NXP Semiconductors
User's Guide

Document identifier: MCUXSDKRDMW320R0GSUG
Rev. 0, March 1, 2021

Getting Started with MCUXpresso SDK for RDMW320-R0

Contents

Chapter 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 MCUXpresso SDK Release Notes for RDMW320-R0 (document MCUXSDKRDMW320R0RN).

For more details about MCUXpresso SDK, see Figure 1.

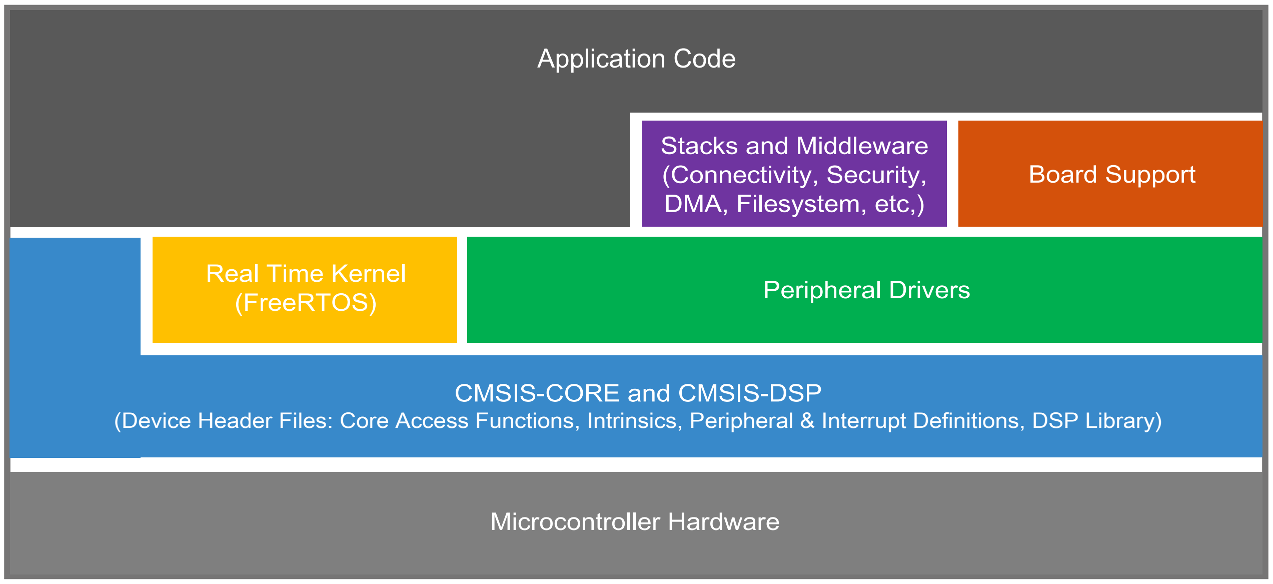

Application Code
Stacks and Middleware (Connectivity, Security, DMA, Filesystem, etc,)
Board Support
Real Time Kernel (FreeRTOS)
Peripheral Drivers
CMSIS-CORE and CMSIS-DSP
(Device Header Files: Core Access Functions, Intrinsics, Peripheral & Interrupt Definitions, DSP Library)
Microcontroller Hardware

Figure 1. MCUXpresso SDK layers

Chapter 2
MCUXpresso SDK board support package folders

MCUXpresso SDK board support package 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 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 it contain. These include (but are not limited to):

- `demo_apps`: Full-featured applications that 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 that show 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 peripherals are used (for example, SPI conversion using DMA).
- `rtos_examples`: Basic FreeRTOS™ OS examples that show the use of various RTOS objects (semaphores, queues, and so on) and interfaces with the MCUXpresso SDK's RTOS drivers.
- `wifi_examples`: Applications that use the NXP wifi and lwip stacks.

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:

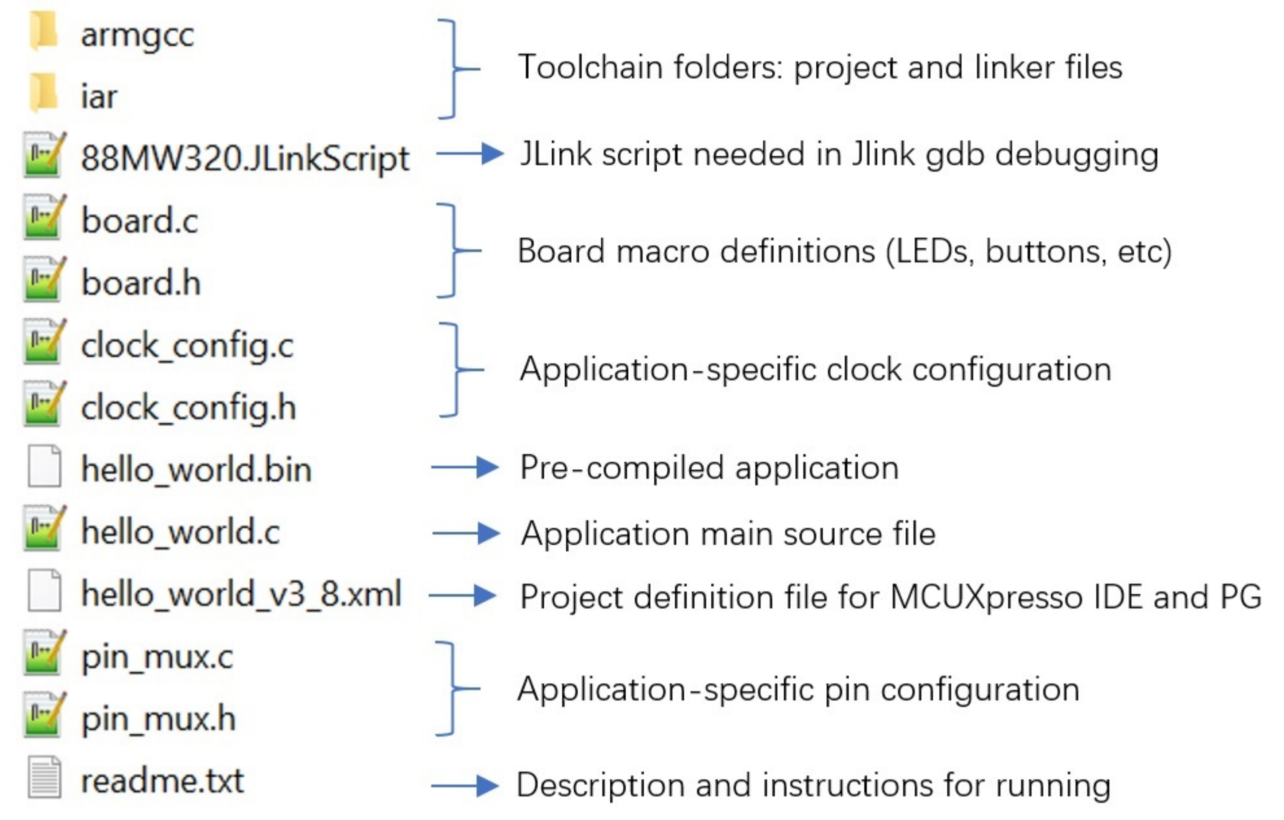

armgcc
iar
Toolchain folders: project and linker files
88MW320.JLinkScript
JLink script needed in Jlink gdb debugging
board.c
board.h
Board macro definitions (LEDs, buttons, etc)
clock_config.c
clock_config.h
Application-specific clock configuration
hello_world.bin
Pre-compiled application
hello_world.c
Application main source file
hello_world_v3_8.xml
Project definition file for MCUXpresso IDE and PG
pin_mux.c
pin_mux.h
Application-specific pin configuration
readme.txt
Description and instructions for running

