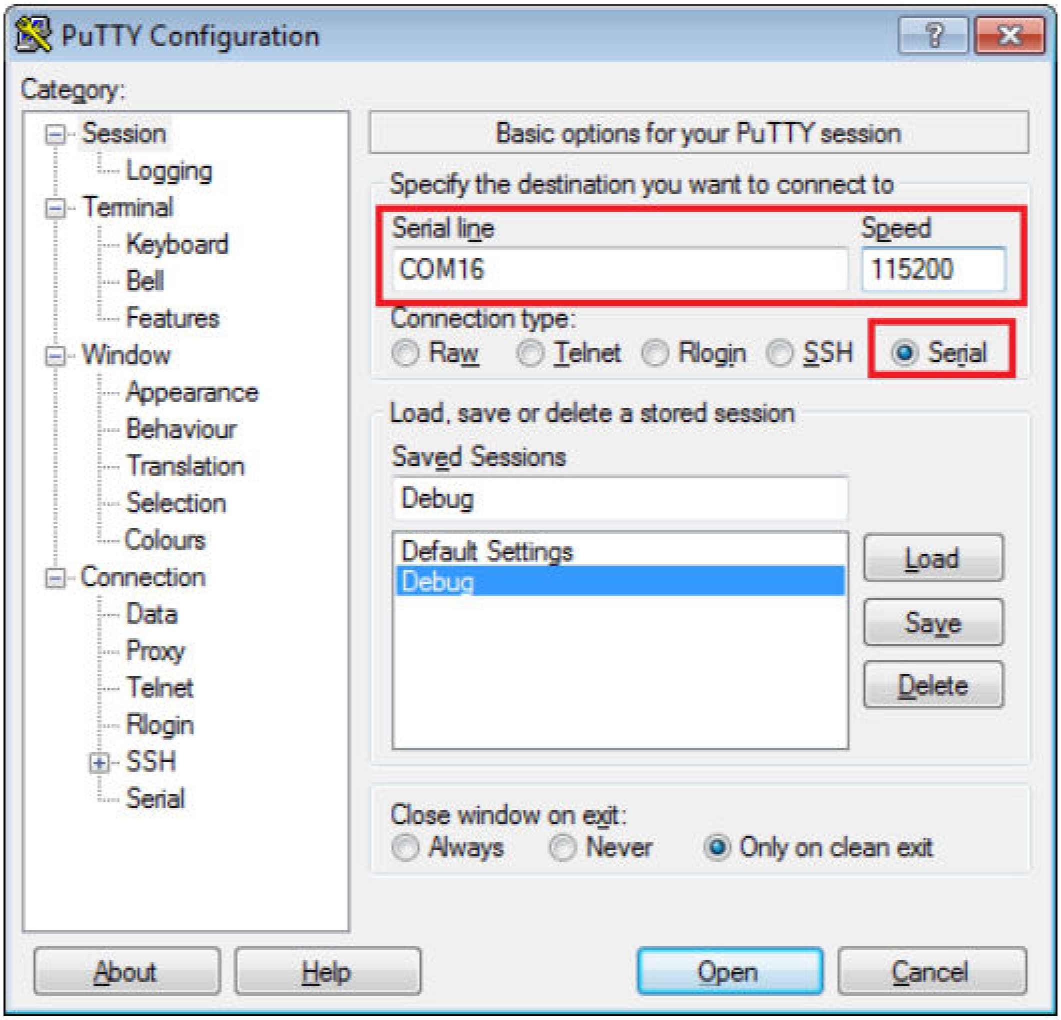


Figure 14. Terminal (PuTTY) configurations

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 MIMX8MM6_M4.
5. After it is connected, the screen should resemble this figure:

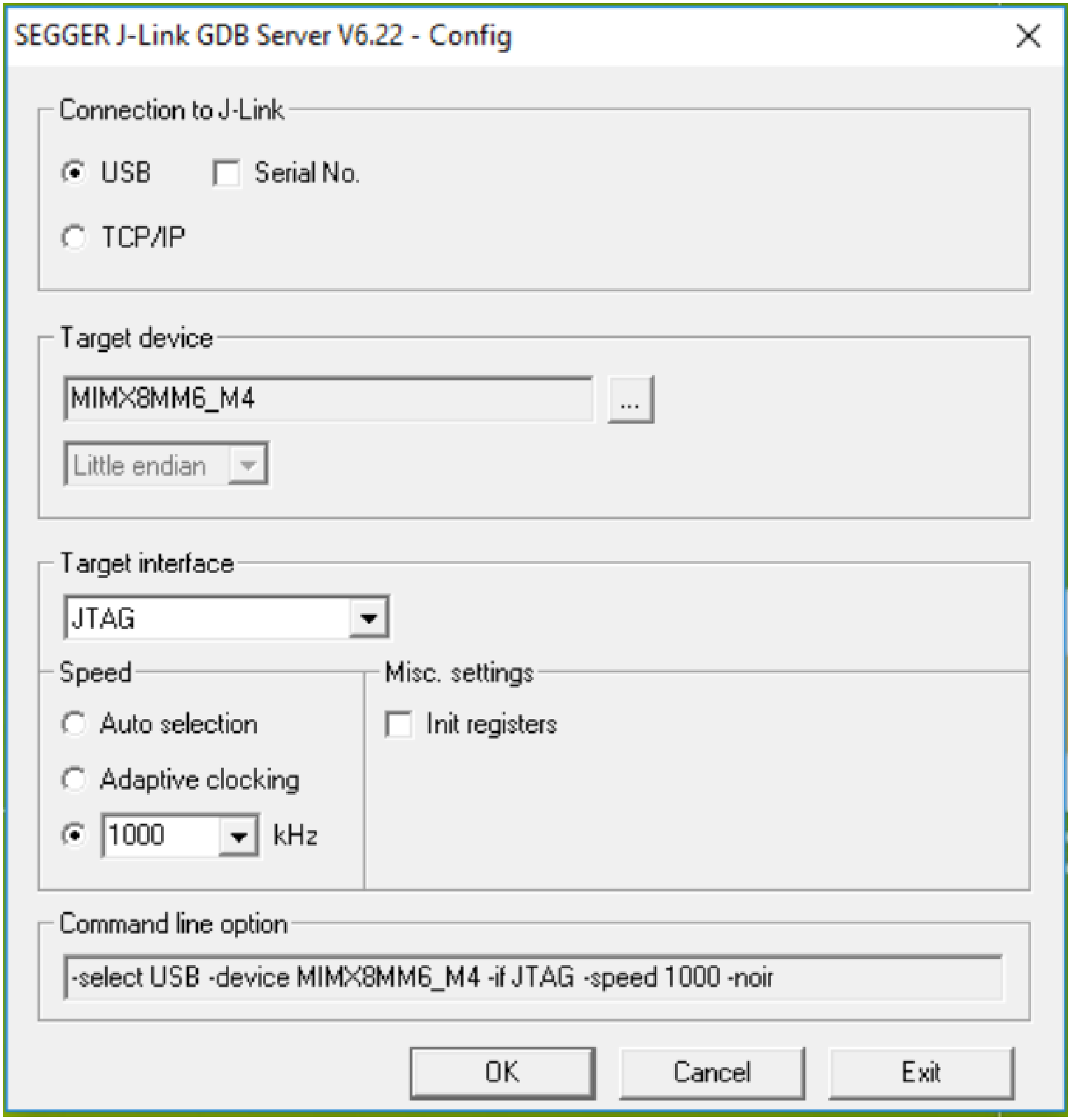


Figure 15. SEGGER J-Link GDB server configuration

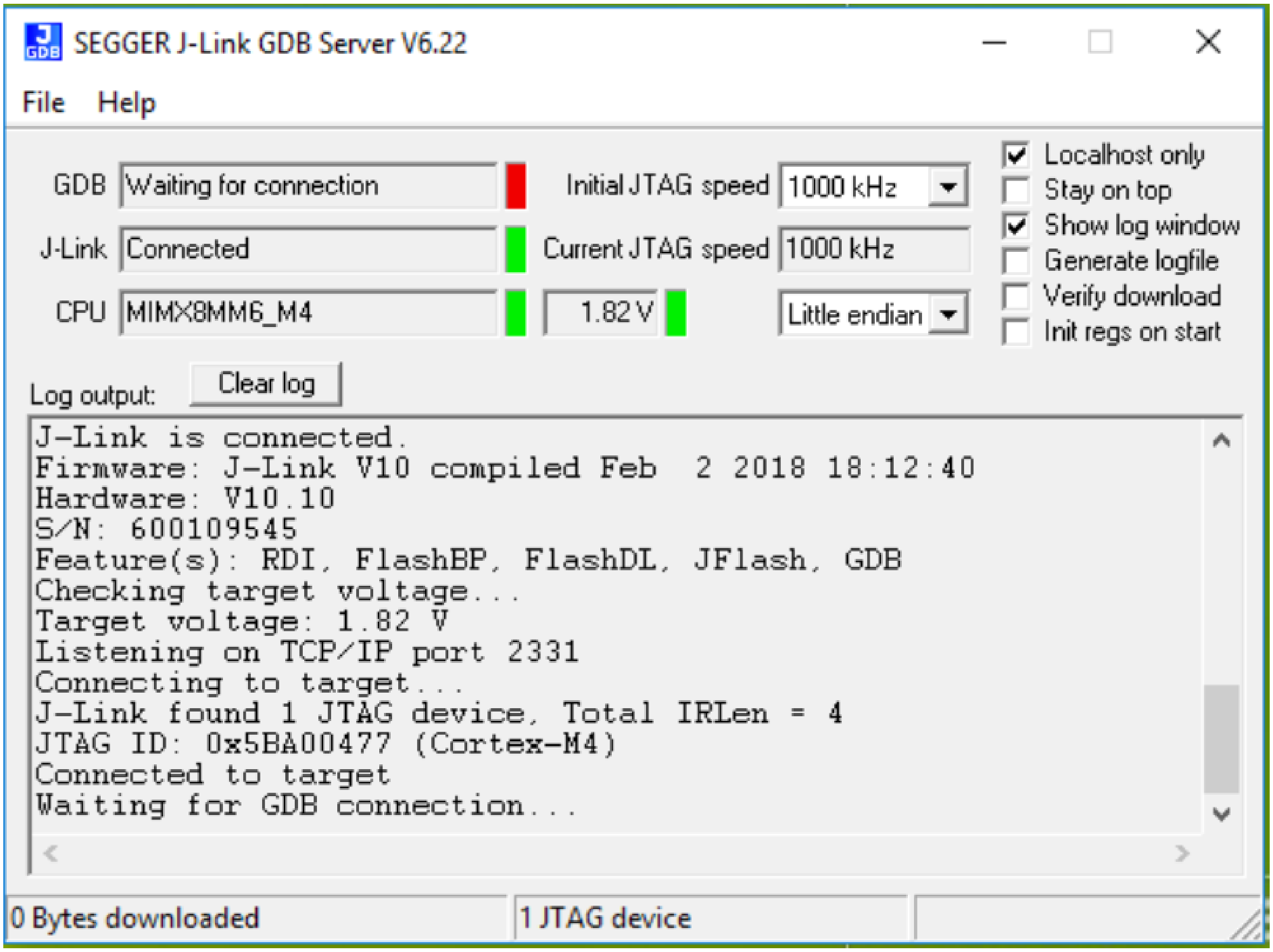


Figure 16. SEGGER J-Link GDB server screen after successful connection

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”.

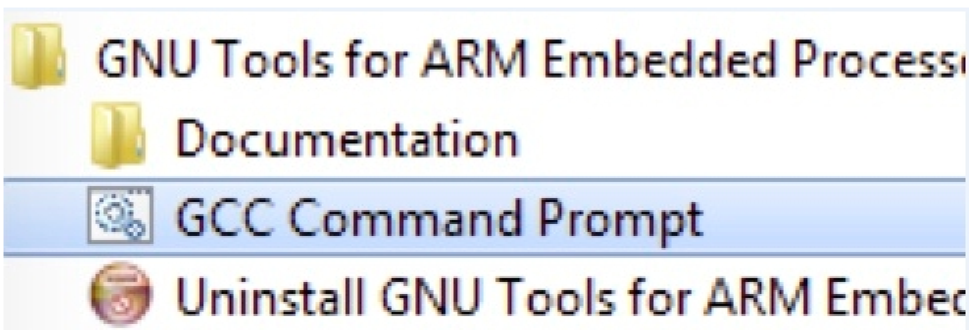


Figure 17. Launch command prompt

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/evkmimx8mm/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”.

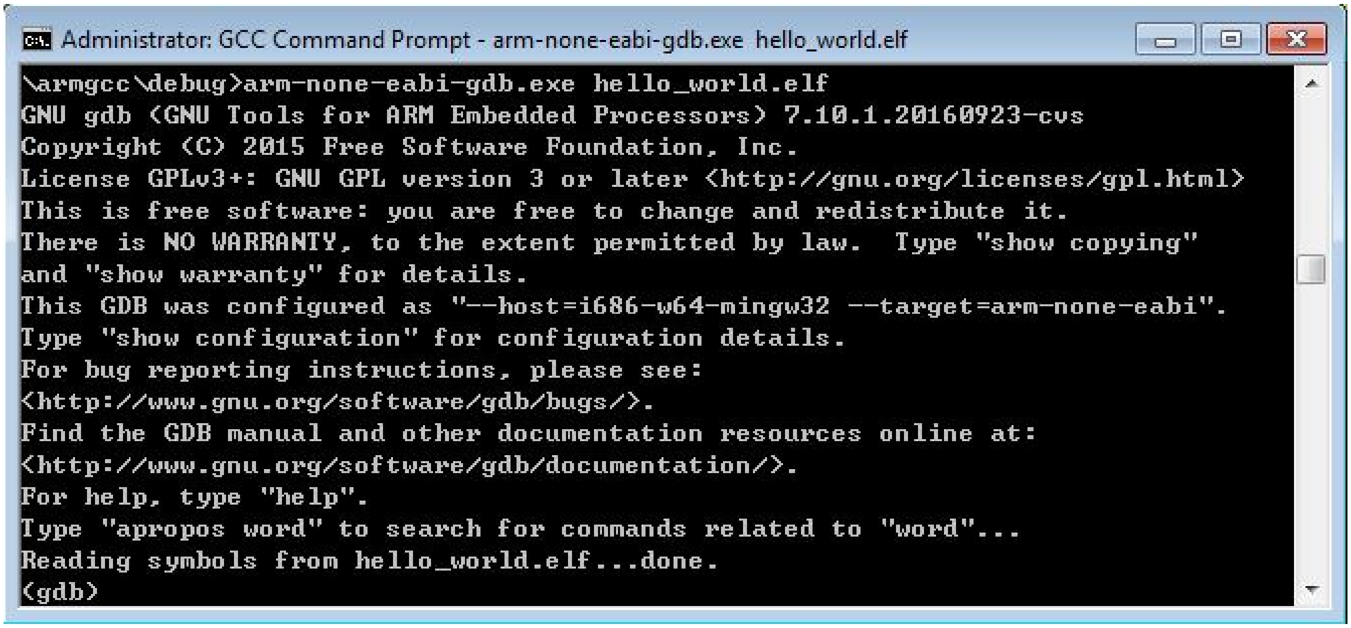


Figure 18. Run arm-none-eabi-gdb

9. Run these commands:
   a. "target remote localhost:2331"
   b. "monitor reset"
   c. "monitor halt"
   d. "load"
10. The application is now downloaded and halted at the reset vector. Execute the “monitor go” command to start the demo application.

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

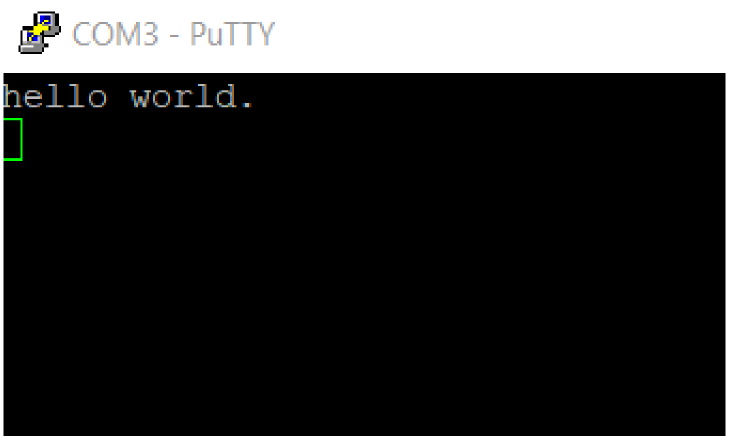


Figure 19. Text display of the hello_world demo

6 Running an application by U-Boot

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

1. Connect the **DEBUG UART** slot on the board to your PC through the USB cable. The Windows® OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.
2. On Windows OS, open the device manager, find **USB serial Port** in **Ports (COM and LPT)**. Assume that the ports are COM9 and COM10. One port is for the debug message from the Cortex®-A53 and the other is for the Cortex®-M7. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name `/dev/ttyUSB*` to determine your debug port. Similar to Windows OS, opening both is beneficial for development.

