MCUXSDKIMX7ULPGSUG

Getting Started with MCUXpresso SDK for EVK-MCIMX7ULP

Rev. 2.14.0 — 27 July 2023 User guide

Document information

Information	Content
Keywords	EVK-MCIMX7ULP, MCIMX7ULP, 7ULP, EVKMCIMX7ULP, Getting Started, MCUXSDKIMX7ULPGSUG
Abstract	This document describes the steps to get started with MCUXpresso SDK for EVK-MCIMX7ULP.

1 Overview

The NXP MCUXpresso software and tools offer comprehensive development solutions designed to optimize, ease, and help accelerate embedded system development of applications based on general purpose, crossover, and Bluetooth-enabled MCUs from NXP. The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications 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 optional RTOS integrations such as FreeRTOS and Azure RTOS, 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.

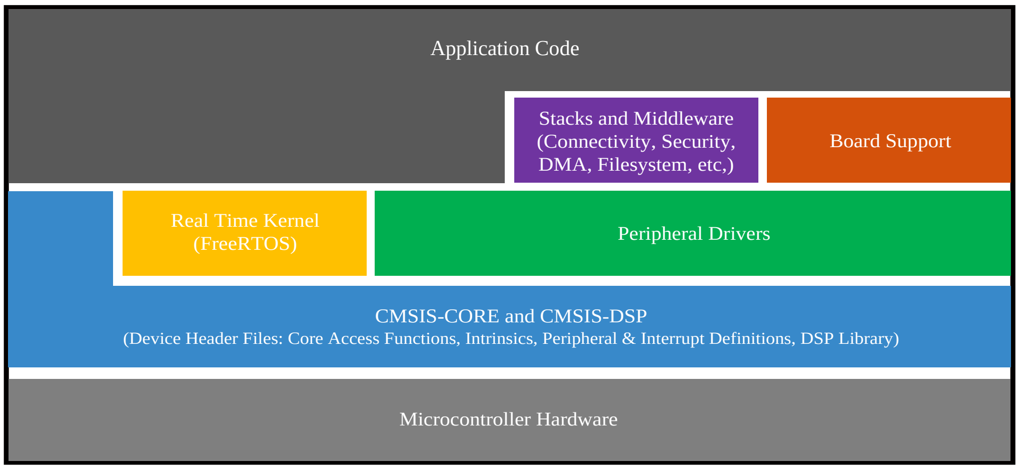

Application Code
Stacks and Middleware (Connectivity, Security, DMA, Filesystem, etc,)
Board Support
Real Time Kernel (FreeRTOS)
Peripheral Drivers
CMSIS-CORE and CMSIS-DSP
(Device Header Files: Core Access Functions, Intrinsics, Peripheral & Interrupt Definitions, DSP Library)
Microcontroller Hardware

Figure 1. MCUXpresso SDK layers

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 subfolders for each example that 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 SDKs 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 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:

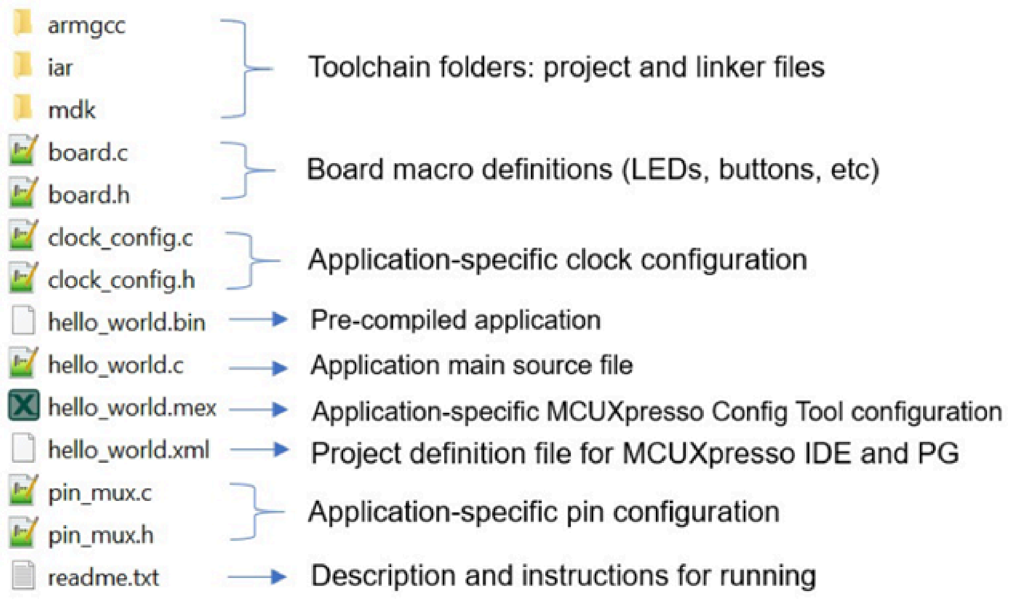

armgcc
iar
mdk
Toolchain folders: project and linker files
board.c
board.h
Board macro definitions (LEDs, buttons, etc)
clock_config.c
clock_config.h
Application-specific clock configuration
hello_world.bin
Pre-compiled application
hello_world.c
Application main source file
hello_world.mex
Application-specific MCUXpresso Config Tool configuration
hello_world.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, various source files are referenced. The MCUXpresso SDK devices folder is the central component to all example applications. It means that 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**.

