

Getting Started with MCUXpresso SDK for i.MX 6UltraLite Derivatives

1 Overview

The MCUXpresso Software Development Kit (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 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 also contains RTOS kernels, a USB host and device stack, and various other middleware to support rapid development on devices.

For supported toolchain versions, see the *MCUXpresso SDK Release Notes Supporting i.MX6 UltraLite Derivatives* (document MCUXSDKIMX6ULRN)

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

Contents

1	Overview.....	1
2	MCUXpresso SDK Board Support Folders.....	2
3	Run a demo application using IAR.....	4
4	Run a demo using ARM® GCC.....	7
5	Run a demo using Manufacturing Tool (MFGTool).....	17
6	Appendix A - How to determine COM port.....	18



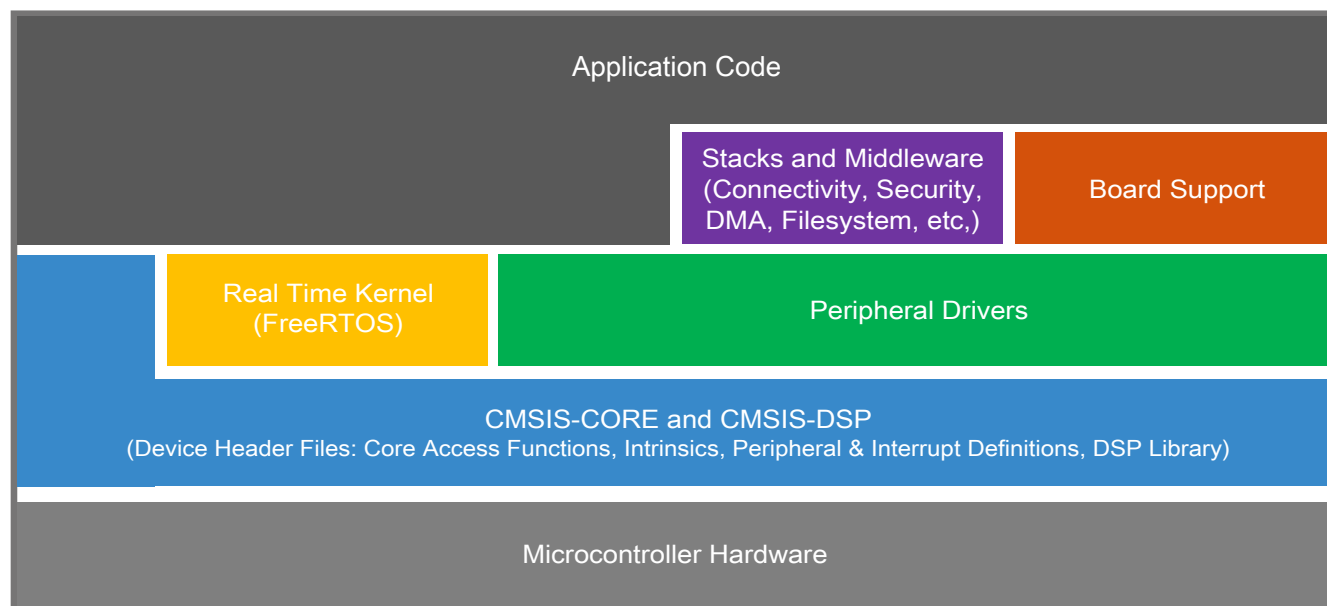


Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK Board Support Folders

MCUXpressoSDK board support provides example applications for NXP development and evaluation 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):

- **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, ADC 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
- **usb_examples:** Applications that use the USB host/device/OTG 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. We'll discuss the `hello_world` example (part of the `demo_apps` folder), but the same general rules apply to any type of example in the <board_name> folder.

In the `hello_world` application folder you see this:

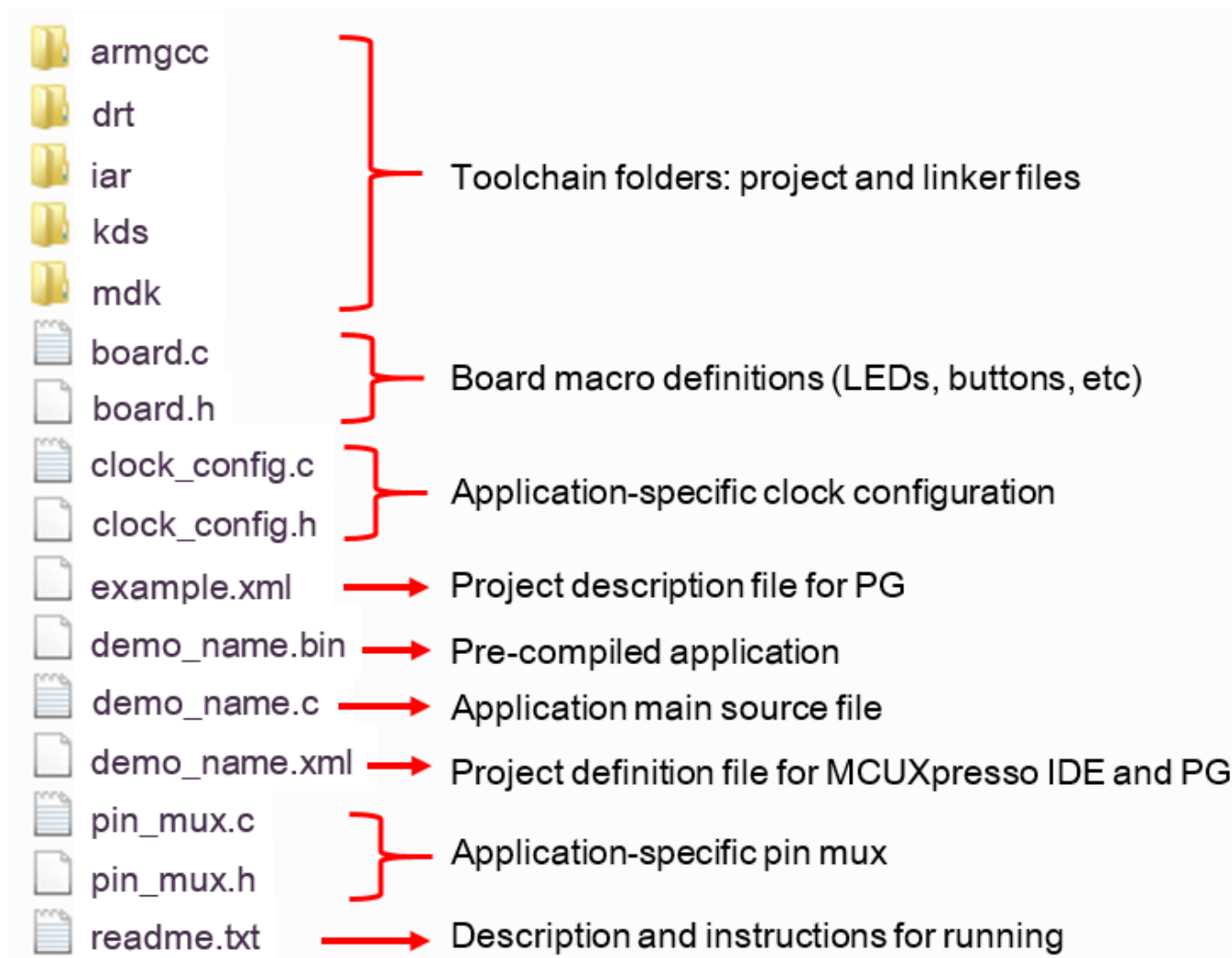


Figure 2. Application folder structure

All files in the application folder are specific to that example, so it's very easy to copy-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, there are a variety of source files 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.

