Getting Started with MCUXpresso SDK Supporting i.MX 7ULP

1   Overview

The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for Kinetis and LPC 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 full demo applications. The MCUXpresso SDK contains FreeRTOS and various other middleware to support rapid development.

For supported toolchain versions, see the MCUXpresso SDK Release Notes Supporting EVK-MCIMX7ULP (document MCUXSDKIMX7ULPRN).

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

Contents

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, including Freedom, Tower System, and LPCXpresso boards. Board support packages are found inside of
the top level boards 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 to classify the type of examples they contain. These
include (but are not limited to):

- cmsis_driver_examples: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- demo_apps: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These
  applications typically use multiple MCU peripherals and may leverage stacks and middleware.
- driver_examples: Simple applications intended to concisely illustrate how to use the MCUXpresso SDK's peripheral
  drivers for a single use case. These applications typically only use a single peripheral, but there are cases where
  multiple are used (for example, SPI conversion using DMA).
- rtos_examples: Basic FreeRTOS™ OS examples showcasing the use of various RTOS objects (semaphores, queues,
  and so on) and interfacing with the MCUXpresso SDK's RTOS drivers
- multicore_examples: Applications for both cores showing the usage of multicore software components and the
  interaction between cores.
- 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 the
MCUXpresso SDK API Reference Manual document (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:
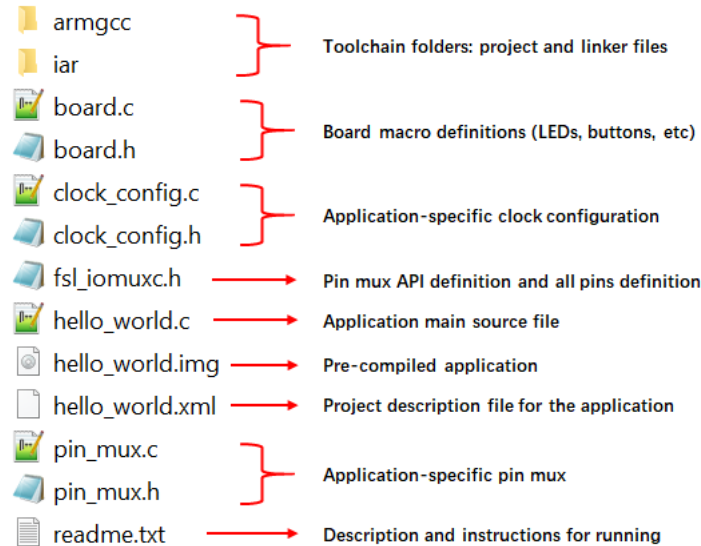
Figure 2. Application folder structure

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 (except MCUXpresso IDE), 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>/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. 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 them could have potential impacts on other projects that depend on them.

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

3 Toolchain introduction

The MCUXpresso SDK release for i.MX 7ULP 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 in the below table.

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 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 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 host

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

4.1.1 Set up toolchain

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

4.1.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset (in other words, compiler, linker, etc.). 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.

4.1.1.2 Add a new system environment variable for ARMGCC_DIR

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

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

4.1.2 Build an example application

To build an example application, follow these steps.

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

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

   For this example, the exact path is:

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:
-- 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
```

Getting Started with MCUXpresso SDK Supporting i.MX 7ULP, Rev. 0, 01/2019

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

4.1.3 Run an example application

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

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
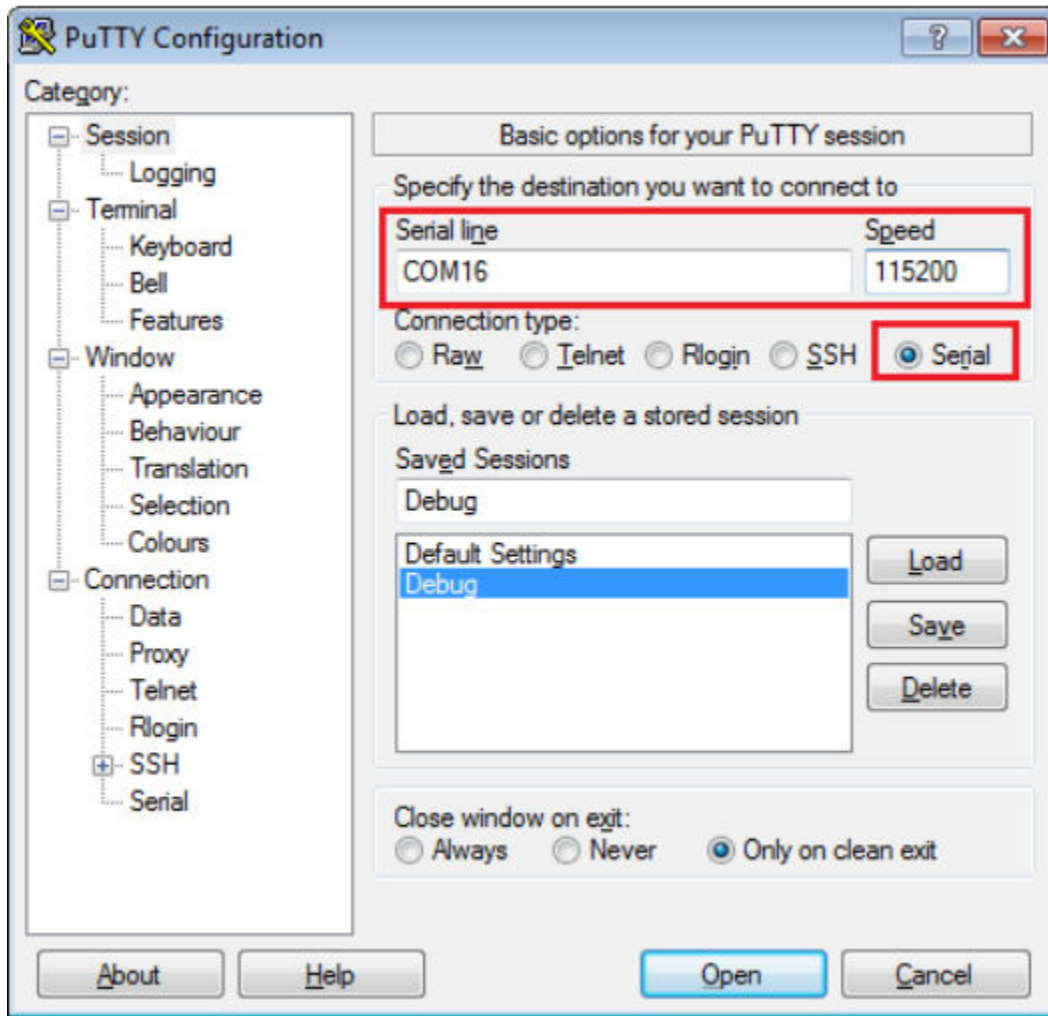
Figure 3. Terminal (PuTTY) configurations

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 device:

```
$ JLinkGDBServer -if JTAG -device
SEGGER J-Link GDB Server  Command Line Version
JLinkARM.dll
Command line: -if JTAG -device
-----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:
```

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

5. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf


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...done.
(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
```

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

```
(gdb) monitor go
```

The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.
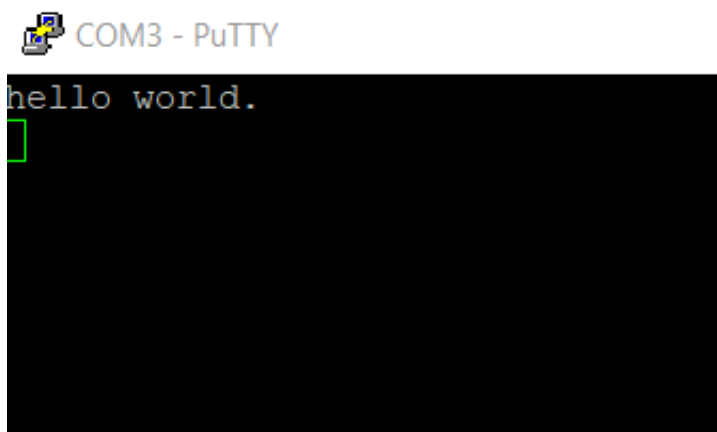
Figure 4. Text display of the hello_world demo

4.2   Windows OS host

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

4.2.1   Set up toolchain

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

4.2.1.1   Install GCC Arm Embedded tool chain

Download and run the installer from developer.arm.com/open-source/gnu-toolchain/gnu-rm. This is the actual toolset (in other words, compiler, linker, and so on). The GCC toolchain should correspond to the latest supported version, as described in the *MCUXpresso SDK Release Notes*. (document MCUXSDKRN).

NOTE
See the *Host Setup* Section in Appendix B for Windows OS 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 tool chain installation path.

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

4.2.2   Build an example application

To build an example application, follow these steps.

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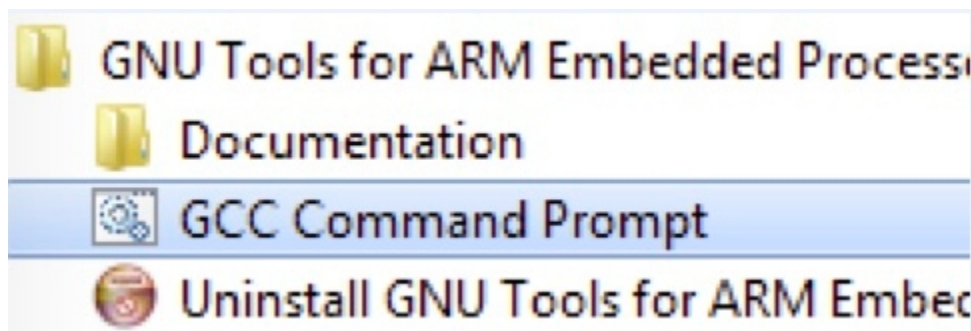

Figure 5. Launch command prompt

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:

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:

4.2.3   Run an example application

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

- 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
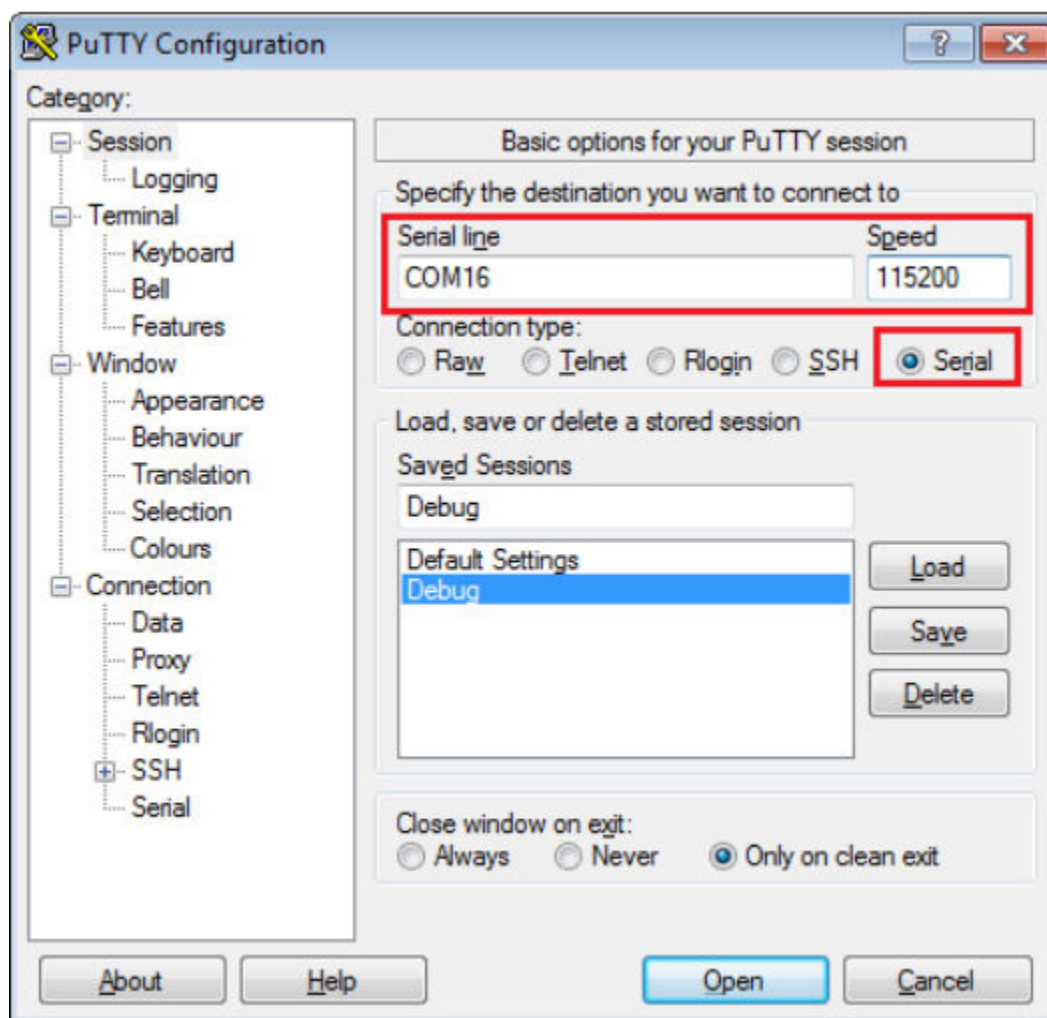
Figure 6. Terminal (PuTTY) configurations

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 .

