MCUXSDKMCIMX93EVKGSUG

Getting Started with MCUXpresso SDK for MCIMX93-EVK

Rev. 2.13.1 — 31 March 2023 User guide

Document information

Information	Content
Keywords	MCUXSDKMCIMX93EVKGSUG, MCIMX93-EVK, MCIMX93EVK, Getting Started, MCUXpresso SDK
Abstract	This document describes the steps to get started with MCUXpresso SDK for MCIMX93-EVK.

1 Overview

The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease and help accelerate embedded system development of applications based on general purpose, crossover and Bluetooth-enabled MCUs from NXP. 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 optional RTOS integrations such as FreeRTOS and Azure RTOS, and device stack to support rapid development on devices.

For supported toolchain versions, see *MCUXpresso SDK Release Notes for MCIMX93-EVK (document MCUXSDKMCIMX93EVKRN)*.

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

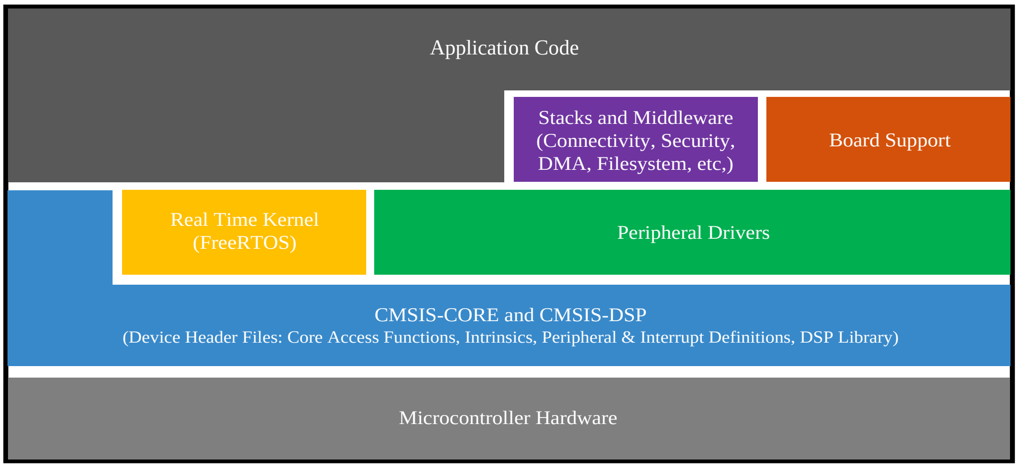


Figure 1. MCUXpresso SDK layers

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*.

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:

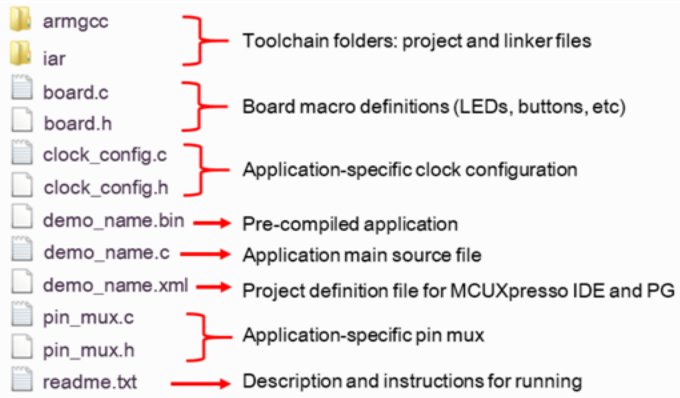


Figure 2. Application folder structure

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>/cmsis_drivers`: All the CMSIS drivers for your specific MCU
- `devices/<device_name>/drivers`: All of the peripheral drivers for your specific MCU
- `devices/<device_name>/<tool_name>`: Toolchain-specific startup code, including vector table definitions
- `devices/<device_name>/utilities`: Items such as the debug console that are used by many of the example applications
- `devices/<devices_name>/project_template`: Project template used in CMSIS PACK new project creation

For examples containing an RTOS, there are references to the appropriate source code. 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 93 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 93 release supports building and debugging with the toolchains listed in Table 1.

The user can choose the appropriate one for development.

For supported toolchain versions, see MCUXpresso SDK Release Notes for MCIMX93-EVK (*document: MCUXSDKMCIMX93EVKRN*).

- 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 93, the patch for IAR and Segger J-Link must be installed. To download, navigate to https://www.nxp.com/webapp/Download?colCode=SDK_MX93_3RDPARTY_PATCH&appType=license). The patch supports SEGGER versions older than v7.62. (does not applies to v7.62).*

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 93 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

Perform the following steps to build the `hello_world` example application.

1. Open the desired demo application workspace. Most example application workspace files can be located using the following path:

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

Using the i.MX 93 EVK hardware platform as an example, the `hello_world` workspace is located in:

```
<install_dir>/boards/mcimx93evk/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 **hello_world** – **debug**.

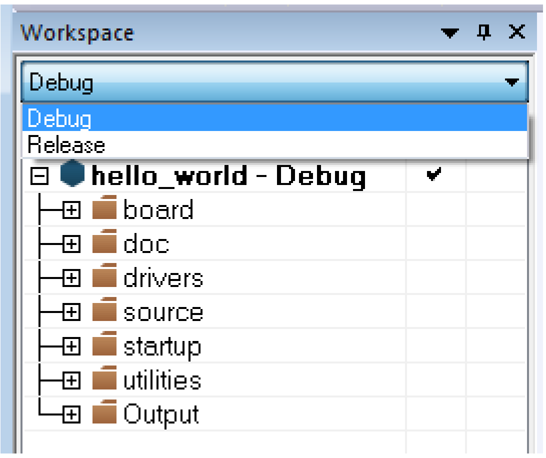


Figure 3. Demo build target selection

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

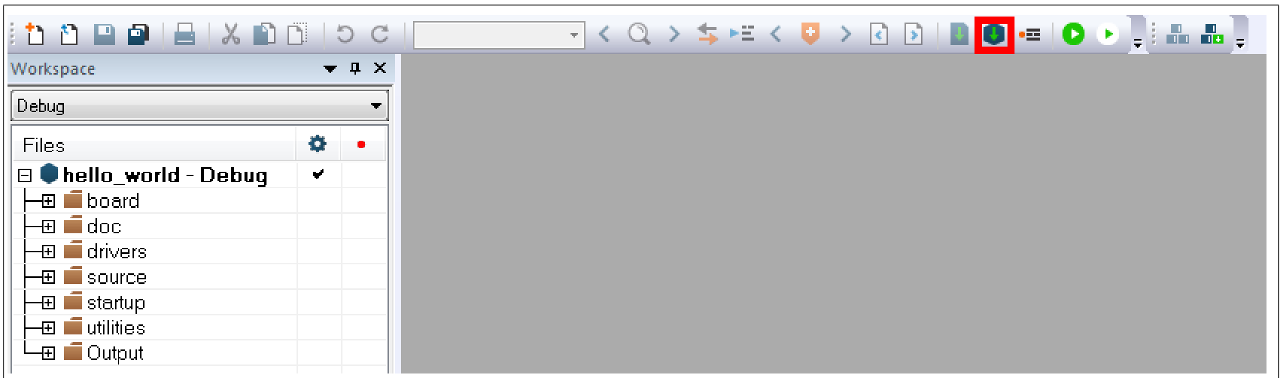


Figure 4. Build the demo application

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, as per the requirement listed in Toolchain introduction.
2. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC USB connector.
3. Connect 12 V ~ 20 V power supply and J-Link Plus to the device.

4. Switch SW1301[3:0] to the M core boot and ensure that the image is not available on the boot source. For example, 0b1010 for MicroSD boot. Keep the SD slot empty.
5. Open the terminal application on the PC, such as PuTTY or TeraTerm, connect to the debug COM port, see Section 8, and configure the terminal with these settings:
   a. 115,200 baud rate
   b. No parity
   c. 8 data bits
   d. 1 stop bit

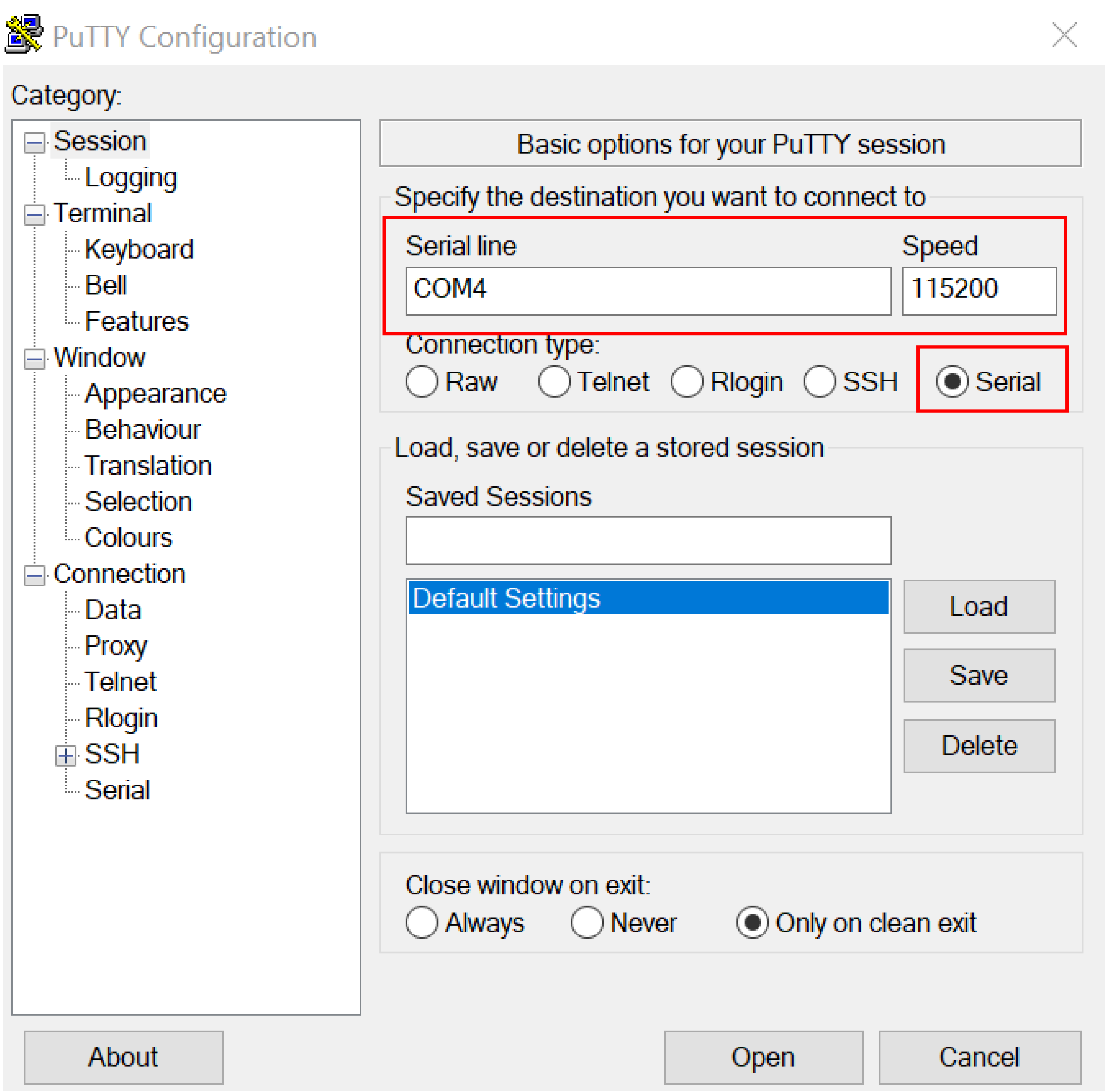


Figure 5. Terminal (PuTTY) configuration

6. Power on the board.
7. In IAR, click **Download and Debug** to download the application to the target.

Figure 6. Download and Debug button

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

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

9. Run the code by clicking **Go** to start the application.

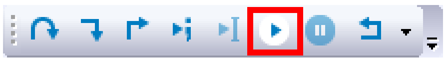


Figure 8. Go button

10. The `hello_world` application is now running and a banner is displayed on the terminal. If the application does not run or the banner is not displayed, check your terminal settings and connections.

Figure 9. Text display of the `hello_world` demo

Note: *If the software is already running on the M core, the debugger loading image into TCM may get HardFault or a data verification issue. NXP recommends you to follow the steps above to use the debugger. Repowering the board is required to restart the debugger.*

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 93 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 Section 9 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-9-2019-q4-major
```

```
$ export PATH= $PATH:/work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
```