Run a demo application using IAR

For examples containing middleware/stacks or a RTOS, there are references to the appropriate source code. Middleware source files are located in the *middleware* folder and RTOSes are in the *rtos* folder. Again, the core files of each of these are shared, so modifying them could have potential impacts on other projects that depend on them.

3 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 MCIMX6UL-EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

3.1 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 MCIMX6UL-EVK hardware platform as an example, the *hello_world* workspace is located in

`<install_dir>/boards/evkmcmx6ul/demo_apps/hello_world/iar/hello_world.eww`

2. Select the desired build target from the drop-down. There are four project configurations (build targets) supported for most MCUXpresso SDK projects:
 - **DDR_debug** - Compiler optimization is set to low, and debug information is generated for the executable. The linker file is RAM linker, where all image sections are put in external DDR RAM. This target should be selected for development and debug.
 - **DDR_release** - Compiler optimization is set to high, and debug information is not generated. The linker file is RAM linker, where all image sections are put in external DDR RAM. This target should be selected for final application deployment.
 - **Debug** - Project configuration is same as DDR_debug target. The linker file is flash linker, where text section is put in external QSPI flash.
 - **Release** - Project configuration is same as DDR_release target. The linker file is flash linker, where text section is put in external QSPI flash.

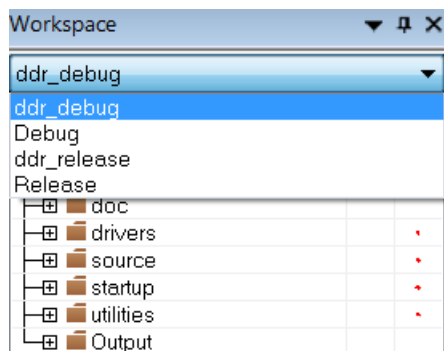


Figure 3. Demo build target selection

3. For this example, select the "hello_world - Debug" target. To build the demo application with flash linker file, click the "Make" button, highlighted in red below.

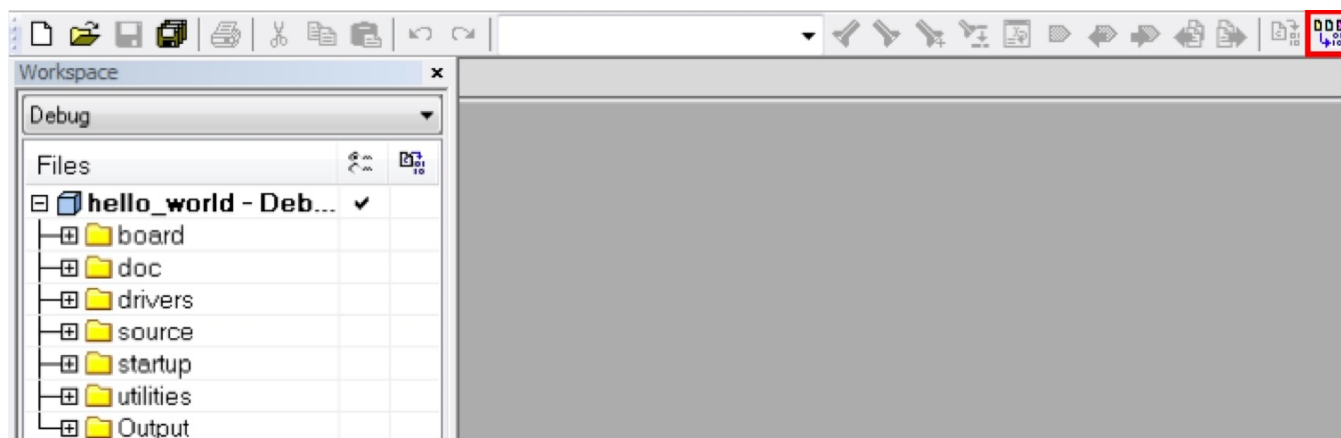


Figure 4. Build the demo application

4. The build completes without errors.

3.2 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 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 or 9600 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