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

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".
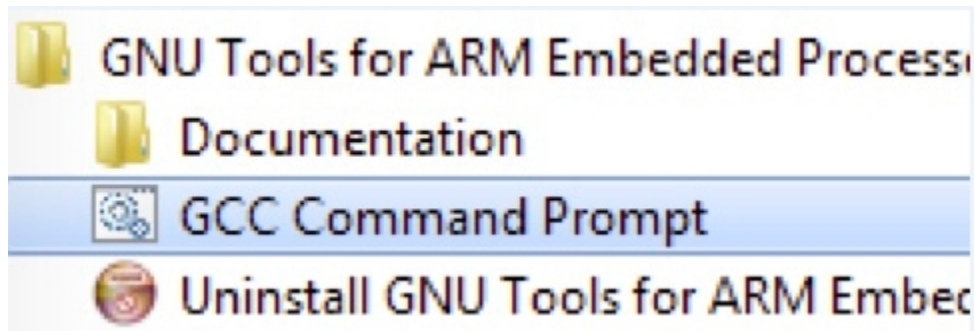
Figure 7. Launch command prompt

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:

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".
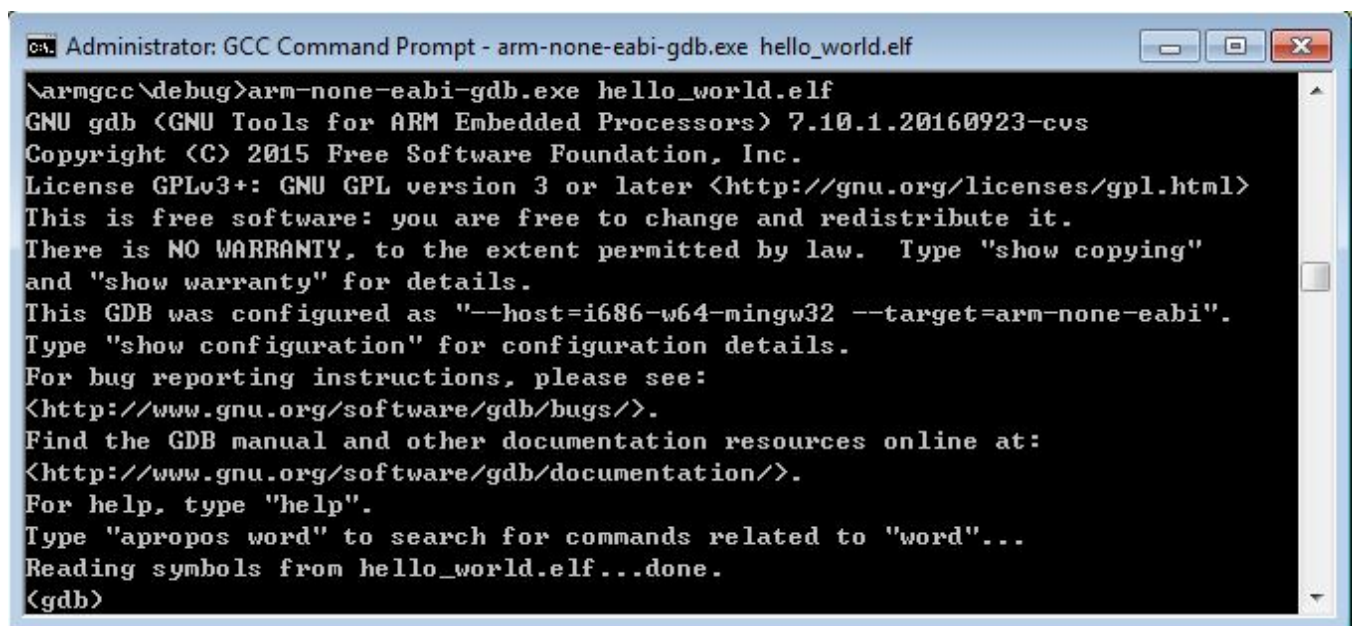
Figure 8. Run arm-none-eabi-gdb

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.
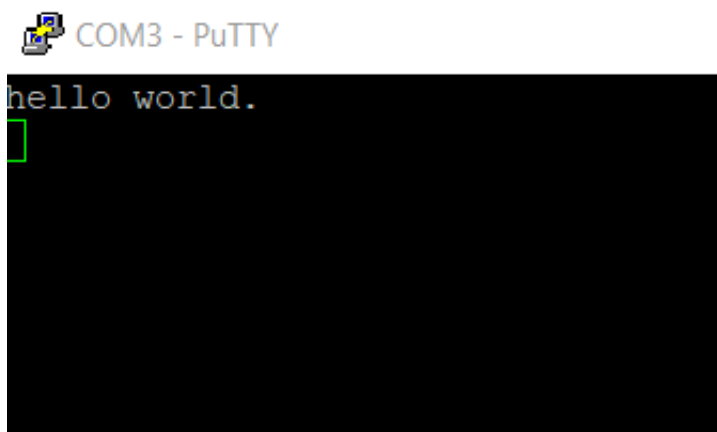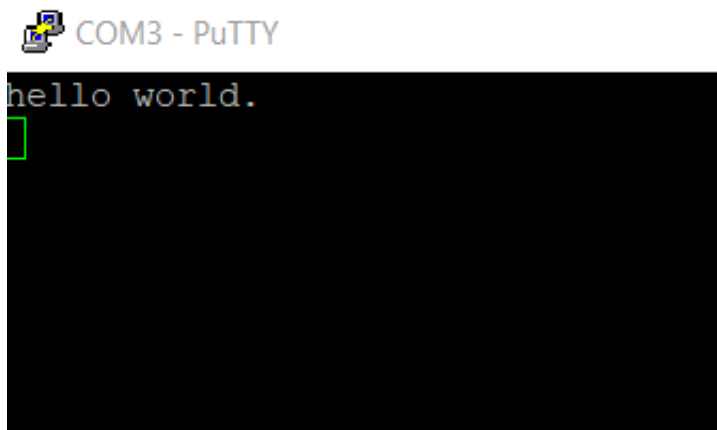
Figure 9. Text display of the hello_world demo

5   Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The hello_world demo application targeted for the MCIMX7ULP-EVK 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

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 path.