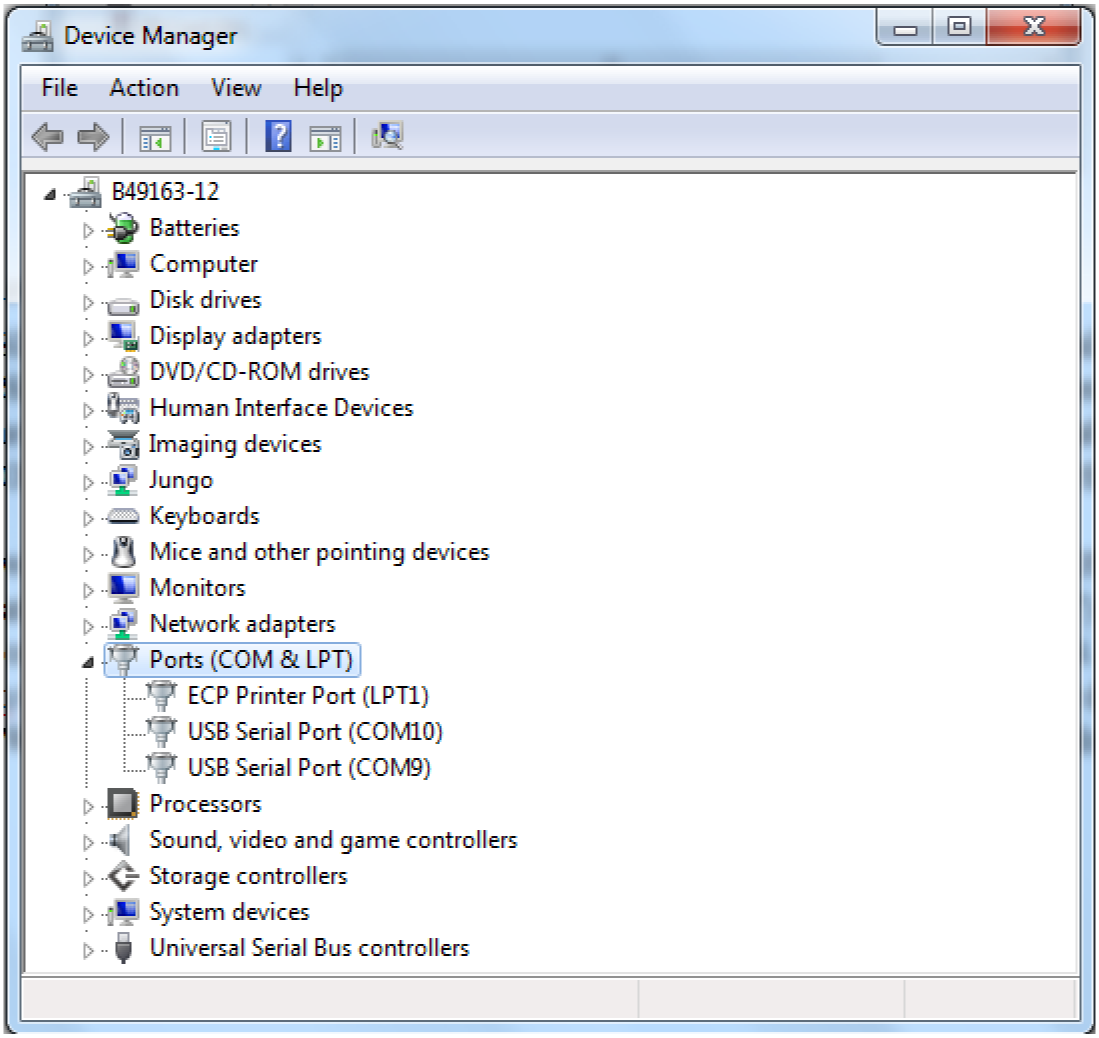


Figure 20. Determining the COM Port of target board

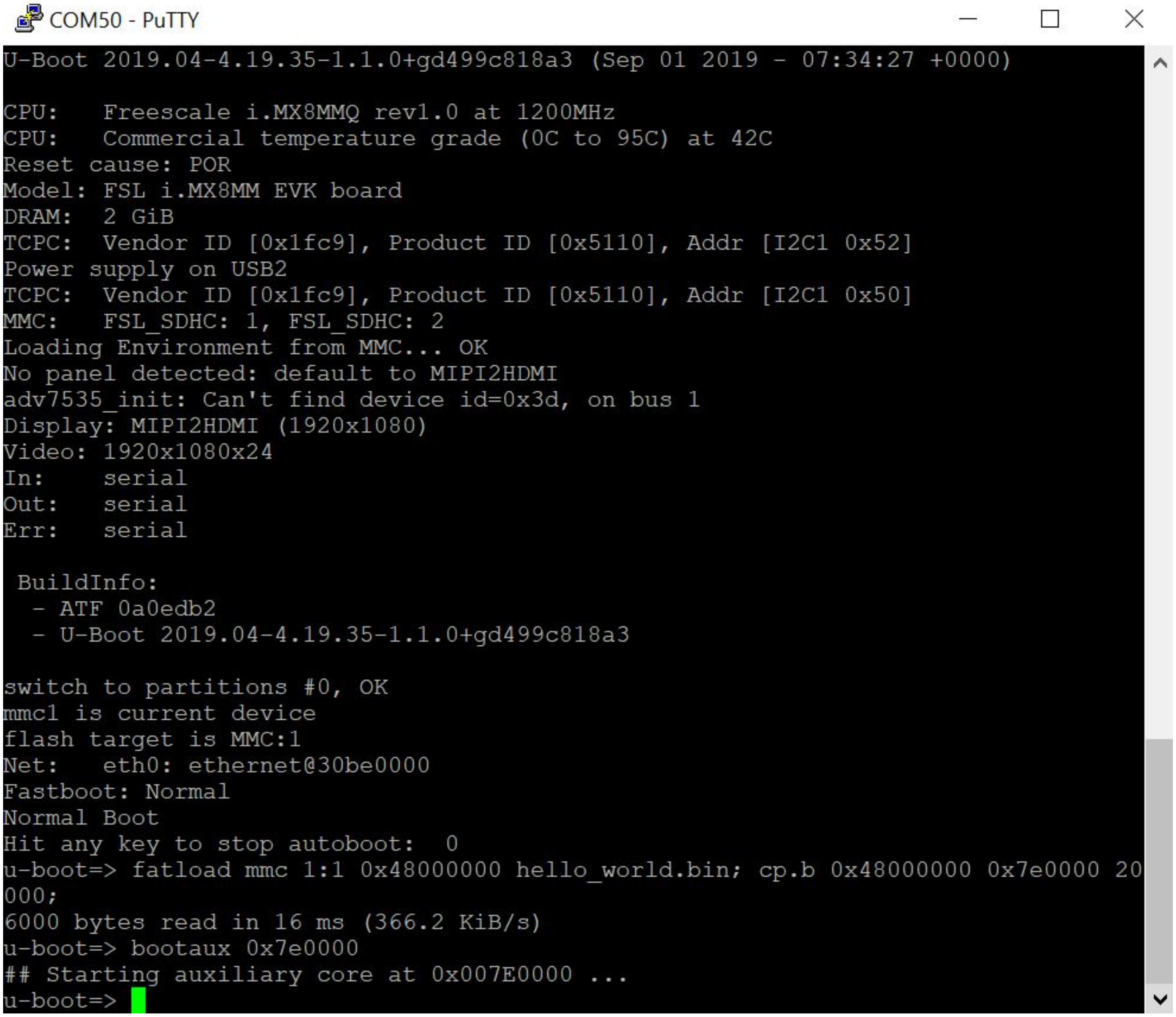
3. Build the application (for example, `hello_world`) to get the bin file (`hello_world.bin`).
4. Prepare an SD card with the prebuilt U-Boot image and copy bin file (`hello_world.bin`) into the SD card. Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dipswitch configuration.
5. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.