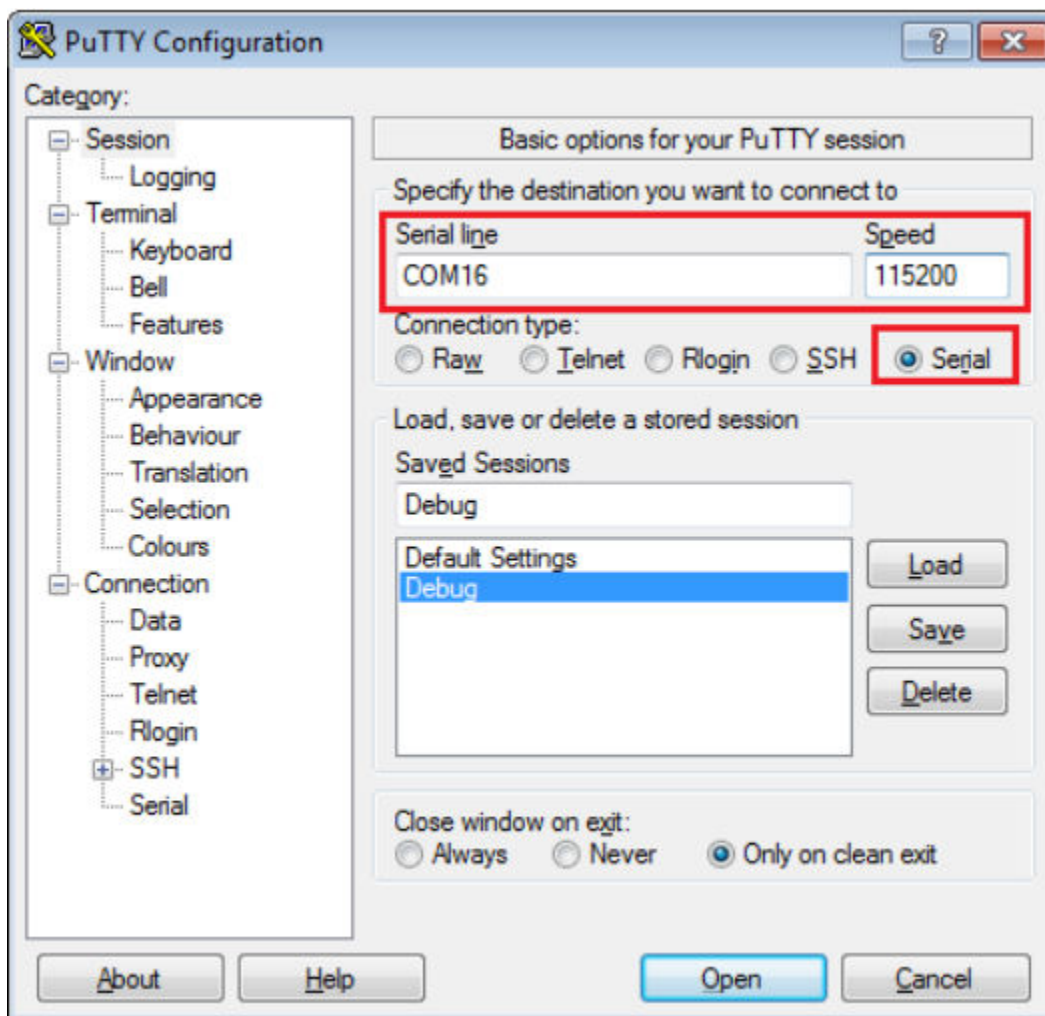


Figure 5. Terminal (PuTTY) configuration

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



Figure 6. Download and Debug button

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

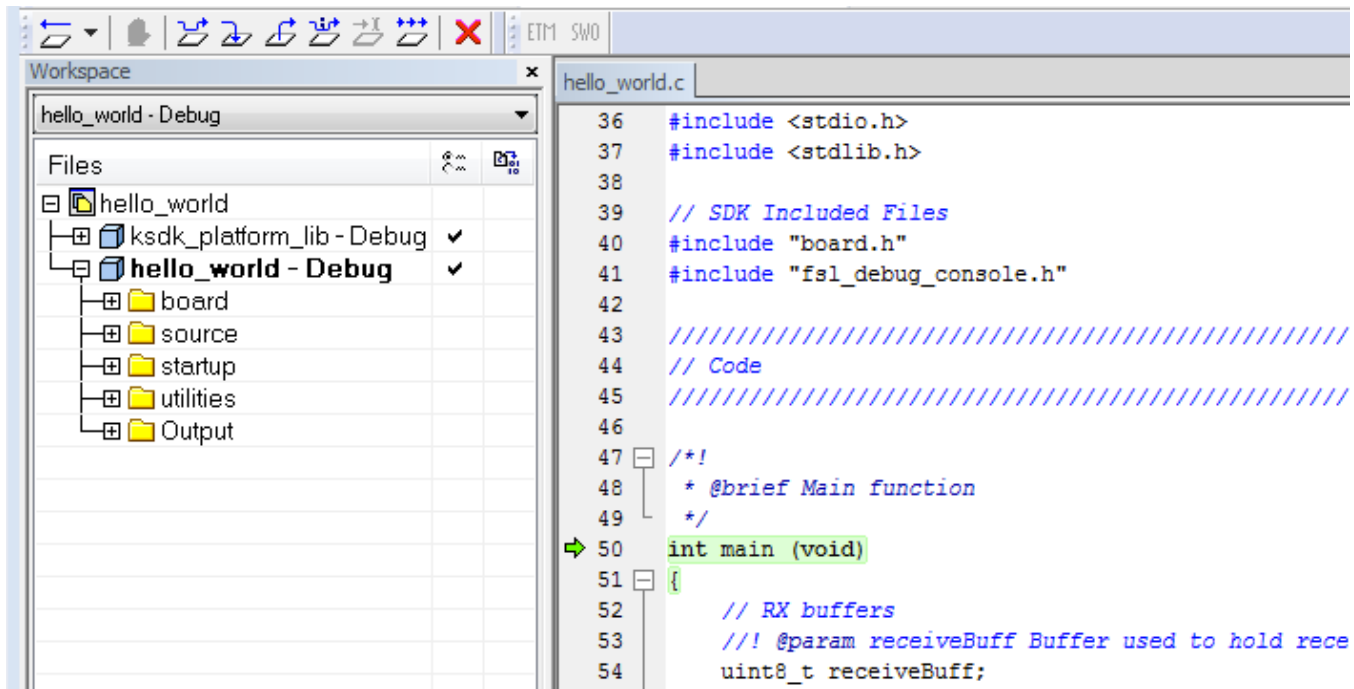


Figure 7. Stop at main() when running debugging

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

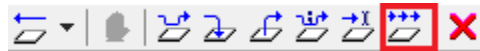


Figure 8. Go button

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

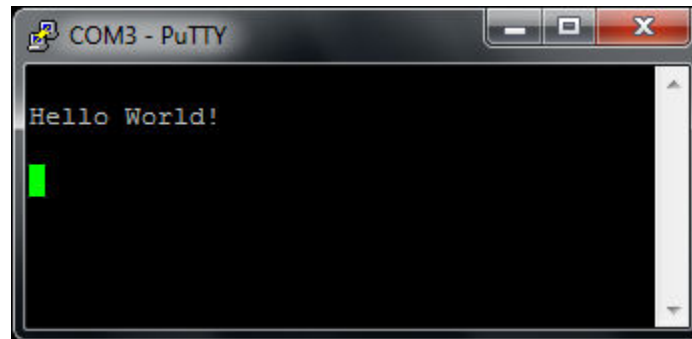


Figure 9. Text display of the hello_world demo

NOTE

If you want to debug QSPI XIP program, use J-Link V6.14 or newer.

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 the MCIMX6UL-EVK hardware platform is used as an example, though these steps can be applied to any board, demo or example application in the MCUXpresso SDK.

4.1 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. There are many ways to use ARM GCC tools, but this example focuses on a Windows operating system environment.

Though not discussed here, ARM GCC tools can also be used with Linux OS. If you want to build an ARM GCC demo under Linux OS, in addition to installing the CMake and ARM GCC toolchain, make sure the ARMGCC_DIR environment variable has been added, and points to the ARM GCC Embedded tool chain installation path.

4.1.1 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).

4.1.2 Install MinGW (only required on Windows OS)

The Minimalist GNU for Windows (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 MCUXpresso 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.

