NXP Semiconductors
User's Guide

Document Number: MCUXSDKIMX7ULPGSUG
Rev. 2, 26 May 2020

Getting Started with MCUXpresso SDK for EVK-MCIMX7ULP

Contents

1 Overview

The MCUXpresso Software Development Kit (MCUXpresso SDK) provides comprehensive software source code to be executed in the i.MX 7ULP M4 core. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications which can be used standalone or collaboratively with the A cores running another Operating System (such as Linux OS Kernel). Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to demo applications. The MCUXpresso SDK also contains RTOS kernels, and device stack, and various other middleware to support rapid development.

For supported toolchain versions, see the *MCUXpresso SDK Release Notes Supporting i.MX 7ULP Derivatives* (document MCUXSDKIMX7ULPRN)

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

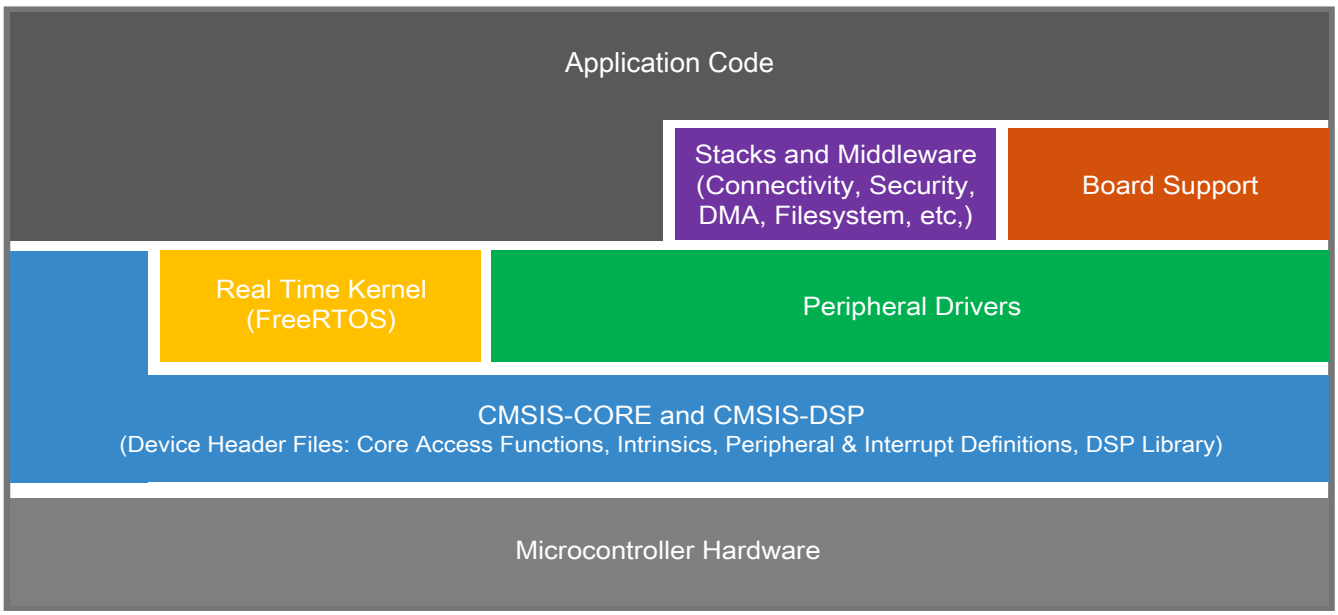

Figure 1. MCUXpresso SDK layers

2 MCUXpresso SDK Board Support Folders

MCUXpresso SDK provides example applications for development and evaluation boards. Board support packages are found inside of the top level <board_name> 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 for each examples they contain. These include (but are not limited to):

- demo_apps: 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.
- rtos_examples: Basic FreeRTOS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the MCUXpresso SDK's RTOS drivers
- cmsis_driver_examples: Simple applications intended to concisely illustrate how to use CMSIS drivers.
- multicore_examples: Simple applications intended to concisely illustrate how to use middleware/multicore stack.
- mmcau_examples: Simple applications intended to concisely illustrate how to use middleware/mmcau stack.

2.1 Example application structure

This section describes how the various types of example applications interact with the other components in the MCUXpresso SDK. To get a comprehensive understanding of all MCUXpresso SDK components and folder structure, see *MCUXpresso SDK API Reference Manual*.

Each `<board_name>` folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. Although we use the `hello_world` example (part of the `demo_apps` folder), the same general rules apply to any type of example in the `<board_name>` folder.

In the `hello_world` application folder you see the following contents:

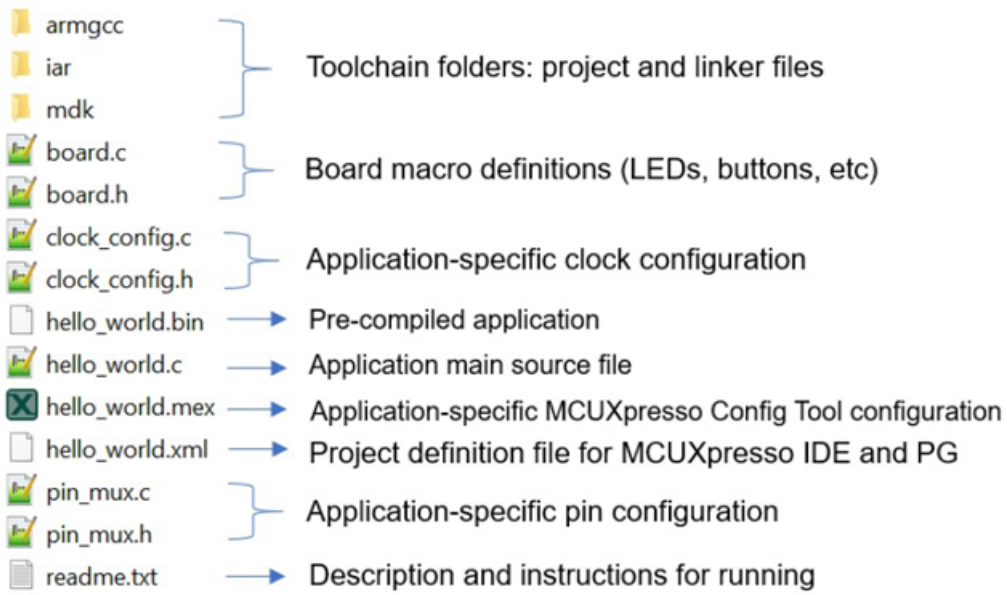

Figure 2. Application folder structure

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

2.2 Locating example application source files

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

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

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

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the `rtos` folder. The core files of each of these are shared, so modifying one could have potential impacts on other projects that depend on that file.

NOTE

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

3 Run a demo application using IAR

This section describes the steps required to build, run, and debug example applications provided in the MCUXpresso SDK. The `hello_world` demo application targeted for the MCIMX7ULP-EVK hardware platform is used as an example, although these steps can be applied to any example application in the MCUXpresso SDK.

3.1 Build an example application

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

   `<install_dir>/boards/evkmcimx7ulp/demo_apps/hello_world/iar/hello_world.eww`

   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**.

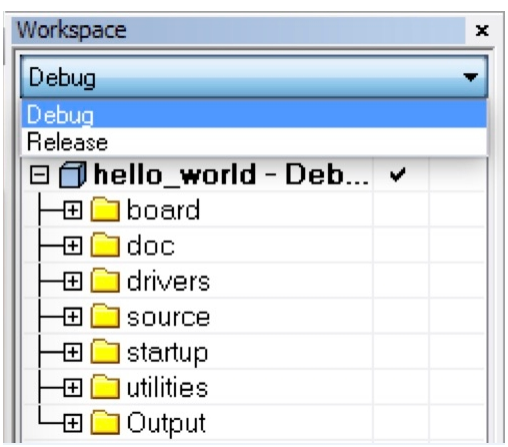

Figure 3. Demo build target selection

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

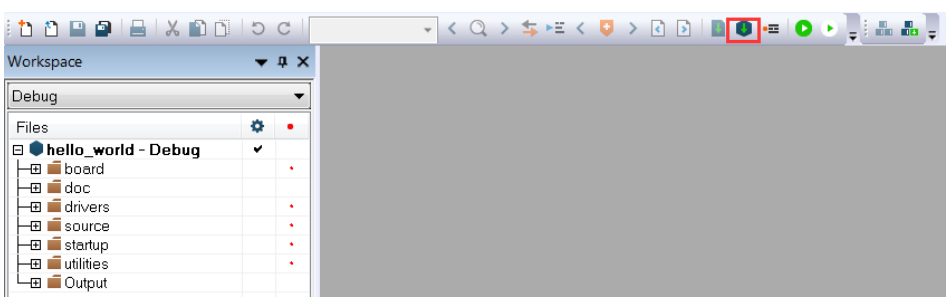

Figure 4. Build the demo application

4. The build completes without errors.

3.2 Run an example application

To download and run the application, perform these steps:

1. This board supports the J-Link debug probe. Before using it, install SEGGER J-Link software, which can be downloaded from www.segger.com.
2. Connect the development platform to your PC via USB cable between the USB-UART MICRO USB connector and the PC USB connector, then connect 5 V power supply and J-Link Plus to the device.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   a. 115200 baud rate
   b. No parity
   c. 8 data bits
   d. 1 stop bit