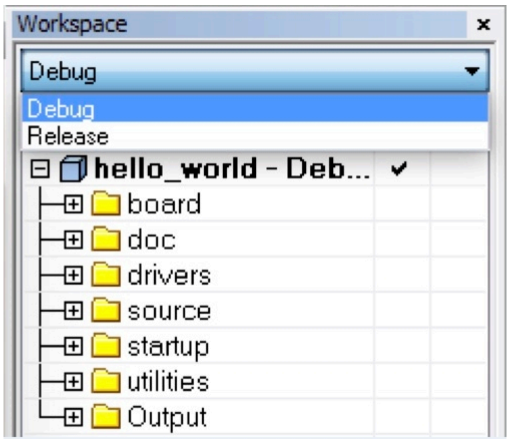


Figure 3. Demo build target selection

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

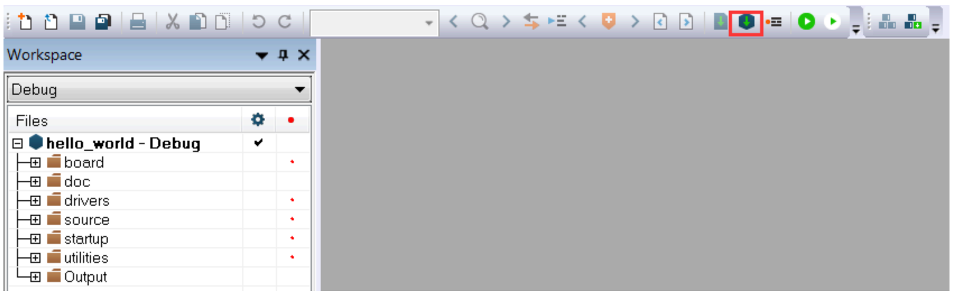


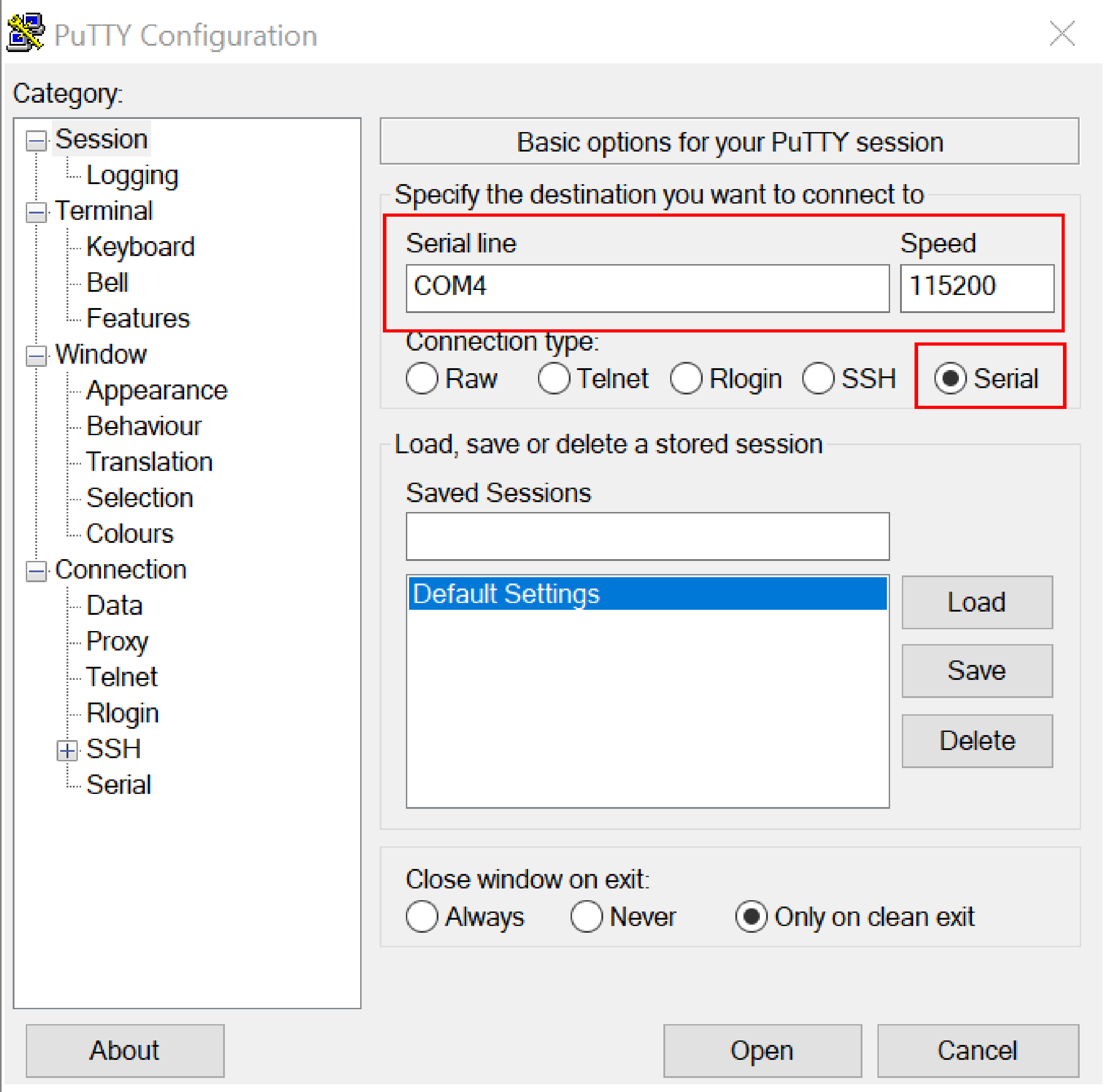
Figure 4. Build the demo application

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



PuTTY Configuration

Category:

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM4

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

Default Settings

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.

