AN14092

Setting Up VS Code for i.MX 8M Linux User Space Cortex-A Development

Rev. 1 — 3 October 2023 Application note

Document Information

Information	Content
Keywords	Visual Studio Code, debugging, Real-Time-Edge, i.MX 8M Cortex-A User-Space.
Abstract	This document describes how to set up Visual Studio Code for developing and debugging User-Space Cortex-A applications on i.MX boards running Real-Time Edge software.

1 Introduction

This document describes how to set up Visual Studio Code for developing and debugging User-Space Cortex-A applications on i.MX 8MP running Real-Time Edge software from a Windows host, using WSL (Windows Subsystem for Linux). While this document uses i.MX 8MP, the same process with minor differences works for the entire i.MX 8M family.

1.1 Steps overview

It is assumed that the host computer has the latest version of Windows 10/11 installed. The main steps to be followed are:

- Setting up WSL1 on the Windows host.
- Installing the development toolchain.
- Installing and configuring VS Code.
- Configuring the board and test application debugging on the i.MX 8MP.

1.2 Software environment

- Download the Real Time Edge Software 2.6.0 for the i.MX 8MP board archive and extract it.
- Extract the `Real-time_Edge_v2.6_IMX8MP-LPDDR4-EVK/real-time-edge/nxp-image-real-time-edge-imx8mp-lpddr4-evk.wic.zst` file to obtain the flashable image.
- Use SD card flashing software such as Balena Etcher to write the `nxp-image-real-time-edge-imx8mp-lpddr4-evk.wic` image to the SD card.

1.3 Hardware setup and equipment

- Development kit: NXP i.MX 8MP EVK LPDDR4
- Micro SD card: SanDisk Ultra 32-GB Micro SDHC I Class 10 was used for the current experiment
- USB-C cable for the debug port
- Ethernet cable

2 Prerequisites

Connect the i.MX 8MP platform to the host Windows PC via USB cable between the DEBUG USB-UART connector and the PC USB connector. Windows finds the serial devices automatically.

Find the COM device with the name COM* to determine your debug port. On the i.MX 8MP, four ports appear. Of the last two, one port is for the debug messages from the Cortex-A53, and the other is for the Cortex-M7. The port number is allocated randomly, so opening both is beneficial for development.

The device manager showing the 4 COM ports exposed by i.MX 8MP is presented on Figure 1. The last two ports highlighted in red are used in this context.

