MCUXSDKRDMW320R0GSUG

Getting Started with MCUXpresso SDK for RDMW320-R0

Rev. 1 — 28 July 2022

User guide

Document information

Information	Content
Keywords	Getting Started, MCUXpresso SDK, RDMW320-R0
Abstract	The MCUXpresso Software Development Kit (SDK) provides comprehensive software support for Kinetis and LPC Microcontrollers.

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.

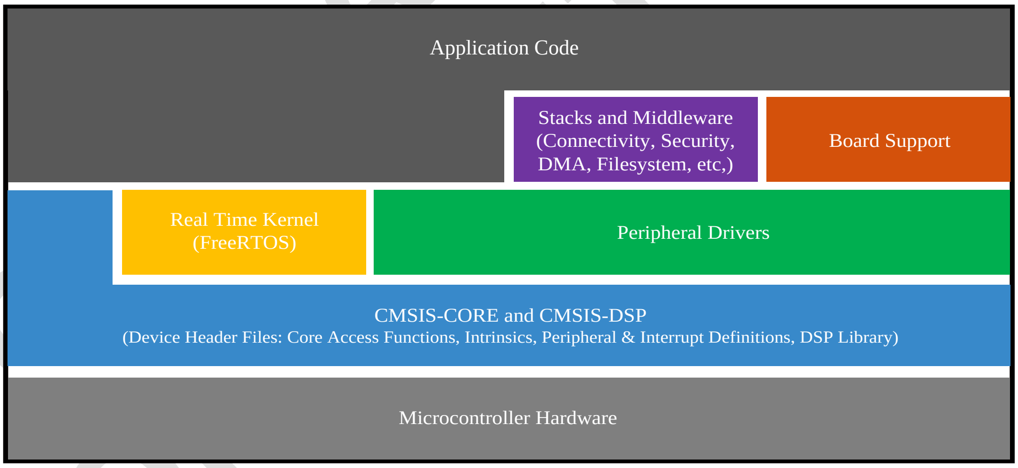

Figure 1. MCUXpresso SDK layers

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:

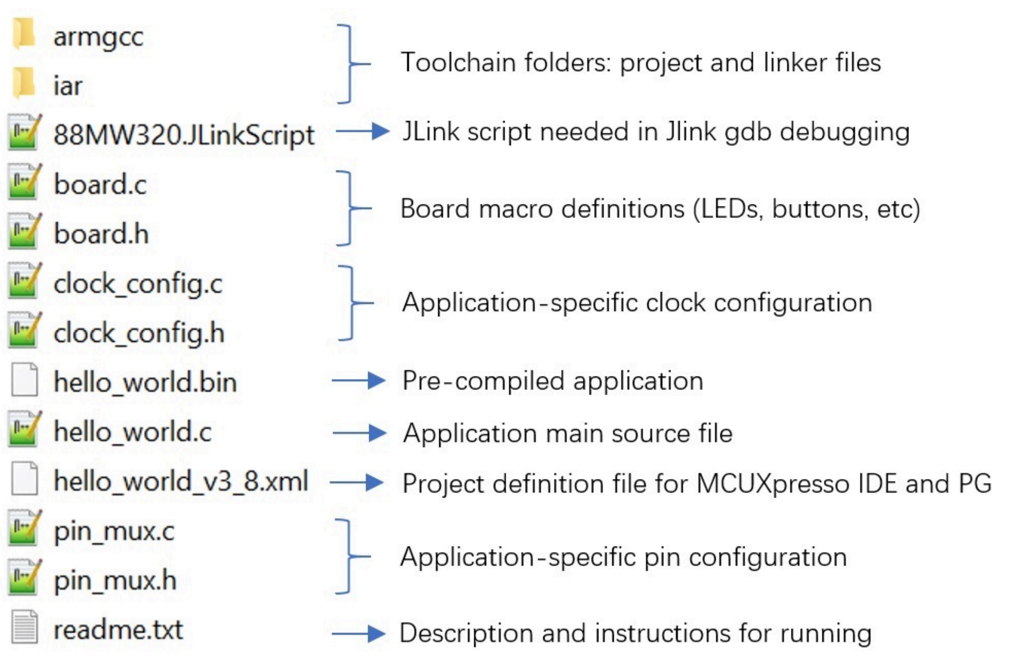

Figure 2. Application folder structure

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.

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.*

To build the `hello_world` example application, perform the following steps:

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**.