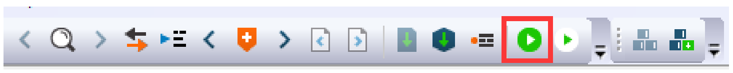
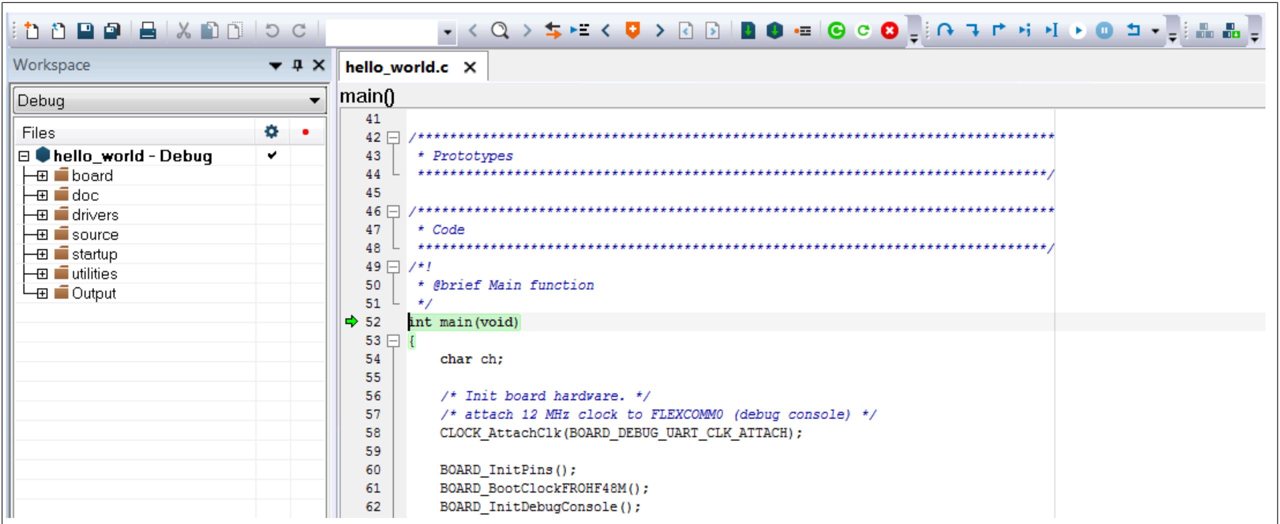


Figure 6. Download and Debug button

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 to start the application.

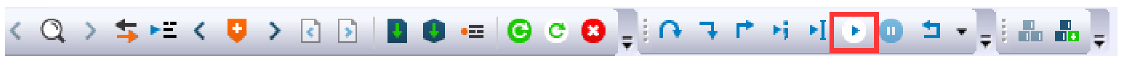
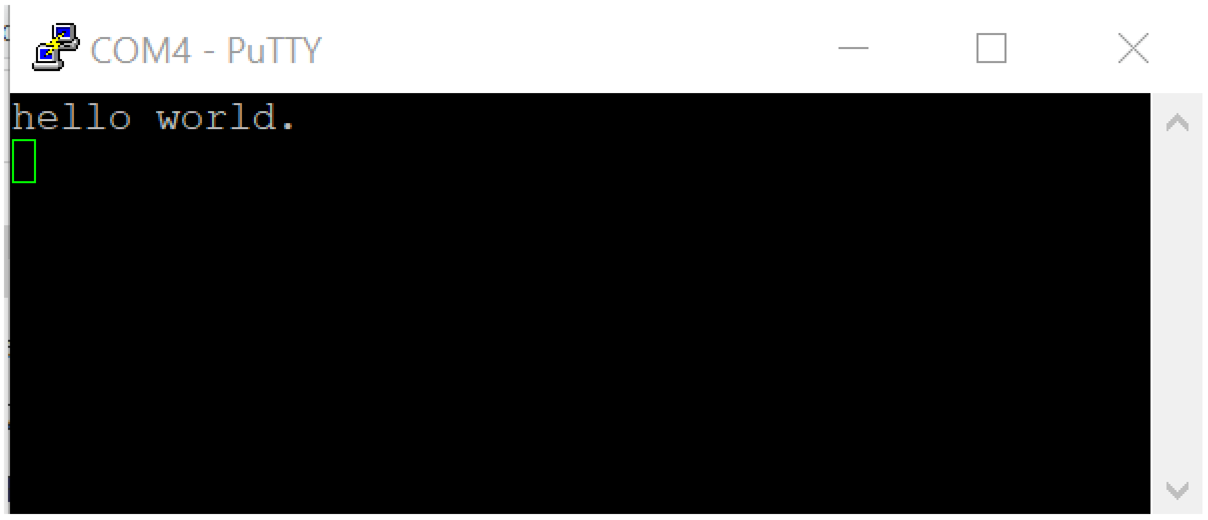


Figure 8. Go button

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.