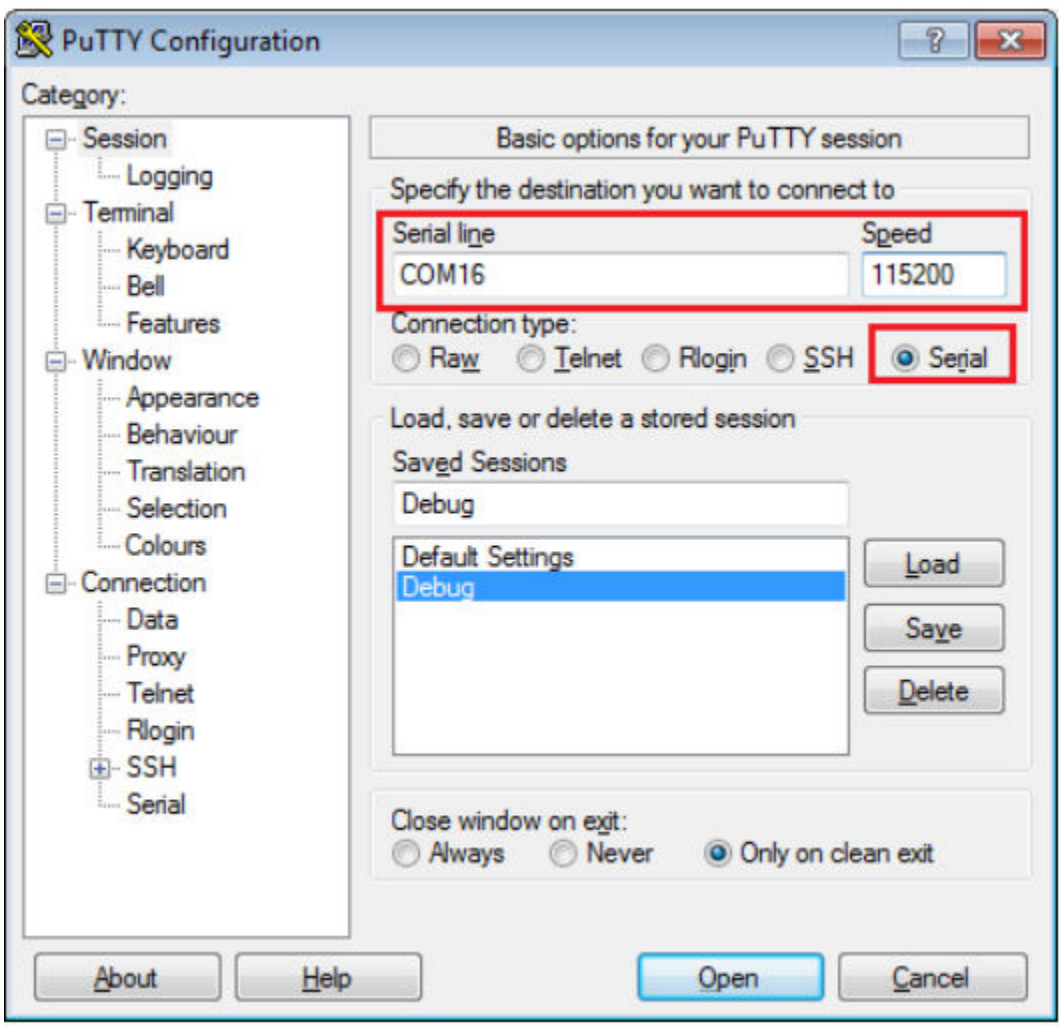

Figure 5. Terminal (PuTTY) configuration

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.

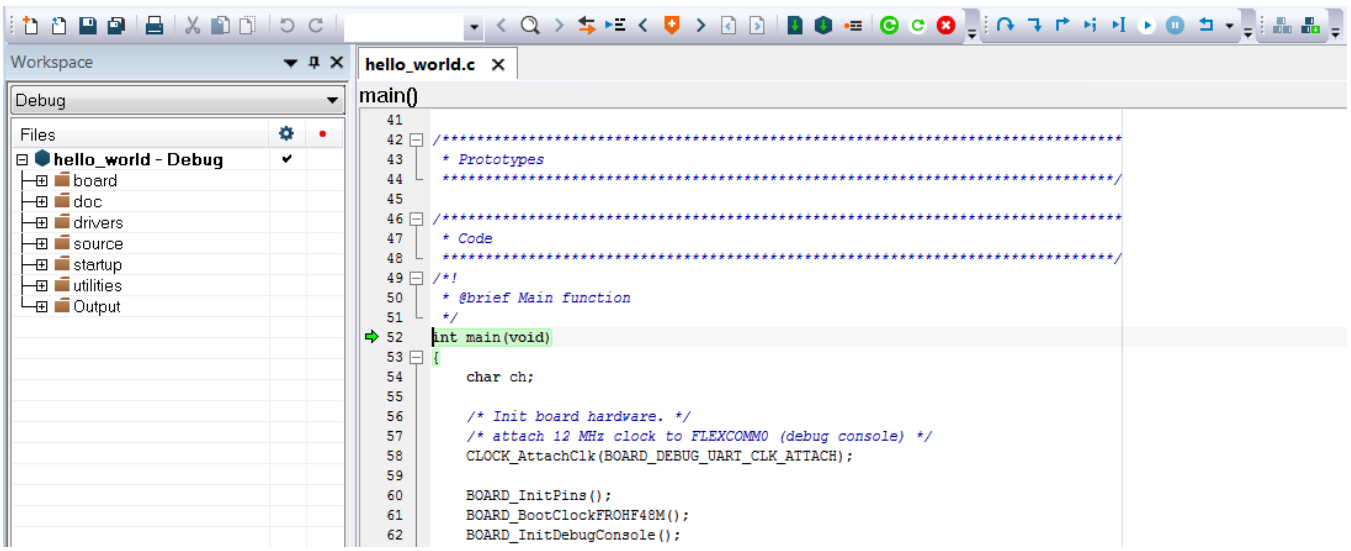

Figure 7. Stop at main() when running debugging

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.

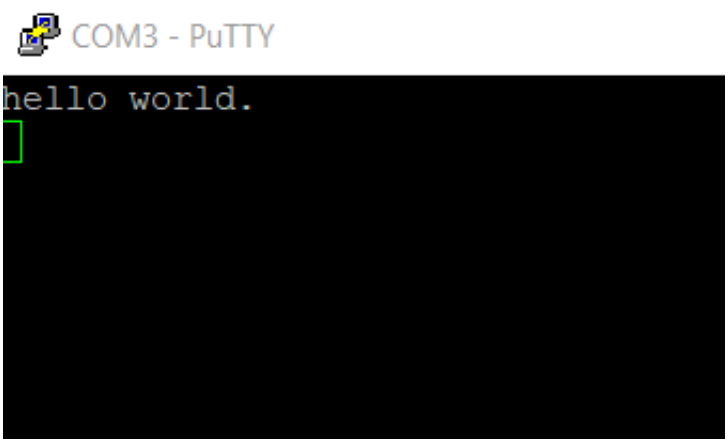

Figure 9. Text display of the hello_world demo

3.3 Debug QSPI FLASH XIP Application

Most demo applications use the RAM linker file by default. If users want to use the flash linker file for QSPI XIP debugging, the linker file for QSPI FLASH needs to be changed from the project default RAM linker file to the FLASH linker file.