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/mcimx93evk/ demo_apps/ hello_world/armgcc`.
2. Run the `build_debug.sh` script on the command line to perform the build. The output is shown as below:

```
$ ./build_debug.sh
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
-- BUILD_TYPE: debug
-- TOOLCHAIN_DIR: /work/platforms/tmp/gcc-arm-none-eabi-9-2019-q4-major
-- BUILD_TYPE: debug
-- The ASM compiler identification is GNU
-- Found assembler: /work/platforms/tmp/gcc-arm-none-eabi-8-2019-q3-update/
bin/arm-none-eabi-gcc
```

```
-- Configuring done
-- Generating done
-- Build files have been written to:
/work/platforms/tmp/nxp/SDK_2.12.0_MCIMX93_EVK/boards/mcimx93evk/demo_apps/
hello_world/armgcc/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_2.12.0_MCIMX93_EVK/boards/mcimx93evk/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

To run a demo application using J-Link GDB Server application, perform the following steps.

1. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC USB connector.
2. Connect 12 V ~ 20 V power supply and J-Link Plus to the device.
3. Switch SW1301[3:0] to the M core boot and ensure that the image is not available on the boot source. For example, 0b1010 for MicroSD boot. Keep the SD slot empty.
4. Open the terminal application on the PC, such as PuTTY or TeraTerm, connect to the debug COM port, see Section 8, and configure the terminal with these settings:
   a. 115200 baud rate, depending on your board (reference `BOARD_DEBUG_UART_BAUDRATE` variable in the `board.h` file)
   b. No parity
   c. 8 data bits
   d. 1 stop bit

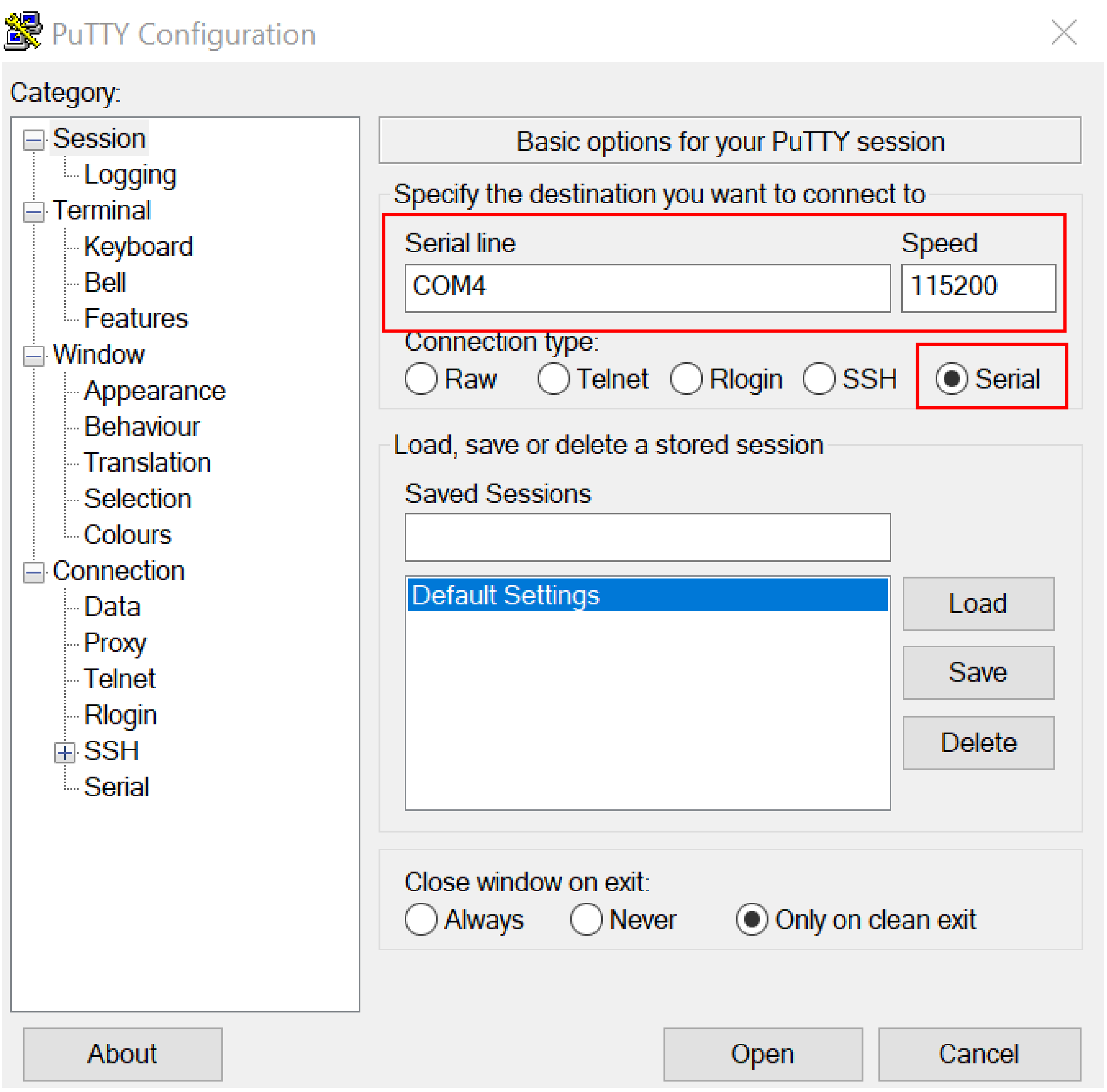


Figure 10. Terminal (PuTTY) configurations

5. Power on the board.
6. 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 MIMX9352_M33 device:

```
$ JLinkGDBServer -jlinkscriptfile /opt/SEGGER/JLink/Devices/NXP/iMX93/
NXP_iMX93_Connect_CortexM33.JLinkScript -device MIMX9352_M33 -if SWD
-----GDB Server start settings-----
GDBInit file:                  none
GDB Server Listening port:     2331
SWO raw output listening port: 2332
Terminal I/O port:             2333
Accept remote connection:      localhost only
Generate logfile:              off
Verify download:               off
Init regs on start:            off
```