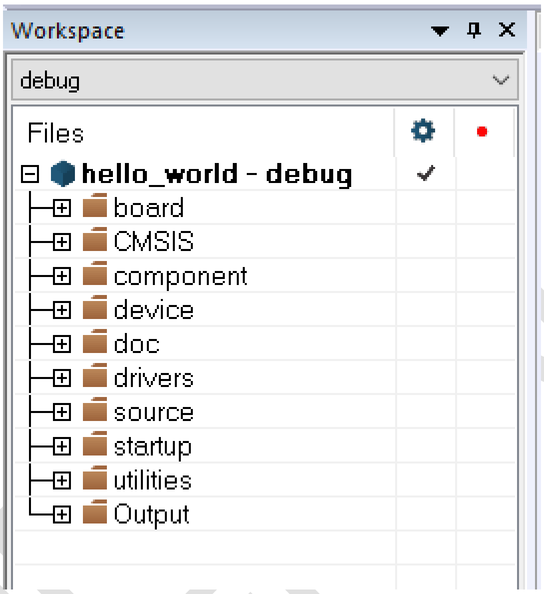

Figure 3. Demo build target selection

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

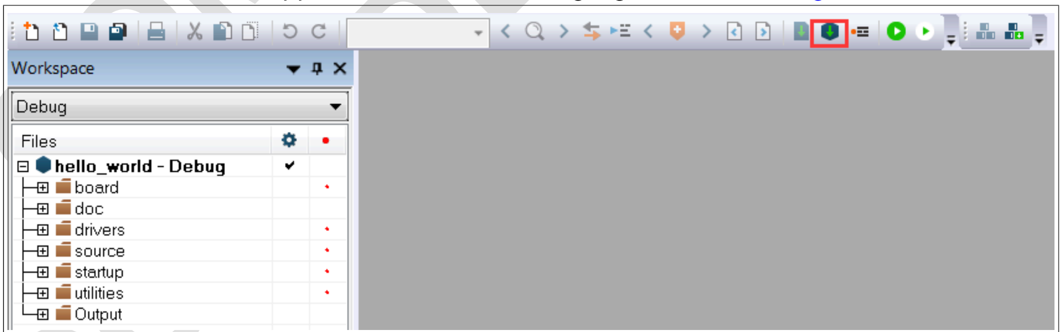

Figure 4. Build the demo application

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 Section 6). 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