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 MCIMX7ULP-EVK hardware platform as an example, the hello_world workspace is located in

    <install_dir>/boards/evkmcimx7ulp/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 – Debug" target.
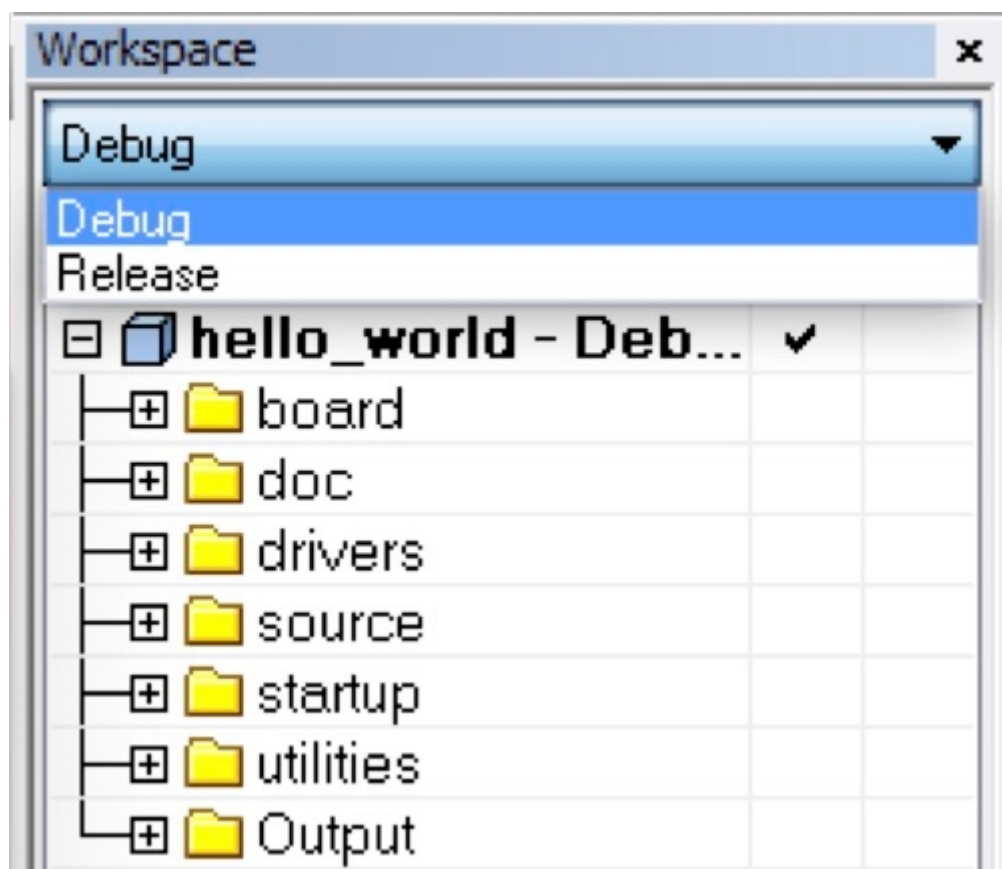
Figure 10. Demo build target selection

3. To build the demo application, click the "Make" button, highlighted in red below.
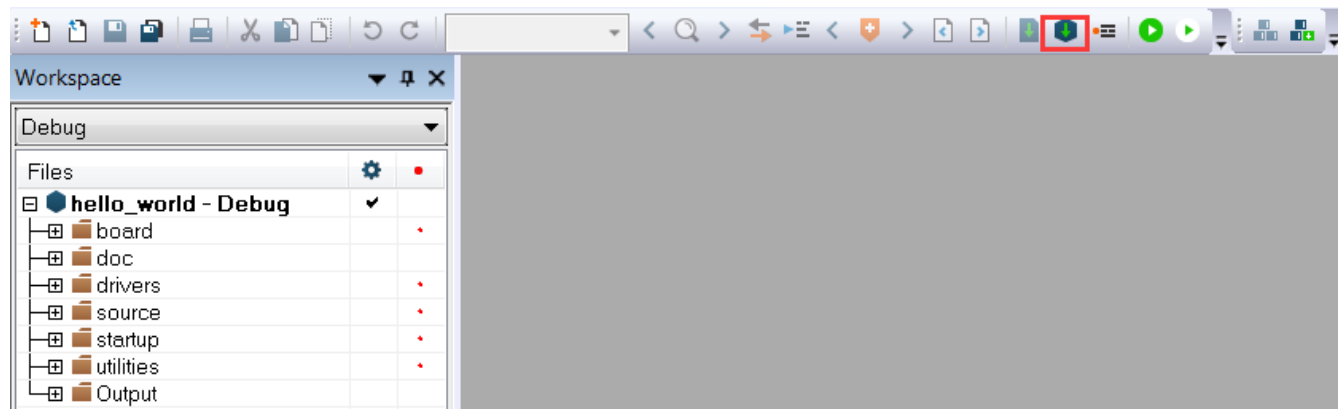
Figure 11. Build the demo application

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 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 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
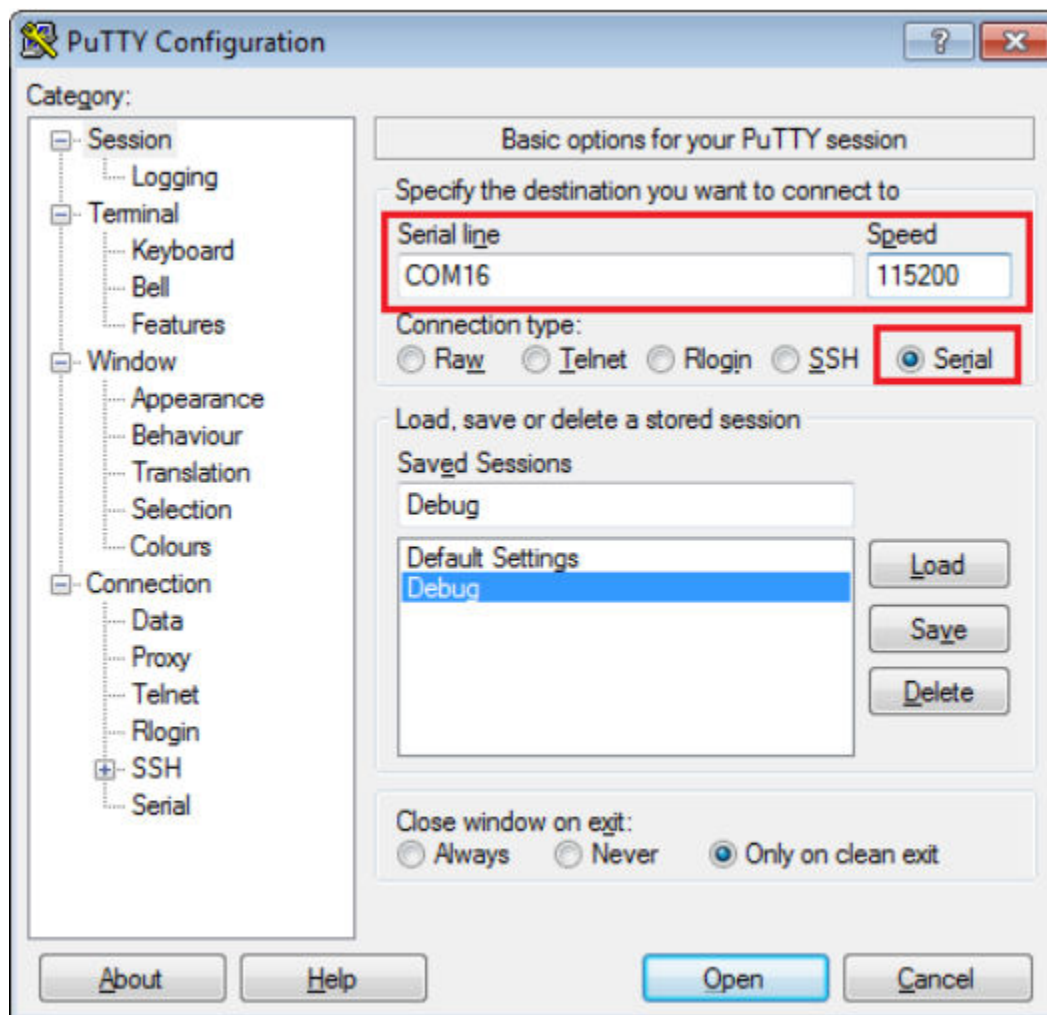
Figure 12. Terminal (PuTTY) configuration