1. Open the hello_world project and select the hello_world top level project as shown below. Once highlighted, one way to open the options is using 'Alt-F7'. This opens the option window. Then, select “Linker”. The “Config” tab is shown by default. Enable Override default and enter the location: *C:\nxp\SDK_2.3_EVK_MCIMX7ULP\devices\MCIMX7U5\iar\MCIMX7U5xxx08_flash.icf*.

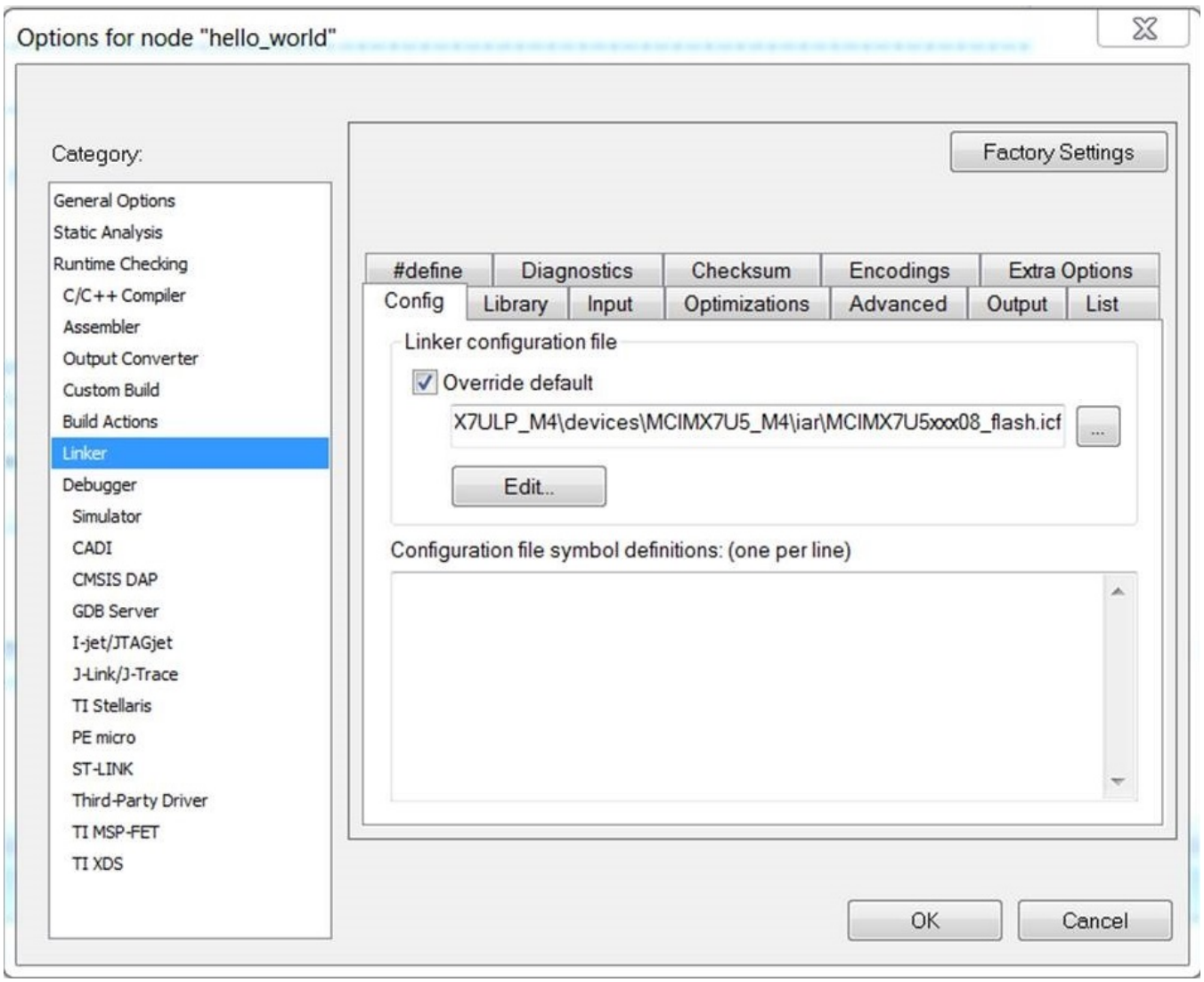

Figure 10. Options for node "hello_world"

Clean the project once the linker control has been configured. Use the key combo 'Alt-P' to clean. Then, make project using the 'F7' key.

2. Select the "Use macro file(s)" and the "Use flash loader(s)" as shown in the following pictures, and start debugging in IAR.