6. Power on the board and hit any key to stop autoboot in the terminals, then enter to U-Boot command line mode. You can then write the image and run it from TCM or DRAM with the following commands:
   a. If the `hello_world.bin` is made from the debug/release target, which means the binary file will run at TCM, use the following commands to boot:
      - `fatload mmc 1:1 0x48000000 hello_world.bin; cp.b 0x48000000 0x7e0000 20000;`
      - `bootaux 0x7e0000`
   b. If the `hello_world.bin` is made from the `ddr_debug`/`ddr_release` target, which means the binary file runs at DRAM, use the following commands:
      - `fatload mmc 1:1 0x80000000 hello_world.bin`
      - `dcache flush`
      - `bootaux 0x80000000`

NOTE

If the Linux OS kernel runs together with M4, make sure the correct dtb file is used. This dtb file reserves resources used by M4 and avoids the Linux kernel from configuring them. Use the following command in U-Boot before running the kernel:

```
setenv fdt_file 'fsl-imx8mm-evk-m4.dtb'
save
```



```
COM50 - PuTTY
U-Boot 2019.04-4.19.35-1.1.0+gd499c818a3 (Sep 01 2019 - 07:34:27 +0000)

CPU:   Freescale i.MX8MMQ rev1.0 at 1200MHz
CPU:   Commercial temperature grade (0C to 95C) at 42C
Reset cause: POR
Model: FSL i.MX8MM EVK board
DRAM:  2 GiB
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C1 0x52]
Power supply on USB2
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C1 0x50]
MMC:   FSL_SDHC: 1, FSL_SDHC: 2
Loading Environment from MMC... OK
No panel detected: default to MIPI2HDMI
adv7535_init: Can't find device id=0x3d, on bus 1
Display: MIPI2HDMI (1920x1080)
Video: 1920x1080x24
In:    serial
Out:   serial
Err:   serial

 BuildInfo:
  - ATF 0a0edb2
  - U-Boot 2019.04-4.19.35-1.1.0+gd499c818a3

switch to partitions #0, OK
mmc1 is current device
flash target is MMC:1
Net:   eth0: ethernet@30be0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot:  0
u-boot=> fatload mmc 1:1 0x48000000 hello_world.bin; cp.b 0x48000000 0x7e0000 20
000;
6000 bytes read in 16 ms (366.2 KiB/s)
u-boot=> bootaux 0x7e0000
## Starting auxiliary core at 0x007E0000 ...
u-boot=>
```

Figure 21. U-Boot command to run application on TCM

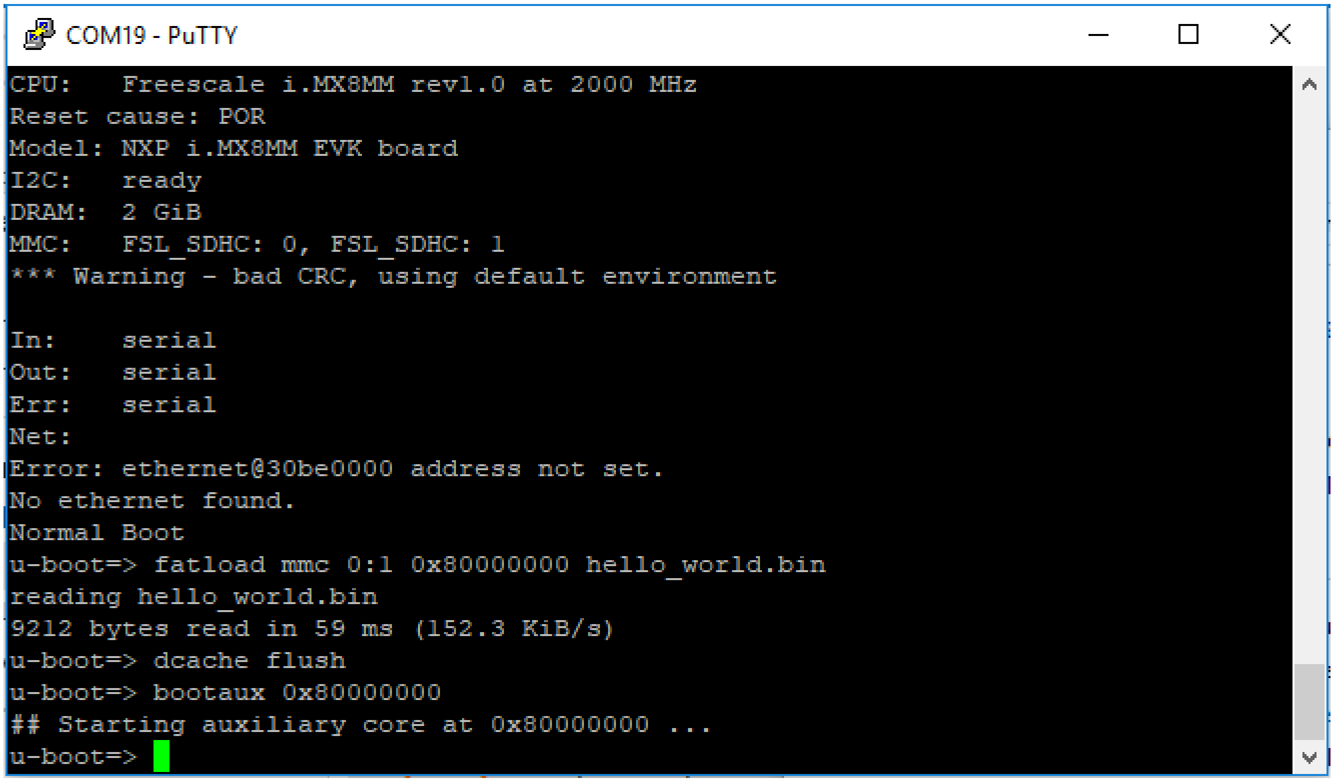


Figure 22. U-Boot command to run application on DRAM

7. Open another terminal application on the PC, such as PuTTY and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit
8. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.