```
Silent mode:                    off
Single run mode:                off
Target connection timeout:      5000 ms
------J-Link related settings------
J-Link Host interface:          USB
J-Link script:                  Devices\NXP
\iMX93\NXP_iMX93_Connect_CortexM33.JLinkScript
J-Link settings file:           none
------Target related settings------
Target device:                  MIMX9352_M33
Target interface:               SWD
Target interface speed:         4000kHz
Target endian:                  little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V9 compiled May  7 2021 16:26:12
Hardware: V9.60
S/N: 59611220
Feature(s): RDI, GDB, FlashDL, FlashBP, JFlash
Checking target voltage...
Target voltage: 1.98 V
Listening on TCP/IP port 2331
Connecting to target...
Connected to target
Waiting for GDB connection...
```

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/mcimx93evk/demo_apps/hello_world/armgcc/debug`
8. Start the GDB client:

```
$ arm-none-eabi-gdb hello_world.elf
GNU gdb (GNU Tools for Arm Embedded Processors 9-2019-q4-major)
 8.3.0.20190709-git
Copyright (C) 2019 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=i686-w64-mingw32 --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)
```

9. 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
0x00000008 in __isr_vector ()
(gdb) monitor reset
Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0x240 lma 0x0
Loading section .text, size 0x3ab8 lma 0x240
Loading section .ARM, size size 0x8 lma 0x3cf8
Loading section .init_array, size 0x4 lma 0x3d00
Loading section .fini_array, size 0x4 lma 0x3d04
Loading section .data, size 0x64 lma 0x3d08
Start address 0x2f4, load size 15724
Transfer rate: 264 KB/sec, 2620 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.

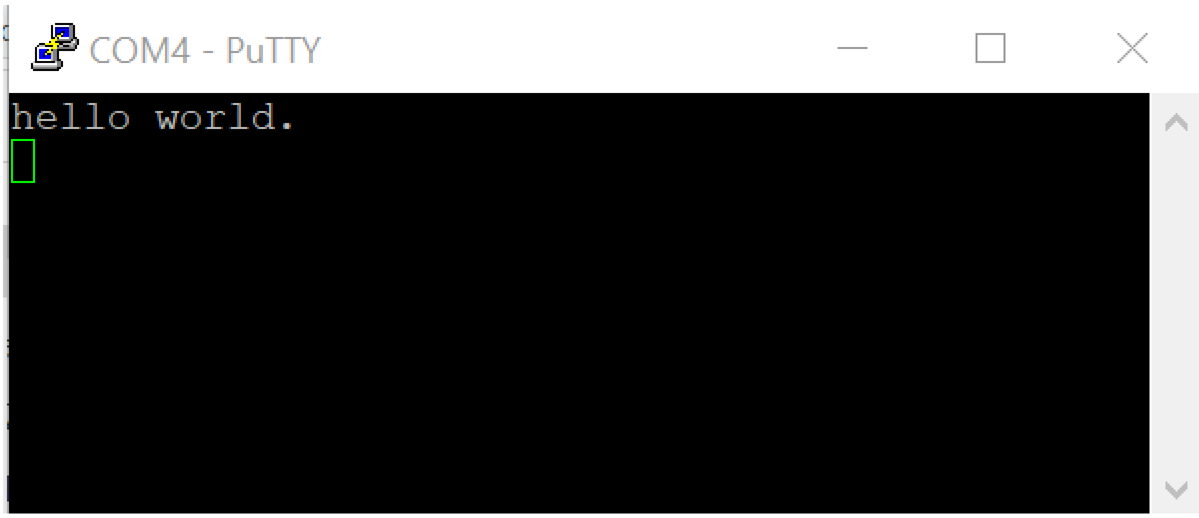


Figure 11. Text display of the hello_world demo

Note: *If the software is already running on the M core, the debugger loading image into TCM may get HardFault or a data verification issue. NXP recommends you to follow the steps above to use the debugger. Repowering the board is required to restart the debugger.*

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 GNU Arm Embedded Toolchain. 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 *MCUXpresso SDK Release Notes*.

Note: *See Section 9 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.

```
C:\Program Files (x86)\GNU Tools Arm Embedded\9 2019-q4-major
```

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

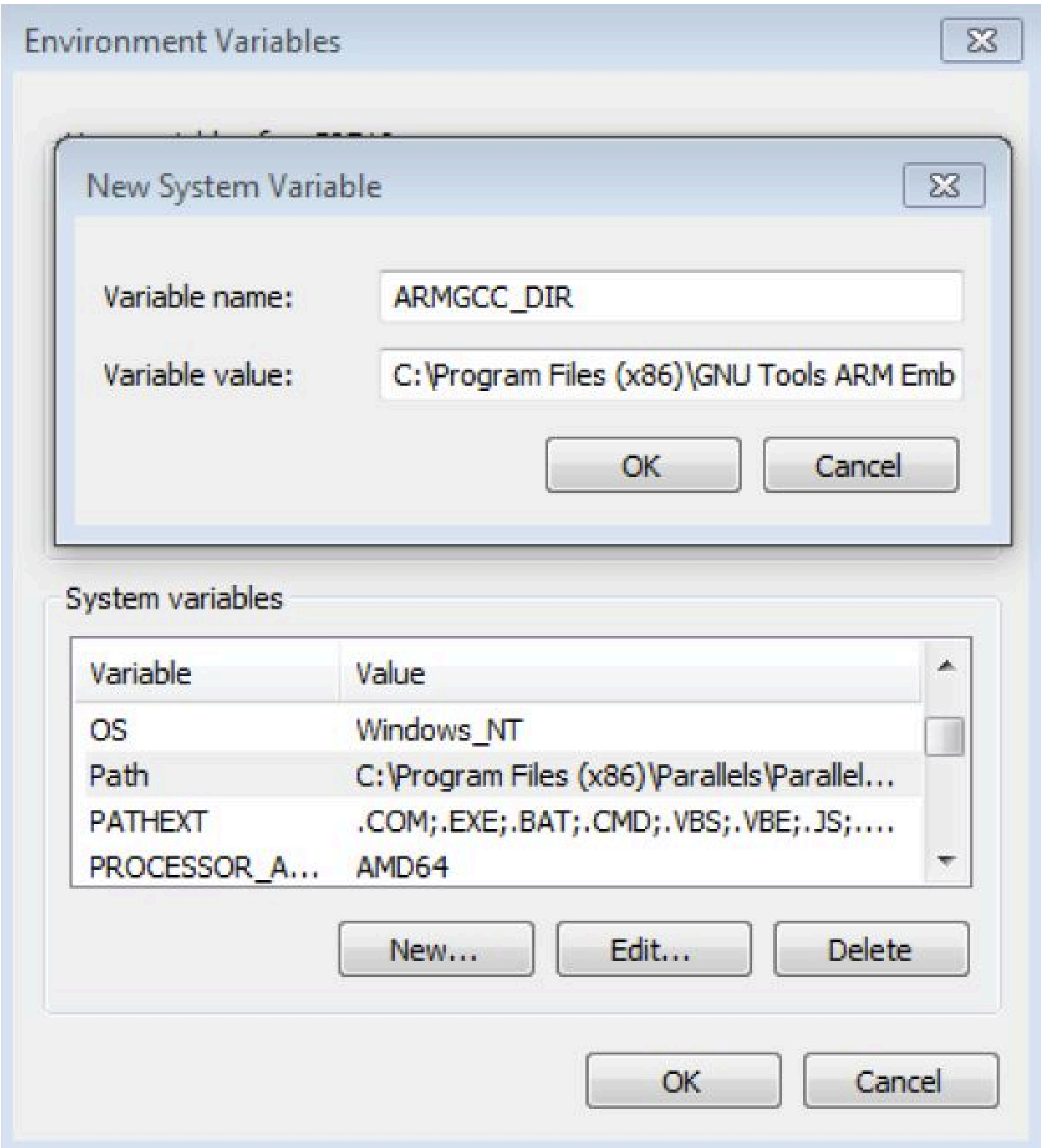


Figure 12. Add ARMGCC_DIR system variable

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**.

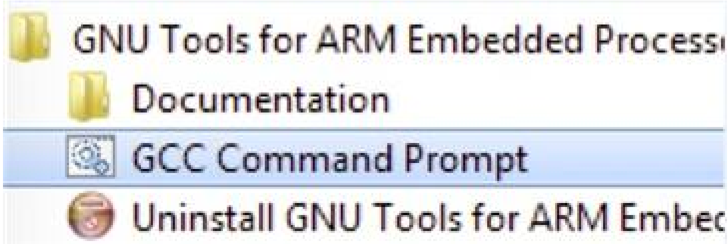


Figure 13. Launch command prompt

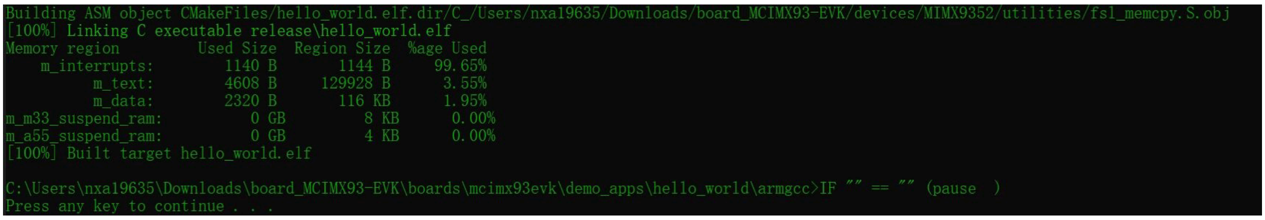
2. Change the directory to the example application project directory, which has a path similar to the following:

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

For this example, the exact path is:

```
<install_dir>/boards/mcimx93evk/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 as shown in Figure 14.



```
Building ASM object CMakeFiles/hello_world.elf.dir/C_/Users/nxa19635/Downloads/board_MCIMX93-EVK/devices/MIMX9352/utilities/fsl_memcpy.S.obj
[100%] Linking C executable release\hello_world.elf
Memory region         Used Size  Region Size  %age Used
    m_interrupts:        1140 B       1144 B     99.65%
          m_text:        4608 B     129928 B      3.55%
          m_data:        2320 B       116 KB      1.95%
m_m33_suspend_ram:          0 GB         8 KB      0.00%
m_a55_suspend_ram:          0 GB         4 KB      0.00%
[100%] Built target hello_world.elf

C:\Users\nxa19635\Downloads\board_MCIMX93-EVK\boards\mcimx93evk\demo_apps\hello_world\armgcc>IF "" == "" (pause  )
Press any key to continue . . .
```

Figure 14. 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.

1. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC USB connector.
2. Connect 12 V ~ 20 V power supply and J-Link Plus to the device.
3. Switch SW1301[3:0] to the M core boot and ensure that the image is not available on the boot source. For example, 0b1010 for MicroSD boot. Keep the SD slot empty.
4. Open the terminal application on the PC, such as PuTTY or TeraTerm, connect to the debug COM port, see Section 8, and configure the terminal with these settings:
    a. 115200 baud rate
    b. No parity
    c. 8 data bits
    d. 1 stop bit

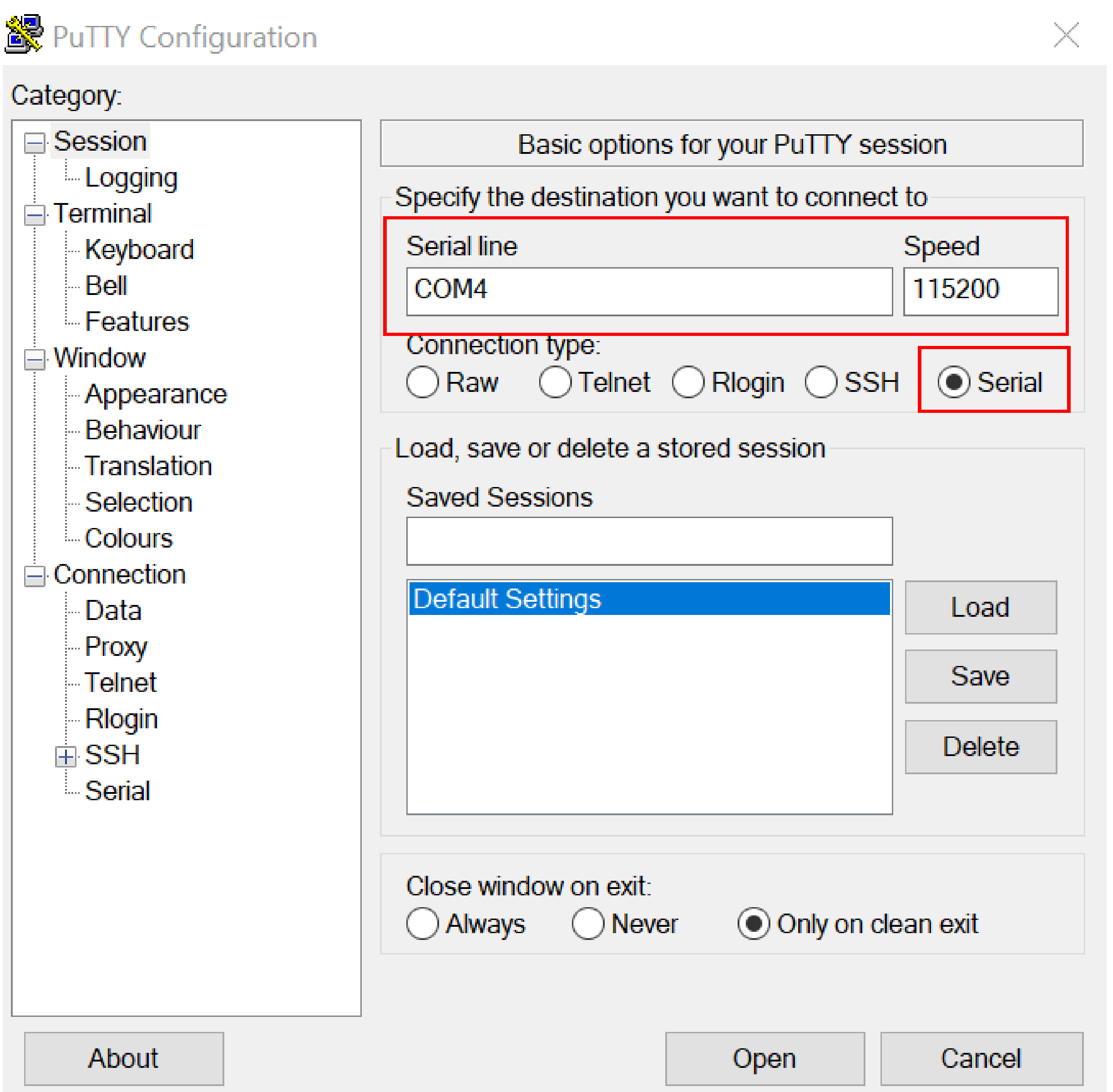


Figure 15. Terminal (PuTTY) configurations

5. Power on the board.
6. 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**.
7. Modify the settings as shown in Figure 16. The target device selection chosen for this example is MIMX9352_M33 .