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

Figure 13. Download and Debug button

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

Getting Started with MCUXpresso SDK Supporting i.MX 7ULP, Rev. 0, 01/2019

Figure 14. Stop at main() when running debugging

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

Figure 15. 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 16. Text display of the hello_world demo

6 Running an application from QSPI flash

This section describes the steps to write a bootable SDK image to QSPI flash 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®-A7 and the other is for the Cortex®-M4. 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.
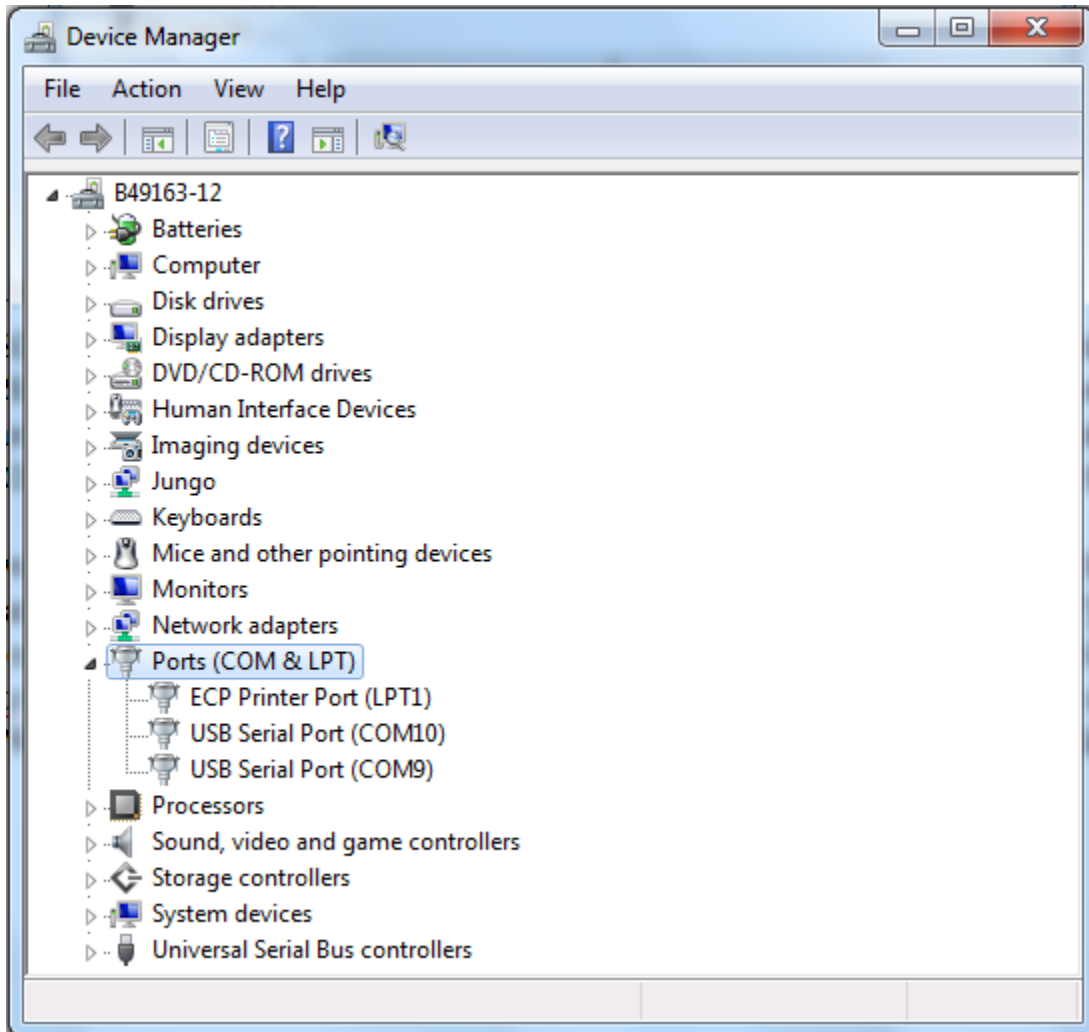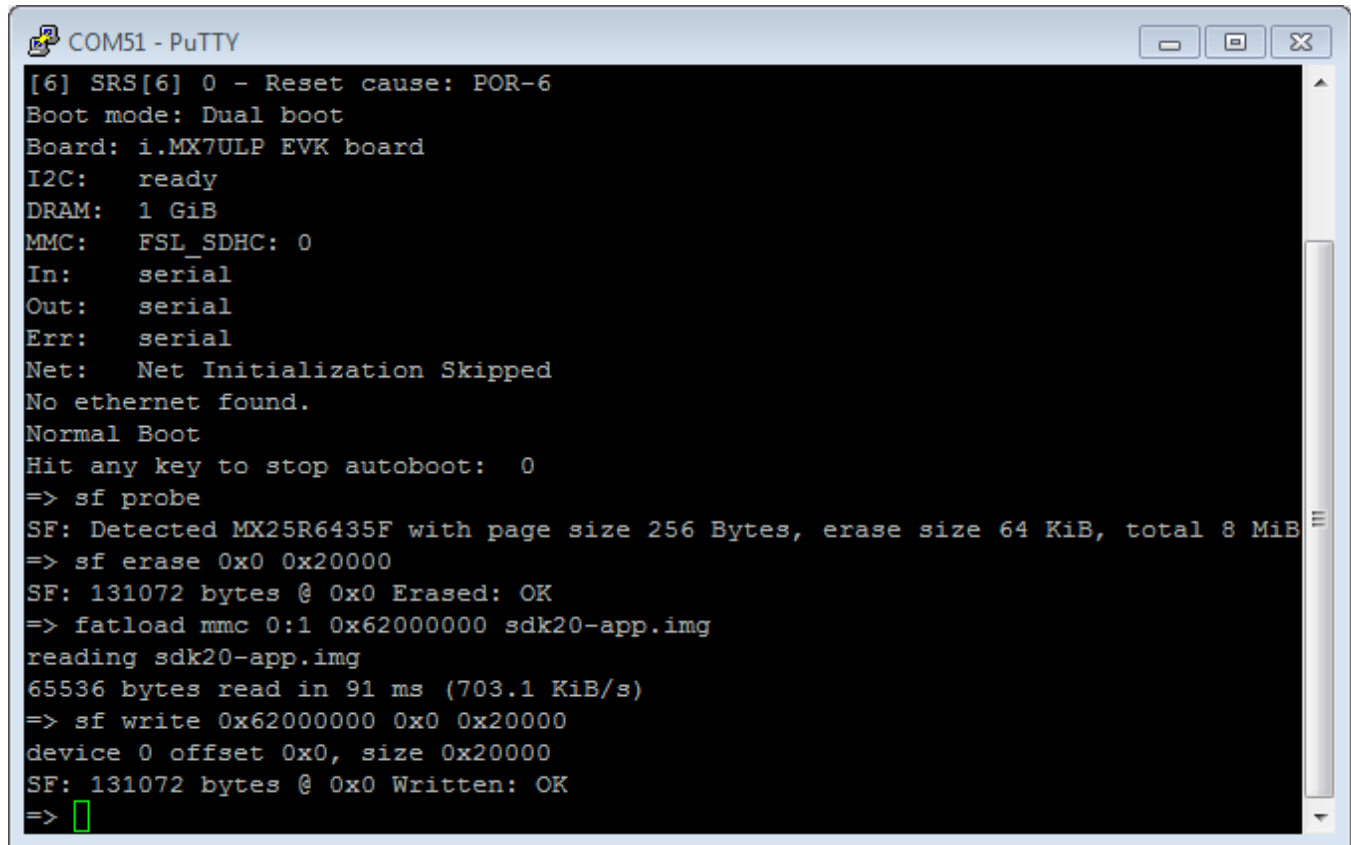
Figure 17. Determining the COM Port of target board