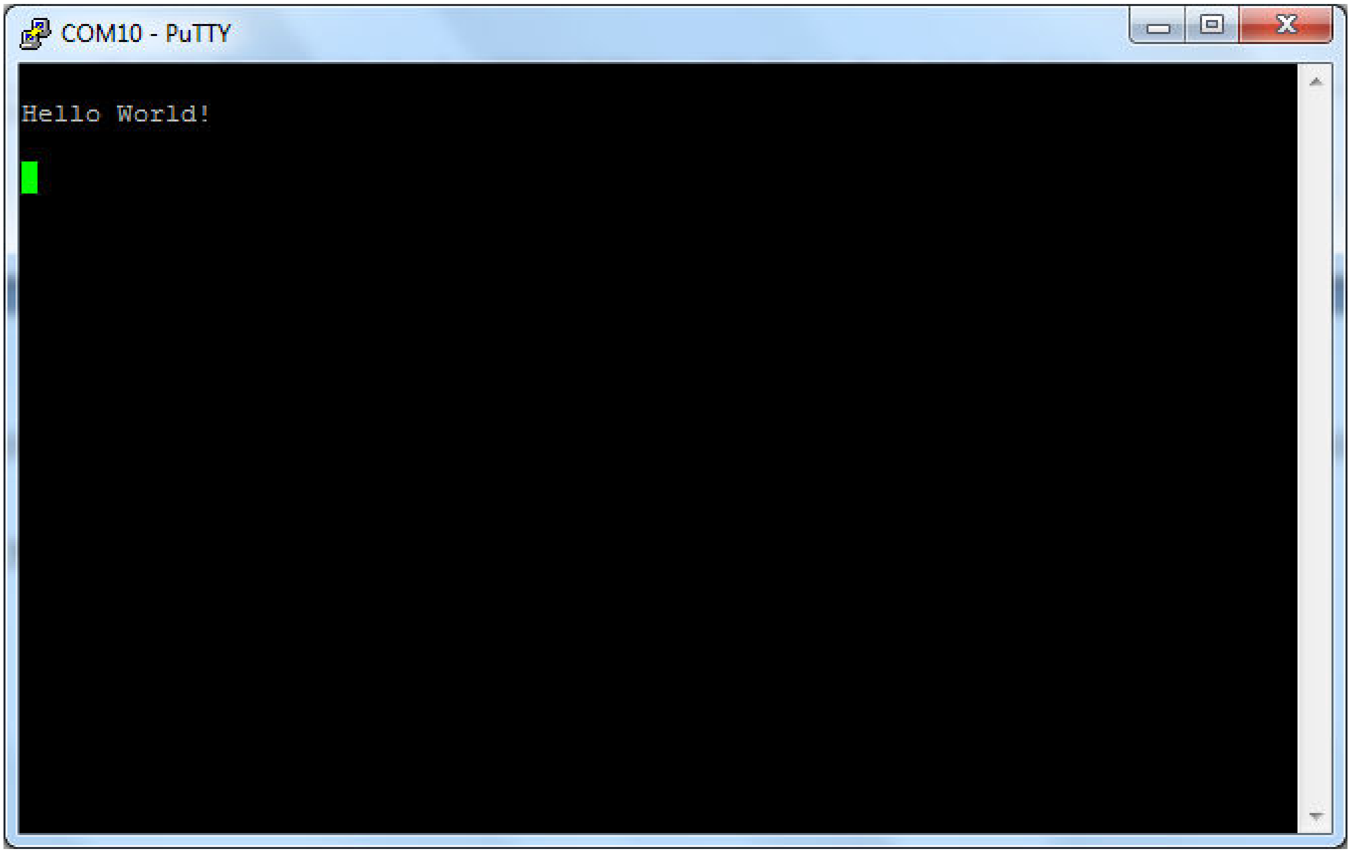


Figure 23. Hello world demo running on Cortex-M7 core

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

7.1 Set up environment

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

7.1.1 Download the MfgTool

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

The tool can be downloaded from github. Use version 1.1.81 or higher for full support for the M4 image. Download libusb-1.0.dll and uuu.exe for Windows OS, or download UUU for Linux. Configure the path so that the executable can later be called anywhere in the command line.

7.1.2 Switch to Download Mode

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

1. Set the board boot mode to Download Mode[SW1101:1000[1-4]].
2. Connect the development platform to your PC via USB cable between the SERIAL port and the PC USB connector. The SERIAL port is J301 on the base board.
3. The PC recognizes the i.MX 8M Mini device as (VID:PID)=(1FC9:013E), which is shown in the figure below.

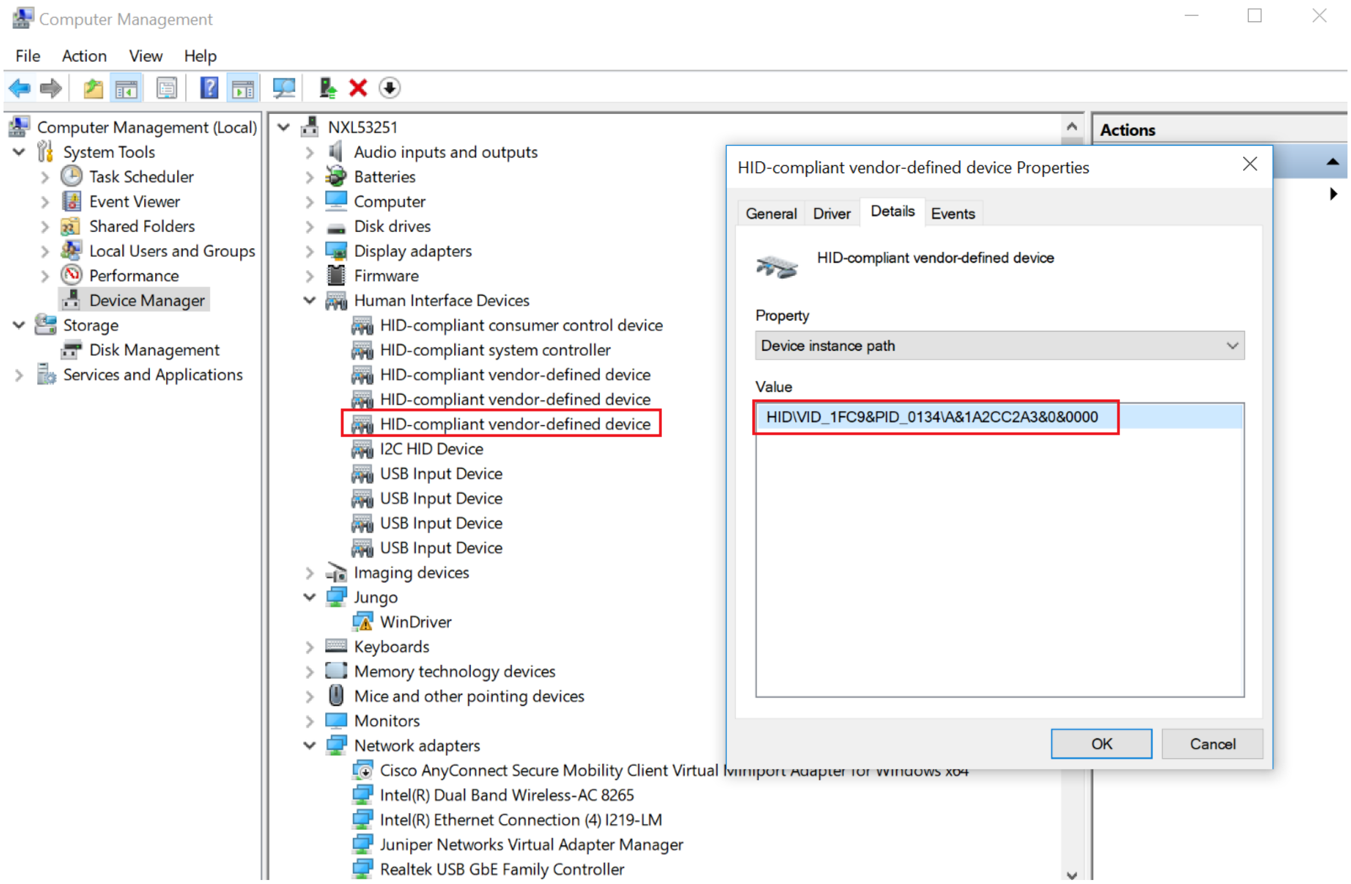


Figure 24. Device as shown in Device Manager

7.2 Build an example application

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

1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path:*<install_dir>/boards/<board_name>/<example_type>/<application_name>/iar*

   Using the i.MX 8M Mini EVK board as an example, the hello_world workspace is located in *<install_dir>/boards/evkmimx8mm/demo_apps/hello_world/iar/hello_world.eww*.