Figure 2. Application folder structure

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

2.2 Locating example application source files

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

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

- *devices/<device_name>*: The device's CMSIS header file, MCUXpresso SDK feature file and a few other files
- *devices/<device_name>/drivers*: All of the peripheral drivers for your specific MCU
- *devices/<device_name>/<tool_name>*: Toolchain-specific startup code, including vector table definitions
- *devices/<device_name>/utilities*: Items such as the debug console that are used by many of the example applications

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.

Chapter 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 `rdmw320_r0` 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

NOTE

First install **Development Tool Package for MCUXpresso SDK** from 88MW32X 802.11n Wi-Fi$^{®}$ Microcontroller SoC to get MW320 device available in IAR.

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

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

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

   Using the RDMW320-R0 hardware platform as an example, the `hello_world` workspace is located in:

   <install_dir>/boards/rdmw320_r0/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**.

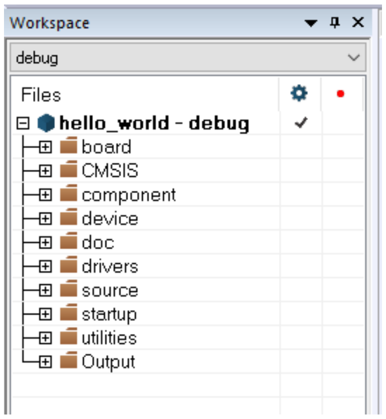


Figure 3. Demo build target selection

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

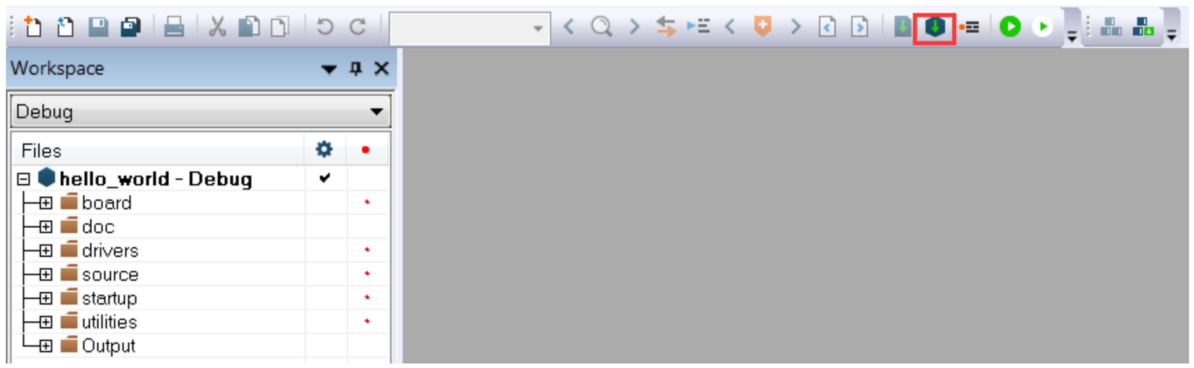


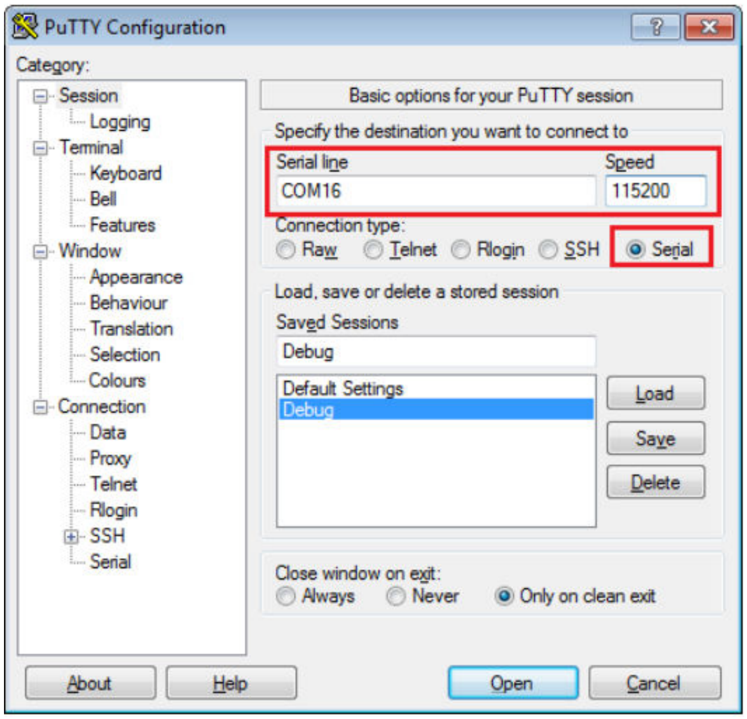
Figure 4. Build the demo application

4. The build completes without errors.

3.2 Run an example application

To download and run the application, perform these steps:

1. Connect jlink probe to RDMW320-R0 jtag interface and Host USB port.
2. Connect the development platform to your PC via USB cable.
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 How to determine COM port). Configure the terminal with these settings:
    a. 115200 or 9600 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



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM16

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions: Debug

Default Settings
Debug

Close window on exit: Always Never [✓ Only on clean exit]