3. Build the application (for example, hello_world) and copy the built binary (sdk20-app.bin file) to the *<install_dir>/ tools/imgutil/evkmcimx7ulp* folder.
4. In the *<install_dir>/tools/imgutil/evkmcimx7ulp*p folder, run mkimg.sh in mingw32 shell to get bootable image file sdk20- app.img.
   - If the image is built with RAM link file, use "mkimg.sh ram" to create the bootable image.
   - If the image is built with flash link file, use "mkimg.sh flash" to create the bootable XIP.
5. Prepare an SD card with the prebuilt U-Boot image and copy the sdk20-app.img generated into the SD card (as shown in Step 4). Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dip switch configuration.
6. 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.
7. 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 QSPI Flash with the following commands (Assume image size is less than 0x20000, otherwise the sf erase and write command size parameter need to be enlarged. If the image size is bigger than 0x20000 (128 KB), change 0x20000 to a number larger or equal to the image size.):
   - sf probe.

- sf erase 0x0 0x20000.
- fatload mmc 0:1 0x62000000 sdk20-app.img.
- sf write 0x62000000 0x0 0x20000.

Figure 18. U-Boot command to run application on QSPI

8. 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
9. Power off and repower on the board.
10. 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.
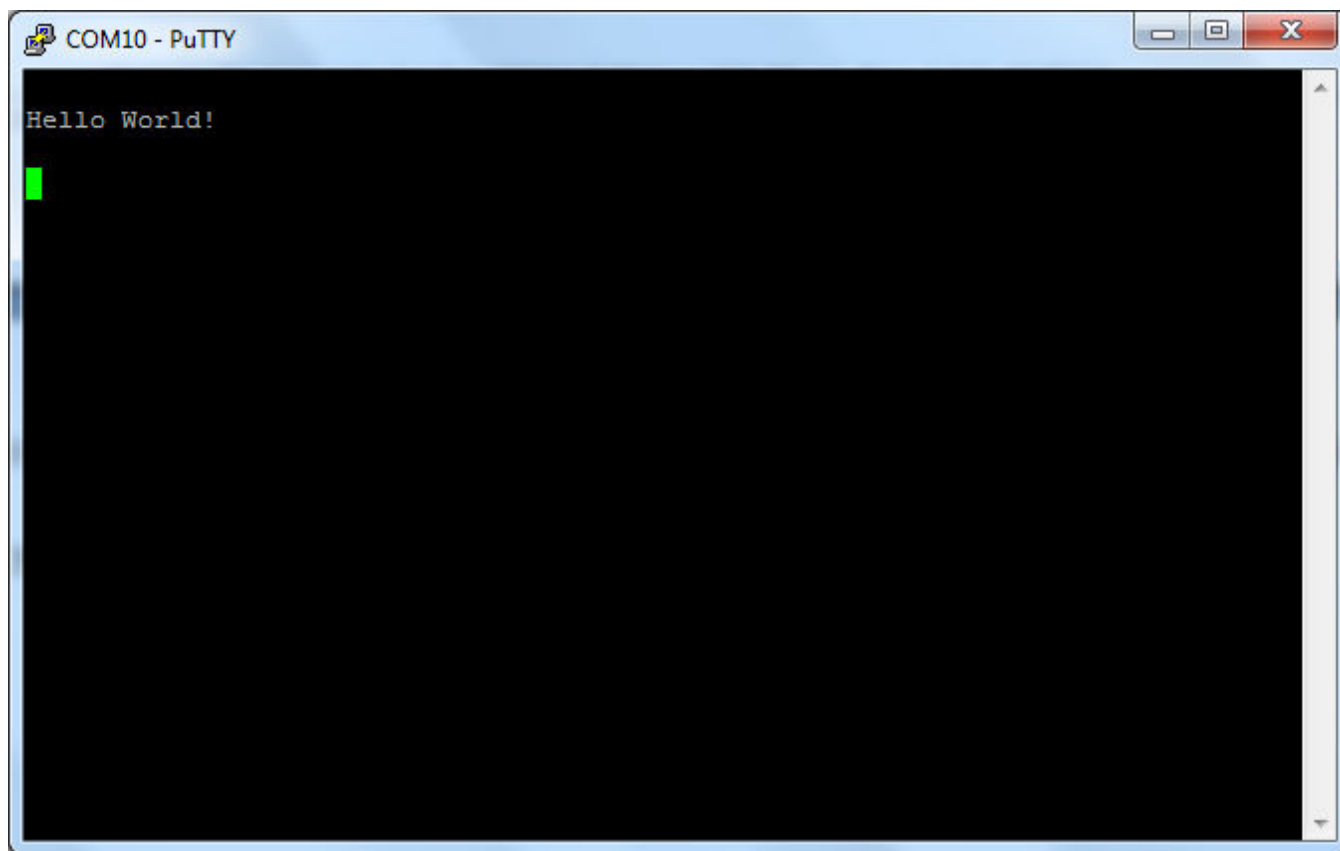
Figure 19. Hello world demo running on Cortex-M4 Core

7 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. **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, another is for Cortex M4.