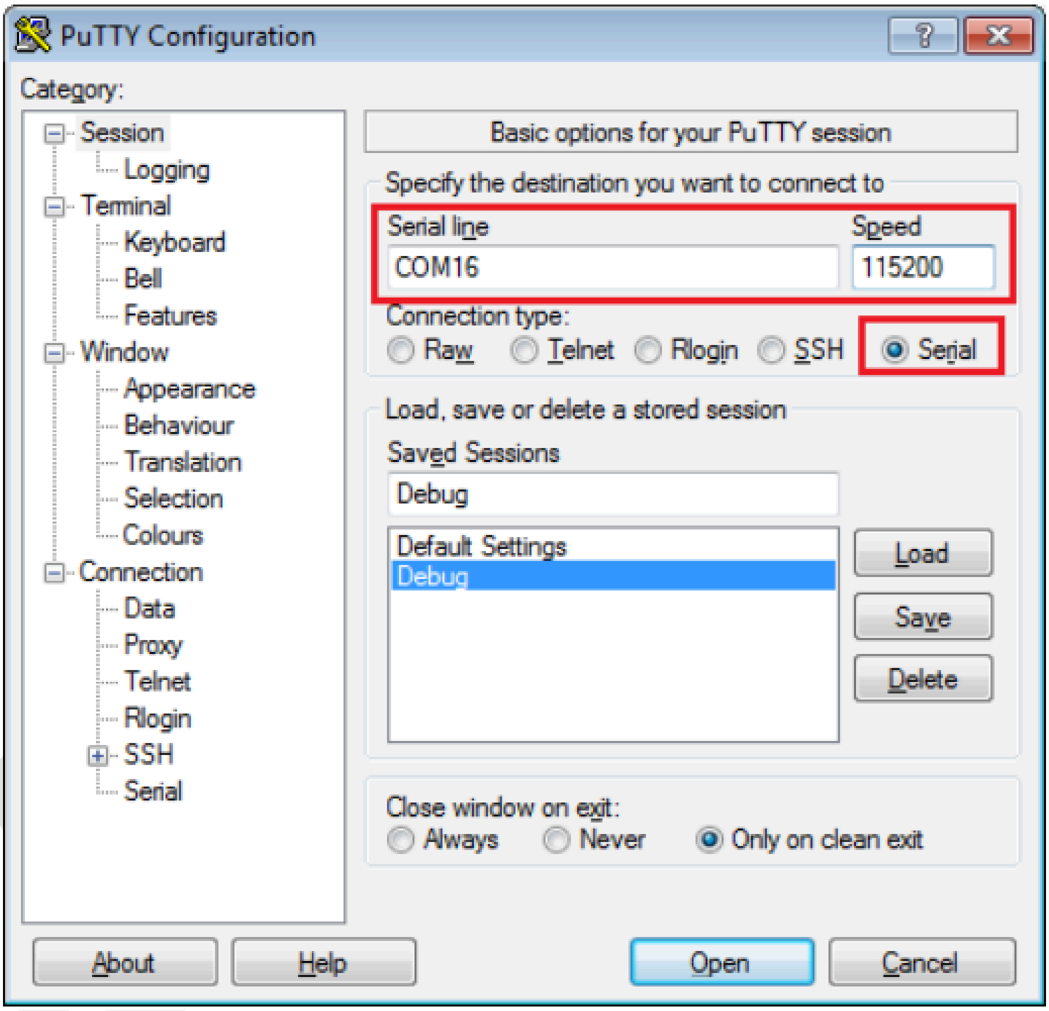

Figure 5. Terminal (PuTTY) configuration

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

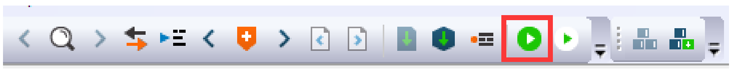

Figure 6. Download and Debug button

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

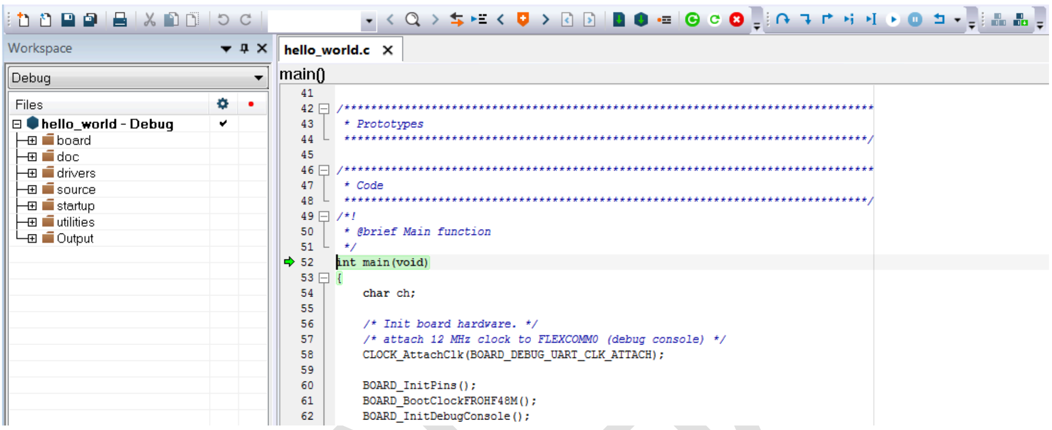

Figure 7. Stop at main() when running debugging

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

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.

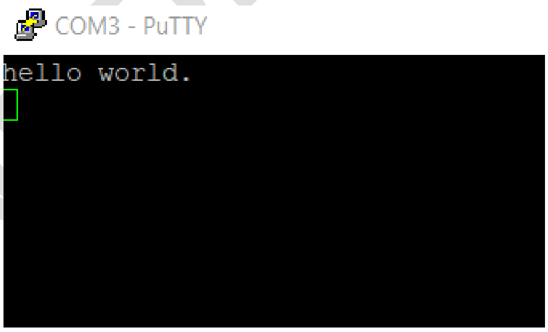

Figure 9. Text display of the hello_world demo

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, and so on). 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**.