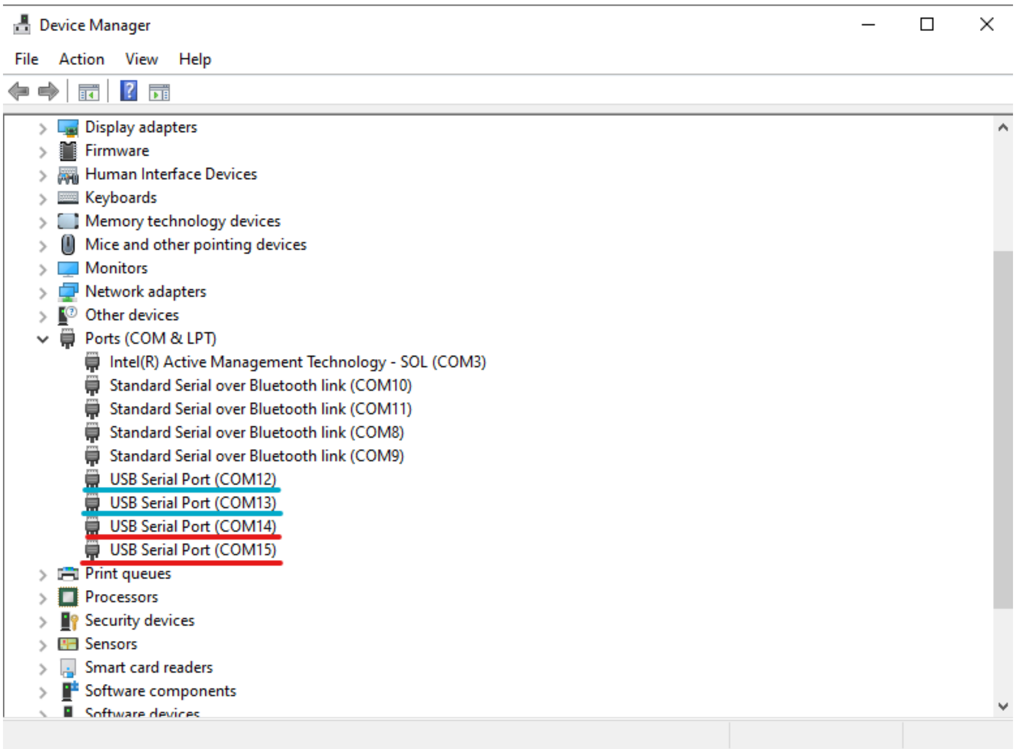

Figure 1. Device manager

Open in your preferred serial terminal emulator (for example, PuTTY) the serial device, set the speed to 115,200 bit/s, 8 data bits, 1 stop bit (115200, 8N1), no parity.

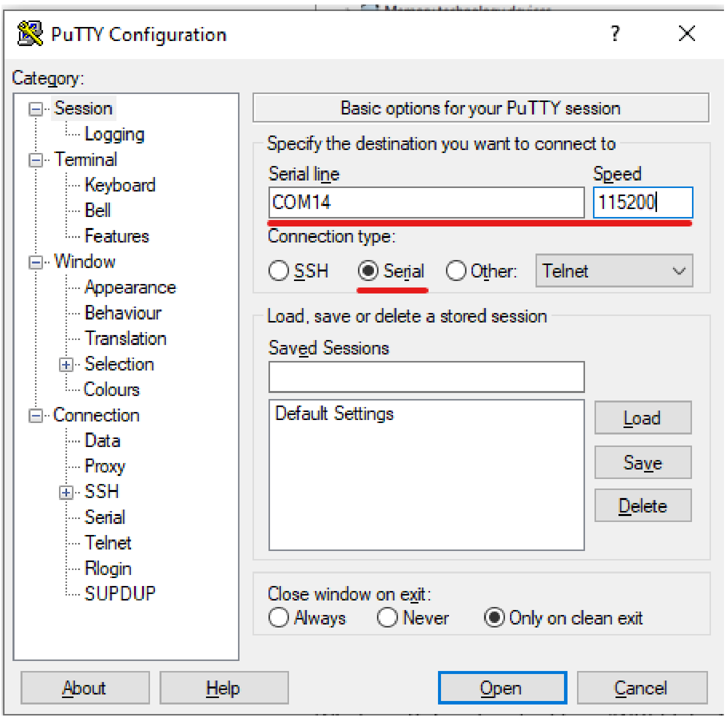

Figure 2. Putty configuration for opening the COM14 port on the board

3 Setting up WSL

Windows Subsystem for Linux (WSL) lets developers install and run a Linux distribution under Windows.