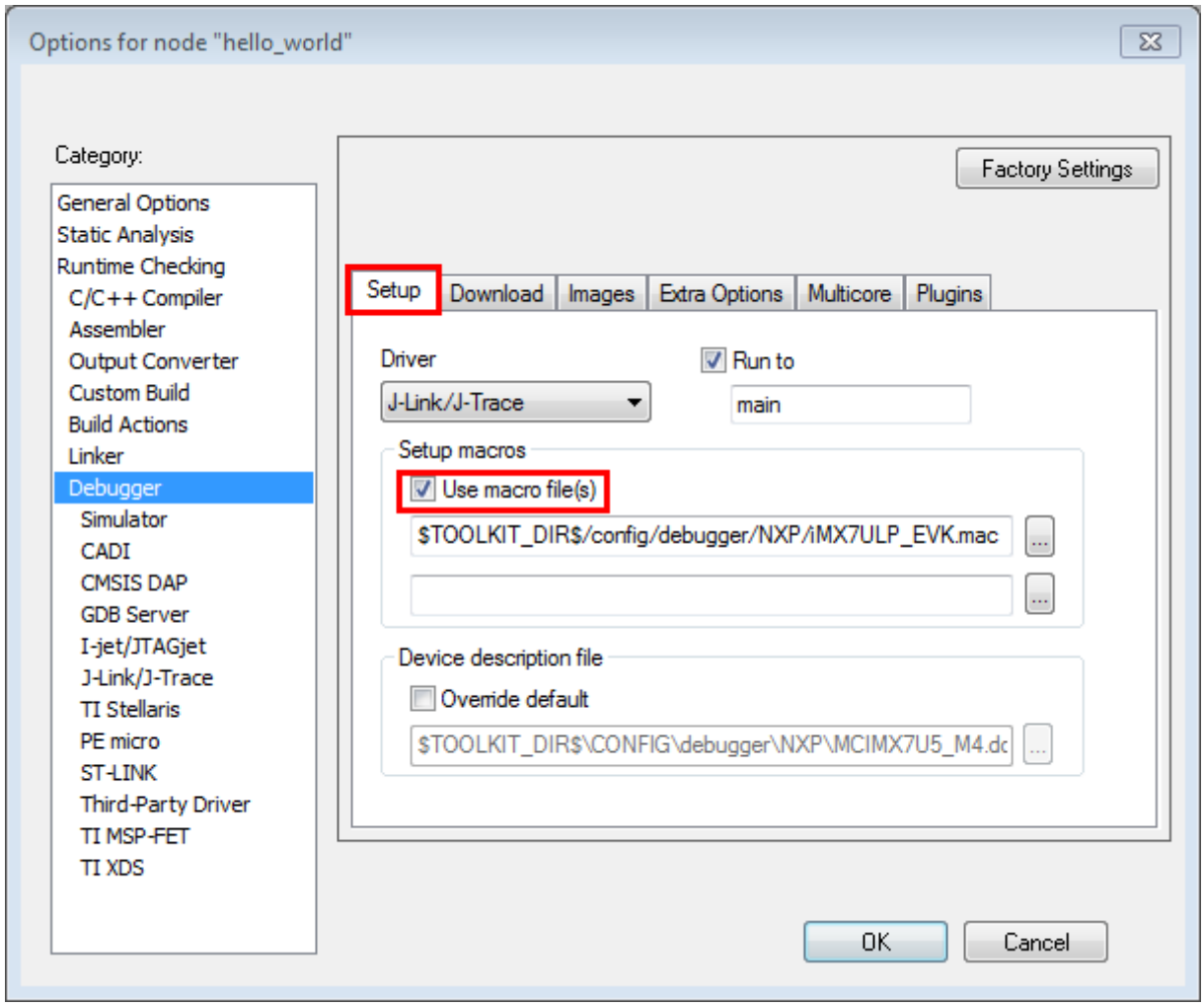

Figure 11. "Use macro file(s)" selection

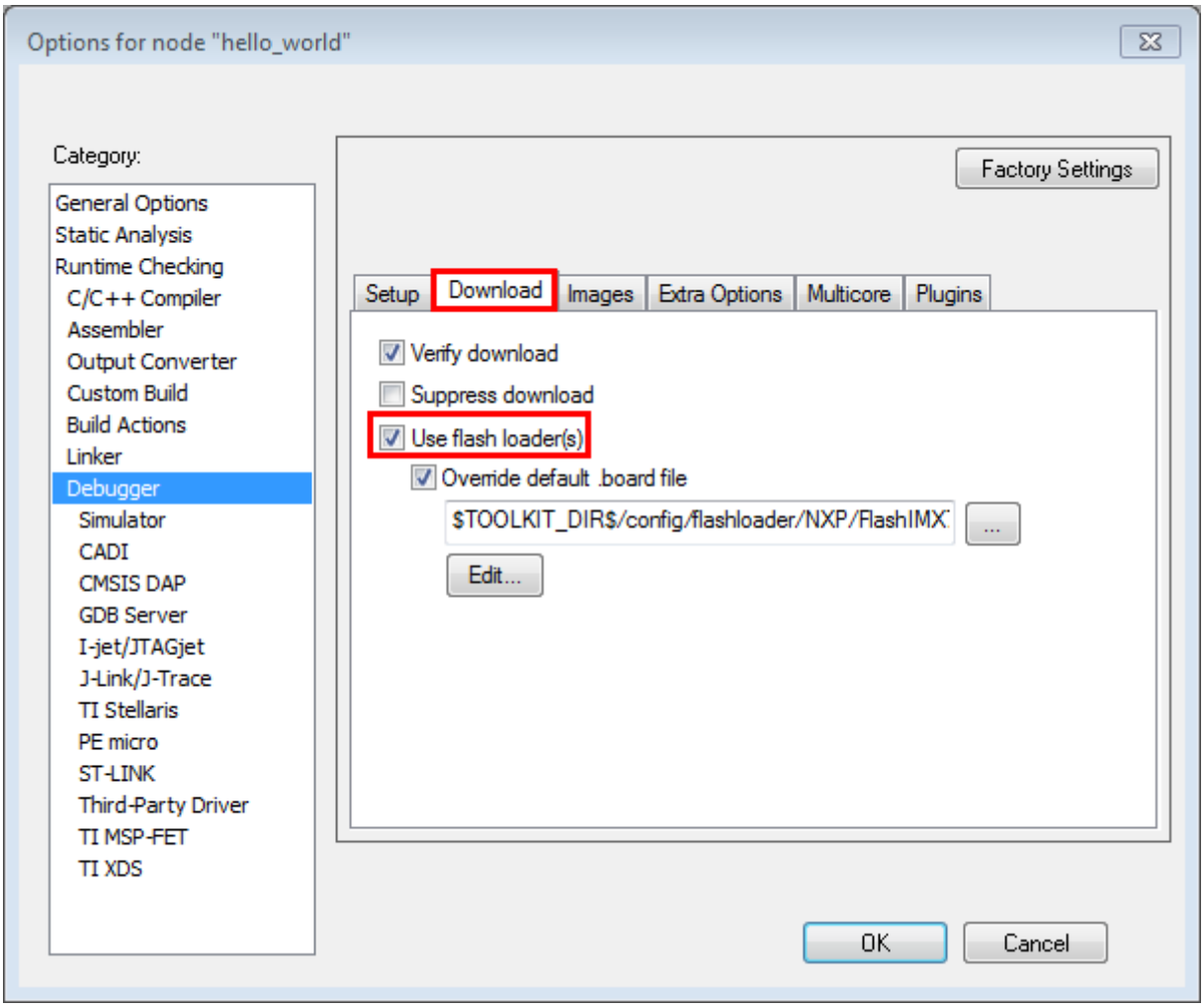

Figure 12. "Use flash loader(s)" selection

4 Running an application from QSPI flash

This section describes the steps to write a bootable SDK image to QSPI flash with the prebuilt U-Boot image for the i.MX processor. The following steps describe how to use the U-Boot:

1. Connect the “DEBUG UART” slot on the board to your PC through the USB cable. The Windows® OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.
2. On Windows OS, open the device manager, find “USB serial Port” in “Ports (COM and LPT)”. Assume that the ports are COM9 and COM10. One port is for the debug message from the Cortex®-A7 and the other is for the Cortex®-M4. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name /dev/ttyUSB* to determine your debug port. Similar to Windows OS, opening both is beneficial for development.