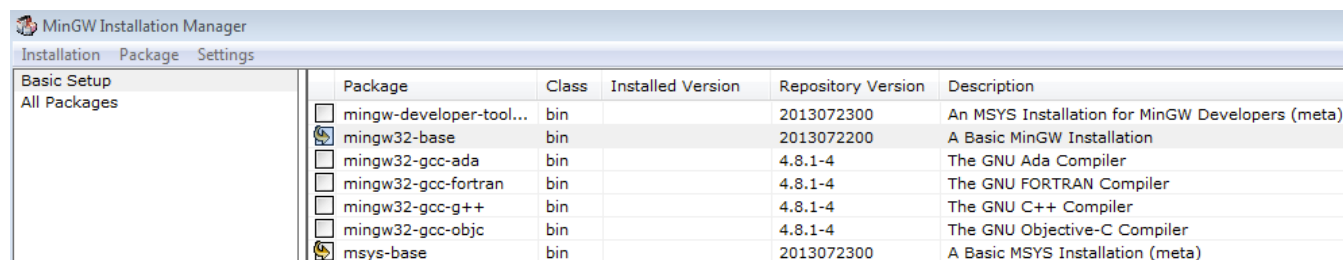


Figure 10. Setup MinGW and MSYS

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

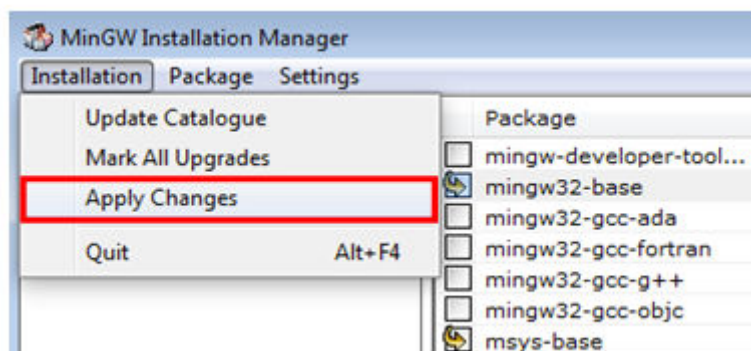


Figure 11. Complete MinGW and MSYS installation

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

`<mingw_install_dir>\bin`

Assuming the default installation path, C:\MinGW, an example is shown below. If the path is not set correctly, the toolchain does not work.

NOTE

If you have "C:\MinGW\msys\x.x\bin" in your PATH variable (as required by KSDK 1.0.0), remove it to ensure that the new GCC build system works correctly.

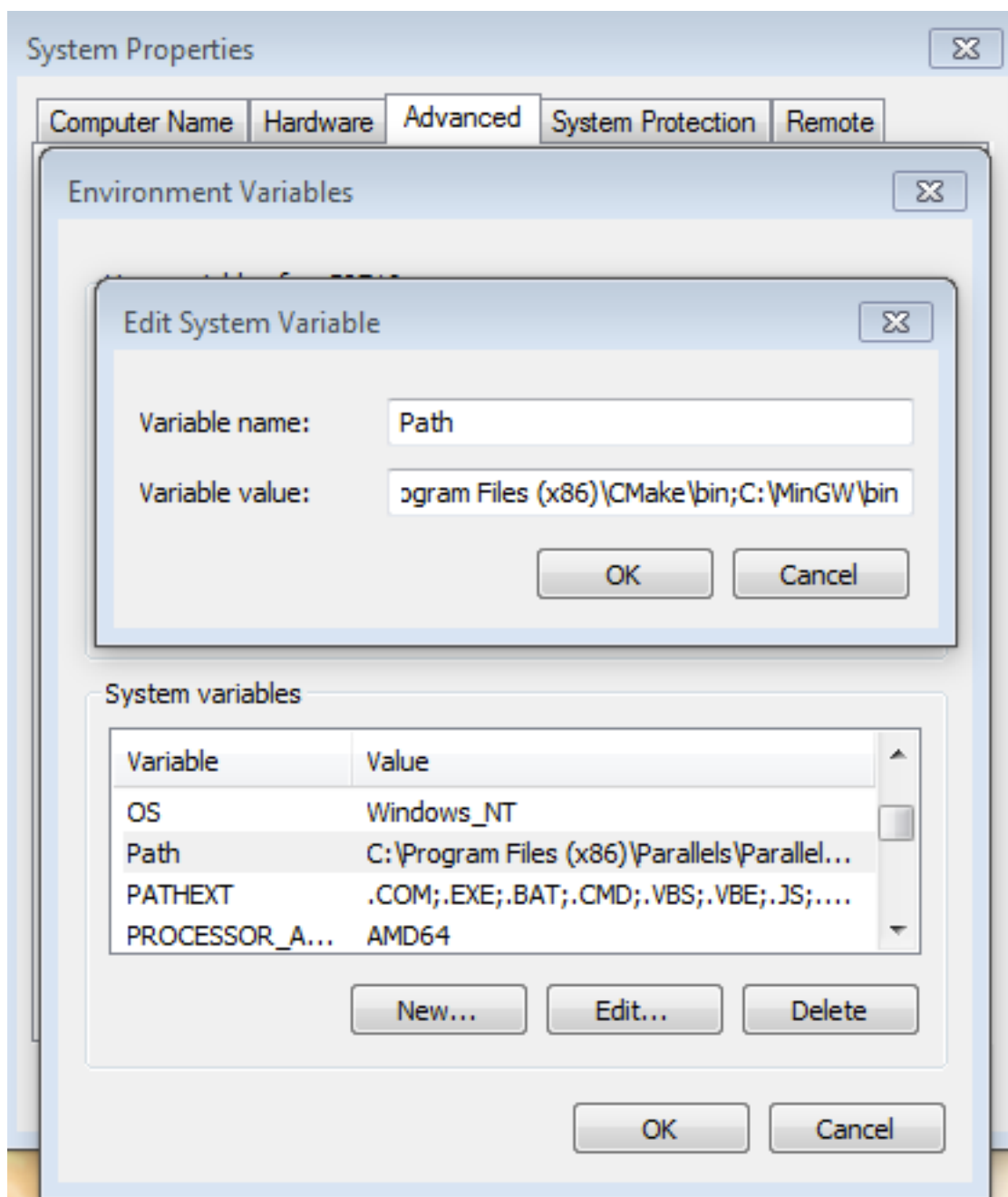


Figure 12. Add Path to systems environment

4.1.3 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, which, for this example, is:

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

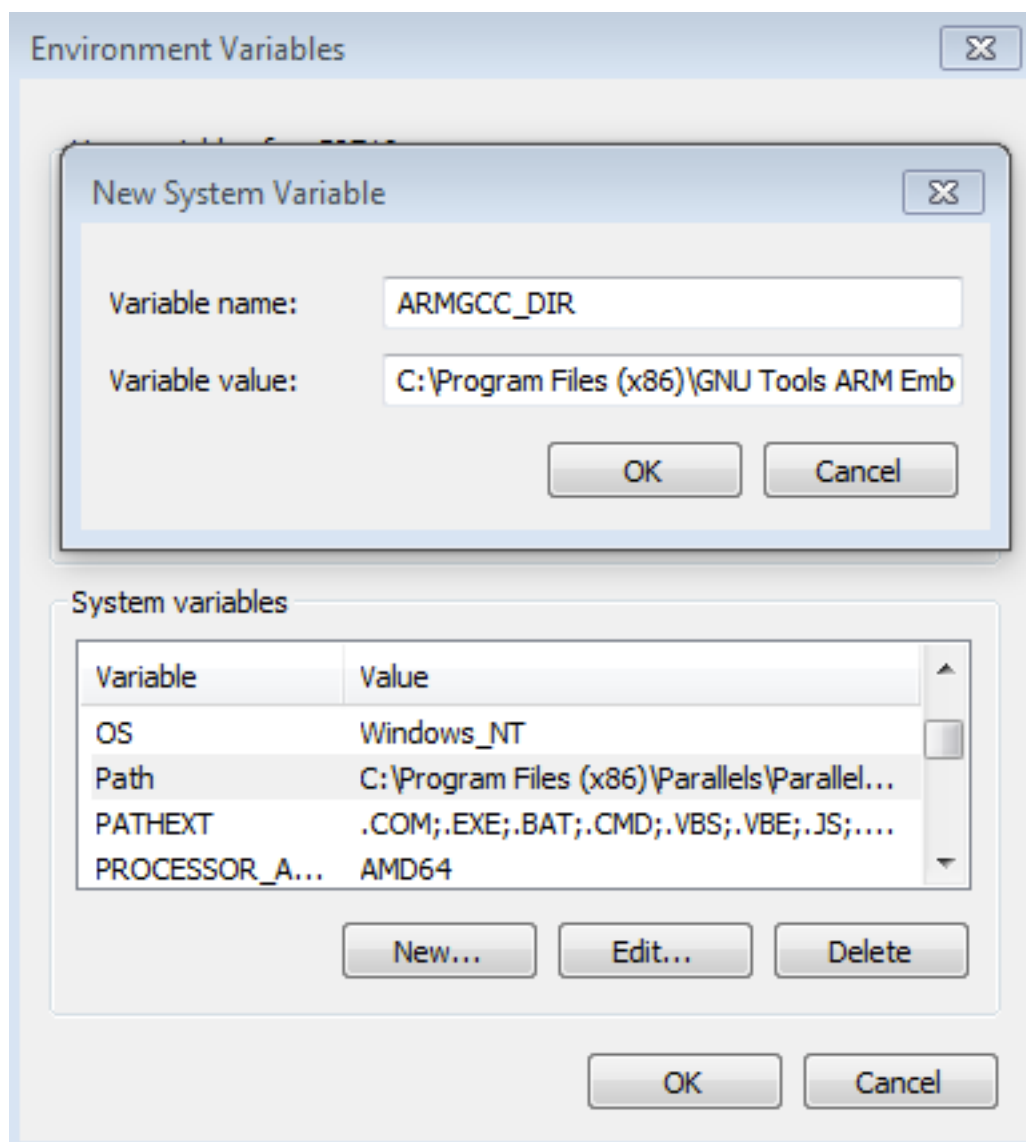


Figure 13. Add ARMGCC_DIR system variable

4.1.4 Install 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.

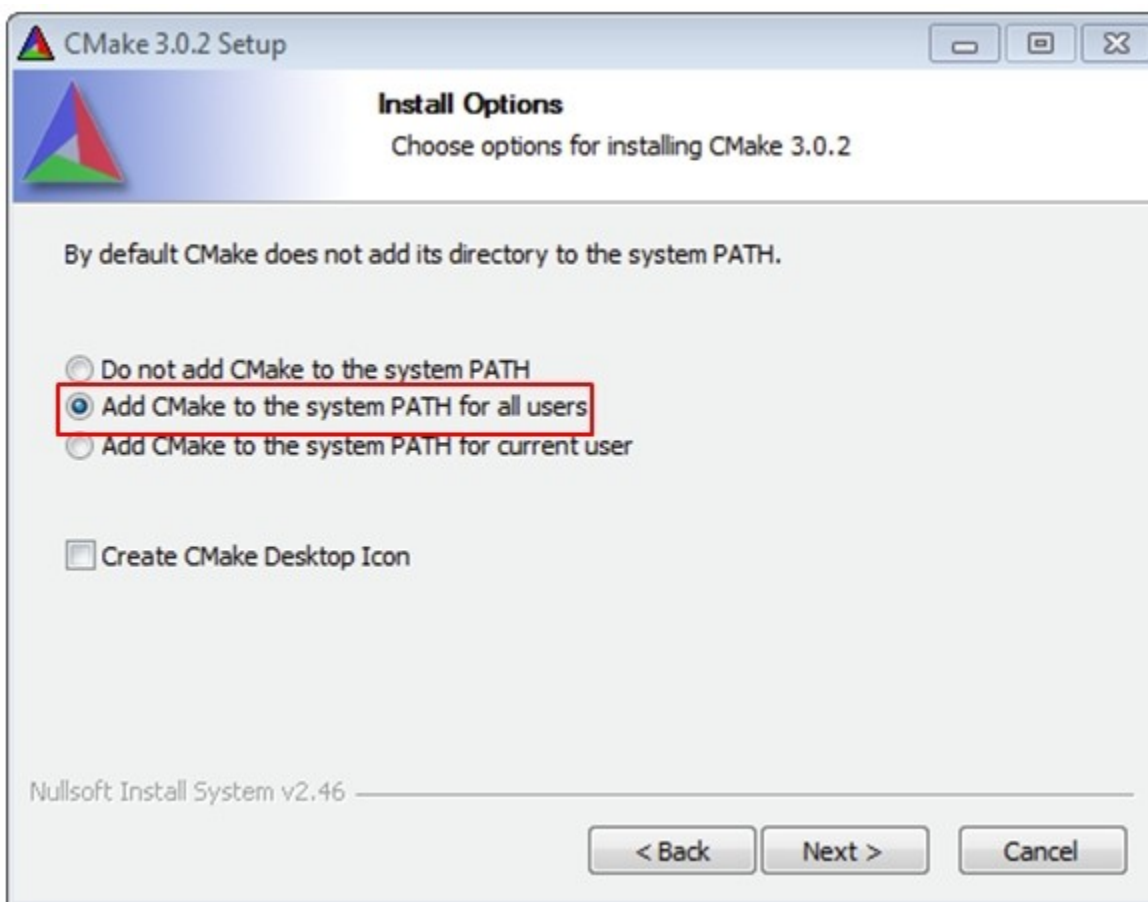


Figure 14. Install CMake

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

4.2 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”.