Figure 5. Terminal (PuTTY) configuration

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

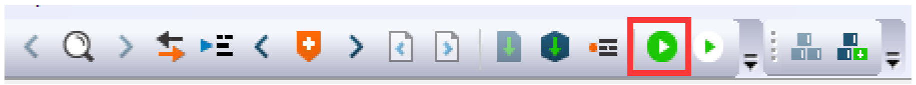
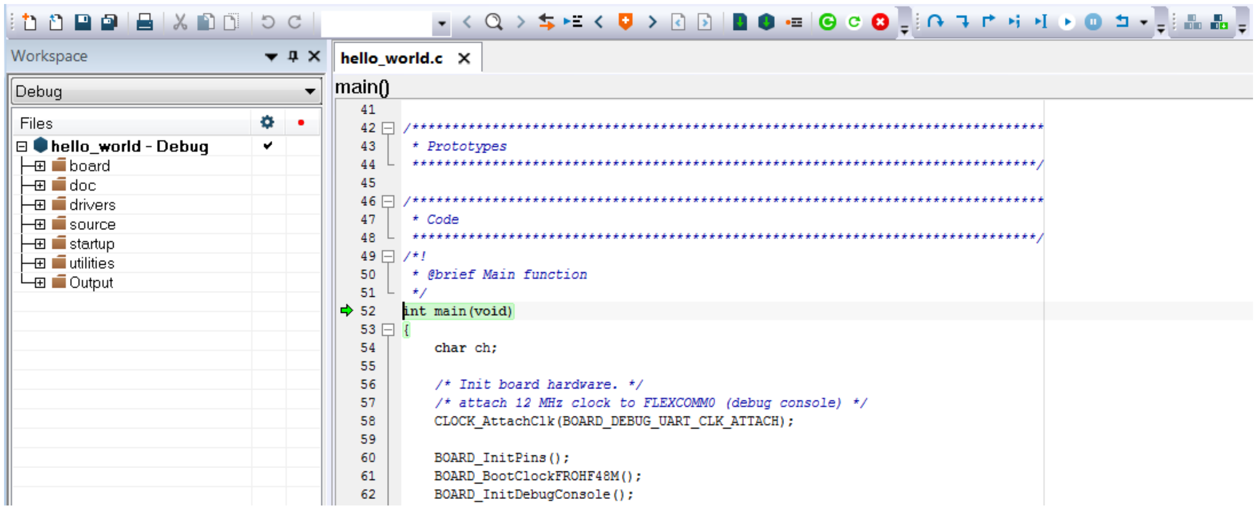


Figure 6. Download and Debug button

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



```
/*******************************************************************************
 * Prototypes
 ******************************************************************************/

/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Main function
 */
int main(void)
{
    char ch;

    /* Init board hardware. */
    /* attach 12 MHz clock to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitPins();
    BOARD_BootClockFROHF48M();
    BOARD_InitDebugConsole();
```

Figure 7. Stop at main() when running debugging

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

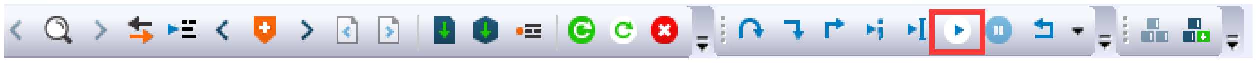
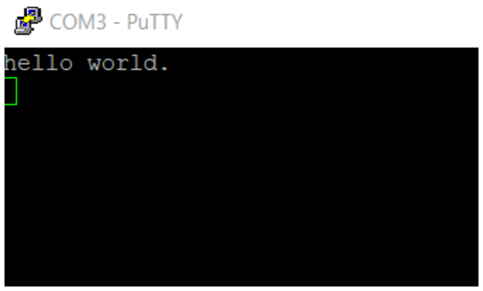


Figure 8. Go button

7. The `hello_world` application is now running and a banner is displayed on the terminal. If it does not appear, check your terminal settings and connections.



```
hello world.
```

Figure 9. Text display of the hello_world demo

Chapter 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 is targeted which is used as an example.

4.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run an 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.

4.1.1 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, etc.). The GCC toolchain should correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes for RDMW320-R0*(document MCUXSDKRDMW320R0RN).

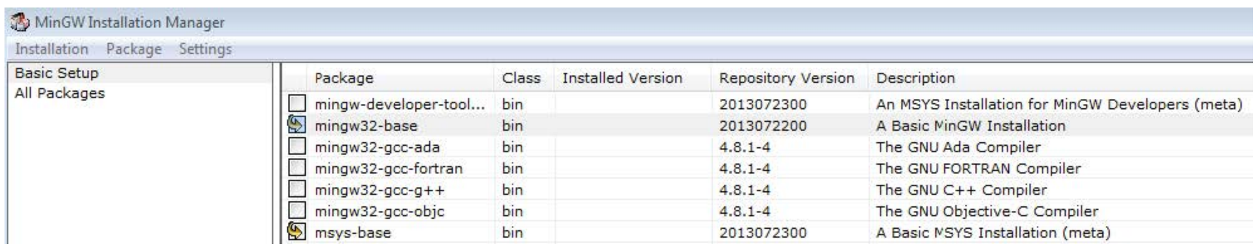
4.1.2 Install MinGW (only required on Windows OS)

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 use 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.
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**.



	Package	Class	Installed Version	Repository Version	Description
☐	mingw-developer-tool...	bin		2013072300	An MSYS Installation for MinGW Developers (meta)
✓	mingw32-base	bin		2013072200	A Basic MinGW Installation
☐	mingw32-gcc-ada	bin		4.8.1-4	The GNU Ada Compiler
☐	mingw32-gcc-fortran	bin		4.8.1-4	The GNU FORTRAN Compiler
☐	mingw32-gcc-g++	bin		4.8.1-4	The GNU C++ Compiler
☐	mingw32-gcc-objc	bin		4.8.1-4	The GNU Objective-C Compiler
✓	msys-base	bin		2013072300	A Basic MSYS Installation (meta)

Figure 10. Set up MinGW and MSYS

4. In the **Installation** menu, click **Apply Changes** and follow the remaining instructions to complete the installation.

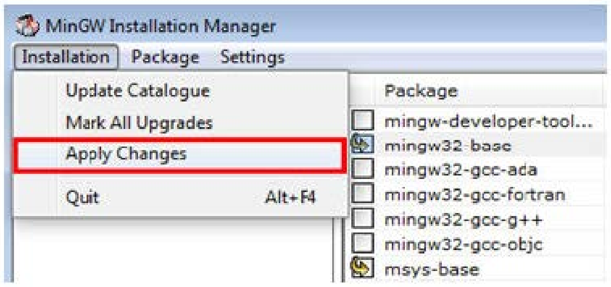


Figure 11. Complete MinGW and MSYS installation

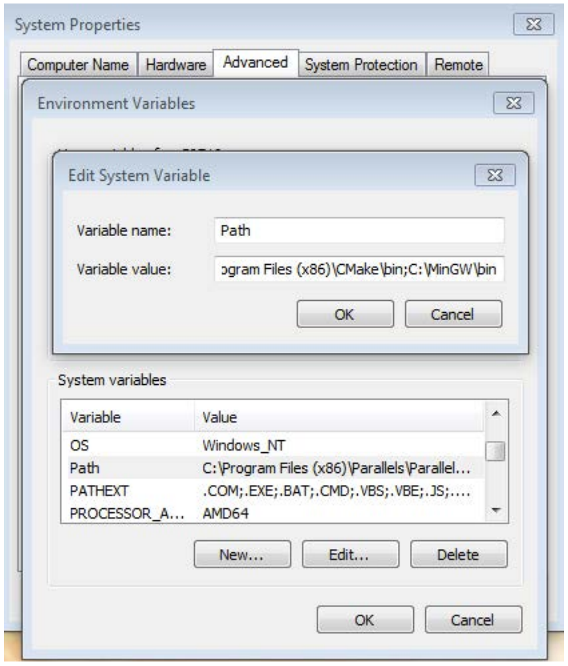
5. Add the appropriate item to the Windows operating system path environment variable. It can be found under **Control Panel->System and Security->System->Advanced System Settings** in the **Environment Variables...** section. The path is:

   <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 will not work.