3.1 Install WSL

1. On the Windows host, open Windows Features and turn on the Windows Subsystem for Linux feature.

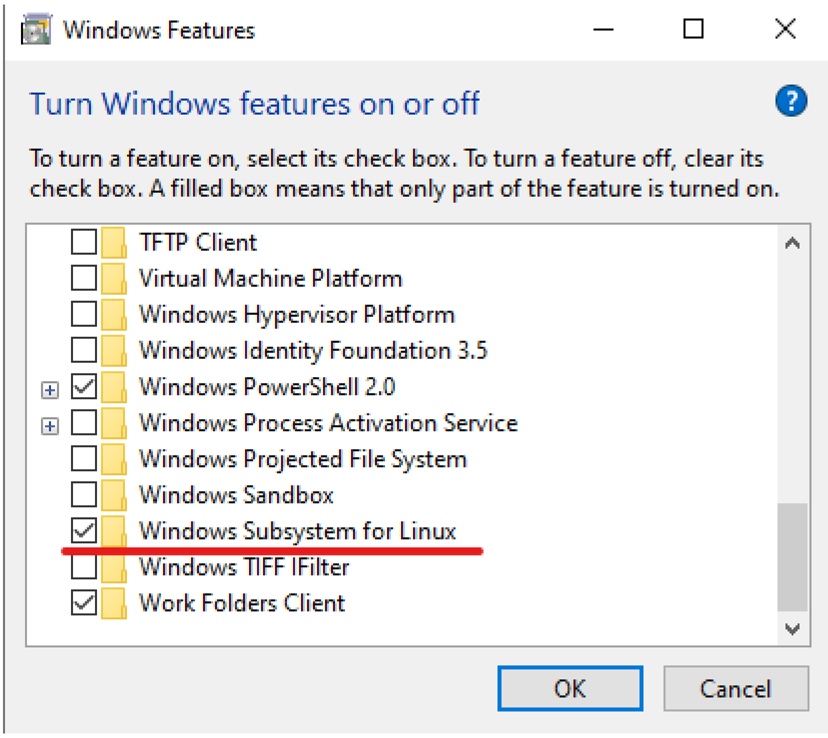

Figure 3. Windows features with WSL enabled

2. Open a CMD window and set the default WSL version to 1:

```
> wsl --set-default-version 1
```

3. From the Microsoft Store, search and download the Ubuntu distribution.
4. After it is installed, open it. A terminal window requiring you to set a user name and password opens. After completing, a bash shell starts.

3.2 Install dependencies

From the opened shell, run the following commands to install the dependencies:

```
$ sudo apt-get -y update
$ sudo apt-get -y install build-essential gdb gdb-multiarch git
```

3.3 Installing development toolchain

To cross-compile applications and have access to the board/software specific libraries, a development toolchain is required. It is obtained from the Real-Time-Edge Yocto project. The project can be set up either on WSL or on any other Ubuntu host.

- Install the Real Time Edge Software 2.6.0 environment.

- Set up the Real-time Edge Yocto environment for the i.MX 8MP. For more details on how to do that, see Section 3 (for dependencies) and Section 5.5. (build guide) from *Real-Time Edge Yocto Project User's Guide* (document REALTIMEEDGEUG), without running the final bitbake command. Instead, run the following one to build the SDK, not the flashable image:

```
$ bitbake nxp-image-real-time-edge -c populate_sdk
```

  The task produces a shell script that can be used to install the SDK. It is found in `<yocto_build_directory>/tmp/deploy/sdk/`.
- Run the script produced above in WSL to install the SDK:

```
$ ./nxp-real-time-edge-glibc-x86_64-nxp-image-real-time-edge
-armv8a-imx8mp-lpddr4-evk-toolchain-2.5.sh
```

Note: *The script asks you if you want to change the default install location in `/opt/nxp-real-time-edge/2.5/`. For this guide, the default location is assumed.*

4 Setting up VS Code

This section describes the details of setting up VS Code.

4.1 Install VS Code

On Windows, install VS Code from the official website.

4.2 Install required extensions

Open VS Code, go to the extensions tab on the left sidebar and install the following extensions:

- C/C++ is the official C/C++ developing extension.
- WSL is used for connecting VS Code to WSL.
- Serial Monitor is used for connecting to the serial port on the board directly from VS Code.
  Note: *This step is optional*

4.3 Create/Open a project

To create or open a project in VS Code connected to WSL, from a WSL shell navigate to the project directory and enter the following command:

```
$ code .
```

5 Configure a project for developing and debugging

For this example, a new project for a hello-world problem is used. Then the required configurations for developing and debugging on the board are made. The basic configuration here is easily transferable to other projects.

This method requires that the board and the host are connected via a network, since `ssh` and `gdbserver` are used.

