AN14120

Debugging Cortex-M with VS Code on i.MX 8M, i.MX 8ULP, and i.MX 9

Rev. 3.0 — 5 May 2025 Application note

Document information

Information	Content
Keywords	AN14120, i.MX 8M, i.MX 93, i.MX 95, Cortex-M, i.MX 8MN, i.MX 8MP, i.MX 8MM, VS Code, MCUXSDK, MCUXpresso SDK, J-Link, SEGGER, Cortex-M debug
Abstract	This document describes cross-compiling, deploying, and debugging an application for the i.MX 8M Family, i.MX 8ULP, and i.MX 9 Cortex-M processor using Microsoft Visual Studio Code.

1 Introduction

This document describes cross-compiling, deploying, and debugging an application for the i.MX 8M Family, i.MX 8ULP, and i.MX 9 family Cortex-M processor using Microsoft Visual Studio Code.

1.1 Software environment

The solution could be implemented both on the Linux and Windows host. For this application note, a Windows PC is assumed, but not mandatory.

- Software versions used:
  - J-Link SEGGER V8.24
  - MCUXpresso for VS Code version 25.3.72
  - MCUX SDK 25.03.00
  - Linux BSP release 6.12.3_1.0.0 with the following images:
    - **i.MX 8M Mini**: `core-image-minimal-imx8mmevk.wic`
    - **i.MX 8M Nano**: `core-image-minimal-imx8mnevk.wic`
    - **i.MX 8M Plus**: `core-image-minimal-imx8mpevk.wic`
    - **i.MX 8ULP**: `core-image-minimal-imx8ulpevk.wic`
    - **i.MX 93**: `core-image-minimal-imx93evk.wic`
    - **i.MX 95**: `core-image-minimal-imx95-19x19-lpddr5-evk.wic`

For detailed steps on how to build these images, refer to *i.MX Linux User Guide* (document IMXLUG) and *i.MX Yocto Project User Guide* (document IMXLXYOCTOUG).

If a Windows PC is used, write the prebuild image on the SD card using Win32 Disk Imager (https://win32diskimager.org/) or Balena Etcher (https://etcher.balena.io/).

If an Ubuntu PC is used, write the prebuild image on the SD card using the below command:

```
$  sudo dd if=<image_name>.wic of=/dev/sd<x> bs=1M status=progress conv=fsync
```

Note: *Check your card reader partition and replace `sd<x>` with your corresponding partition.*

1.2 Hardware setup and equipment

- Development kit:
  - NXP i.MX 8MM EVK LPDDR4
  - NXP i.MX 8MN EVK LPDDR4
  - NXP i.MX 8MP EVK LPDDR4
  - NXP i.MX 93 EVK for 11x11 mm LPDDR4
  - NXP i.MX 8ULP EVK LPDDR4
  - NXP i.MX 95 EVK for 19x19 mm LPDDR5
- Micro SD card: SanDisk Ultra 32-GB Micro-SDHC; Class 10 is used for the current experiment.
- Micro-USB (i.MX 8M) or Type-C (i.MX 9) cable for debug port.
- SEGGER J-Link debug probe.
- MCU-link debug probe – This is mandatory for the i.MX 95 debug using the serial download method.

2 Prerequisites

Before starting to debug, several prerequisites must be met to have a properly configured debug environment.

2.1 PC Host – i.MX board debug connection

To establish the hardware debug connection, perform the following steps:

1. Connect the i.MX board to the host PC via the DEBUG USB-UART and PC USB connector using a USB cable. The Windows OS finds the serial devices automatically.
2. In *Device Manager*, under *Ports (COM & LPT)*, find two or four connected USB serial ports (COM <port_number>). One of the ports is used for the debug messages generated by the Cortex-A core, and the other is for the Cortex-M core.
   Before determining the right port needed, remember:
   - **[i.MX 95]**: There are four ports available in the *Device Manager*. The first port is for Cortex-M7 debug. The third port is for Cortex-A debug. The fourth port is for Cortex-M33 debug. Counting the debug ports in ascending order.
   - **[i.MX 8MP, i.MX 8ULP, i.MX 93]**: There are four ports available in the *Device Manger*. The last port is for Cortex-M debug and the second to last port is for Cortex-A debug, counting debug ports in ascending order.
   - **[i.MX 8MM, i.MX 8MN]**: There are two ports available in *Device Manager*. The first port is for Cortex-M debug and the second port is for Cortex-A debug, counting debug ports in ascending order.
3. Open the right debug port using your preferred serial terminal emulator (for example PuTTY) by setting the following parameters:
   - Speed to 115200 bps
   - 8 data bits
   - 1 stop bit (115200, 8N1)
   - No parity
4. Connect the SEGGER/MCU-link debug probe between the JTAG connector of the EVK and the PC.