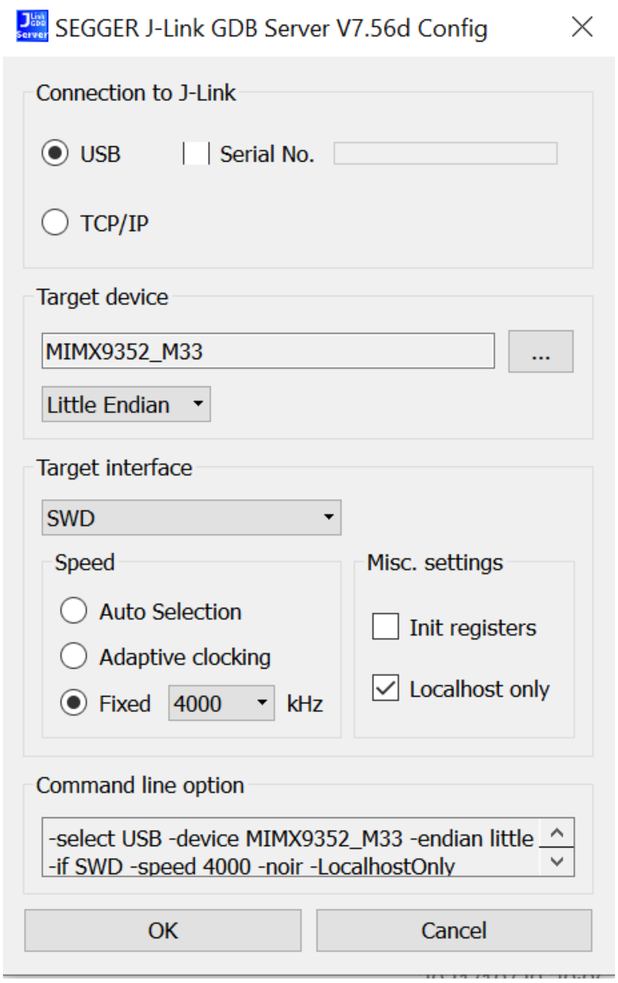


Figure 16. SEGGER J-Link GDB server configuration

8. After GDB server is running, the screen should resemble Figure 17:

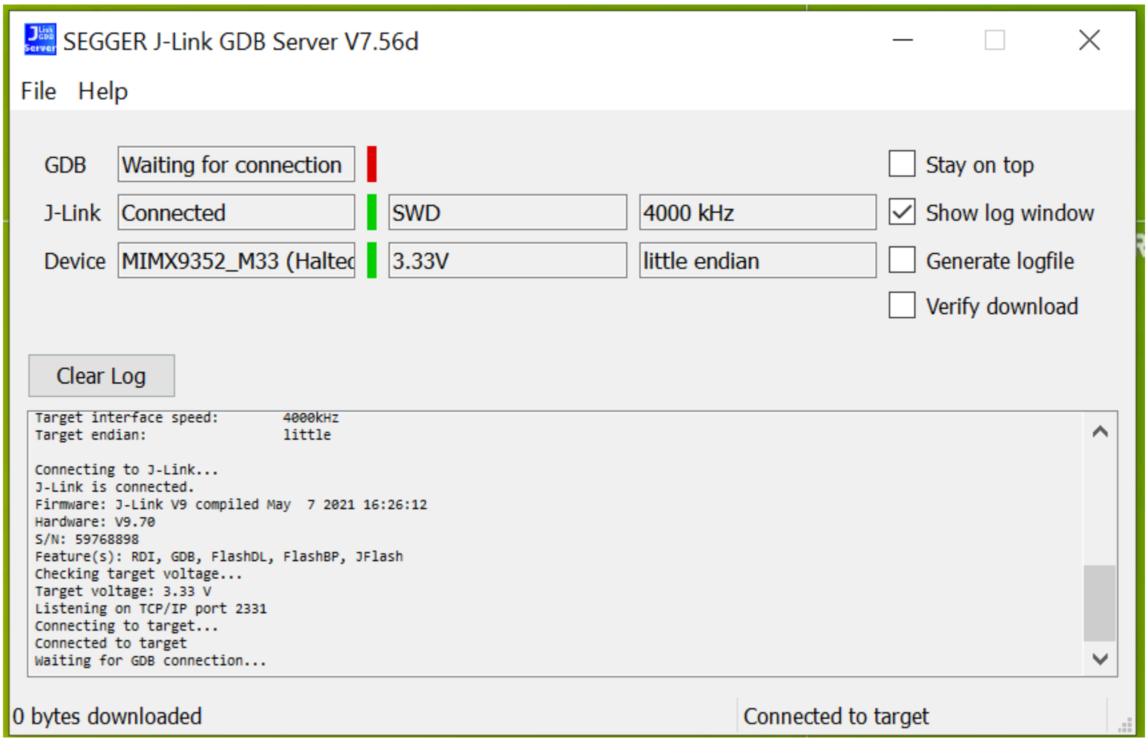


Figure 17. SEGGER J-Link GDB server screen after successful connection

9. 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**.

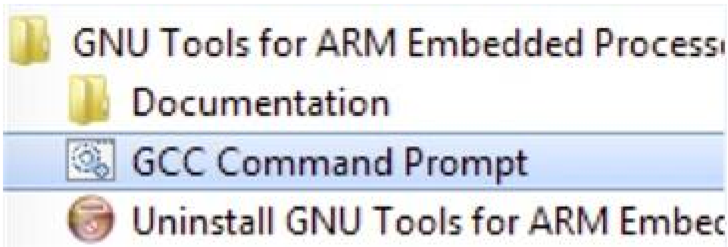


Figure 18. Launch command prompt

10. 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/mcimx93evk/demo_apps/hello_world/armgcc/debug
```

11. Run the command of `arm-none-eabi-gdb.exe <application_name>.elf`. For this example, it is `arm-none-eabi-gdb.exe hello_world.elf`.

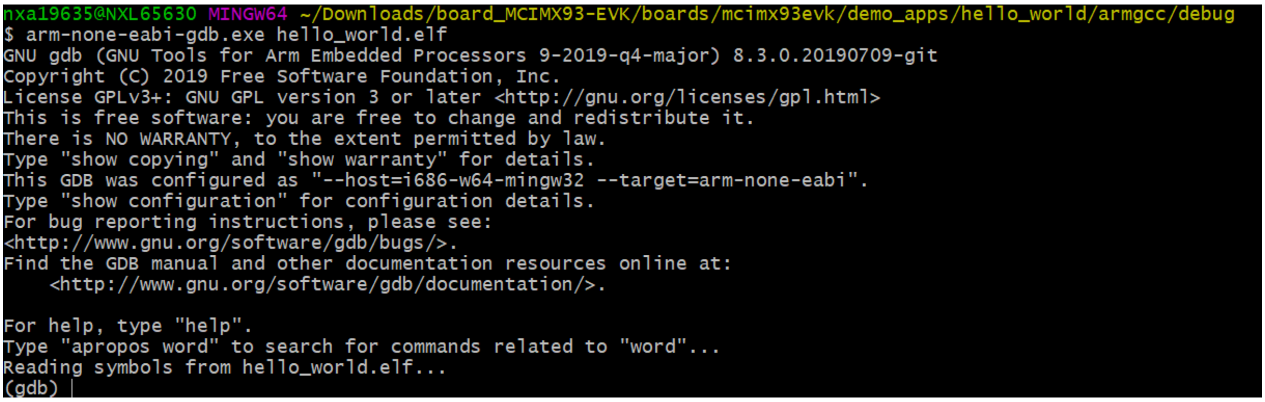


Figure 19. Run arm-none-eabi-gdb

12. Run these commands:
    a. `target remote localhost:2331`
    b. `monitor reset`
    c. `monitor halt`
    d. `load`
13. 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.

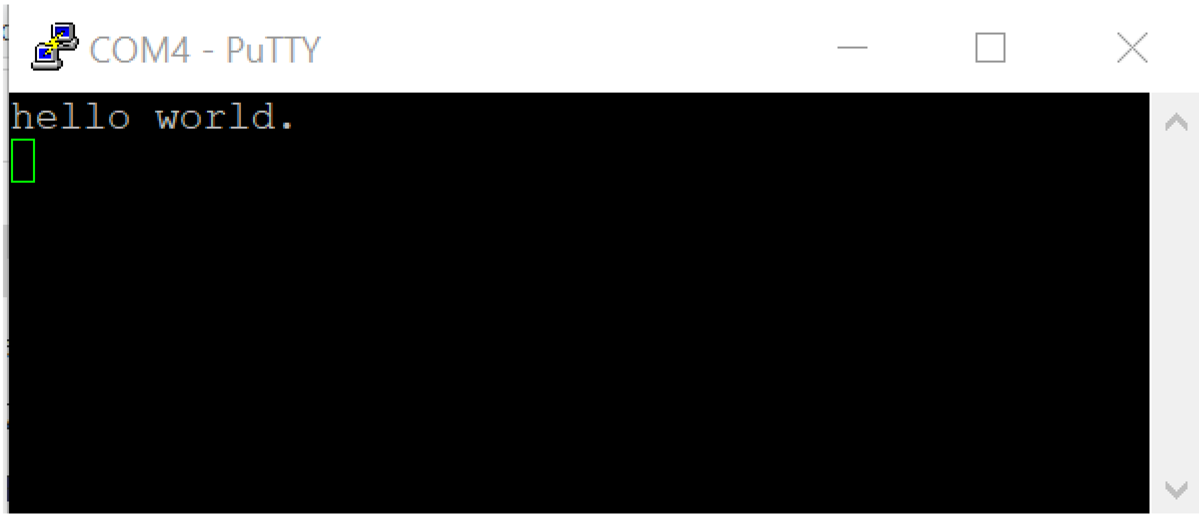


Figure 20. Text display of the hello_world demo

Note: *If the software is already running on the M core, the debugger loading image into TCM may get HardFault or a data verification issue. NXP recommends you to follow the steps above to use the debugger. Repowering the board is required to restart the debugger.*

6 Running an application by U-Boot

This section describes the steps to write a bootable SDK bin file to TCM 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 & LPT)**. Assume that the ports are COM71 - COM74. COM73 is for the debug message from the Cortex-A55 and COM74 is for the Cortex-M33. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name `/dev/ttyUSB*` to determine your debug port. Similar to Windows OS, opening both is beneficial for development.

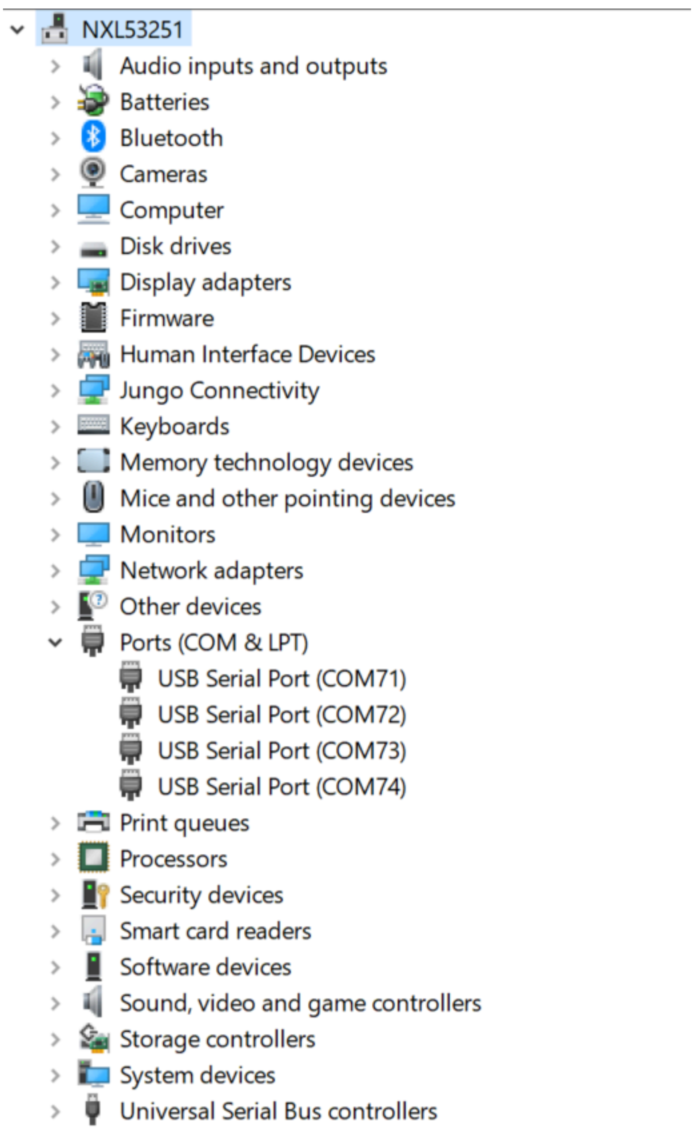


Figure 21. Determining the COM Port of target board