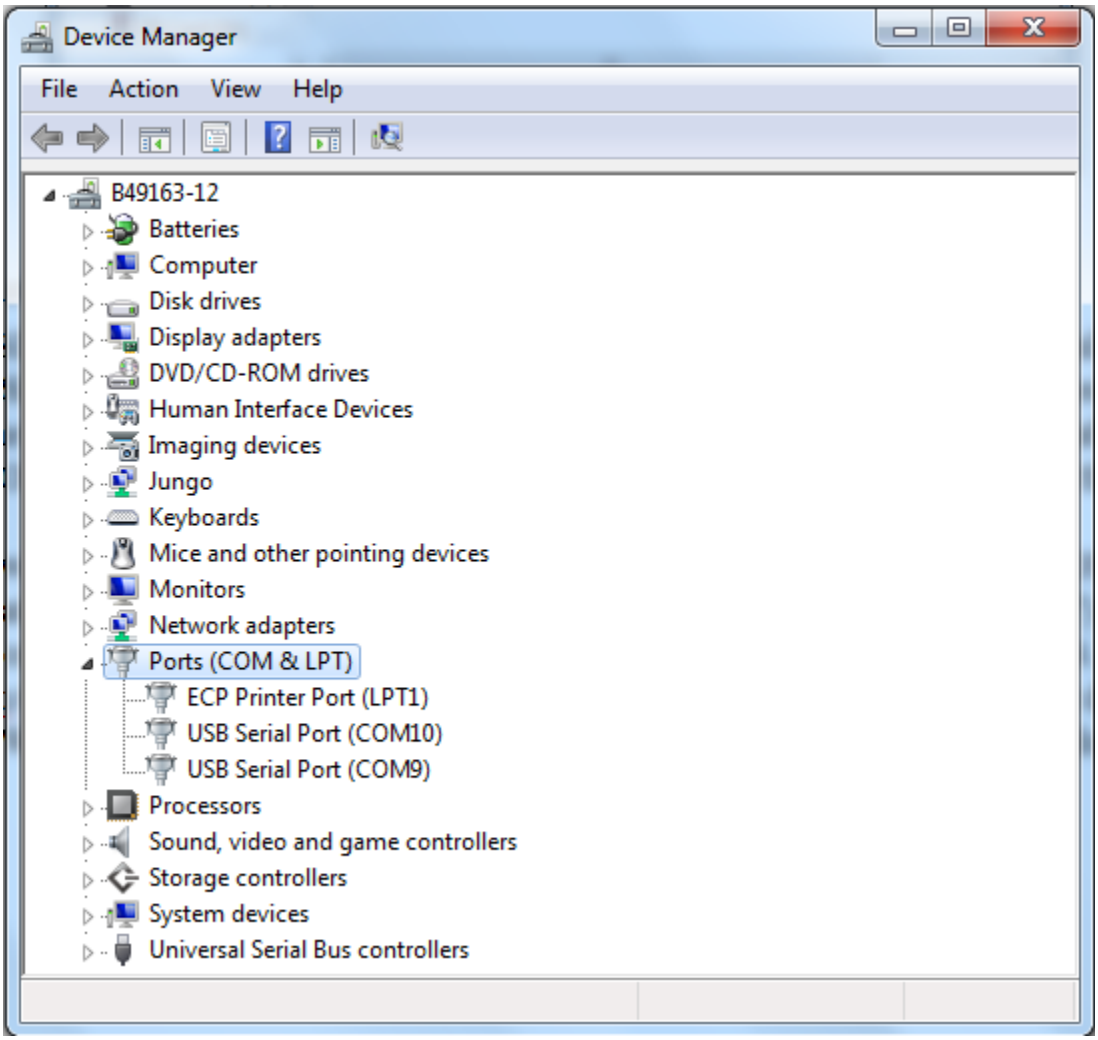

Figure 13. Determining the COM port of target board

3. Build the application (for example, hello_world) and copy the built binary (sdk20-app.bin file) to the *<install_dir>/tools/imgutil/evkmcimx7ulp* folder.
4. In the *<install_dir>/tools/imgutil/evkmcimx7ulp*p folder, run mkimg.sh in mingw32 shell to get bootable image file sdk20- app.img.
   - If the image is built with RAM link file, use "mkimg.sh ram" to create the bootable image.
   - If the image is built with flash link file, use "mkimg.sh flash" to create the bootable XIP.
5. Prepare an SD card with the prebuilt U-Boot image and copy the sdk20-app.img generated into the SD card (as shown in Step 4). Then, insert the SD card to the target board. Make sure to use the default boot SD slot and check the dip switch configuration.
6. Open your preferred serial terminals for the serial devices, setting the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity, then power on the board.