If the i.MX board JTAG interface has no guided connector, the orientation is determined by aligning the red wire to the pin 1, as in Figure 1.

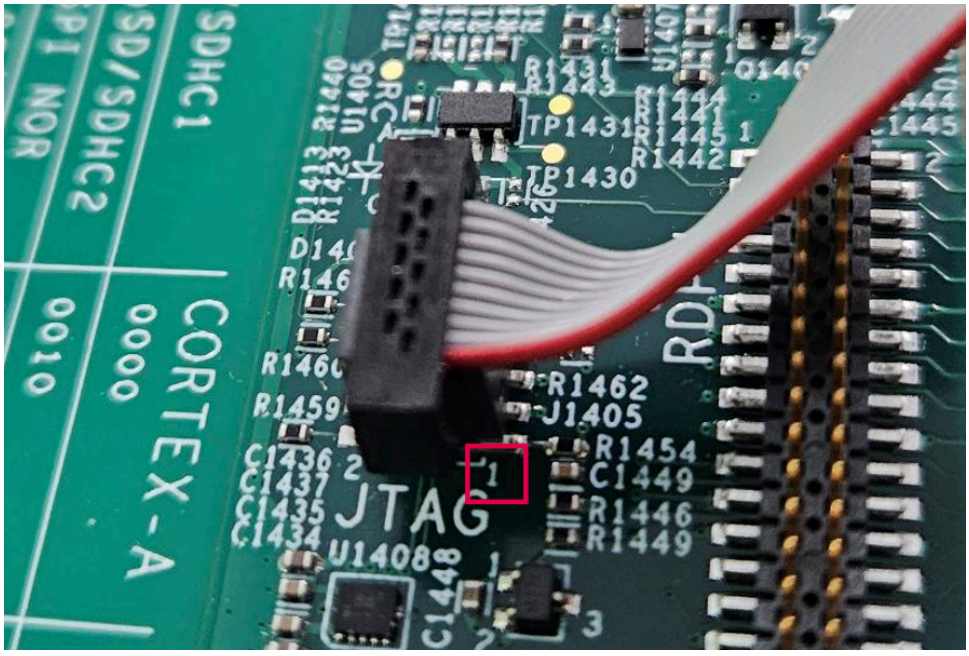

Figure 1. JTAG connection

2.2 VS Code configuration

To download and configure the VS Code, perform the following steps:

1. Download and install the latest version of Microsoft Visual Studio Code from the official website. If using Windows as the host OS, choose the "Download for Windows" button from the Visual Studio Code main page.

Figure 2. Download Microsoft Visual Studio Code

2. After installing Visual Studio Code, open it and choose the "Extensions" tab or press the Ctrl + Shift + X combination.