3. Build the application (for example, `hello_world`) to get the bin file (`hello_world.bin`).
4. Prepare an SD card with the prebuilt Linux BSP flashed and copy bin file (`hello_world.bin`) into the SD card.

5. Insert the SD card to the target board. Make sure to switch SW1301[3:0] is configured to MicroSD A core boot 0x0010.
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 A55 terminal.
8. Enter to U-Boot command line mode. You can then write the image and run it from TCM with the following commands:
   - `fatload mmc 1:1 80000000 hello_world.bin; cp.b 0x80000000 0x201e0000 0x10000;`
   - `bootaux 0x1ffe0000 0`
9. The `hello_world` application is now running and a banner is displayed on the M33 terminal. If this is not true, check your terminal settings and connections.

Figure 22. Hello world demo running on Cortex-M33 core

7 Program flash.bin to SD/eMMC with UUU

This section describes the steps to use the UUU to run the example applications provided in the MCUXpresso SDK. Download the flash.bin to `emmc/sd` with UUU. The `hello_world` demo application targeted for the i.MX 93 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 Universal Upgrade Utility

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 accessed from corresponding Linux BSP release. 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.

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 [SW1301[3:0] = 0011(Single Boot Mode)/1011(Low Power Boot Mode)].
2. Connect the development platform to your PC via USB cable between the DBG USB connector (J1401) and the PC USB connector.
3. Connect J403 (USB1) to PC USB connector for downloading.
4. The PC recognizes the i.MX 93 device as (VID:PID)=(1FC9:0146), as shown in Figure 23.

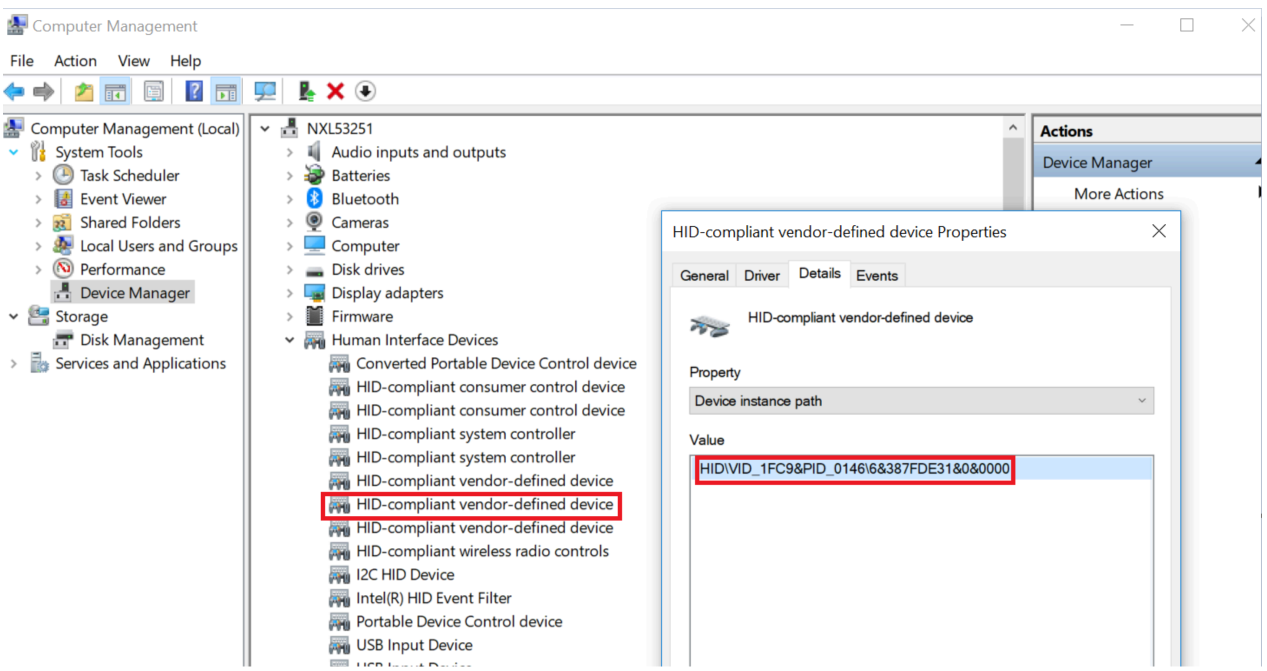


Figure 23. Device as shown in Device Manager

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 93 EVK board as an example, the workspace is located in:

```
<install_dir>/boards/mcimx93evk/demo_apps/hello_world/iar/hello_world.eww
```

2. Select the desired build target from the drop-down. For this example, select **hello_world – debug**.
3. To build the demo application, click **Make**.

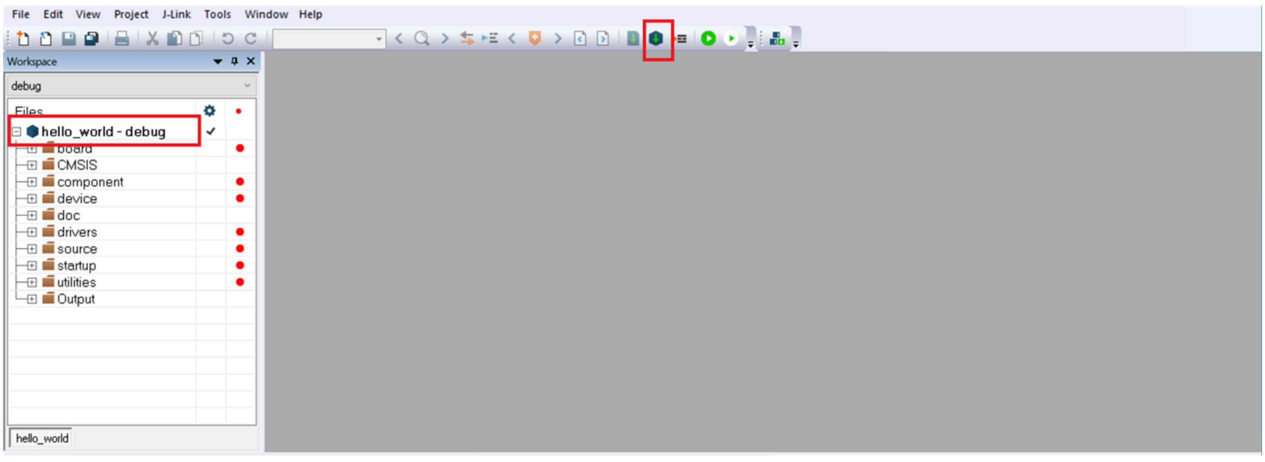


Figure 24. Demo build target selection

4. The build completes without errors.

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 DBG USB connector (J1401) and the PC. It provides console output while using UUU.
2. Connect the J403 (USB1) 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 Section 8). Configure the terminal with these settings:
   a. 115200 baud rate
   b. No parity
   c. 8 data bits
   d. 1 stop bit

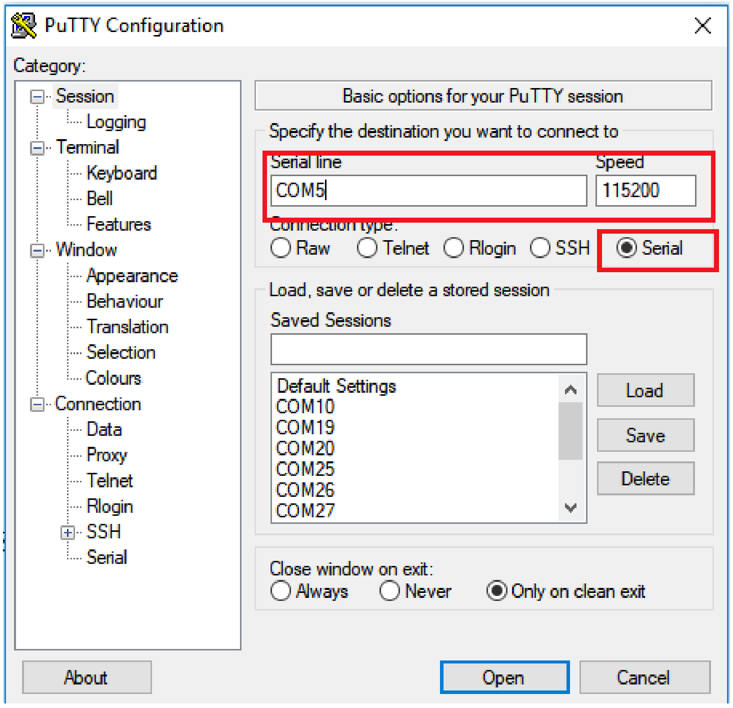


Figure 25. Terminal (PuTTY) configuration

4. Get the boot images and the imx-mkimage source repository from corresponding Linux BSP release. The boot images required to be put into imx-mkimage/i.MX9 are:
   ```
   - u-boot-imx93evk.bin-sd (rename to u-boot.bin)
   - u-boot-spl.bin-imx93evk-sd (rename to u-boot-spl.bin)
   - bl31-imx93.bin (rename to bl31.bin)
   - mx93a0-ahab-container.img
   - lpddr4_dmem_1d_v202201.bin
   - lpddr4_dmem_2d_v202201.bin
   - lpddr4_imem_1d_v202201.bin
   - lpddr4_imem_2d_v202201.bin
   ```
5. Copy binary generated by IAR build into imx-mkimage/i.MX9, and rename it to m33_image.bin.
6. Make flash.bin with imx-mkimage.
   ```
   make SOC=iMX9 flash_singleboot_m33 (for single boot mode)
   ```
   or
   ```
   make SOC=iMX9 flash_lpboot (for low power boot mode)
   ```
7. Power on the board.
8. Type the UUU command to the flash image.
   ```
   uuu -b emmc flash.bin (for single boot on eMMC)
   uuu -b sd flash.bin (for single boot on SD)
   ```
   For low power boot, a single boot flash.bin is needed besides the target flash.bin.
   ```
   uuu -b emmc <singleboot flash.bin> flash.bin (for lowpower boot on eMMC)
   uuu -b sd <singleboot flash.bin> flash.bin (for lowpower boot on SD)
   ```
   The UUU puts the platform into fast boot mode and automatically flashes the target bootloader to emmc/sd. The command line and fast boot console is as shown in Figure 26.

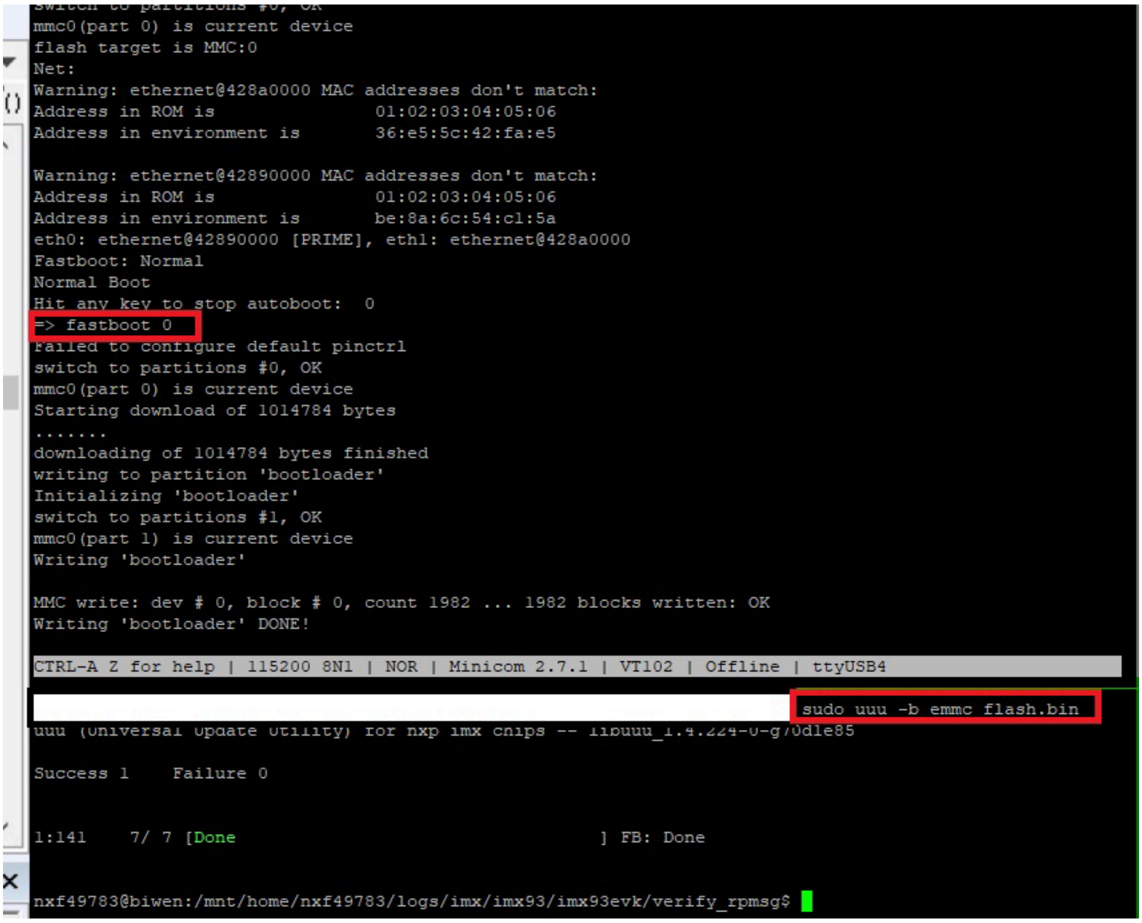


Figure 26. Command line and fast boot console output when executing UUU

9. Then, power off the board and change the boot mode to the corresponding one.
   - For single-boot mode:
     – when boot device is emmc, then `SW1301[3:0] = 0000`;
     – when boot device is sd, then `SW1301[3:0] = 0010`.
   - For low-power boot mode:
     – when boot device is emmc, then `SW1301[3:0] = 1000`;
     – when boot device is sd, then `SW1301[3:0] = 1010`.
10. Power on the board again.

8 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 27.

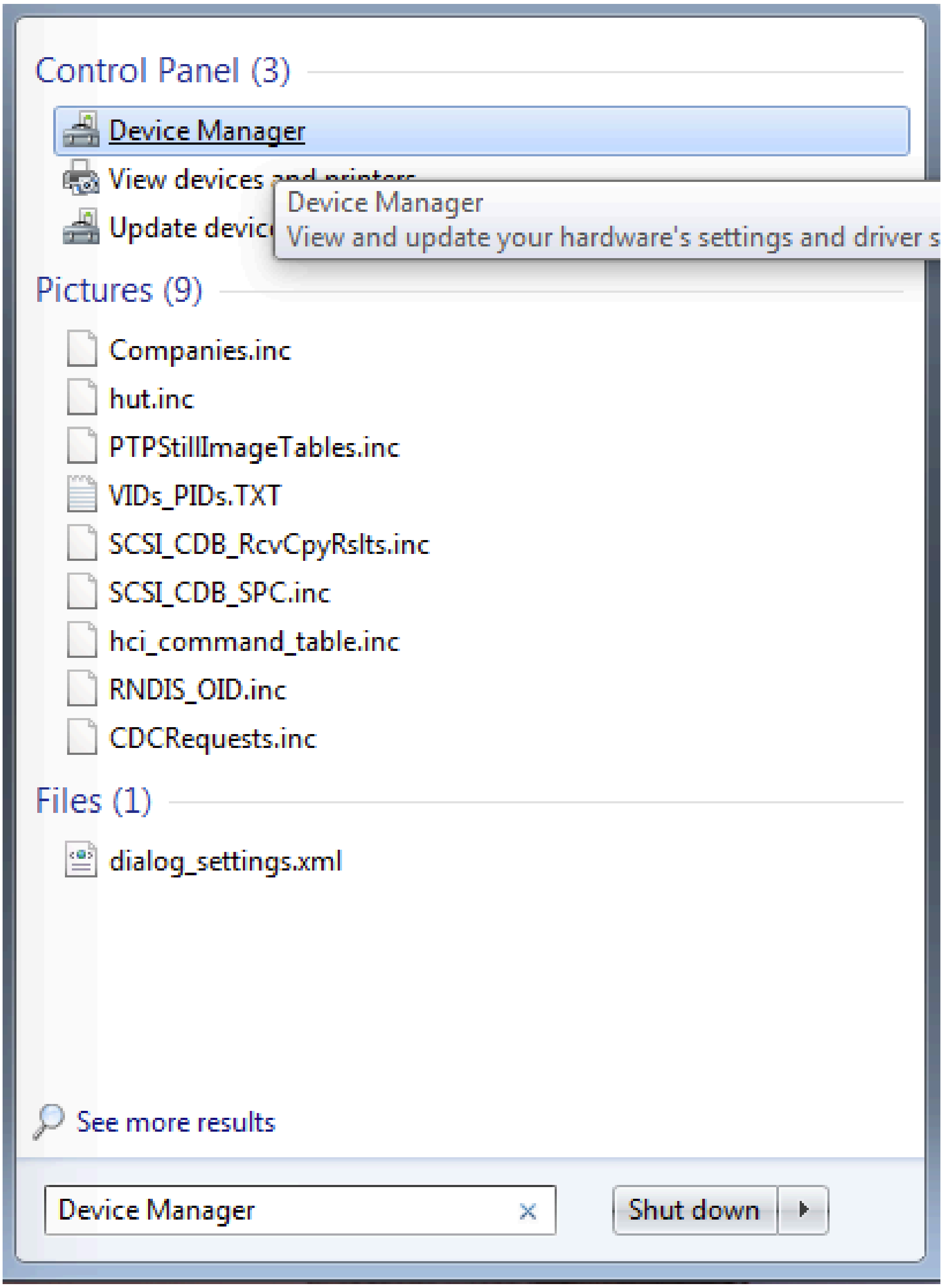


Figure 27. Device manager

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.
   a. **USB-UART** interface

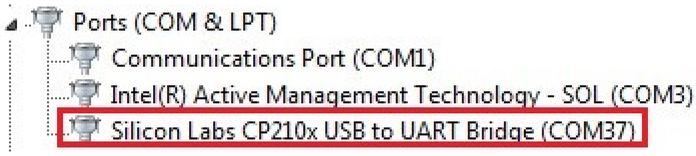


Figure 28. USB-UART interface

9 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 **mingw32-base** and **msys-base** are selected under **Basic Setup**.

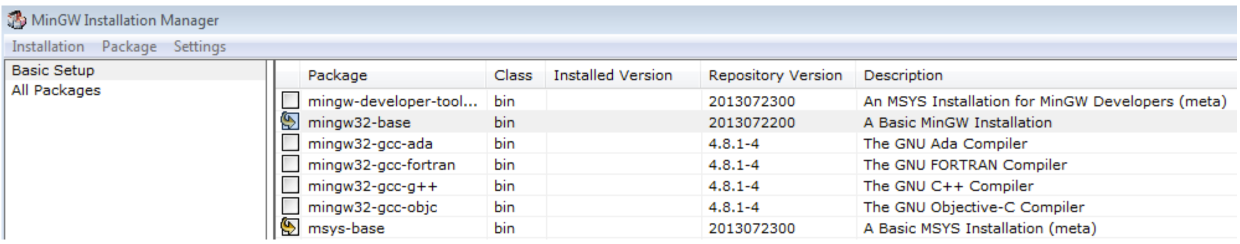


Figure 29. Setup MinGW and MSYS

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

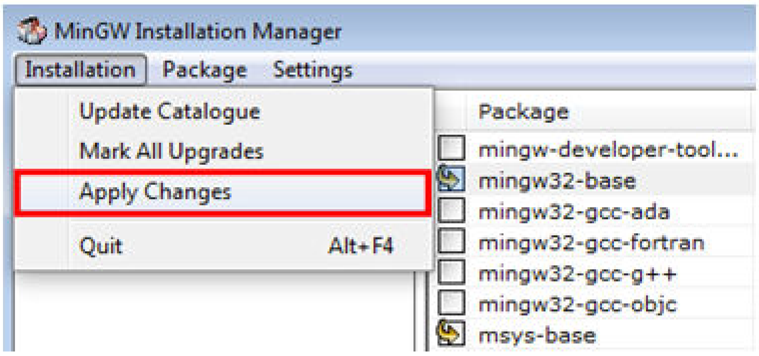


Figure 30. Complete MinGW and MSYS installation

5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel**->**System and Security**->**System**->**Advanced System Settings** in the **Environment Variables...** section. The path is: <mingw_install_dir>\bin.
   Assuming the default installation path, `C:\MinGW`, an example is as shown in Figure 31. 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.*

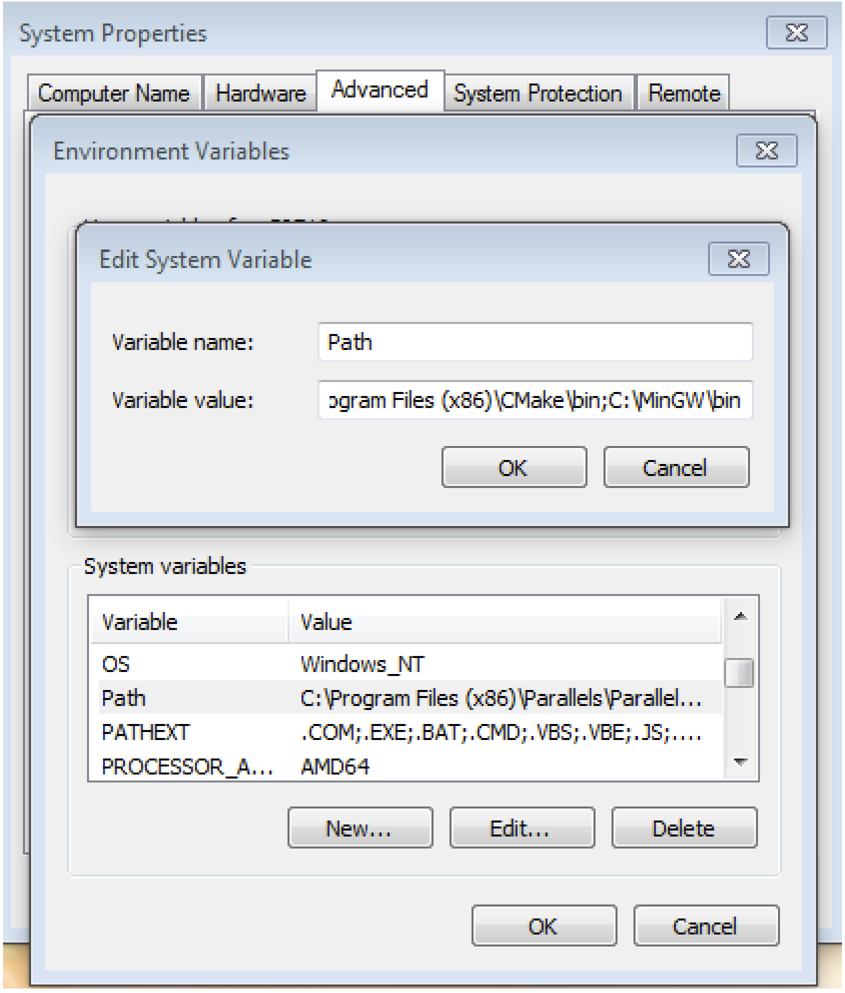


Figure 31. Add Path to systems environment

- 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.

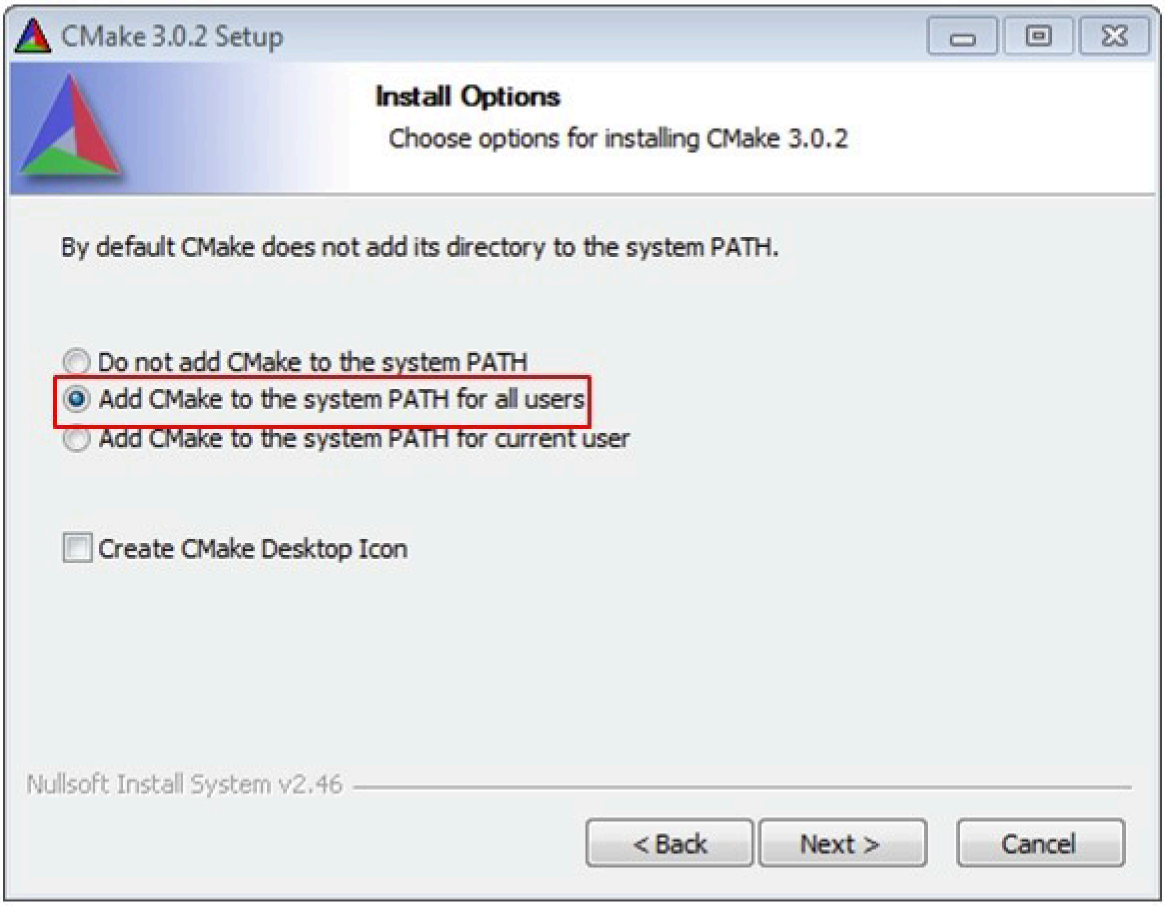


Figure 32. Install CMake

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

10 Revision history

This table summarizes revisions to this document.

Table 2. Revision history

Revision number	Date	Substantive changes
2.12.0	20 July 2022	Updated for MCUXpresso SDK v2.12.0
2.12.1	30 September 2022	Updated for MCUXpresso SDK v2.12.1
2.13.0	16 December 2022	Updated for MCUXpresso SDK v2.13.0
2.13.1	31 March 2022	Updated for MCUXpresso SDK v2.13.1

11 Legal information

11.1 Definitions

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

11.2 Disclaimers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

11.3 Trademarks

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

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

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

Contents

Date of release: 31 March 2023
Document identifier: MCUXSDKMCIMX93EVKGSUG