7. Power on the board and hit any key to stop autoboot in the terminals, then enter to U-Boot command line mode. You can then write the image and run it from QSPI Flash with the following commands (Assume image size is less than 0x20000, otherwise the sf erase and write command size parameter need to be enlarged. If the image size is bigger than 0x20000 (128 KB), change 0x20000 to a number larger or equal to the image size.):
   - sf probe.
   - sf erase 0x0 0x20000.
   - fatload mmc 0:1 0x62000000 sdk20-app.img.
   - sf write 0x62000000 0x0 0x20000.

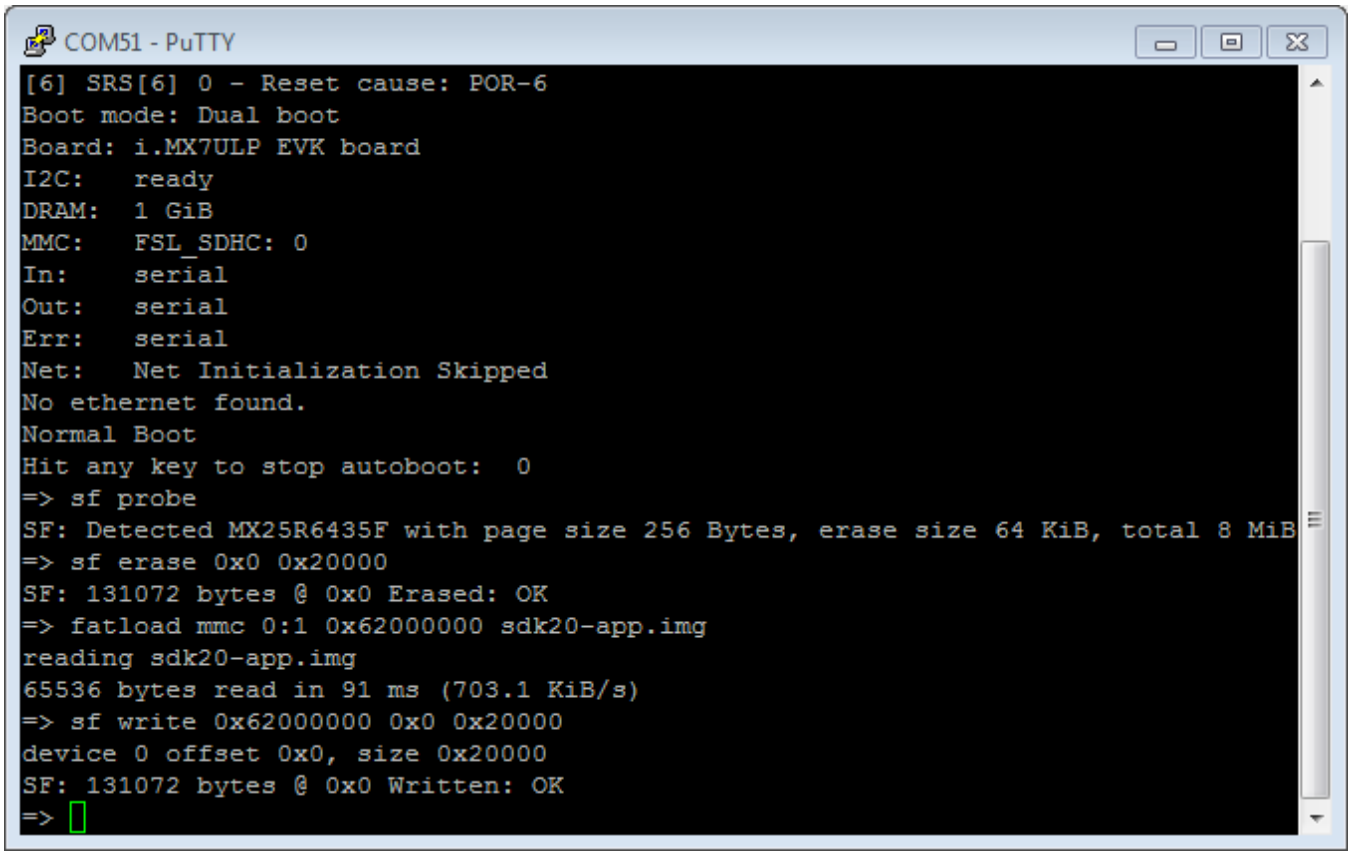

Figure 14. U-Boot command to run application on QSPI

8. Open another terminal application on the PC, such as PuTTY and connect to the debug COM port (to determine the COM port number, see Appendix A). Configure the terminal with these settings:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit
9. Power off and repower on the board.
10. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.