2. **Windows:** 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:
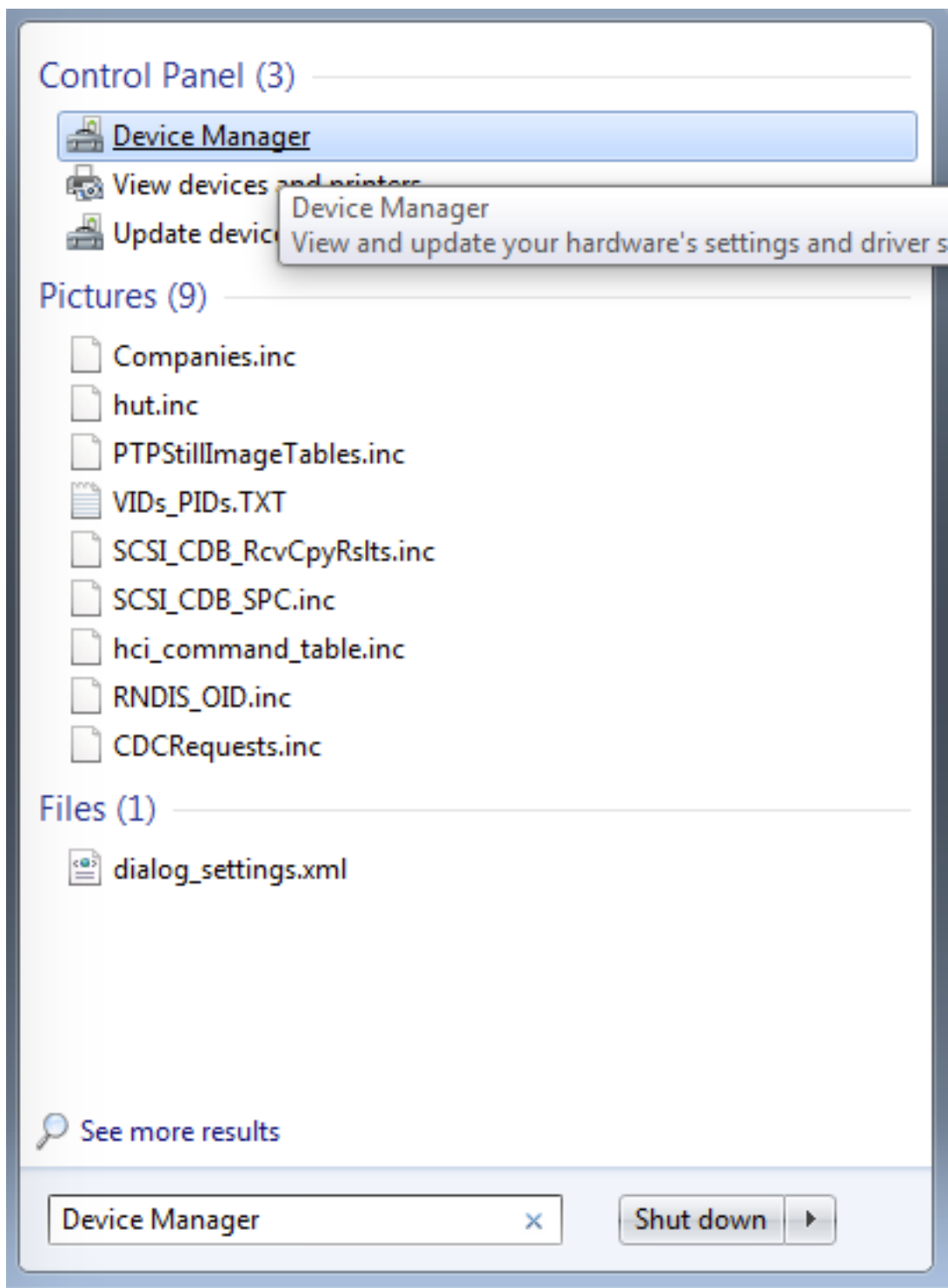
Figure 20. Device manager

3. 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:
   a. USB-UART interface
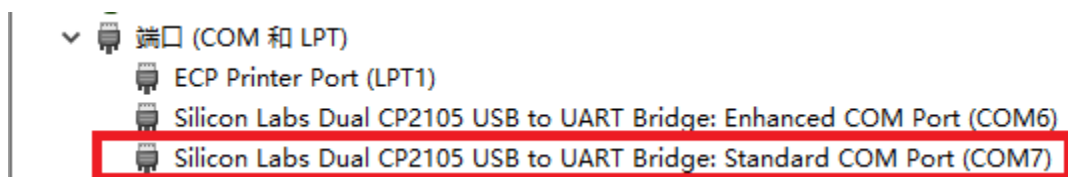
Figure 21. USB-UART interface

8 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.

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.
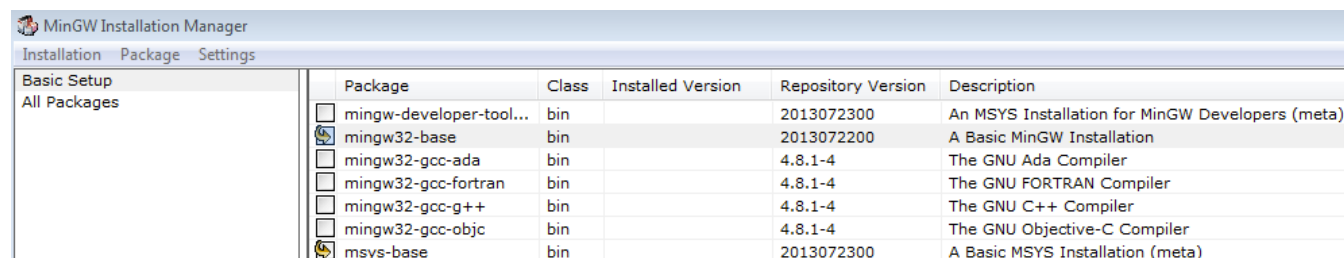
Figure 22. Setup MinGW and MSYS

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

Getting Started with MCUXpresso SDK Supporting i.MX 7ULP, Rev. 0, 01/2019
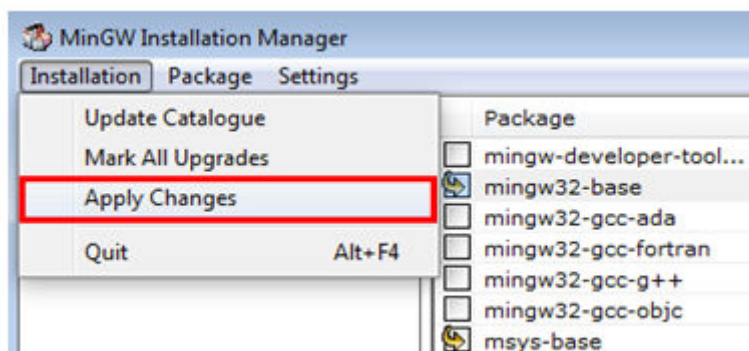
Figure 23. Complete MinGW and MSYS installation

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 shown below. If the path is not set correctly, the toolchain does not work.