2. Select the desired build target from the drop-down. For this example, select the “hello_world – flash_debug” target.

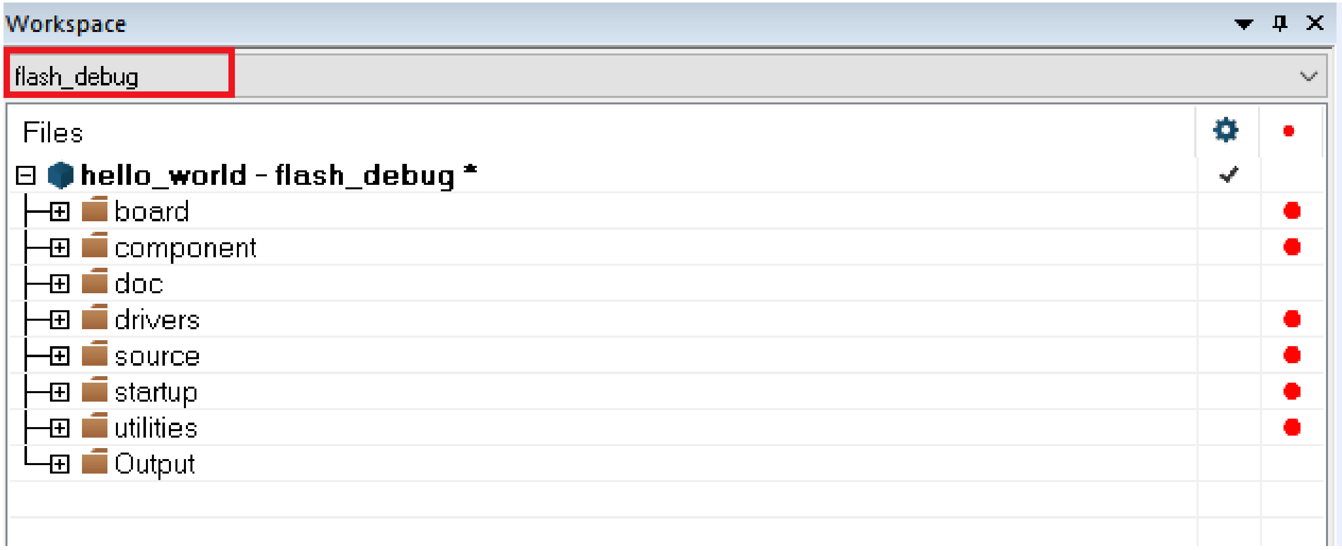


Figure 25. Demo build target selection

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

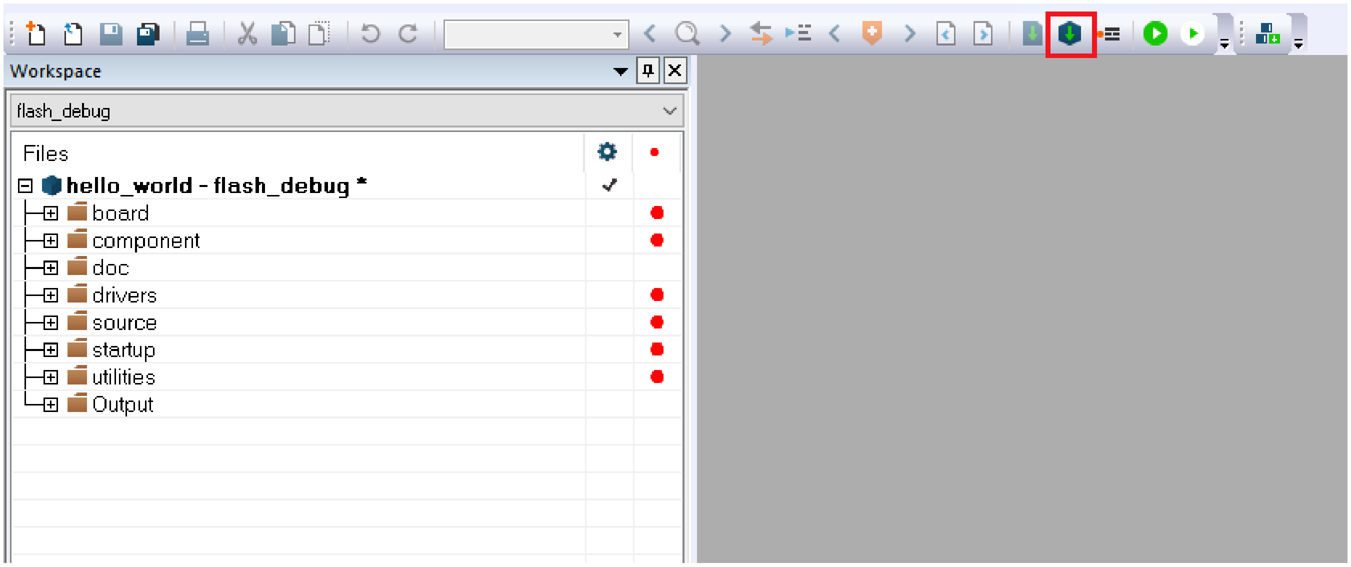


Figure 26. Build the demo application

4. The build completes without errors.
5. Rename the generated “hello_world.bin” to “m4_flash.bin”, then copy it to the uuu tool directory.

7.3 Run an example application

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

1. Connect the development platform to your PC via USB cable between the J901 USB DEBUG connector and the PC. It provides console output while using UUU.
2. Connect the J301 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

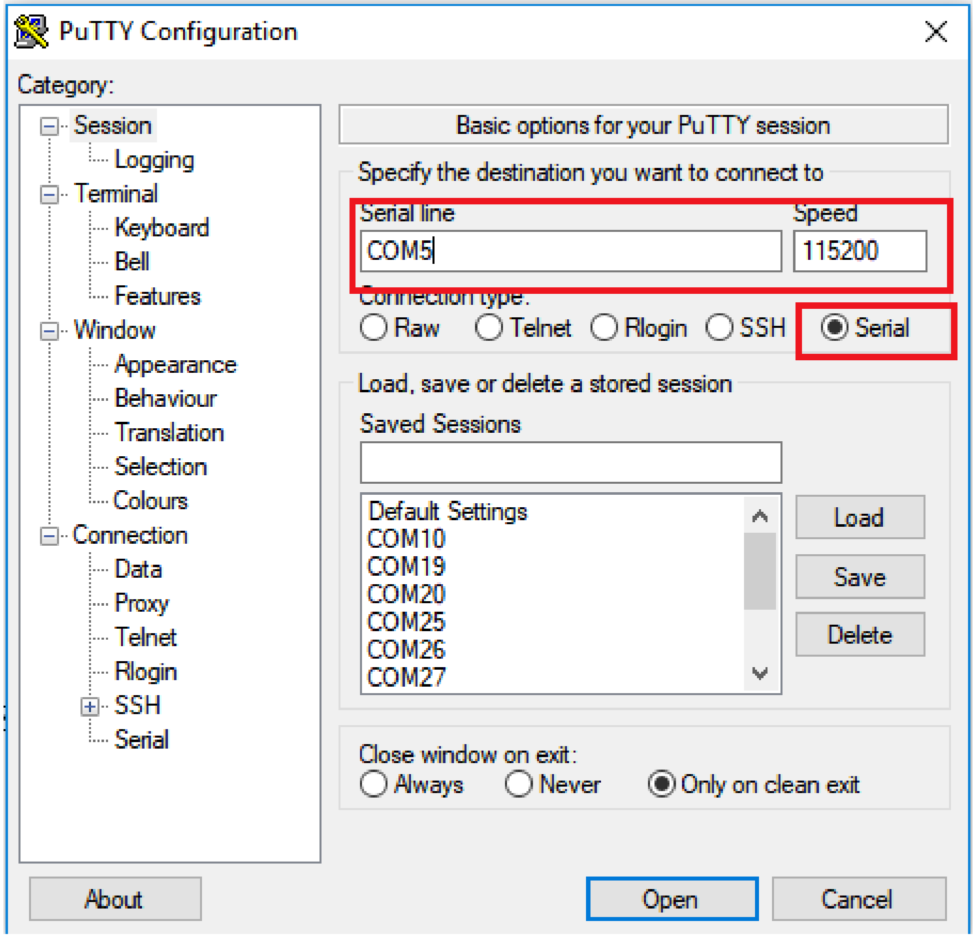


Figure 27. Terminal (PuTTY) configuration

4. Get the fspi version U-Boot image from release package and rename it to uboot_flash.bin.
5. In the command line, execute uuu with the *-b qspi* switch: `uuu -b qspi m4_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:

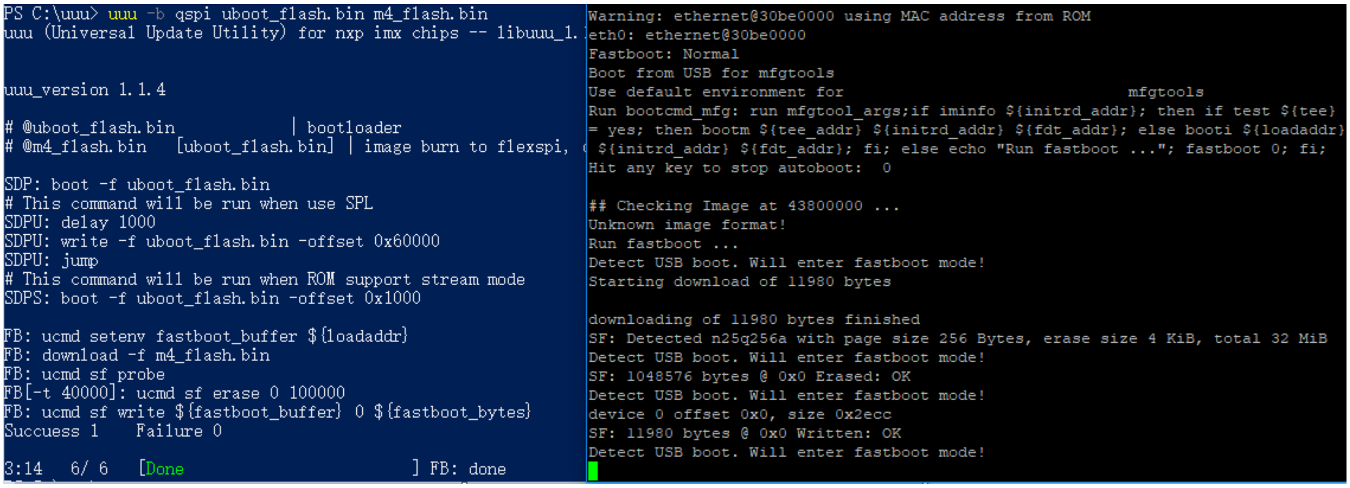


Figure 28. Command line and fast boot console output when executing UUU

6. Then, power off the board, change the boot mode to MicroSD Mode [SW1101:0110000000, SW1102:1000111100], and power on the board again.
7. Use following command in U-Boot to kickoff m7:

```
sf probe
bootaux 0x8000000
```