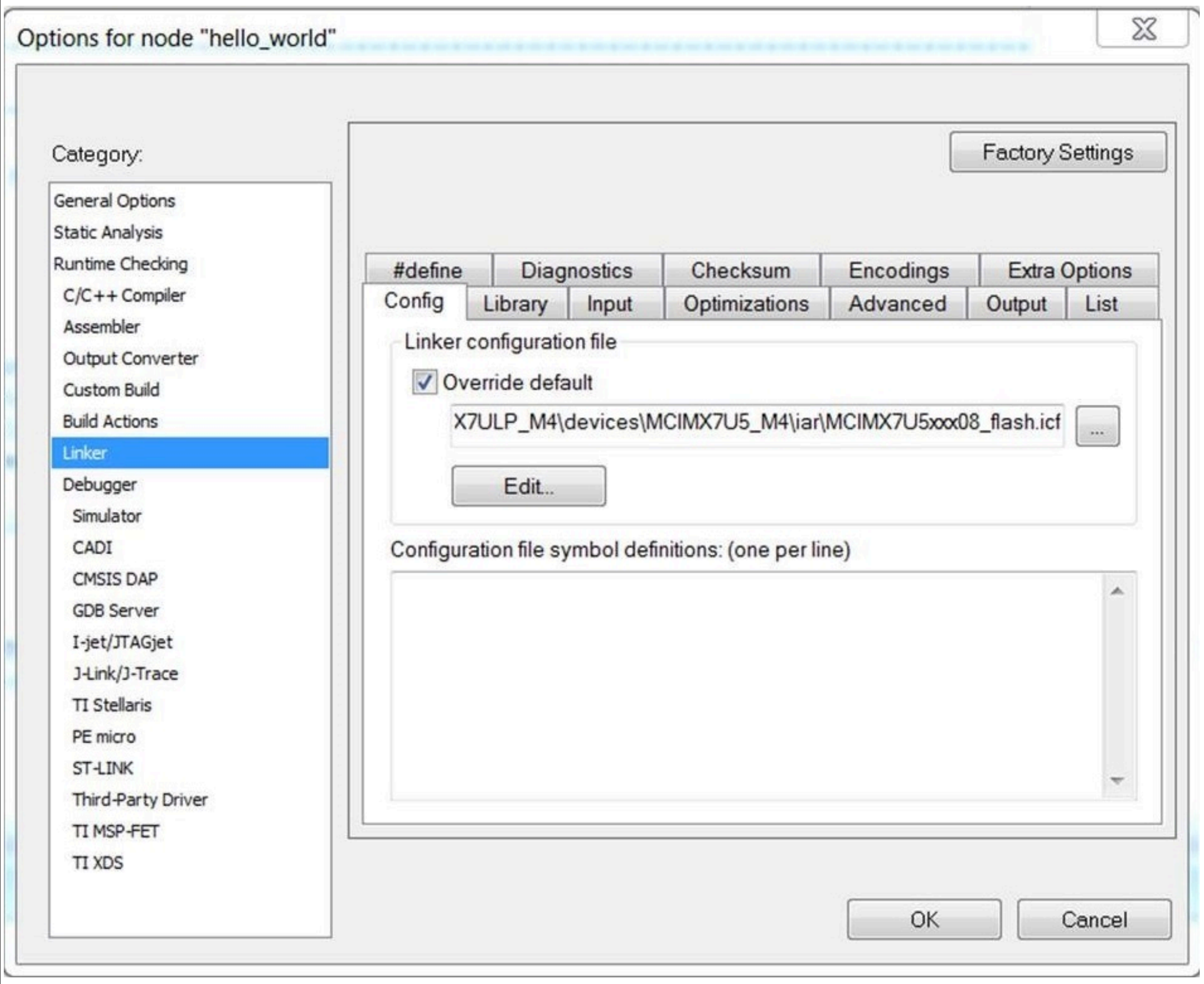
```
hello world.
```

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 must 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*.