5.1 Create a project

Run the following commands to create and open a project:

```
$ mkdir demo-proj
$ cd demo-proj
$ code .
```

5.2 Create sources

- In the project, create a `hello-world.c` file with a simple program:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Hello, World!\n");

    return 0;
}
```

- Create a simple Makefile with the following contents:

```
CC ?= gcc

CFLAGS ?= -Wall -Wextra

DEBUGFLAGS = -Og -g

TARGET = hello.bin
SOURCE = hello-world

OBJECTS = $(TARGET).o

.PHONY: all clean

all: $(TARGET)

$(TARGET): $(OBJECTS)
 $(CC) $(CFLAGS) $(DEBUGFLAGS) -o $@ $^

$(OBJECTS): $(SOURCE).c
 $(CC) $(CFLAGS) $(DEBUGFLAGS) -c -o $@ $^
```

The Makefile builds the `hello-world.c` executable into a binary called `hello.bin`, using the environmentally defined compiler (CC) and CFLAGS, falling back on defaults if not set.

5.3 VS Code project configuration

The `.vscode` folder in the `demo-proj` project folder contains the project configuration files. If the folder was not created automatically, do it manually:

```
$ mkdir .vscode
```

Then, create/edit the following files:

- `settings.json`, this file contains values for variables that are used in configuring the workspace in the other files.
  Contents:

```
{
 /* Target Device Settings */
 "TARGET_IP":"169.254.158.177",

 /* Project Settings */
 "PROGRAM":"hello.bin",

 /* SDK Configuration */
 "ARCH":"aarch64-poky-linux",
 "OECORE_NATIVE_SYSROOT":"/opt/nxp-real-time-edge/2.5/sysroots/x86_64-pokysdk-linux",
 "SDKTARGETSYSROOT": "/opt/nxp-real-time-edge/2.5/sysroots/armv8a-poky-linux",

 /* SDK Constants */
 "CC_PREFIX": "${config:OECORE_NATIVE_SYSROOT}/usr/bin/${config:ARCH}/${config:ARCH}-",
 "CXX": "${config:CC_PREFIX}g++ -march=armv8-a+crc+crypto -fstack-protector-strong  -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-security --sysroot=${config:SDKTARGETSYSROOT}",
 "CC": "${config:CC_PREFIX}gcc -march=armv8-a+crc+crypto -fstack-protector-strong  -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-security --sysroot=${config:SDKTARGETSYSROOT}",
 "CPP": "${config:CC_PREFIX}gcc -E -march=armv8-a+crc+crypto -fstack-protector-strong  -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-security --sysroot=${config:SDKTARGETSYSROOT}",
}
```

  Where:
  - `TARGET_IP:` is the IP of the i.MX board that we want to debug.
  - `PROGRAM:` is the compiled executable name.
  - `ARCH:` is the architecture that we want to compile for.
  - `OECORE_NATIVE_SYSROOT:` is the location of the native sysroot.
  - `SDKTARGETSYSROOT:` is the location of the target SDK sysroot.
  - `CC_PREFIX:` is the path prefix of the cross-compiler binaries.
  - `CXX/CC/CPP:` is the full path of a cross-compiler binary complete with the default flags set by the SDK's environment setup script (`/opt/nxp-real-time-edge/2.5/environment-setup-armv8a-poky-linux`).

  They can be modified or removed as needed, except for the sysroot parameter that is required for cross-compiling.
- `c_cpp_properties.json` describes the C/C++ extension configuration. Here is the setting of the IncludePath where IntelliSense looks for header files and the compiler path.

Contents:

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**",
                "${config:SDKTARGETSYSROOT}/usr/include/**"
            ],
            "compilerPath": "${config:CC_PREFIX}gcc",
            "intelliSenseMode": "linux-gcc-arm64",
```

```
            "browse": {
                "path": [
                    "${workspaceFolder}/**",
                    "${config:SDKTARGETSYSROOT}/usr/include/**"
                ],
                "limitSymbolsToIncludedHeaders": true
            }
        }
    ],
    "version": 4
}
```