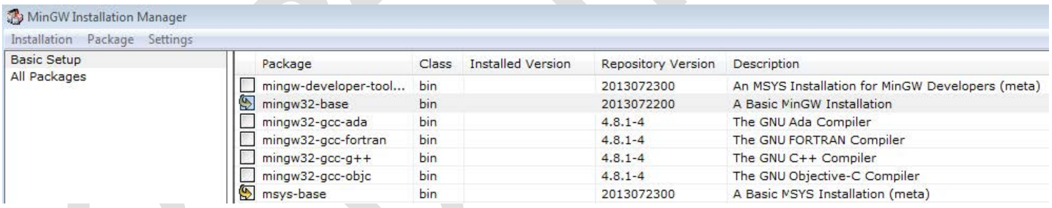

Figure 10. Set up MinGW and MSYS

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

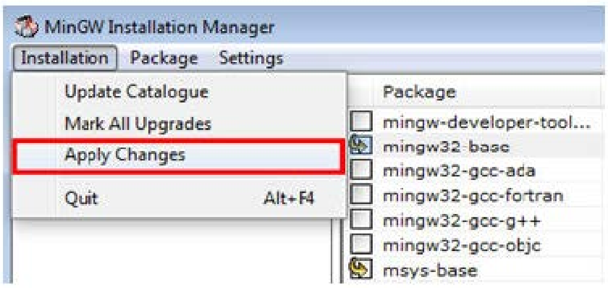

Figure 11. Complete MinGW and MSYS installation

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.*

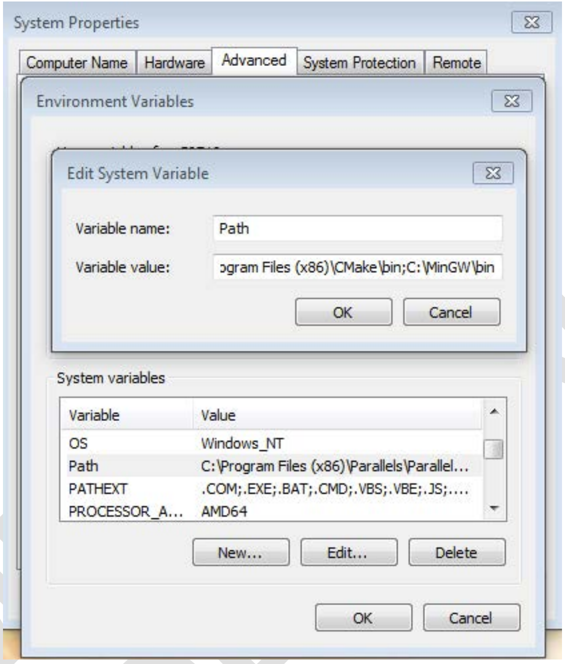

Figure 12. Add Path to systems environment

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.

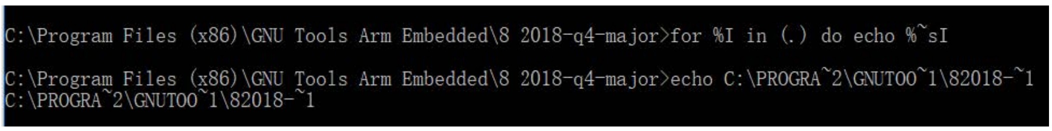

Figure 13. Convert path to short path

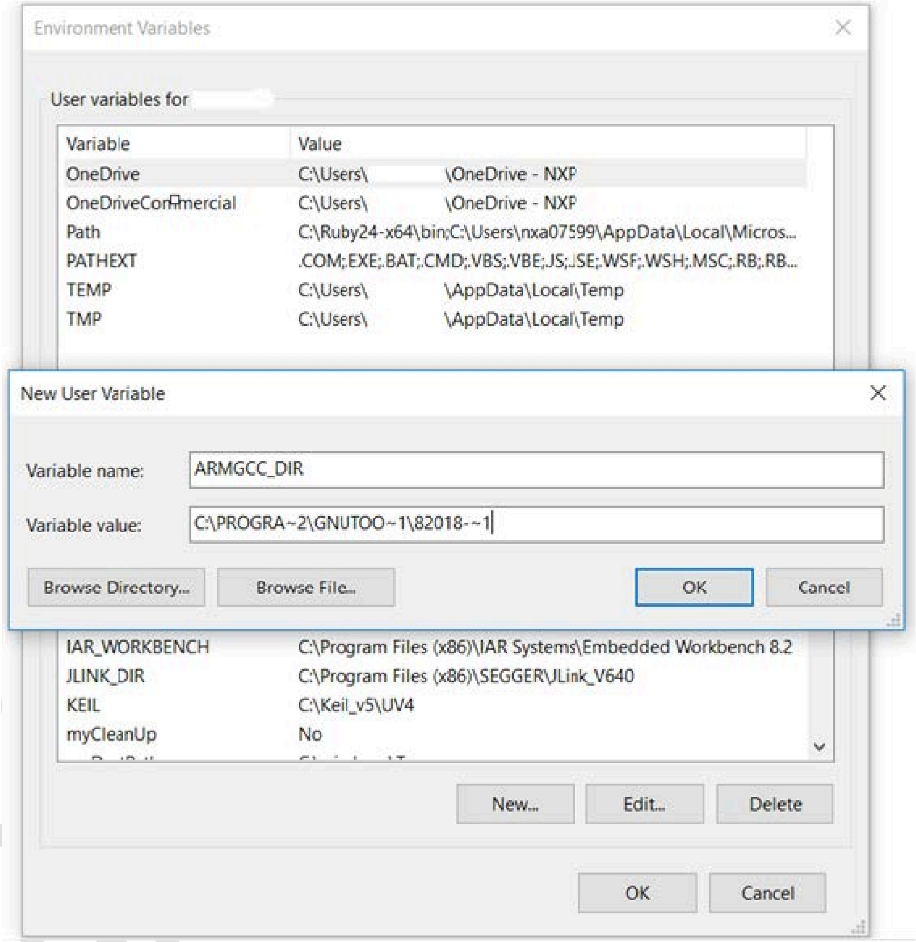

Figure 14. Add ARMGCC_DIR system variable

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.