Linker configuration file

☑ Override default

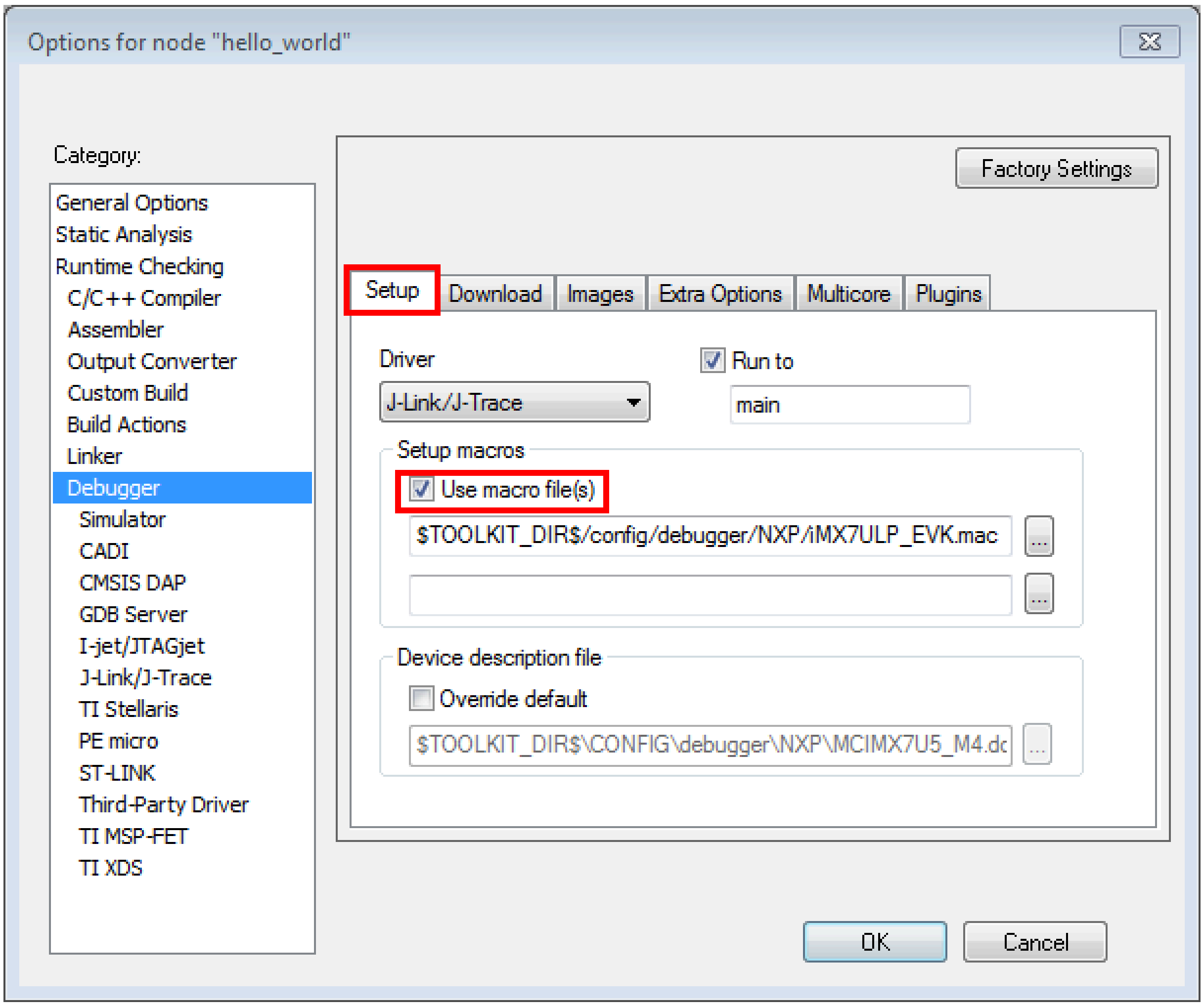
X7ULP_M4\devices\MCIMX7U5_M4\iar\MCIMX7U5xxx08_flash.icf

Configuration file symbol definitions: (one per line)

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 files" and