NOTE

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



Variable name: Path

Variable value: ɔgram Files (x86)\CMake\bin;C:\MinGW\bin

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\Parallels\Parallel...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Figure 12. Add Path to systems environment

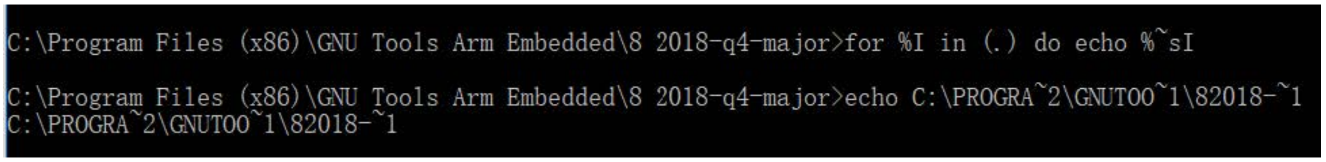
4.1.3 Add a new system environment variable for ARMGCC_DIR

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

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

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

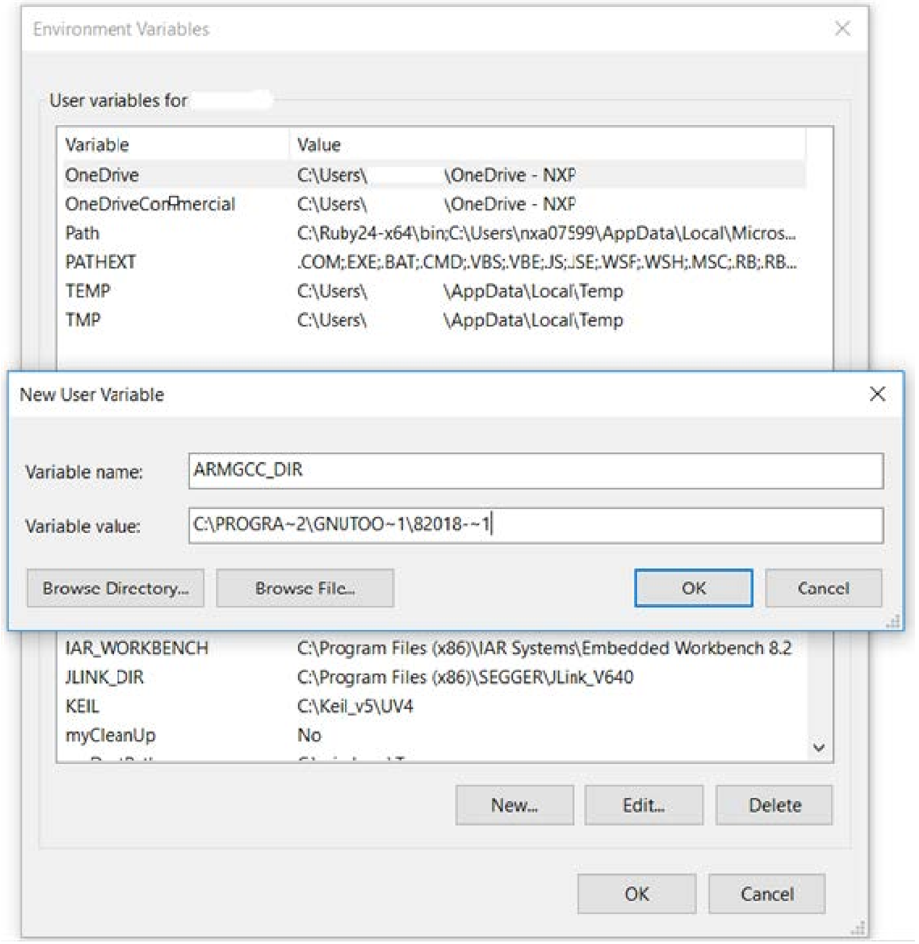
Short path should be used for path setting. You could convert the path to short path by running the `for %I in (.) do echo %~sI` command in above path. Figure 13 and Figure 14 show an example for setting `ARMGCC_DIR` for armgcc 8 version. Similar operation also works for armgcc 9.



```
C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>for %I in (.) do echo %~sI

C:\Program Files (x86)\GNU Tools Arm Embedded\8 2018-q4-major>echo C:\PROGRA~2\GNUTOO~1\82018-~1
C:\PROGRA~2\GNUTOO~1\82018-~1
```

Figure 13. Convert path to short path



Environment Variables

User variables for

Variable	Value
OneDrive	C:\Users\ \OneDrive - NXP
OneDriveCommercial	C:\Users\ \OneDrive - NXP
Path	C:\Ruby24-x64\bin;C:\Users\nxa07599\AppData\Local\Micros...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.RB;.RB...
TEMP	C:\Users\ \AppData\Local\Temp
TMP	C:\Users\ \AppData\Local\Temp
IAR_WORKBENCH	C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.2
JLINK_DIR	C:\Program Files (x86)\SEGGER\JLink_V640
KEIL	C:\Keil_v5\UV4
myCleanUp	No

New User Variable

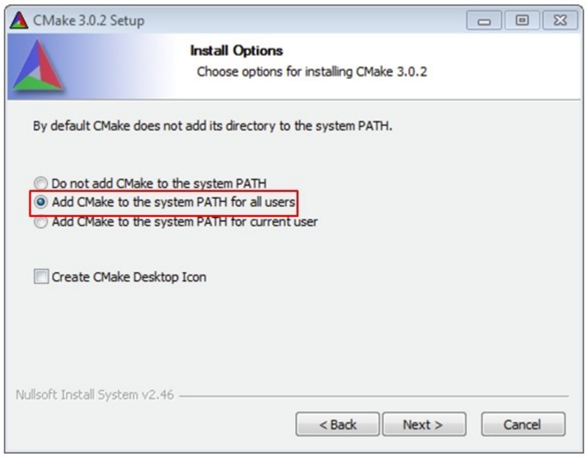
Variable name: ARMGCC_DIR

Variable value: C:\PROGRA~2\GNUTOO~1\82018-~1

Figure 14. Add ARMGCC_DIR system variable

4.1.4 Install CMake

1. Download CMake 3.0.x from CMAKE.
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.



CMake 3.0.2 Setup

Install Options

Choose options for installing CMake 3.0.2

By default CMake does not add its directory to the system PATH.