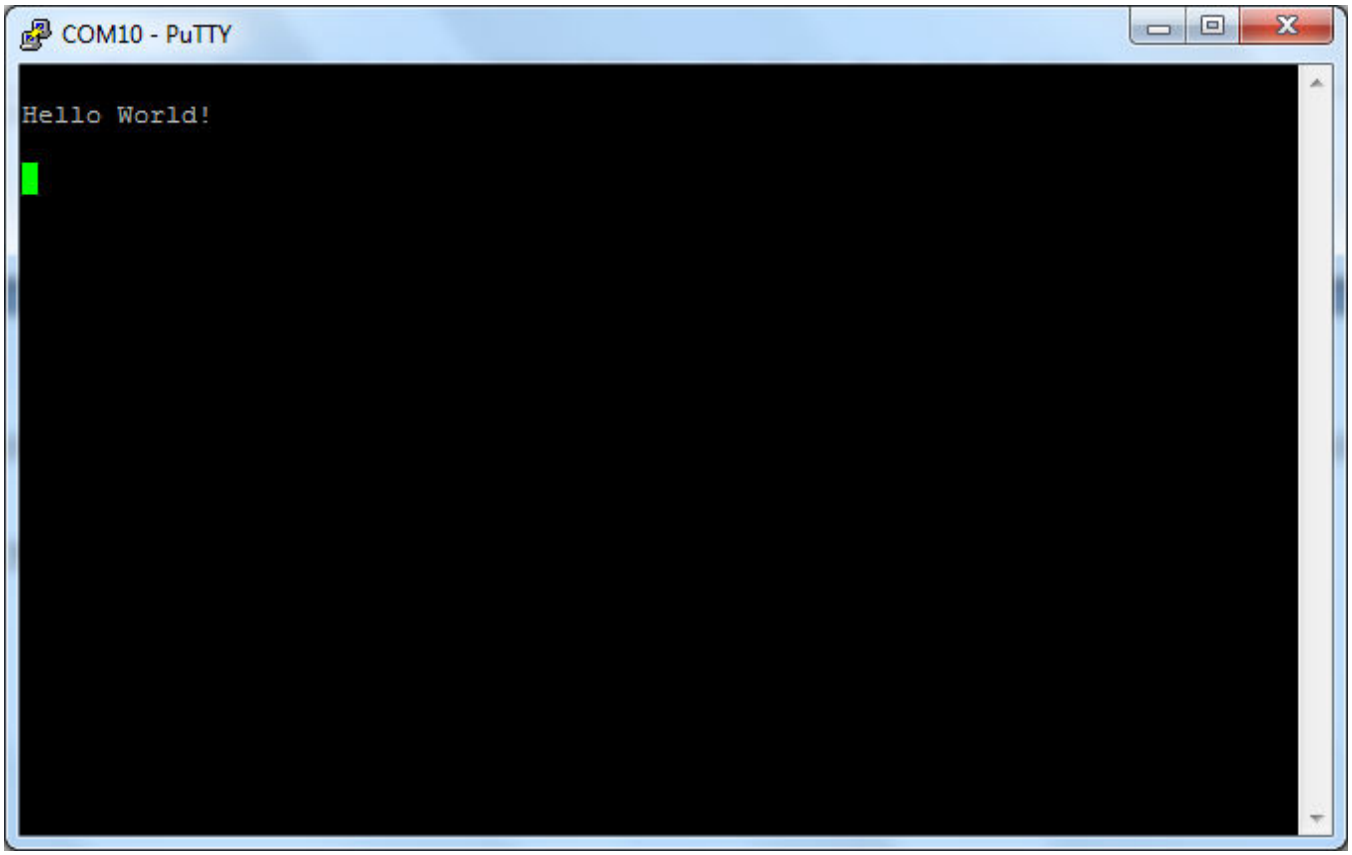

Figure 15. Hello world demo running on Cortex-M4 core

5 How to determine COM port

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

1. **Linux**: The serial port can be determined by running the following command after the USB Serial is connected to the host:

   ```
   $ dmesg | grep "ttyUSB"
                       [503175.307873] usb 3-12: cp210x converter now attached to ttyUSB0
                       [503175.309372] usb 3-12: cp210x converter now attached to ttyUSB1
   ```

   There are two ports, one is Cortex-A core debug console and the other is for Cortex M4.

2. **Windows**: To determine the COM port open Device Manager in the Windows operating system. Click on the **Start** menu and type **Device Manager** in the search bar.

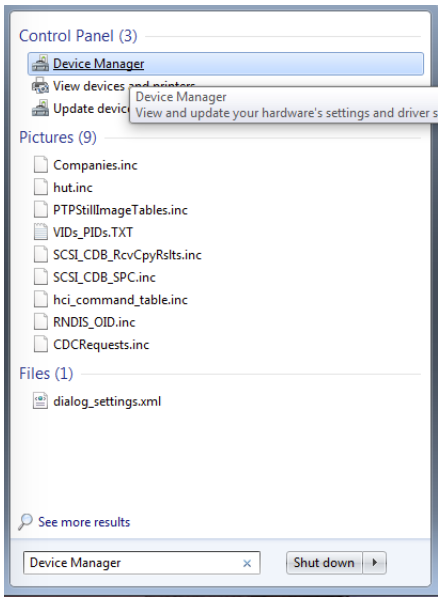

Figure 16. Device Manager

3. In the Device Manager, expand the **Ports (COM & LPT)** section to view the available ports. The COM port names will be different for all the NXP boards.

   a. **USB-UART** interface

Figure 17. 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.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

Document Number MCUXSDKIMX7ULPGSUG
Revision 2, 26 May 2020