the "Use flashloader" as shown in the following pictures, and start debugging in IAR.



Driver: J-Link/J-Trace

[✓] Run to: main

Setup macros

[✓] Use macro file(s)

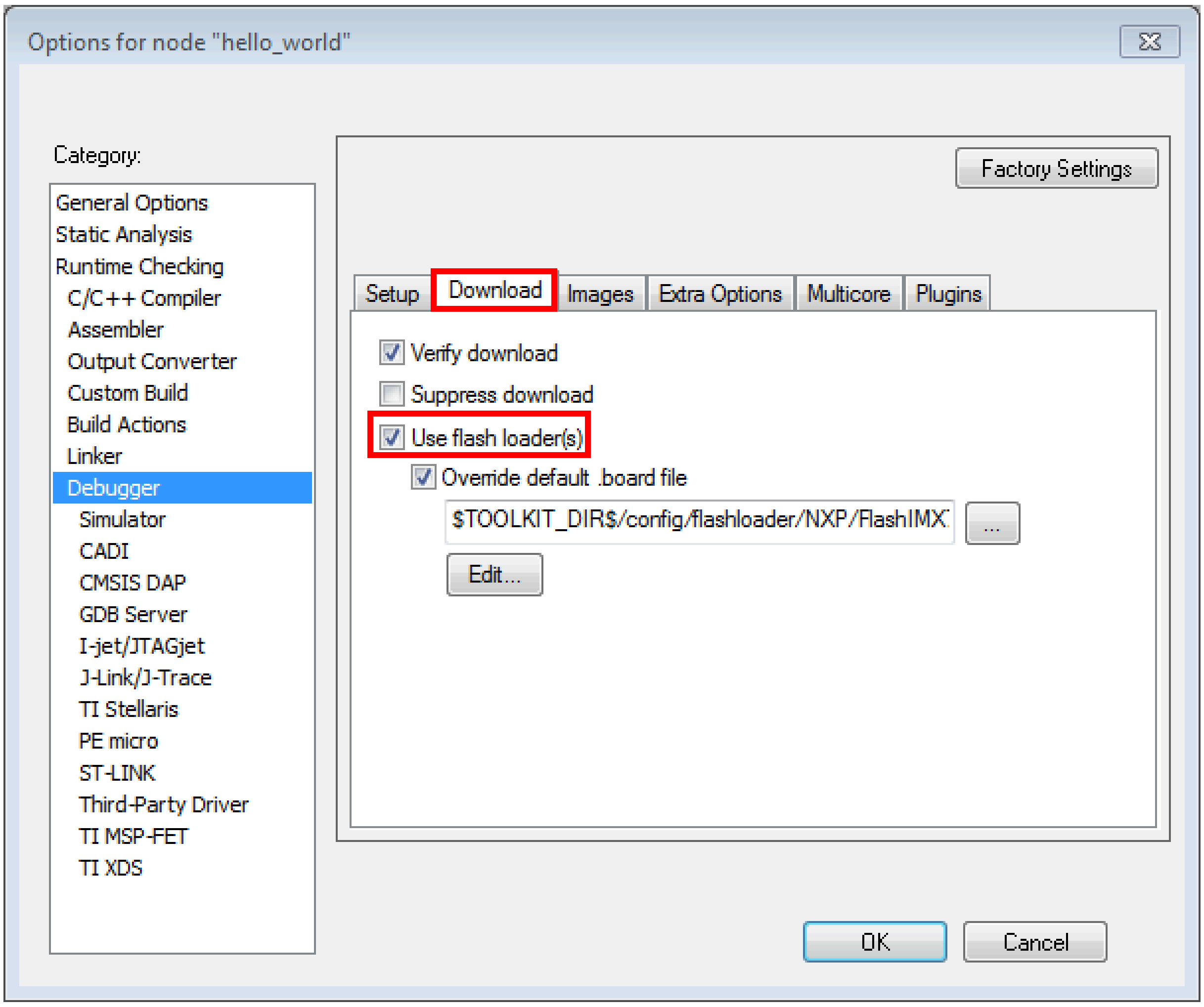
$TOOLKIT_DIR$/config/debugger/NXP/iMX7ULP_EVK.mac