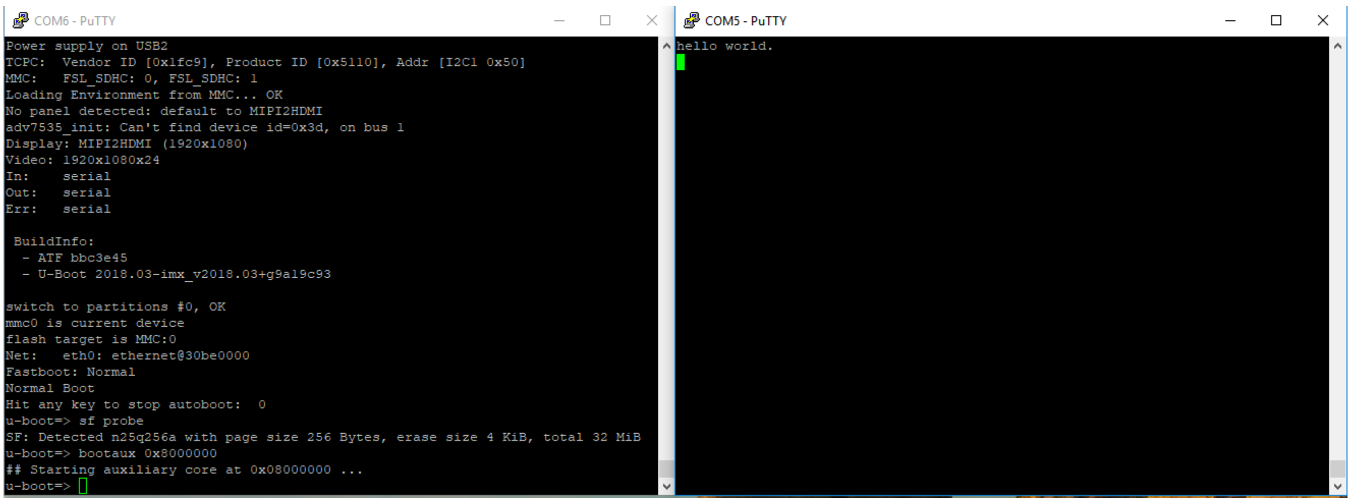


Figure 29. Console output from QSPI Boot

Appendix A How to determine COM port

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

1. To determine the COM port, open the Windows operating system Device Manager. This can be achieved by going to the Windows operating system Start menu and typing **Device Manager** in the search bar, as shown in Figure A-1.

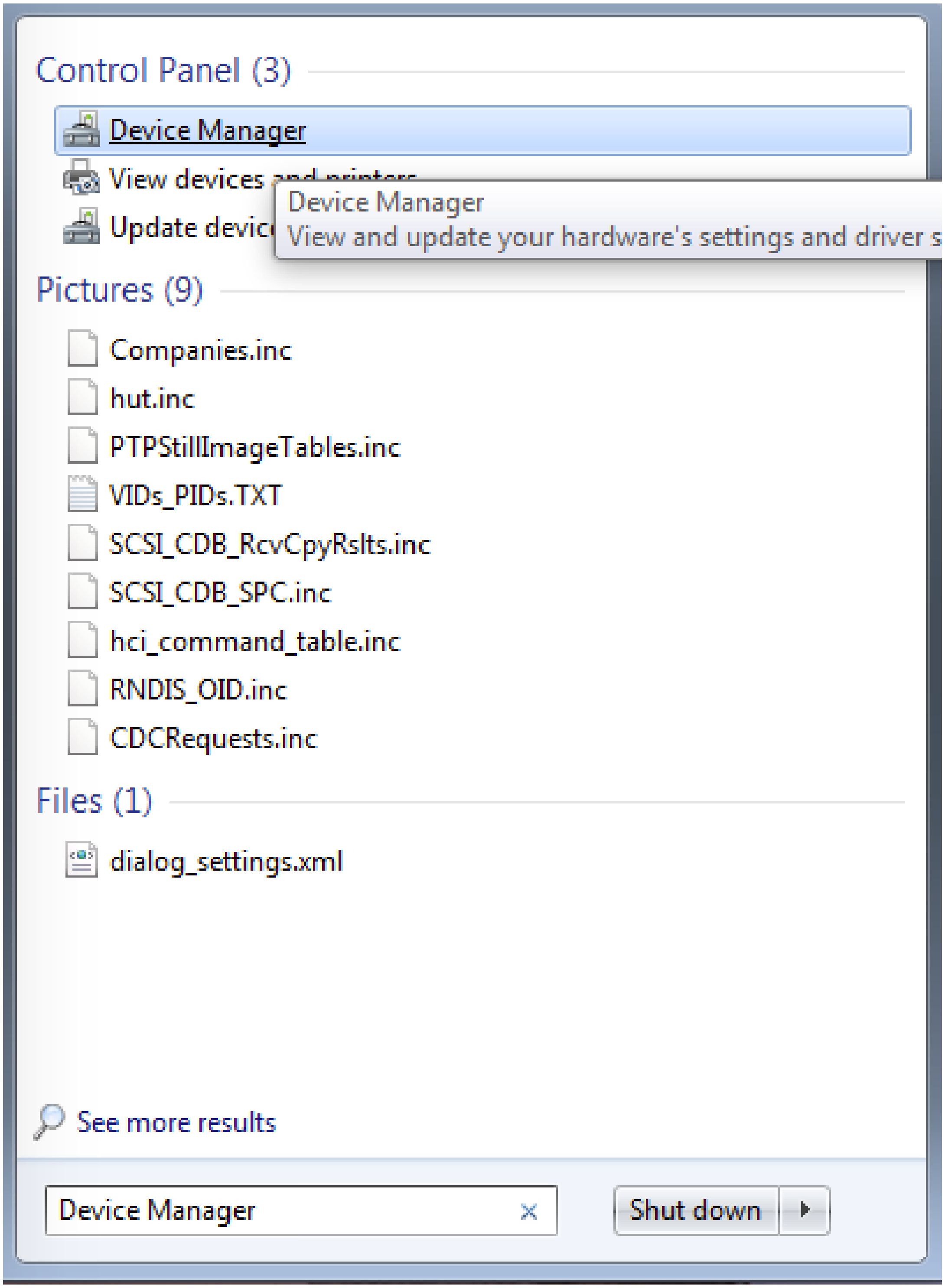


Figure A-1. Device manager

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

Appendix B Host setup

An MCUXpresso SDK build requires that some packages are installed on the Host. Depending on the used Host operating system, the following tools should be installed.

Linux:

- Cmake

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

Windows:

- MinGW

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

  a. Download the latest MinGW mingw-get-setup installer from sourceforge.net/projects/mingw/files/Installer/.
  b. 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.