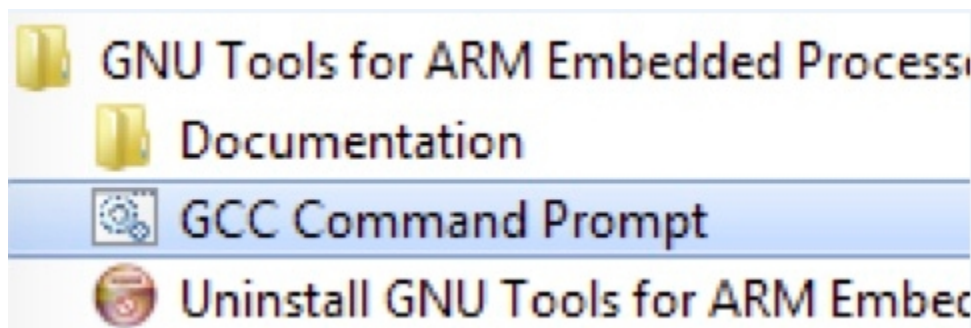


Figure 15. Launch command prompt

2. Change the directory to the example application project directory, which has a path like this:

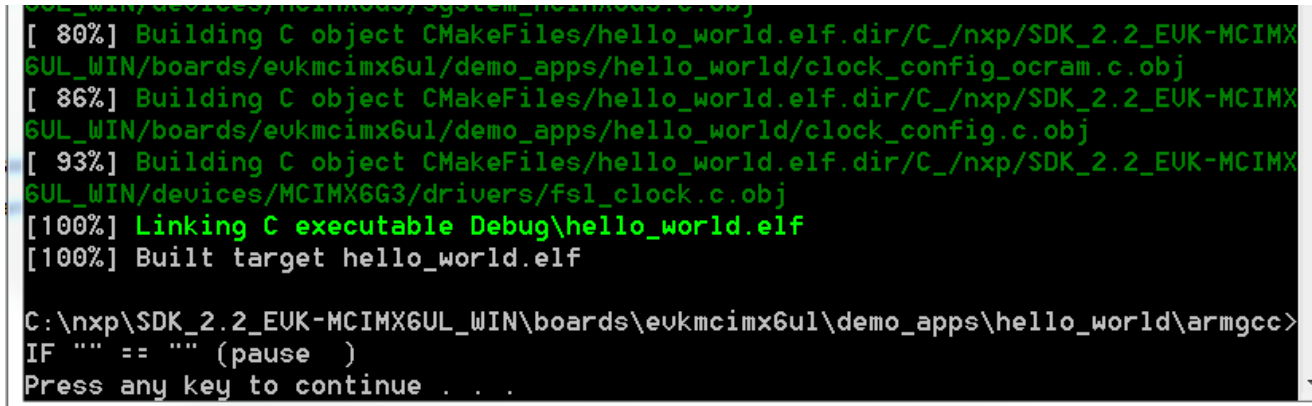
`<install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc`

For this example, the exact path is: `<install_dir>/examples/evkmcmx6ul/demo_apps/hello_world/armgcc`

NOTE

To change directories, use the 'cd' command.

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. (On Linux platform, there is corresponding shell script "build_debug.sh" to perform the build). The output is shown in this figure:



```

[ 80%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.2_EUK-MCIMX6UL_WIN/boards/evkmcmx6ul/demo_apps/hello_world/clock_config_ocram.c.obj
[ 86%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.2_EUK-MCIMX6UL_WIN/boards/evkmcmx6ul/demo_apps/hello_world/clock_config.c.obj
[ 93%] Building C object CMakeFiles/hello_world.elf.dir/C:/nxp/SDK_2.2_EUK-MCIMX6UL_WIN/devices/MCIMX6G3/drivers/fsl_clock.c.obj
[100%] Linking C executable Debug\hello_world.elf
[100%] Built target hello_world.elf

C:\nxp\SDK_2.2_EUK-MCIMX6UL_WIN\boards\evkmcmx6ul\demo_apps\hello_world\armgcc>
IF "" == "" (pause )
Press any key to continue . . .
  
```

Figure 16. hello_world demo build successful

NOTE

There are several other batch commands in the same folder to build different target binary. See Section 3.1, "Build an example application", to see what those four project configurations (build targets) mean.

4.3 Run an example application

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

1. This board supports the J-Link debug probe. Before using it, install SEGGER software, which can be downloaded from <http://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 the 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 serial port number (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
 - a. 115200 or 9600 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

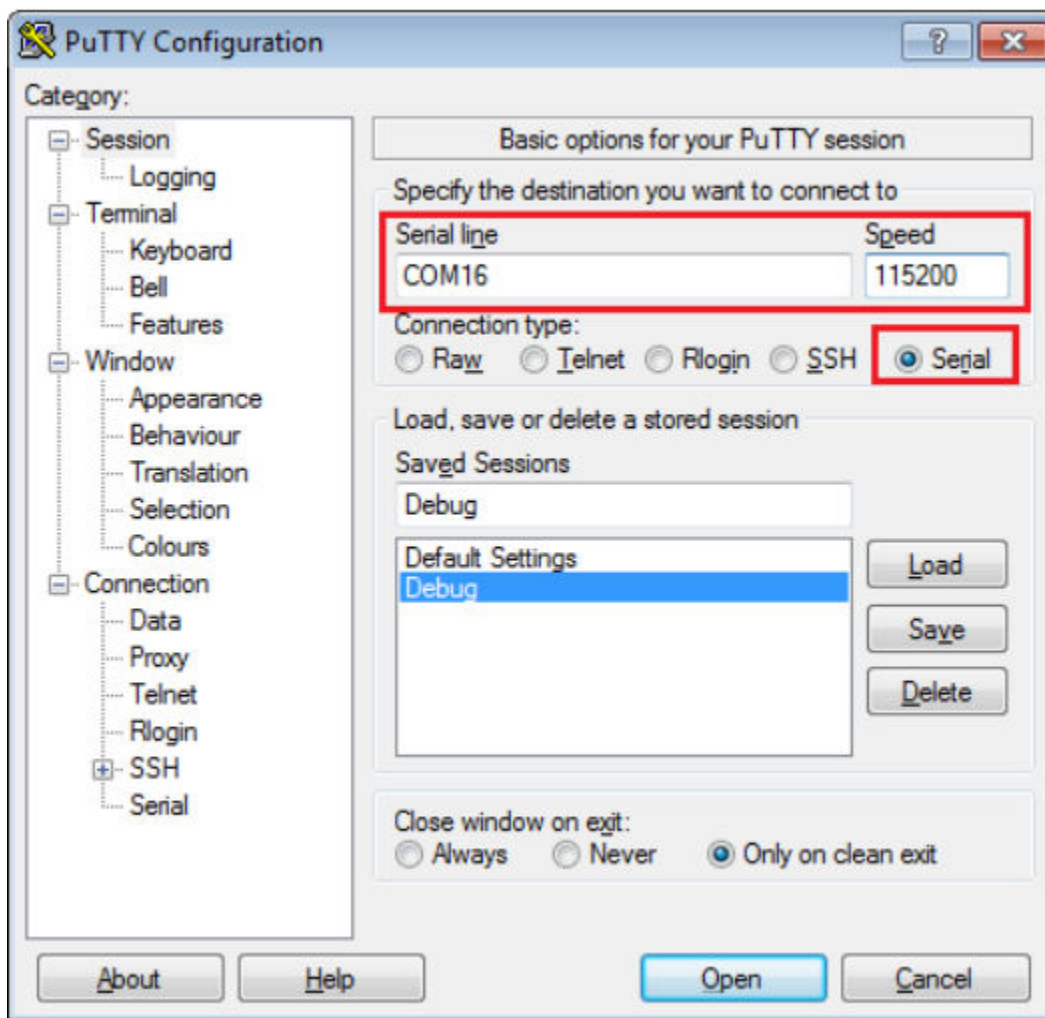


Figure 17. Terminal (PuTTY) configurations

4. Since it is required to run the `ddr_init.jlinkscript` to initialize DDR before loading the example application, use command window to run `JLinkGDBServer` with the `-scriptfile` option. Assuming the J-Link software is installed, change to the directory that contains the software like "`C:\Program Files (x86)\SEGGER\JLink_V614`", run the command "`JLinkGDBServer -device MCIMX6G3 -scriptfile "<install_dir>/boards/<board_name>/<example_type>/<application_name>/ddr_init.jlinkscript"`".
5. After it is connected, the screen should resemble this figure:

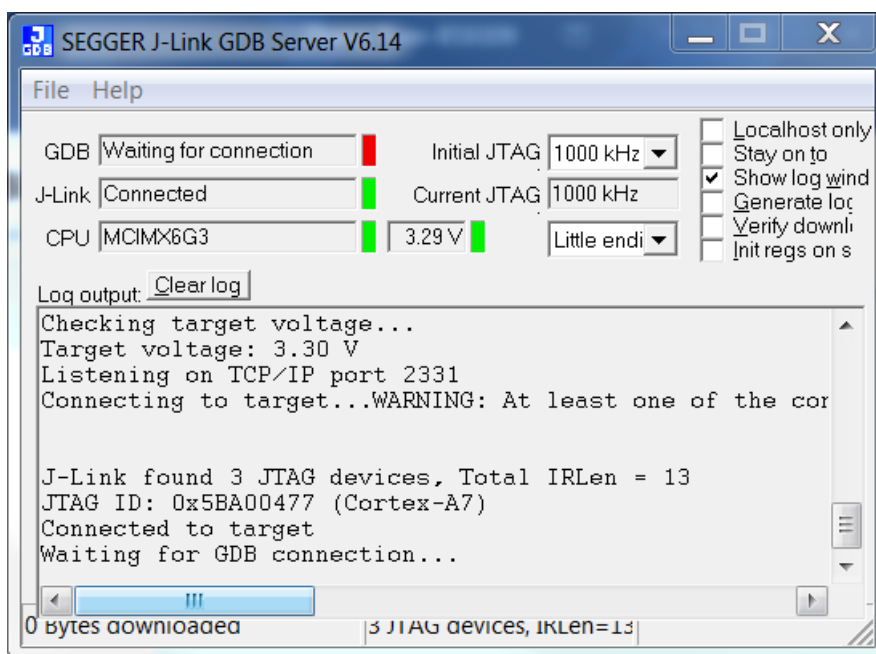


Figure 18. SEGGER J-Link GDB Server screen after successful connection

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”.

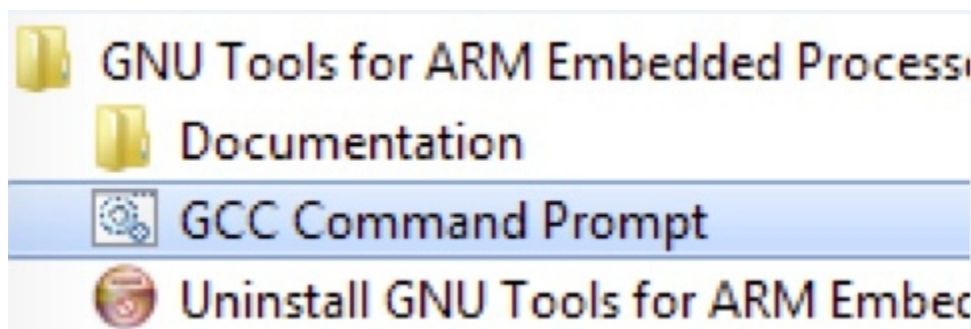


Figure 19. Launch command prompt

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/evkmcmx6ul/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”.


```

GCC Command Prompt - arm-none-eabi-gdb.exe hello_world.elf
MCIMX6G3xxx05_ram.ld      build_dds_release.sh      cmake_install.cmake
Makefile                  build_debug.bat

C:\npx\SDK_2.2_EVK-MCIMX6UL_WIN\boards\evkmcimx6ul\demo_apps\hello_world\armgcc>
cd Ddr_debug

C:\npx\SDK_2.2_EVK-MCIMX6UL_WIN\boards\evkmcimx6ul\demo_apps\hello_world\armgcc\
Ddr_debug>arm-none-eabi-gdb.exe hello_world.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.10.1.20160616-cvs
Copyright (C) 2015 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...done.
(gdb)

```

Figure 20. Run arm-none-eabi-gdb

9. Run these commands:
 - a. "target remote localhost:2331"
 - b. "monitor reset"
 - c. "monitor halt"
 - d. "load"
 - e. "monitor go"
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.

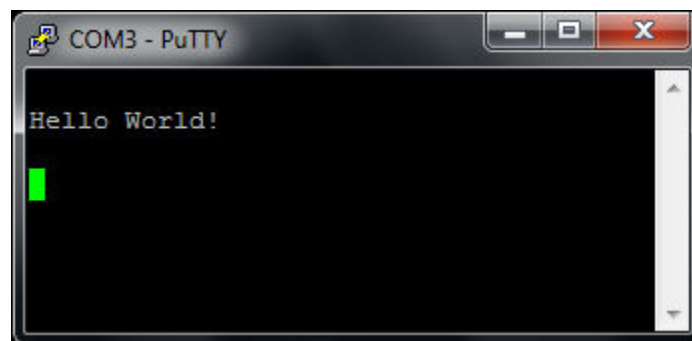


Figure 21. Text display of the hello_world demo

5 Run a demo using Manufacturing Tool (MFGTool)

The manufacturing tool, named MFGTool, is a tool that runs on a computer and is used to download images to different devices on an i.MX board. The tar.gz file can be found with the pre-built images.

5.1 Configuring Manufacturing Tool (MFGTool)

The following steps describe how to configure the MFGTool:

1. Download IMX6_L4.1.15_2.0.0_MFG_TOOL from www.nxp.com
2. Extract IMX6_L4.1.15_2.0.0_MFG_TOOL to get mfgtools-with-rootfs and mfgtools-without-rootfs, then continue to extract mfgtools-with-rootfs to get the mfgtools folder. Override the contents in the mfgtools folder with the files provided in the `<sdk_dir>/tools/mfgtools` folder. It is also important to replace the ucl2.xml file.