- `tasks.json` is used to override or add new tasks. It runs the Makefile when the VS Code build command is executed and defines the debug task.
  Contents:

```
{
    "version": "2.0.0",
    /* Configure Yocto SDK Constants from settings.json */
    "options": {
        "env": {
            "CXX": "${config:CXX}",
            "CC": "${config:CC}",
            "CPP": "${config:CPP}"
        }
     },
     /* Configure integrated VS Code Terminal */
     "presentation": {
        "echo": false,
        "reveal": "always",
        "focus": true,
        "panel": "dedicated",
        "showReuseMessage": true,
    },
    "tasks": [
        /* Configure launch.json (debug) preLaunchTask Task */
        {
            "label": "imx-deploy-gdb",
            "isBackground": true,
            "problemMatcher":{
                "base": "$gcc",
                "background": {
                    "activeOnStart": true,
                    "beginsPattern":  "Deploying to target",
                    "endsPattern":  "Starting GDB Server on Target"
                }
            },
            "type": "shell",
            "command": "sh",
            "args": [
                "imx-deploy-gdb.sh",
                "${config:TARGET_IP}",
                "${config:PROGRAM}"
            ],
            "dependsOn": ["build"],
        },
        /* Configure Build Task */
        {
            "label": "build",
            "type": "shell",
```



```
                "command": "make clean; make -j$(nproc)",
                "problemMatcher": ["$gcc"]
            },
        ]
    }
```

Where:

– `options.env:` sets the environment variables for compilation to the ones that we set in `settings.json.`
– `tasks[0]:` defines the debug task. Runs the `build` task first, then the `imx-deploy-gdb.sh` script described in Section 5.4 to connect to the board and launch the debugging session.
– `tasks[1]:` defines the build task. This example uses a simple make command but can be edited as needed to suit other projects.

- `launch.json` is a VS Code file to configure debug settings. It runs the `imx-deploy-gdb` task above.

```
{
    "version": "0.2.0",
    "configurations": [{
        "name": "GDB debug",
        "type": "cppdbg",
        "request": "launch",
        "program": "${config:PROGRAM}",
        "args": [],
        "stopAtEntry": true,
        "cwd": "${workspaceFolder}",
        "environment": [],
        "MIMode": "gdb",
        "targetArchitecture": "arm64",
        "preLaunchTask": "imx-deploy-gdb",
        "setupCommands": [{
            "description": "Enable pretty-printing for gdb",
            "text": "-enable-pretty-printing",
            "ignoreFailures": true
        }],
        "miDebuggerPath": "/usr/bin/gdb-multiarch",
        "miDebuggerServerAddress": "${config:TARGET_IP}:3000",
    }]
}
```

Where:

– `configurations.program:` is the final executable name.
– `configurations.args:` is arguments to pass to the program on execution.
– `configurations.preLaunchTask:` is the imx-deploy-gdb task from `tasks.json`.
– `configurations.miDebuggerPath:` is the path to the gdb binary that supports the target architecture.
– `configurations.miDebuggerServerAddress:` is the address and port of the gdb-server (launched by the deploy script) on the board.

5.4 Debugger deploy script

Create the `imx-deploy-gdb.sh` script in the root of the project that has the following contents:

```
#!/bin/bash
readonly TARGET_IP="$1"
readonly PROGRAM="$2"
readonly TARGET_DIR="/home/root"

# Must match startsPattern in tasks.json
```

```
echo "Deploying to target"

# kill gdbserver on target and delete old binary
ssh root@${TARGET_IP} "sh -c '/usr/bin/killall -q gdbserver; rm -rf
 ${TARGET_DIR}/${PROGRAM}  exit 0'"

# send the program to the target
scp ${PROGRAM} root@${TARGET_IP}:${TARGET_DIR}

# Must match endsPattern in tasks.json
echo "Starting GDB Server on Target"

# start gdbserver on target
ssh -t root@${TARGET_IP} "sh -c 'cd ${TARGET_DIR}; gdbserver localhost:3000
 ${PROGRAM}'"
```

The final file structure of the project is shown on Figure 4.