Device description file

[ ] Override default

$TOOLKIT_DIR$\CONFIG\debugger\NXP\MCIMX7U5_M4.dc

Figure 11. "Use macro files" selection



Verify download [✓]
Suppress download [ ]
Use flash loader(s) [✓]
Override default .board file [✓]
$TOOLKIT_DIR$/config/flashloader/NXP/FlashIMX

Figure 12. "Use flashloader " 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.

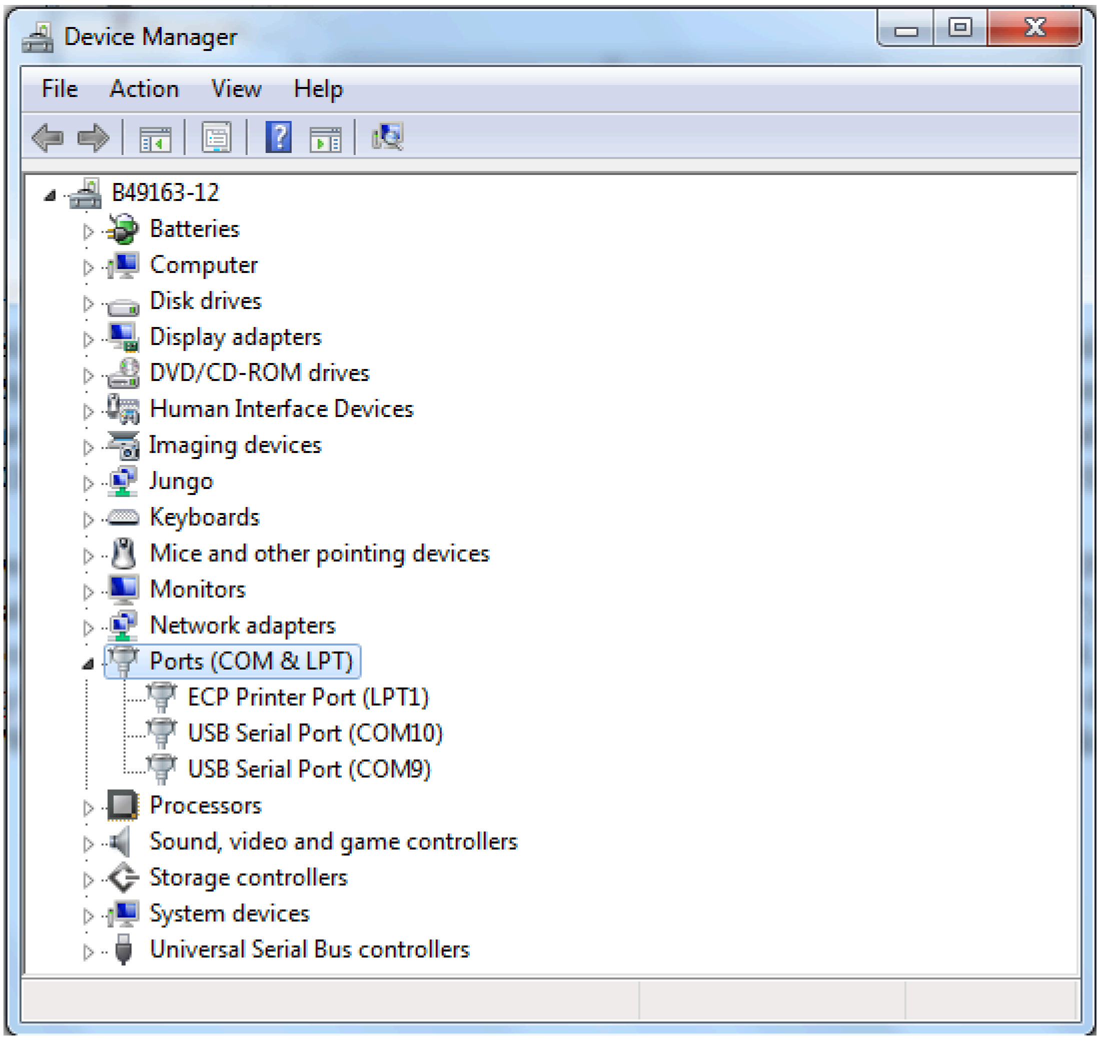
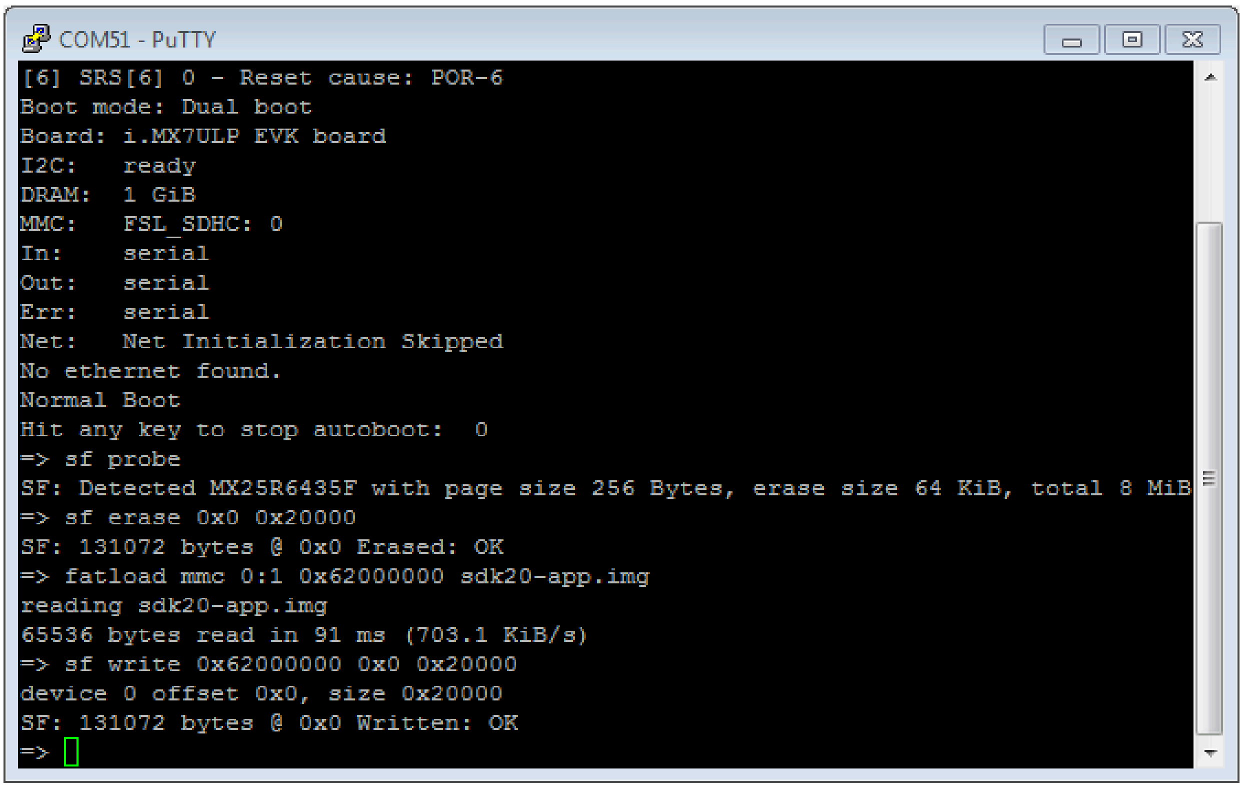


Figure 13. Determining the COM port of target board

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 bit/s, 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 that image size is less than 0x20000, otherwise the sf erase and write command size parameter must 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.