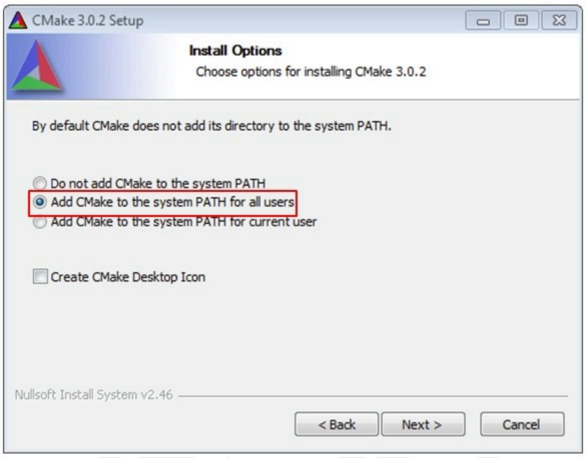

Figure 15. Install CMake

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**.

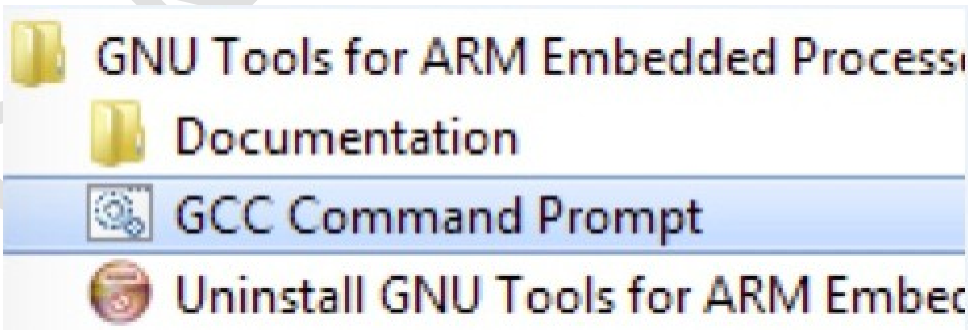

Figure 16. Launch command prompt

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 Section 6). 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