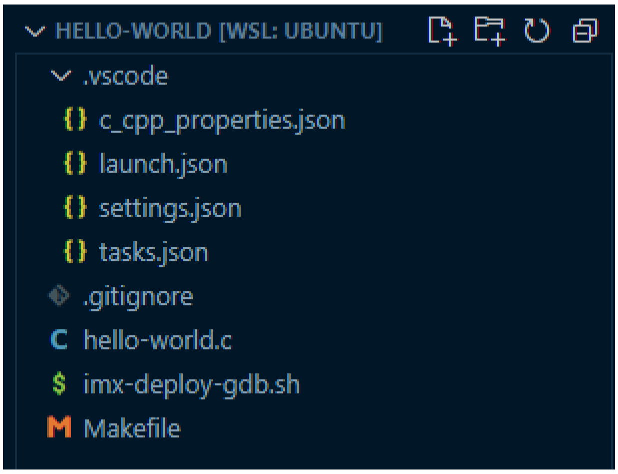

Figure 4. Final demo project structure

5.5 Setup and target configuration

- Connect the board and the PC via Ethernet, connect the serial port to the PC.
- After a few moments, the i.MX board must have an IP address on the connected interface. You can check it by running the following command via the serial terminal and checking the Ethernet address:

```
[I.MX Board Serial]
$ ip a s
```

- The IP address of the board must match the one we have configured in `settings.json`. For this example, manually set the board IP to the one in `settings.json`, but any other method should work. To set the IP of the board, run the following command:

```
[I.MX Board Serial]
$ ifconfig eth0 169.254.158.177
```

- Test the connection between the PC and the board by using ping and then confirm that `ssh` is working. Run these commands on WSL:

```
[WSL]
$ ping 169.254.158.177
$ ssh root@169.254.158.177
```


5.6 Debugging your program

- Set a breakpoint in the main function of the `hello-world.c` file.
- From the top bar, press Run -> Start Debugging.
- VS Code compiles the executable, sends it to the board and launches a debugging session that stops at the set breakpoint, as shown on Figure 5:

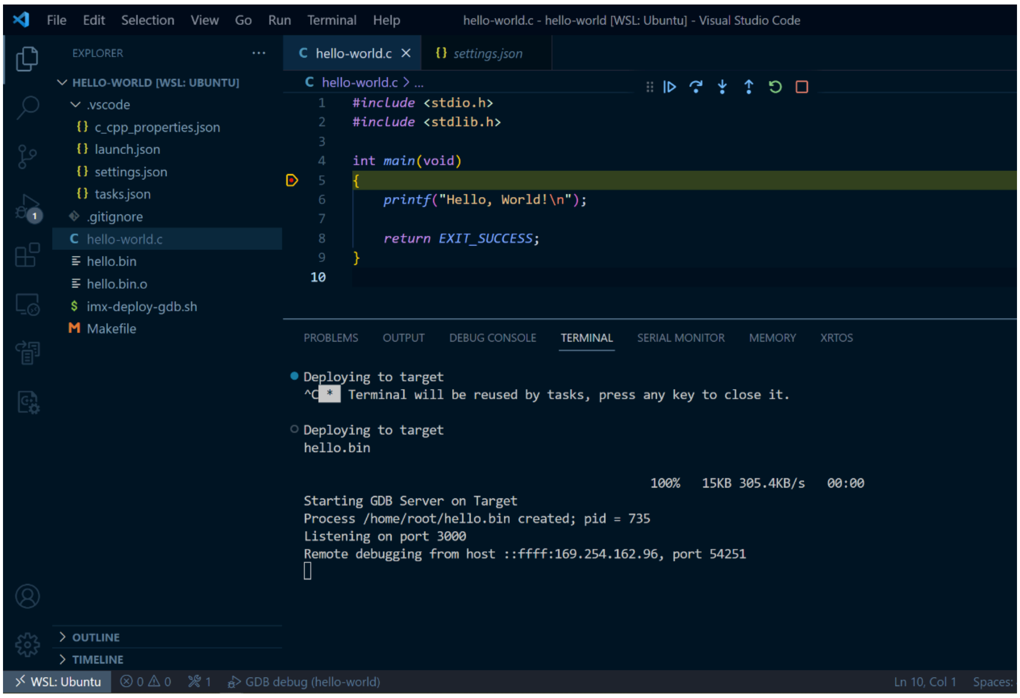

Figure 5. Demo application remote debugging session

6 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN

ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7 Revision history

Table 1 summarizes the revisions to this document.

Table 1. Revision history

Revision number	Release date	Description
1	03 October 2023	Initial public 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.

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.

i.MX — is a trademark of NXP B.V.

Contents

Date of release: 3 October 2023
Document identifier: AN14092