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

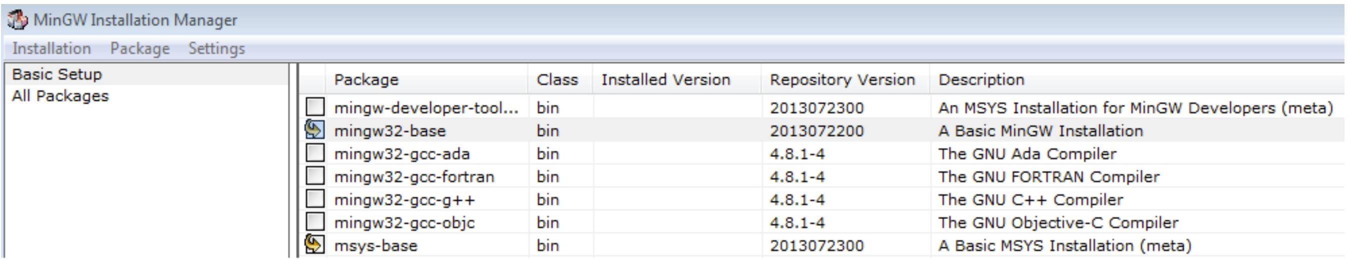


Figure B-1. Setup MinGW and MSYS

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

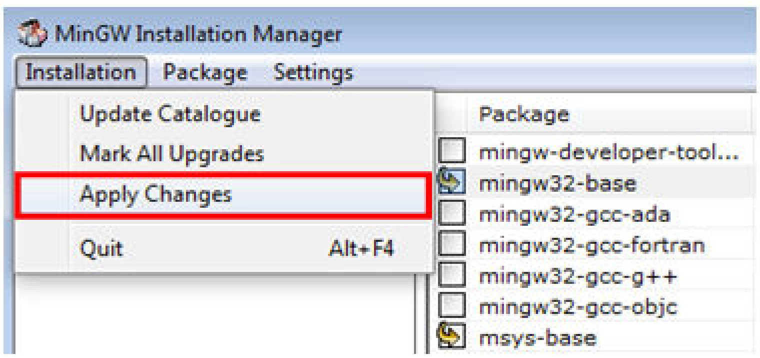


Figure B-2. Complete MinGW and MSYS installation

  e. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is: <mingw_install_dir>\bin.

Assuming the default installation path, `C:\MinGW`, an example is as shown in Figure B-3. 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.

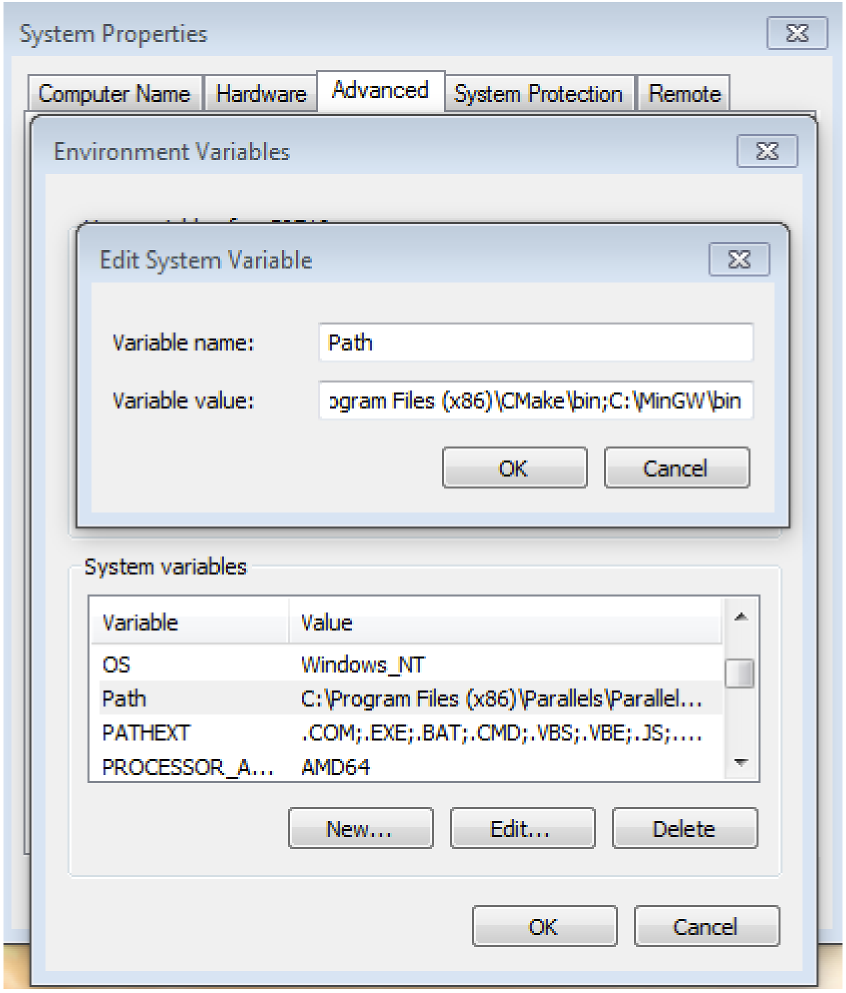


Figure B-3. Add Path to systems environment

- Cmake

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

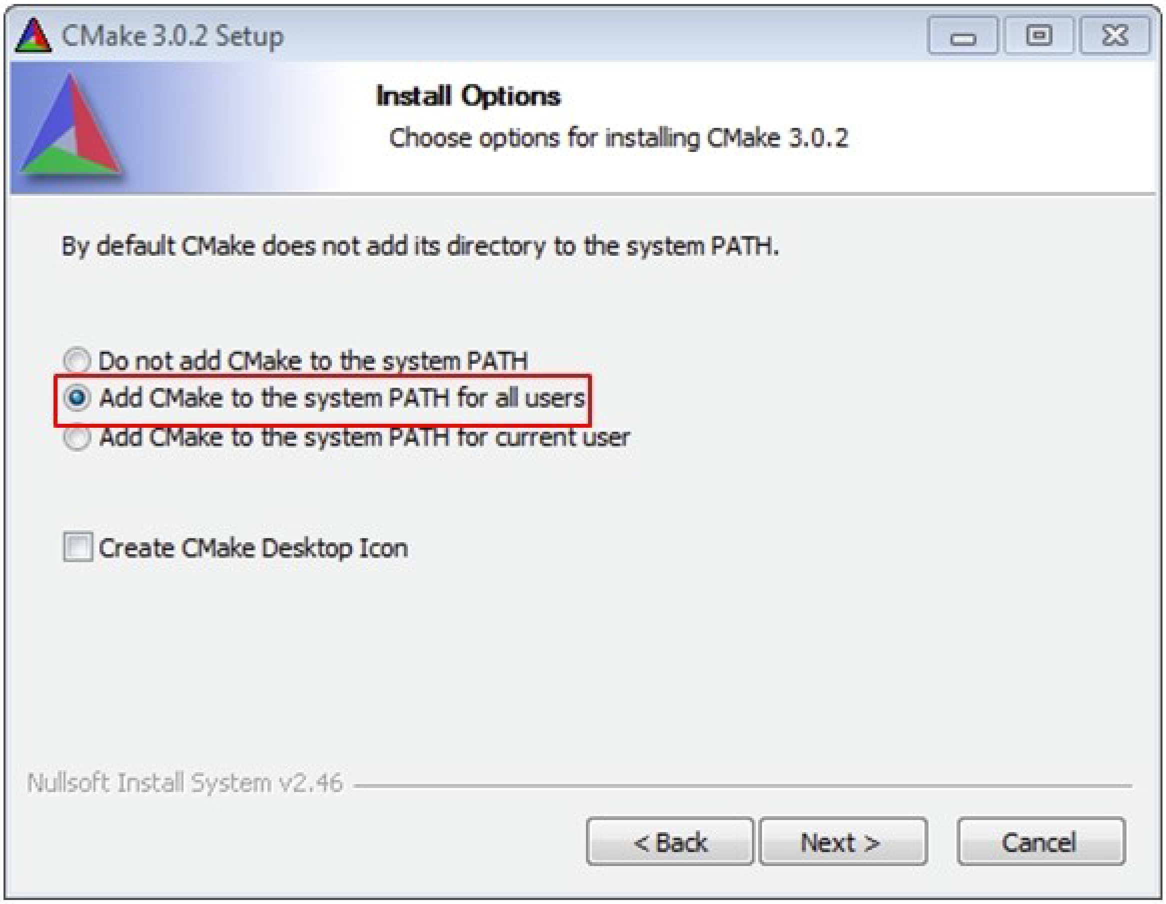


Figure B-4. Install CMake

c. Follow the remaining instructions of the installer.

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

Document Number MCUXSDKIMX8MMGSUG
Revision 0, 10/2019