<div align="center">

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.

</div>
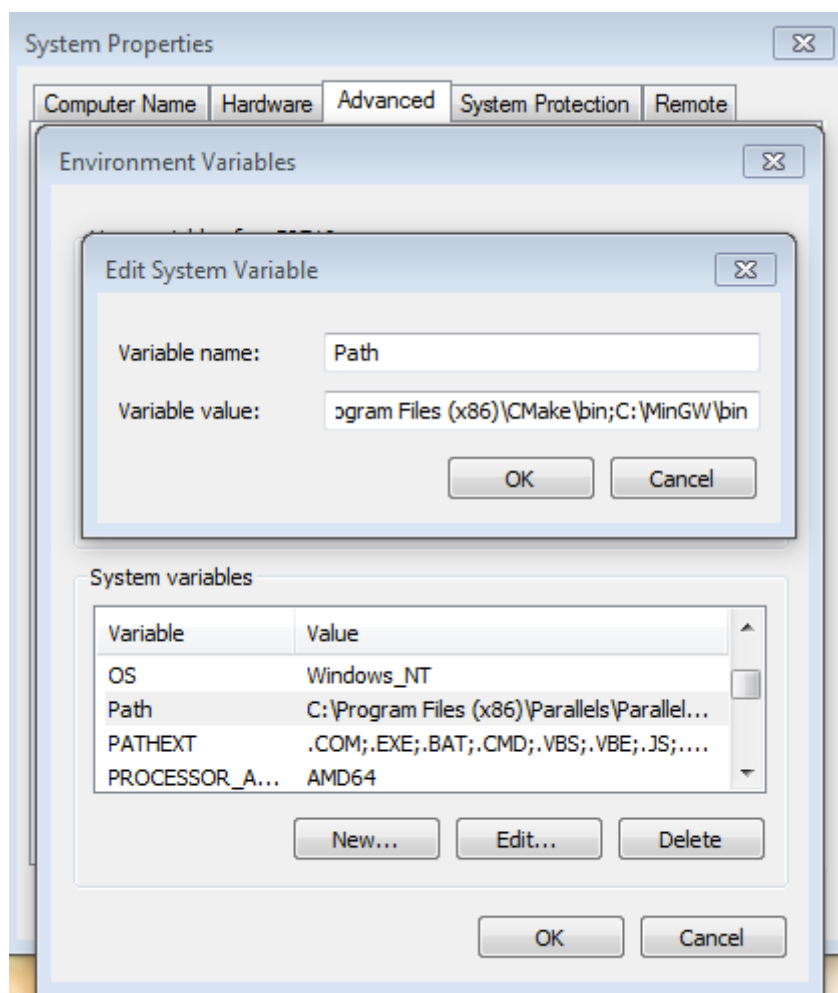
Figure 24. Add Path to systems environment

- 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.
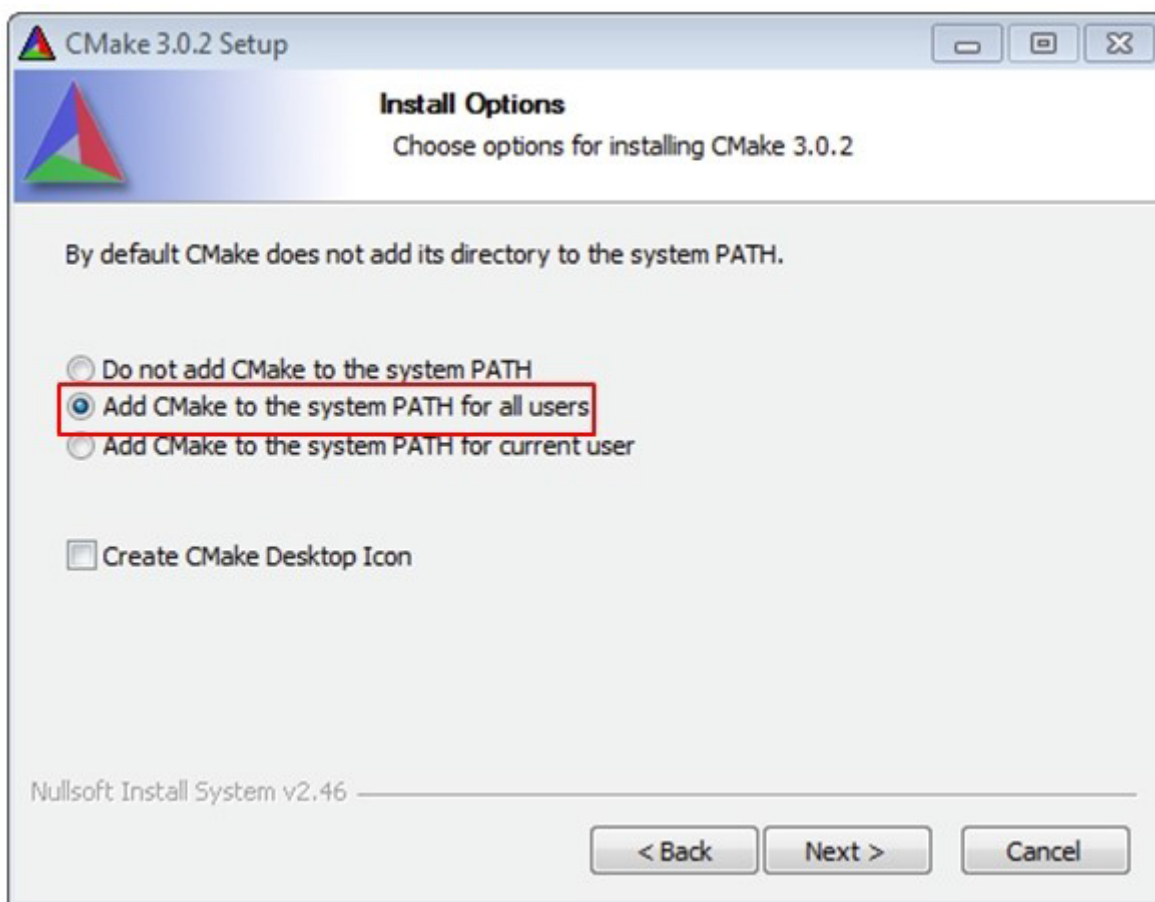
Figure 25. Install Cmake

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

Document Number MCUXSDKIMX7ULPGSUG
Revision 0, 01/2019