5.2 Using Manufacturing Tool (MFGTool)

The following steps describe how to use the MFGTool:

1. Build the application and copy the built binary (.bin file) to the `<sdk_dir>/tools/imgutil/<board>` folder and rename to `sdk20-app.bin`.
2. In the `sdk_dir/tools/imgutil/<board>` folder, run the proper `mkimage.sh` command as following to get bootable image file `sdk20-app.img`.
 - a. **QSPI image:** If the application is built with the RAM link file (under `DDR_debug` or `DDR_release` target) and wants to be loaded from flash to RAM then run, use `"mkimage.sh ram"` to create the bootable image.
 - b. **QSPI XIP image:** If the application is built with the flash link file (under `debug` or `release` target) and wants to run on flash directly, use `"mkimage.sh flash"` to create the bootable XIP image.
 - c. **SD image:** If the application is built with the RAM link file and wants to be loaded from MicroSD to RAM and run, use `"mkimage.sh sd"` to create the bootable image.
3. Copy `sdk20-app.img` file made with `imgutil` to `Profiles/Linux/OS Firmware/files` folder in MFGTool.
4. Connect a USB cable from a computer to the USB OTG port on the board.
5. Connect a USB cable from the OTG-to-UART port to the computer for console output.
6. Set the boot pin to Serial download mode (see the *Quick Start Guide, Evaluation Kit, Based on i.MX 6UltraLite Applications Processor*). Run `mfgtool2-sdk20-mx6ul-evk-qspi-nor-n25q256a.vbs` to write `sdk20-app.img` (built with `"mkimage.sh ram"` or `"mkimage.sh flash"`) to QSPI flash or run `mfgtool2-sdk20-mx6ul-evk-sdcard.vbs` to write `sdk20-app.img` (built with `"mkimage.sh sd"`) to MicroSD.
7. Switch boot mode to Internal Boot and set boot devices to QSPI flash or MicroSD card, then power on the board. Then, the application should be running.

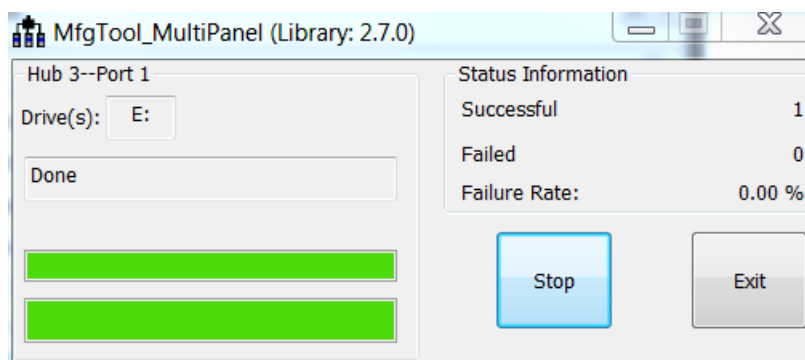


Figure 22. Programming QSPI flash with the manufacturing tool -- image downloading

NOTE

The readme.txt will be found in the imgutil folder and mfgtools folder.

6 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 below:

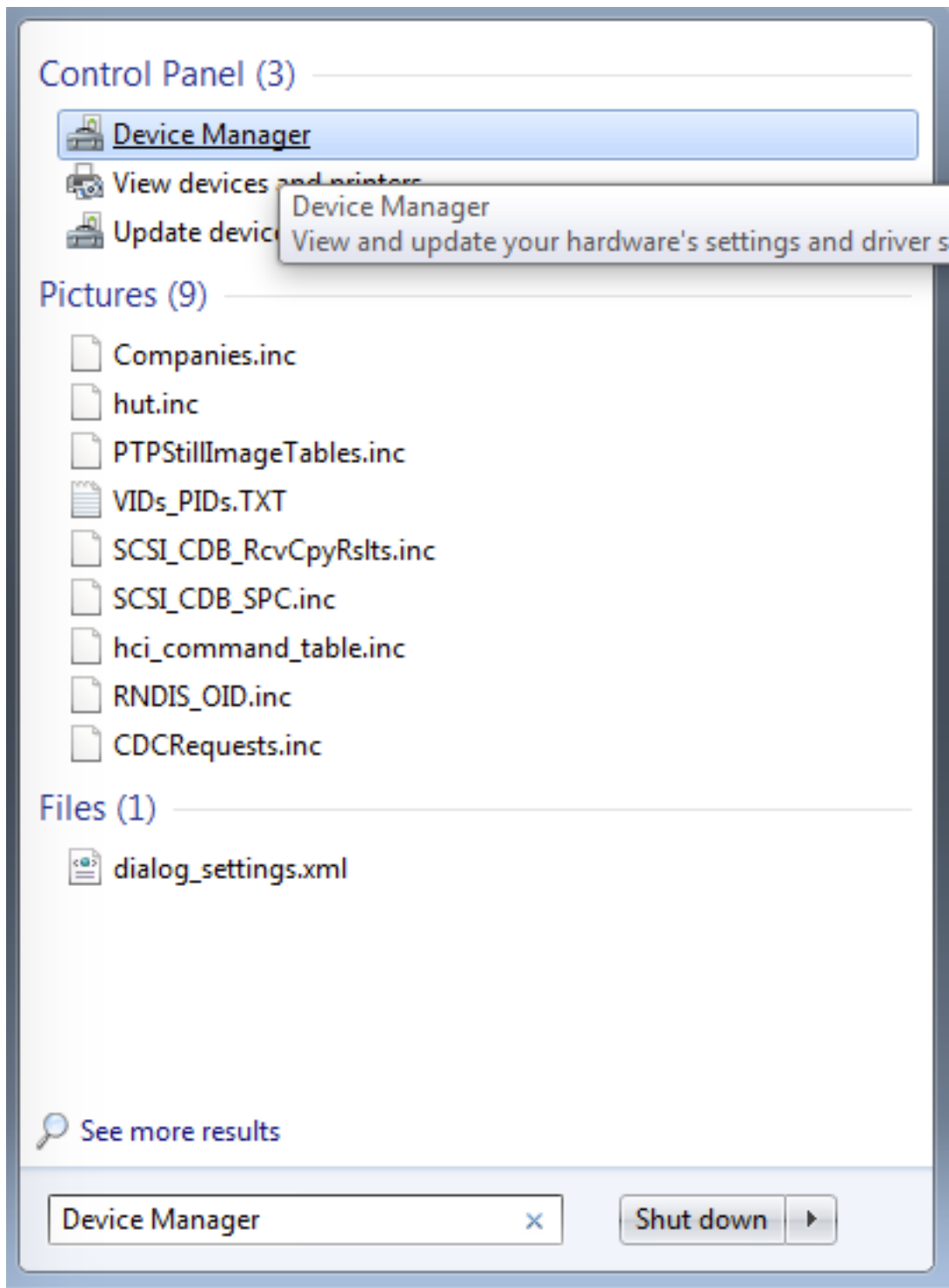


Figure 23. Device manager

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

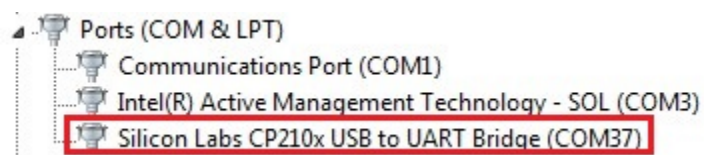


Figure 24. USB-UART interface

How to Reach Us:**Home Page:**nxp.com**Web Support:**nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM, ARM Powered, Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2017 NXP B.V.

Document Number MCUXSDKIMX6ULGSUG
Revision 0, 06/2017