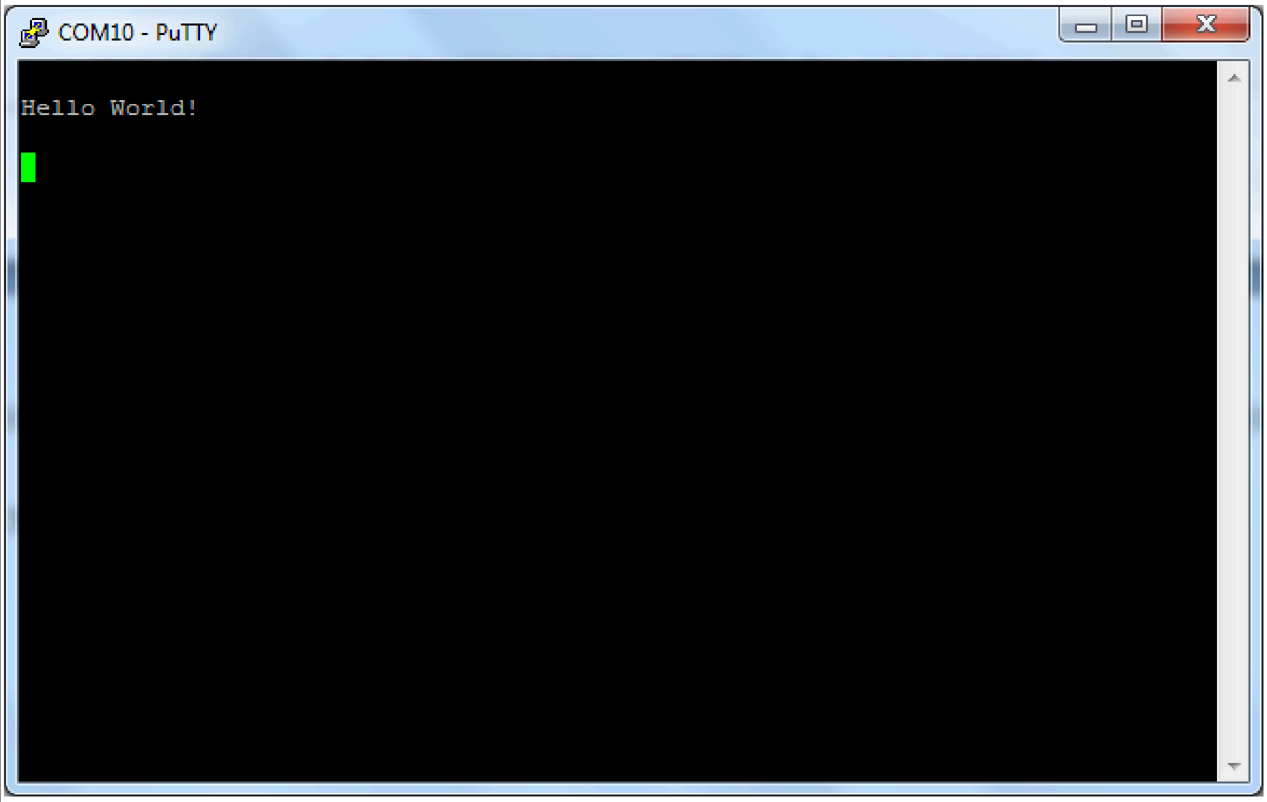
- sf erase 0x0 0x20000.
- fatload mmc 0:1 0x62000000 sdk20-app.img.
- sf write 0x62000000 0x0 0x20000.



```
[6] SRS[6] 0 - Reset cause: POR-6
Boot mode: Dual boot
Board: i.MX7ULP EVK board
I2C:   ready
DRAM:  1 GiB
MMC:   FSL_SDHC: 0
In:    serial
Out:   serial
Err:   serial
Net:   Net Initialization Skipped
No ethernet found.
Normal Boot
Hit any key to stop autoboot:  0
=> sf probe
SF: Detected MX25R6435F with page size 256 Bytes, erase size 64 KiB, total 8 MiB
=> sf erase 0x0 0x20000
SF: 131072 bytes @ 0x0 Erased: OK
=> fatload mmc 0:1 0x62000000 sdk20-app.img
reading sdk20-app.img
65536 bytes read in 91 ms (703.1 KiB/s)
=> sf write 0x62000000 0x0 0x20000
device 0 offset 0x0, size 0x20000
SF: 131072 bytes @ 0x0 Written: OK
=>
```

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.



```
Hello World!
```

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

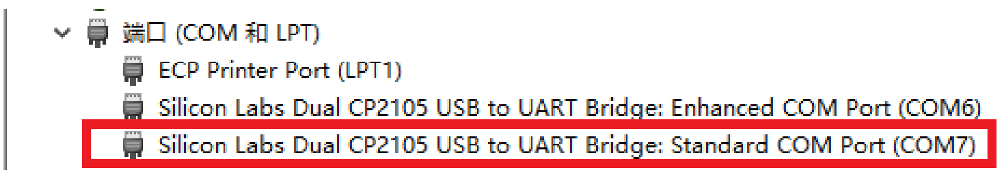
5 How to determine COM port

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 the **Start** menu and type **Device Manager** in the search bar.
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



端口 (COM 和 LPT)
ECP Printer Port (LPT1)
Silicon Labs Dual CP2105 USB to UART Bridge: Enhanced COM Port (COM6)
Silicon Labs Dual CP2105 USB to UART Bridge: Standard COM Port (COM7)

Figure 16. USB-UART interface

6 How to define IRQ handler in CPP files

With MCUXpresso SDK, users could define their own IRQ handler in application level to override the default IRQ handler. For example, to override the default `PIT_IRQHandler` define in `startup_DEVICE.s`, application code like app.c can be implement like:

```
c
void PIT_IRQHandler(void)
{
    // Your code
}
```

When application file is CPP file, like app.cpp, then `extern "C"` should be used to ensure the function prototype alignment.

```
cpp
extern "C" {
    void PIT_IRQHandler(void);
}
void PIT_IRQHandler(void)
{
    // Your code
}
```

7 Revision history

This table summarizes revisions to this document.

Table 1. Revision history

Revision number	Date	Substantive changes
2.13.0	22 December 2022	Updated for MCUXpresso SDK v2.13.0
2.14.0	27 July 2023	Updated for MCUXpresso SDK v2.14.0

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 document 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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell 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

Date of release: 27 July 2023