- Do not add CMake to the system PATH
- [✓ Add CMake to the system PATH for all users]
- Add CMake to the system PATH for current user

- [ ] Create CMake Desktop Icon

Figure 15. 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.
5. Make sure sh.exe is not in the Environment Variable PATH. This is a limitation of `mingw32-make`.

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**.

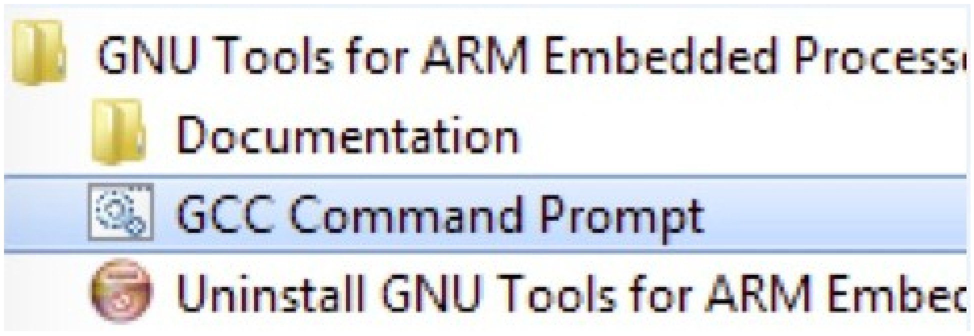

GNU Tools for ARM Embedded Process
Documentation
GCC Command Prompt
Uninstall GNU Tools for ARM Embec

Figure 16. 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

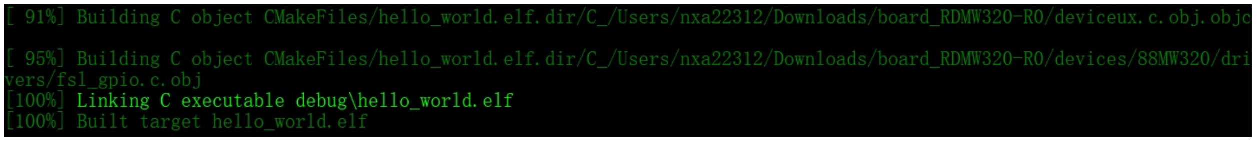
   For this example, the exact path is:

   <install_dir>/boards/rdmw320_r0/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 **build_debug.bat** file in Windows Explorer to build it. The output is as shown in Figure 17.



```
[ 91%] Building C object CMakeFiles/hello_world.elf.dir/C_/Users/nxa22312/Downloads/board_RDMW320-R0/deviceux.c.obj.objc

[ 95%] Building C object CMakeFiles/hello_world.elf.dir/C_/Users/nxa22312/Downloads/board_RDMW320-R0/devices/88MW320/dri
vers/fsl_gpio.c.obj
[100%] Linking C executable debug\hello_world.elf
[100%] Built target hello_world.elf
```

Figure 17. hello_world demo build successful

4.3 Run an example application

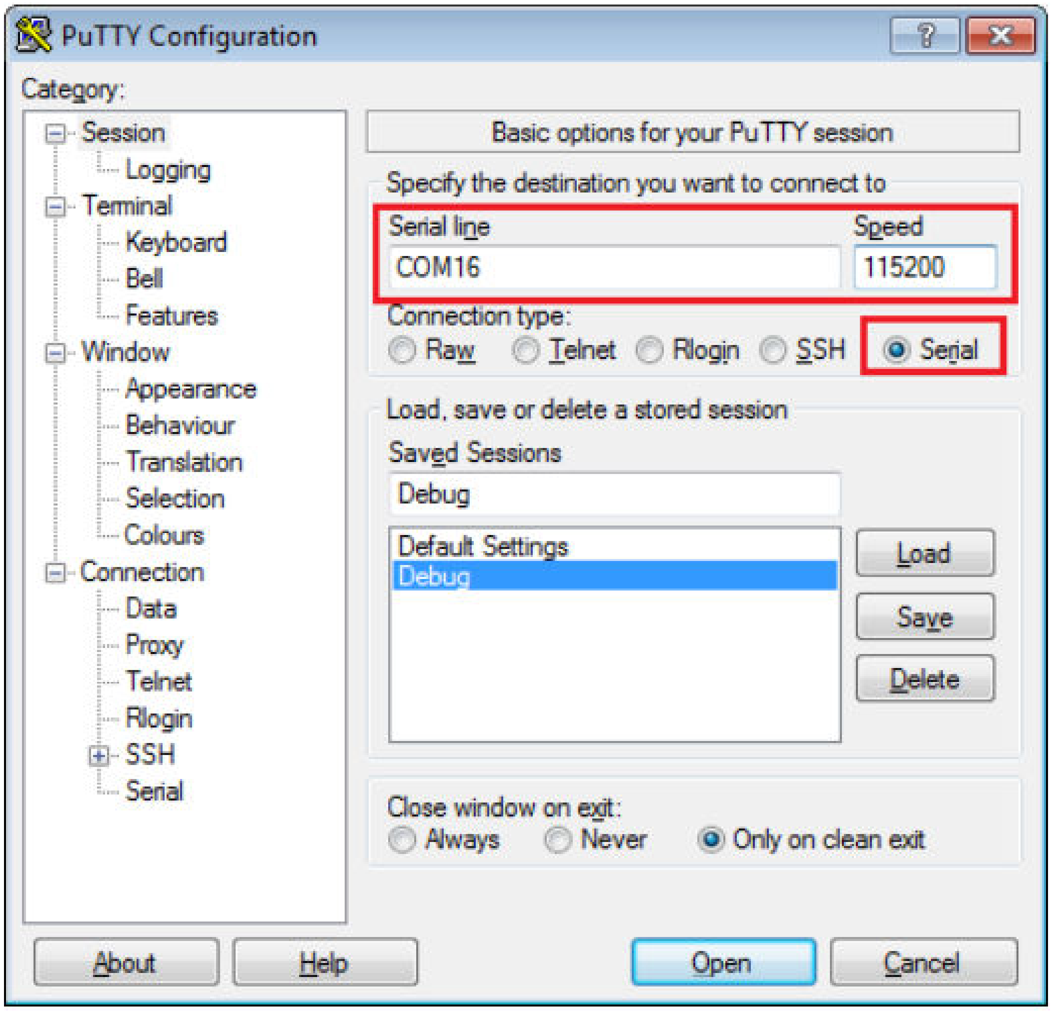
This section describes steps to run a demo application using J-Link GDB Server application. To perform this exercise, make sure you have a standalone J-Link pod that is connected to the debug interface of your board.