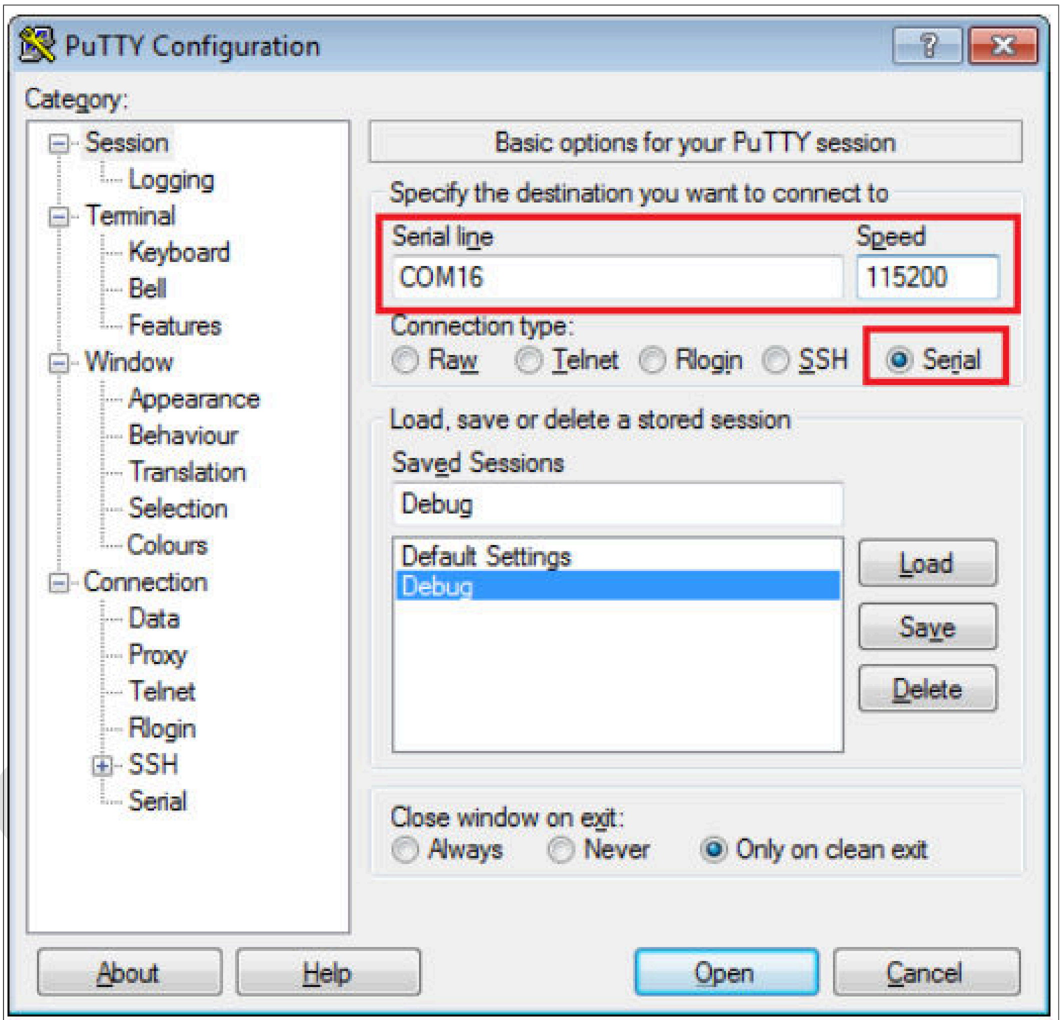

Figure 18. Terminal (PuTTY) configurations

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.

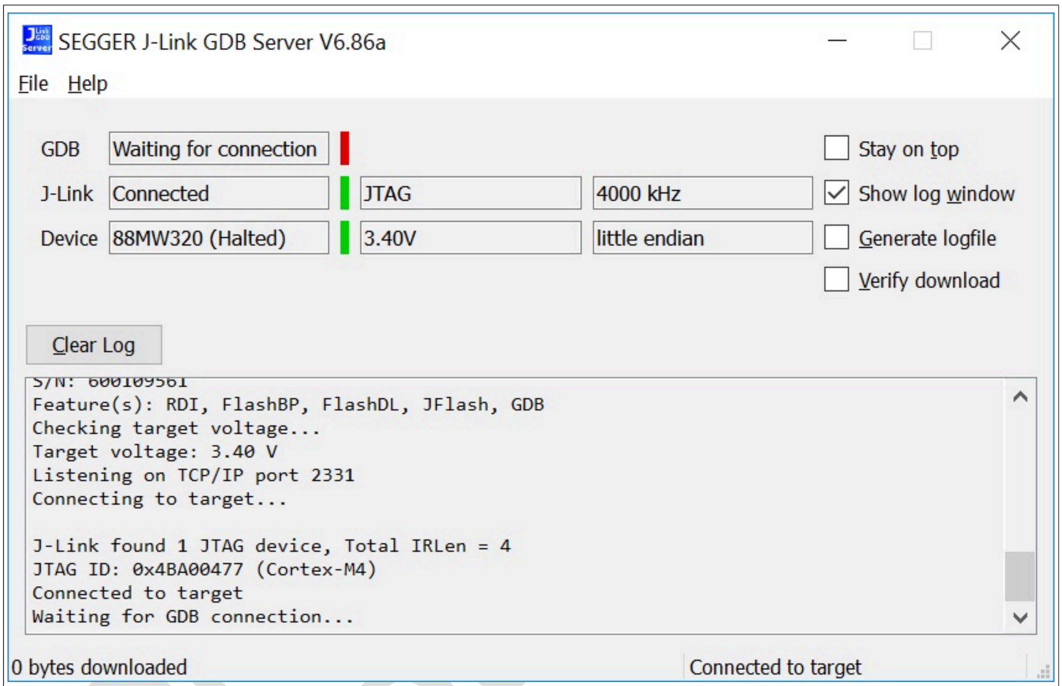

Figure 19. SEGGER J-Link GDB Server screen after successful connection

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**.

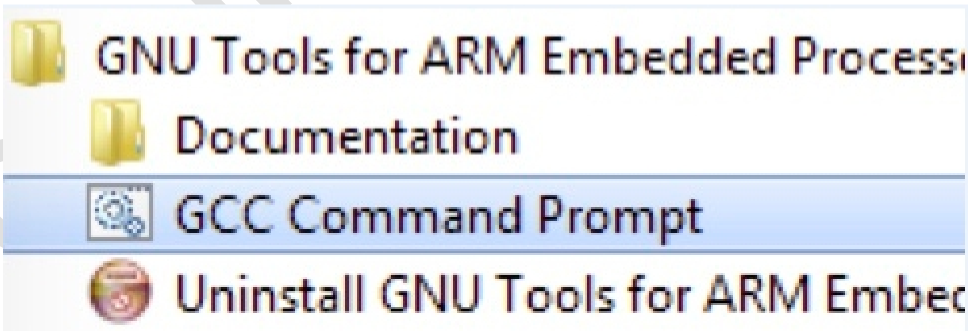

Figure 20. Launch command prompt

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`.