NOTE

Some hardware platforms require hardware modification in order to function correctly with an external debug interface.

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

1. This board supports the J-Link debug probe. Before using it, install SEGGER software, which can be downloaded from SEGGER.
2. Connect the development platform to your PC via USB cable between the USB connector and the PC USB connector. Connect a standalone J-Link debug pod to the SWD/JTAG connector of the board.
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 How to determine COM port). Configure the terminal with these settings:
    a. 115200 or 9600 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



Serial line: COM16
Speed: 115200

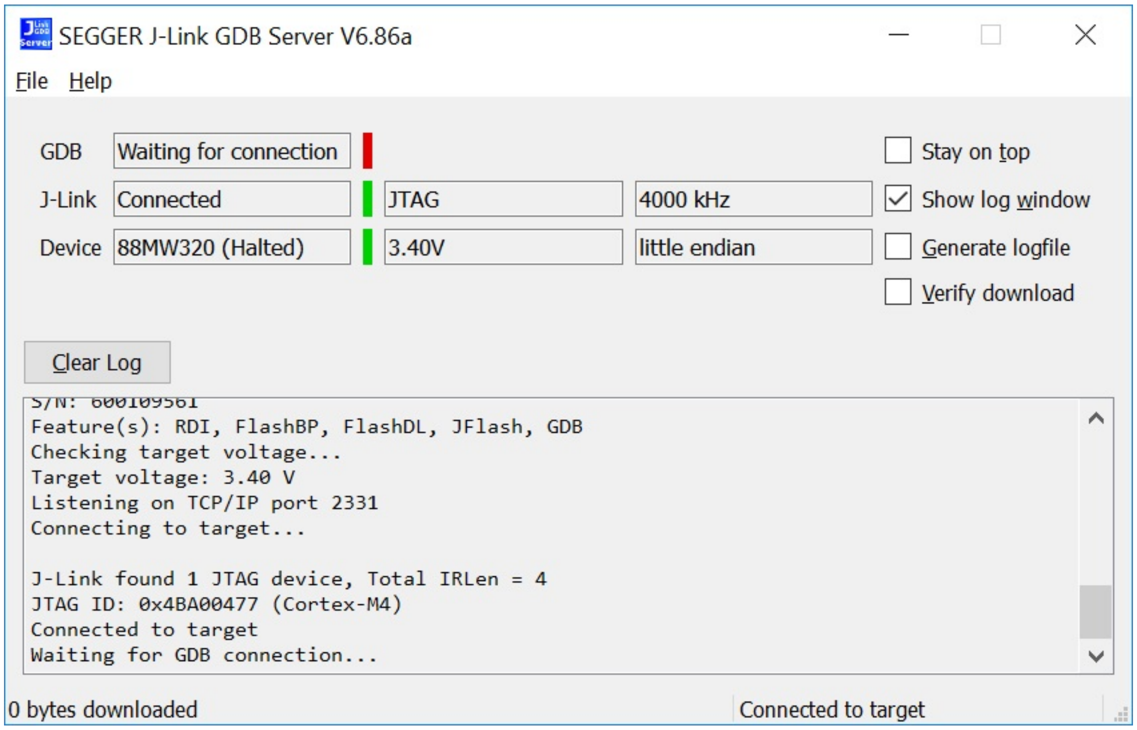
Connection type: ○ Raw ○ Telnet ○ Rlogin ○ SSH [✓ Serial]

Saved Sessions: Debug

Close window on exit: ○ Always ○ Never [✓ Only on clean exit]

Figure 18. Terminal (PuTTY) configurations

4. Open the J-Link GDB Server application. Open Windows command window.
   Type "C:\Program Files (x86)\SEGGER\JLink_<version>\JLinkGDBServer.exe" -jlinkscriptfile <install_dir>\boards\rdmw320_r0\demo_apps\hello_world\88MW320.JLinkScript -device 88MW320.
5. After it is connected, the screen should resemble Figure 19.


SEGGER J-Link GDB Server V6.86a
File Help
GDB Waiting for connection
J-Link Connected JTAG 4000 kHz
Device 88MW320 (Halted) 3.40V little endian
Stay on top
Show log window
Generate logfile
Verify download
Clear Log
0 bytes downloaded
Connected to target

```
S/N: 600109561
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.40 V
Listening on TCP/IP port 2331
Connecting to target...

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x4BA00477 (Cortex-M4)
Connected to target
Waiting for GDB connection...
```

Figure 19. 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**.

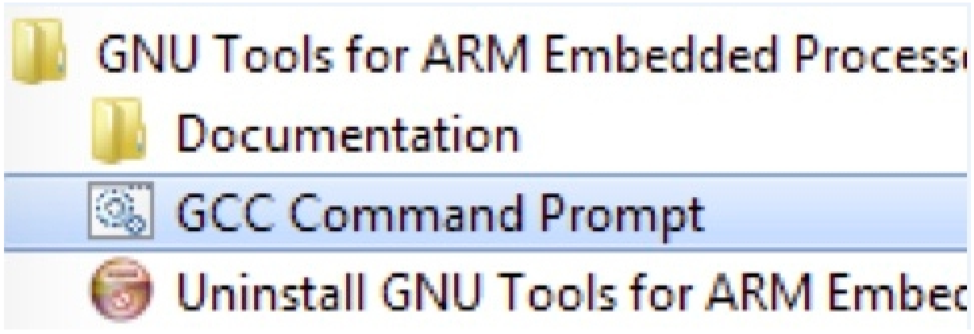

GNU Tools for ARM Embedded Process
Documentation
GCC Command Prompt
Uninstall GNU Tools for ARM Embec

Figure 20. 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:

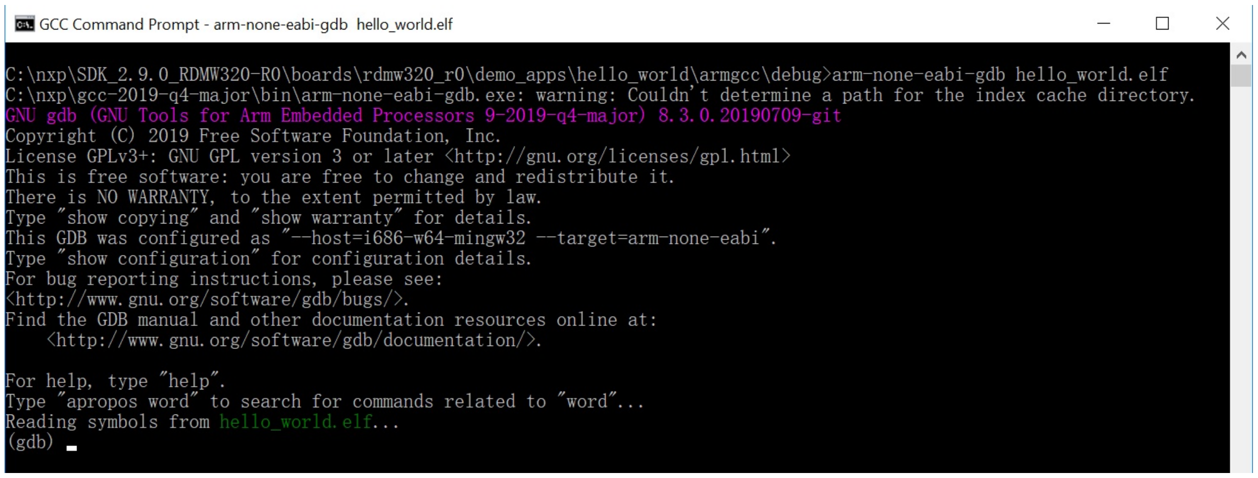
   <install_dir>/boards/<board_name>/<example_type>/<application_name>/armgcc/debug

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

   For this example, the path is:

   <install_dir>/boards/rdmw320_r0/demo_apps/hello_world/armgcc/debug

8. Run the `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 hello_world.elf

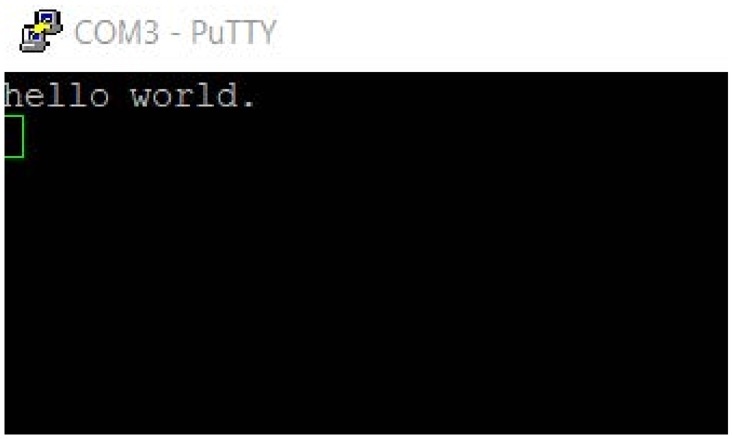
```
C:\nxp\SDK_2.9.0_RDMW320-R0\boards\rdmw320_r0\demo_apps\hello_world\armgcc\debug>arm-none-eabi-gdb hello_world.elf
C:\nxp\gcc-2019-q4-major\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Tools for Arm Embedded Processors 9-2019-q4-major) 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)
```

Figure 21. Run arm-none-eabi-gdb

9. Run these commands:
   a. `target remote localhost:2331`
   b. `monitor reset`
   c. `monitor halt`
   d. `load`
   e. `continue`
10. After Command a to Command d, the application is downloaded and halted at the reset vector. Use Command e 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.



COM3 - PuTTY

```
hello world.
```

Figure 22. Text display of the hello_world demo

Chapter 5
Boot a demo application

To make the application boot on board reset, boot2 is needed. Boot2 serves as secondary stage boot-loader. It can properly configure hardware for application and load partition table in flash. It can also figure out which firmware image to start up, load the application from flash to RAM if necessary, and kick it off.

Users need to write boot2, partition table and the firmware image to Flash. Any poorly-programming will result in boot failure. In addition, if the application needs to load WiFi firmware, the WiFi firmware also needs to be flashed.

5.1 Create partition table

Partition table can be described in a configuration file in `txt` format, including the partition component information, such as, name, address, size, etc. Users may use *<install_dir>\tools\mw_img_conv\bin\mw_img_conv* to convert the configuration file to binary. For example,

```
# mw_img_conv layout layout.txt layout.bin
```

Then, the partition table binary file is ready to use.

5.2 Create bootable MCU firmware

Bootable MCU firmware is the binary file that converted from the compiled *.bin* file. When user compiles an SDK example project, a binary file will be created, like *hello_world.bin*.

NOTE

Because the debugger downloads the application without considering XIP flash offset setting in FLASHC, we create a special linker file with manual offset by default to make the wifi example debuggable (without conflicting the partition table area in flash). To make WiFi examples bootable, change the linker file to the one in devices, e.g. *<sdk_path>\devices\88MW320\iar\88MW320_xx_xxxx_flash.icf*, and re-build the application to get the correct bootable firmware.

User may use *<install_dir>\tools\mw_img_conv\mw320\mkimg.sh* to covert the compiled binary file to bootable MCU firmware. For example:

```
# ./mkimg.sh hello_world.bin
```

Now *hello_world.fw.bin* is created and can be written to flash.

5.3 Create bootable WiFi firmware

Bootable WiFi firmware is the binary file converted from raw WiFi firmware. The raw WiFi firmware must not be compressed. Users may use *<install_dir>\tools\mw_img_conv\bin\mw_img_conv* to covert the raw WiFi firmware to bootable WiFi firmware. For example:

```
#  mw_img_conv wififw mw30x_uapsta_W14.88.36.p144.bin mw30x_uapsta_W14.88.36.p144.fw.bin
```

Then the bootable wifi firmware is ready to use.

5.4 Flash the device

Now follow the address in the partition table to write the images to flash.

For example, boot2 address is 0, which means we need to write it to flash offset 0. Partition table is located at a fixed address of `0x4000`, so it should be written to flash offset `0x4000`.

Here's an example to write flash with *jlink.exe* tool.

1. Connect a USB cable between the host PC and Mini USB port on the target board.
2. Connect j-link probe between the host PC and JTAG port on the target board.
3. Open *Jlink.exe* and execute the following commands:

```
   J-Link>connect
   Device>88MW320
   TIF>s
   Speed>
   J-Link>exec SetFlashDLNoRMWThreshold = 0xFFFF        // SET RMW threshold to 64kB, so size
<64KB will be RMW.
   J-Link>loadbin <install_dir>\tools\boot2\boot2.bin 0x1F000000
   J-Link>loadbin <install_dir>\tools\boot2\layout.bin 0x1F004000
   J-Link>loadbin <install_dir>\tools\mw_img_conv\bin\hello_world.fw.bin 0x1F010000
   J-Link>loadbin
<install_dir>\boards\rdmw320_r0\wifi_examples\common\mw30x_uapsta_W14.88.36.p144.fw.bin
0x1F150000
```

Now reset your board and then the application is running.