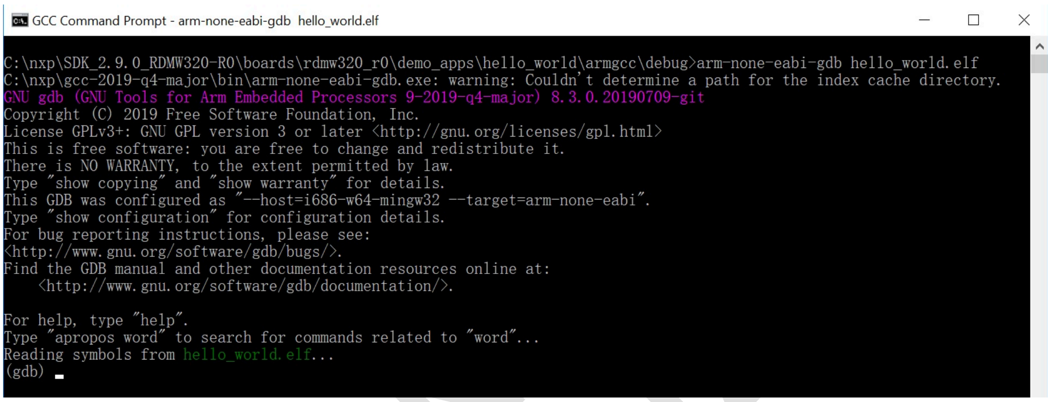

Figure 21. Run arm-none-eabi-gdb

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.

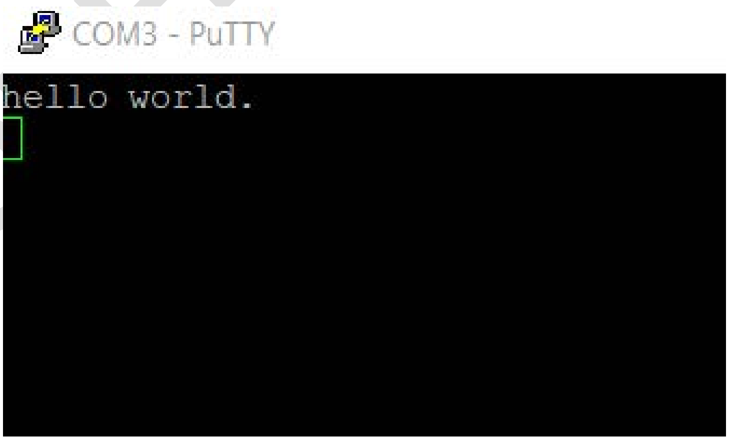

Figure 22. Text display of the hello_world demo

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.

Note:

Now we directly provide Bootable WiFi Firmware at location:

\boards\rdmw320_r0\wifi_examples\common

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\mw320\mw_img_conv.py* to convert the configuration file to binary. For example,

```
# mw_img_conv.py 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: *For IAR and armgcc compiler, 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 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
\mw320\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.

6 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. To open the manager, go to the Windows operating system **Start** menu and type **Device Manager** in the search bar, as shown in Figure 23.

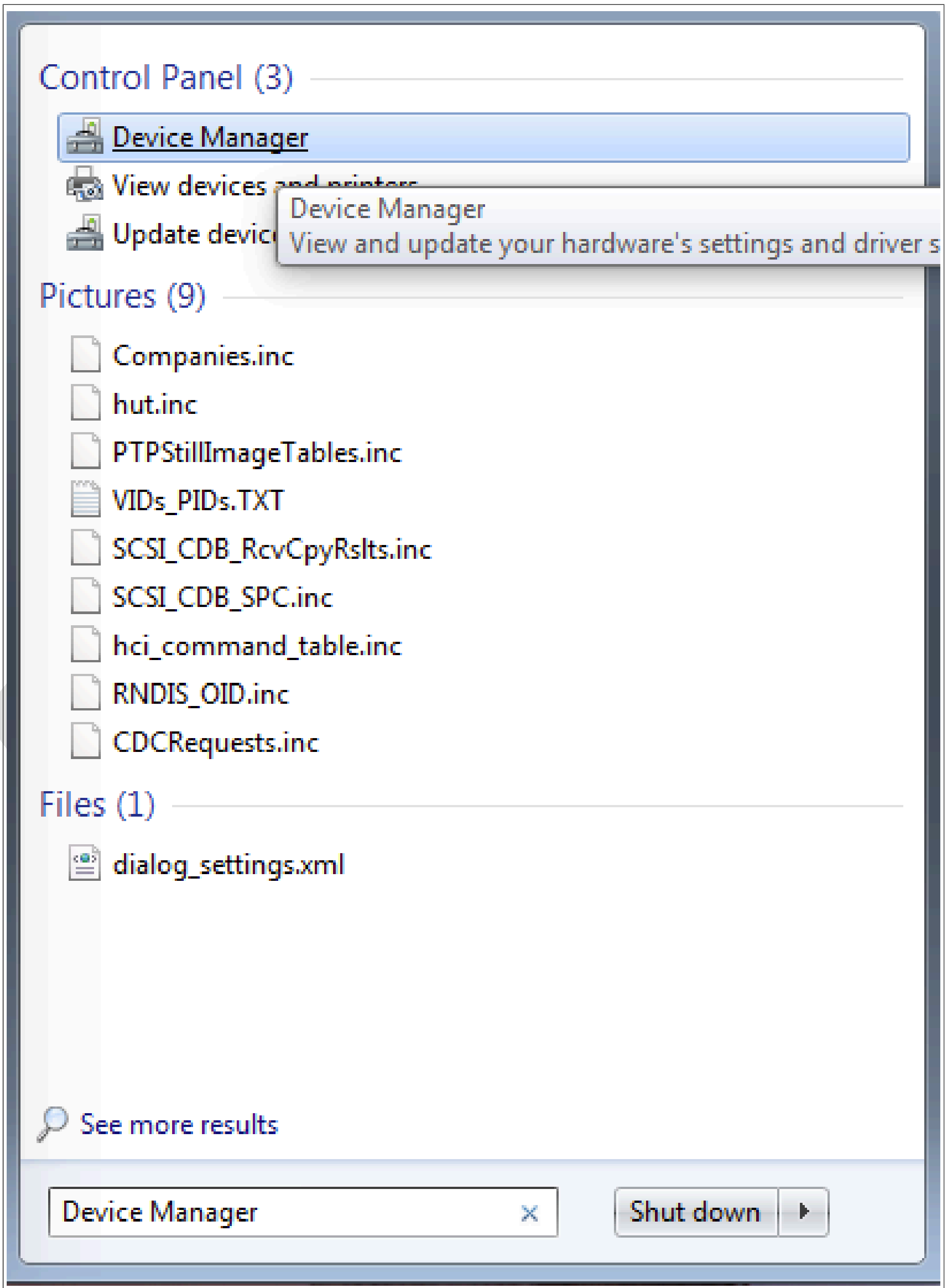

Figure 23. Device Manager

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

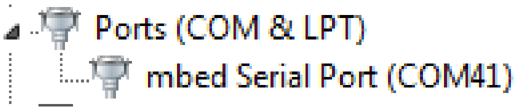

Figure 24. OpenSDA – CMSIS-DAP/mbed/DAPLink interface

b. OpenSDA – P&E Micro:

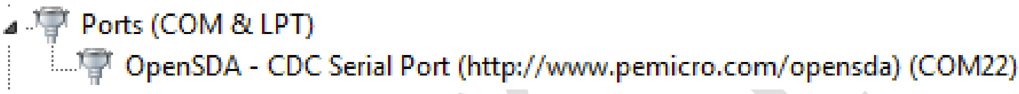

Figure 25. OpenSDA – P&E Micro

c. OpenSDA – J-Link:

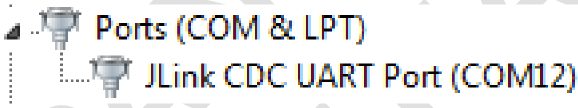

Figure 26. OpenSDA – J-Link

d. P&E Micro OSJTAG:

Figure 27. P&E Micro OSJTAG

e. LPC-Link2:

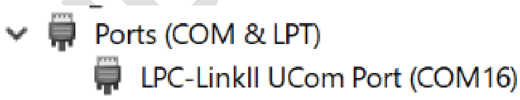

Figure 28. LPC-Link2

f. FTDI UART:

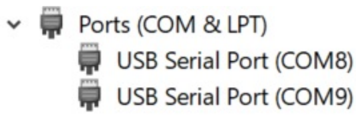

Figure 29. FTDI UART

7 Revision history

Rev.	Date	Description
1	28 July 2022	Updated for MCUXpresso SDK for MW320 release: • Updated code in Section 5.1 • Removed Section Create bootable WIFI firmware and add the information in Section 5 • Updated code in Section 5.3
0	14 March 2022	Initial release

8 Legal information

8.1 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.

8.2 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.

8.3 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.

Contents

1	**Overview**	**2**
2	**MCUXpresso SDK board support package folders**	**2**
2.1	Example application structure	3
2.2	Locating example application source files	3
3	**Run a demo application using IAR**	**4**
3.1	Build an example application	4
3.2	Run an example application	5
4	**Run a demo using Arm® GCC**	**7**
4.1	Set up toolchain	7
4.1.1	Install GCC Arm Embedded tool chain	7
4.1.2	Install MinGW (only required on Windows OS)	8
4.1.3	Add a new system environment variable for ARMGCC_DIR	9
4.1.4	Install CMake	10
4.2	Build an example application	11
4.3	Run an example application	12
5	**Boot a demo application**	**15**
5.1	Create partition table	16
5.2	Create bootable MCU firmware	16
5.3	Flash the device	16
6	**How to determine COM port**	**17**
7	**Revision history**	**19**
8	**Legal information**	**20**

Date of release: 28 July 2022
Document identifier: MCUXSDKRDMW320R0GSUG