Appendix A
How to determine COM port

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

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

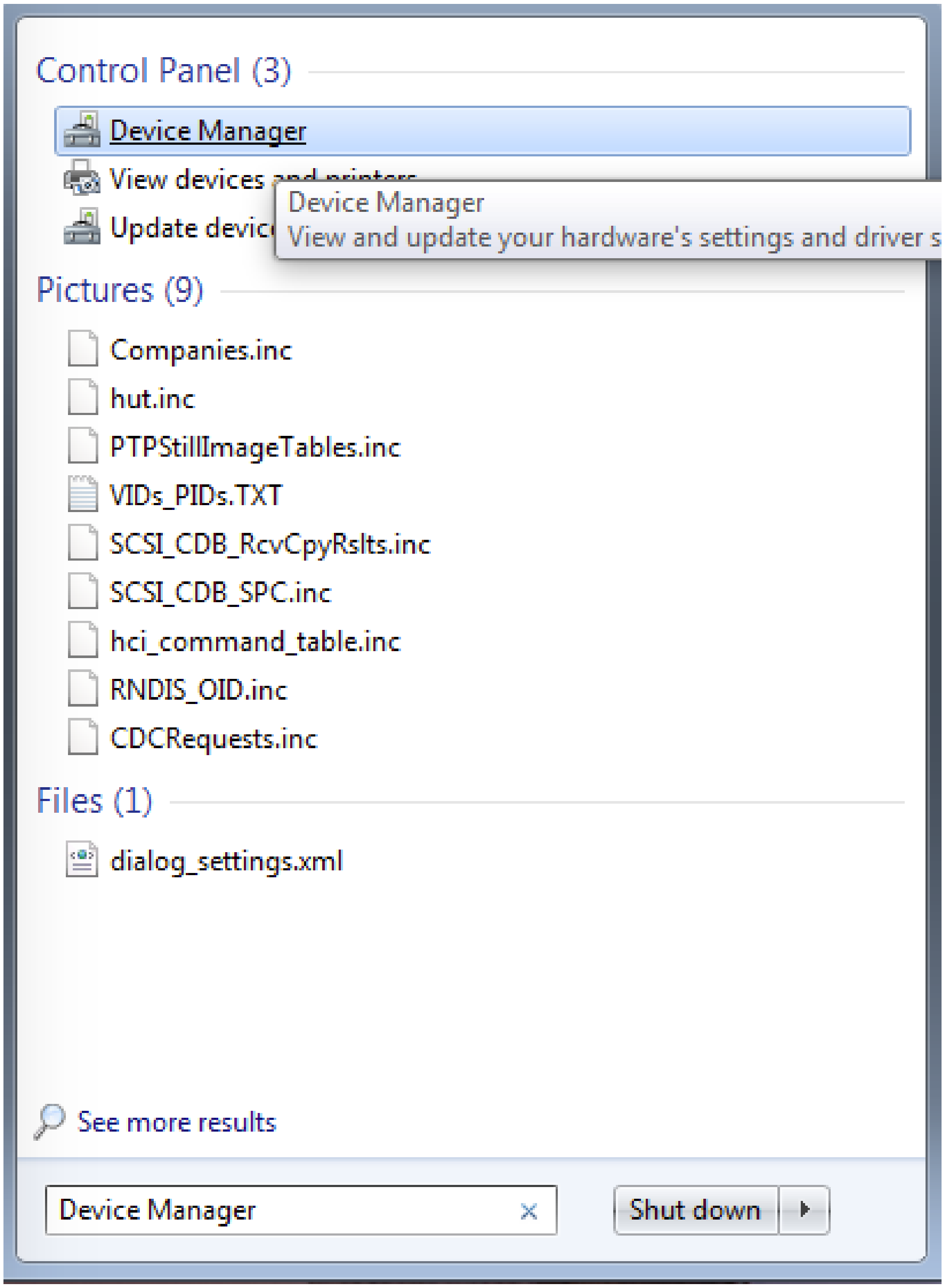
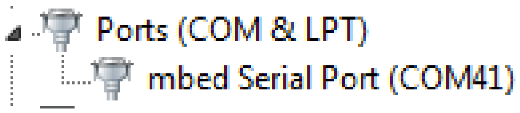


Figure 23. Device Manager

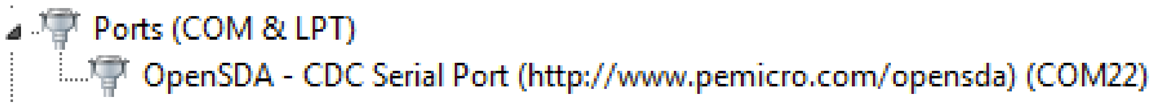
2. In the **Device Manager**, expand the **Ports (COM & LPT)** section to view the available ports. Depending on the NXP board you're using, the COM port can be named differently.
   a. OpenSDA – CMSIS-DAP/mbed/DAPLink interface:



Ports (COM & LPT)
mbed Serial Port (COM41)

Figure 24. OpenSDA – CMSIS-DAP/mbed/DAPLink interface

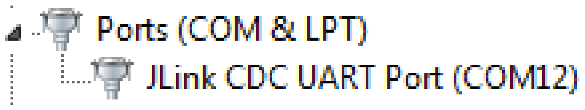
b. OpenSDA – P&E Micro:



Ports (COM & LPT)
OpenSDA - CDC Serial Port (http://www.pemicro.com/opensda) (COM22)

Figure 25. OpenSDA – P&E Micro

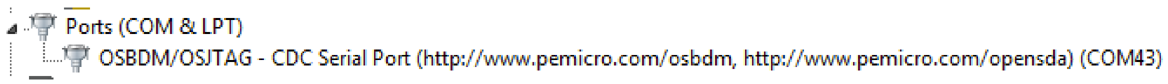
c. OpenSDA – J-Link:



Ports (COM & LPT)
JLink CDC UART Port (COM12)

Figure 26. OpenSDA – J-Link

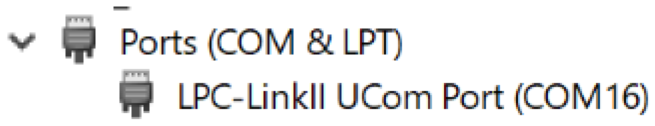
d. P&E Micro OSJTAG:



Ports (COM & LPT)
OSBDM/OSJTAG - CDC Serial Port (http://www.pemicro.com/osbdm, http://www.pemicro.com/opensda) (COM43)

Figure 27. P&E Micro OSJTAG

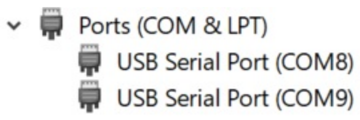
e. LPC-Link2:



Ports (COM & LPT)
LPC-LinkII UCom Port (COM16)

Figure 28. LPC-Link2

f. FTDI UART:



Ports (COM & LPT)
USB Serial Port (COM8)
USB Serial Port (COM9)

Figure 29. FTDI UART