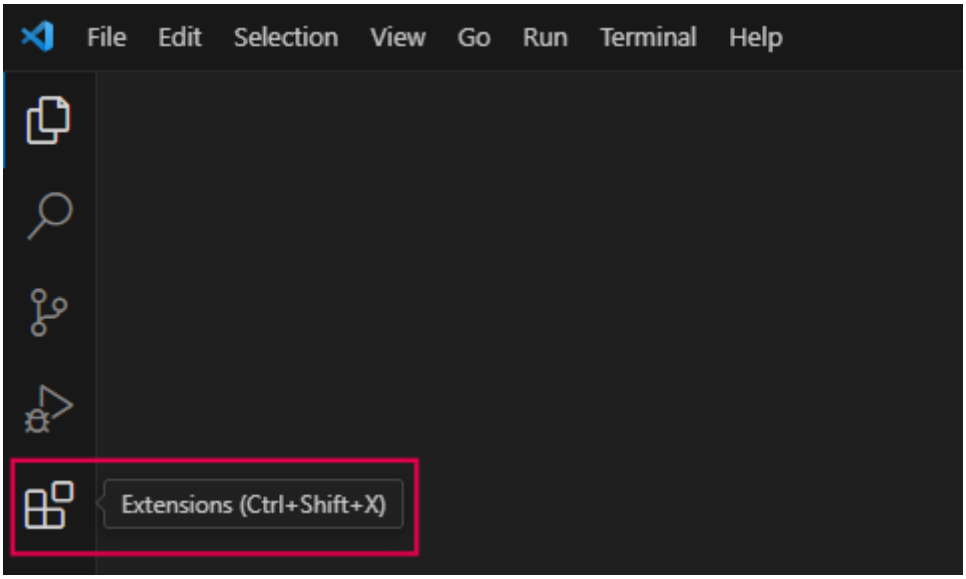

Figure 3. Microsoft Visual Studio Code Extensions tab

3. In the dedicated Search bar, type *MCUXpresso for VS Code* and install the extension. A new tab appears in the left side of the VS Code window.

Figure 4. Install MCUXpresso for VS Code extension

2.3 MCUXpresso extension configuration

To configure the MCUXpresso extension, perform the following steps:

1. Click the MCUXpresso extension dedicated tab from the left side bar. From the *QUICKSTART PANEL*, click *Open MCUXpresso Installer* and give permission for downloading the installer.
2. The installer window appears in a short time. Click *MCUXpresso SDK Developer* and on *SEGGER J-Link* then click the *Install* button. The installer installs the needed software for archives, toolchain, Python support, Git, and debug probe.
   Note: *For i.MX 95, to use the serial download method described in Section 4 "Prepare the board for the debugger", install LinkServer and MCUXpresso Secure Provisioning Tool as well.*

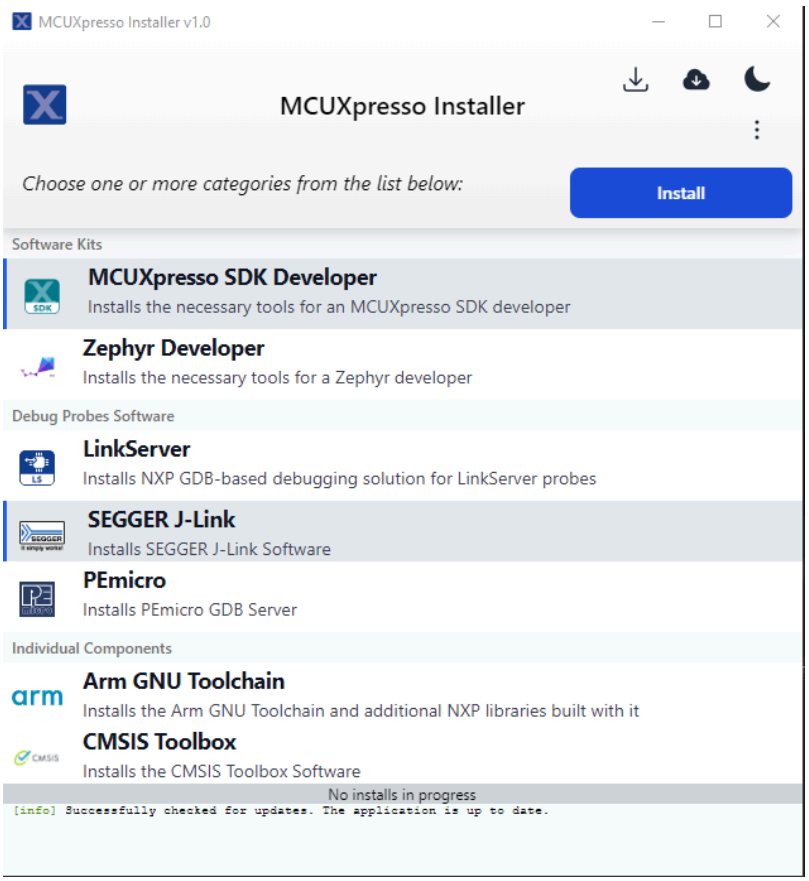

Figure 5. Install SEGGER J-Link software

After all packages are installed, be sure that the J-Link probe is connected to the host PC. Then, check if the probe is also available in the MCUXpresso extension under the *DEBUG PROBES* view, as shown in Figure 6.

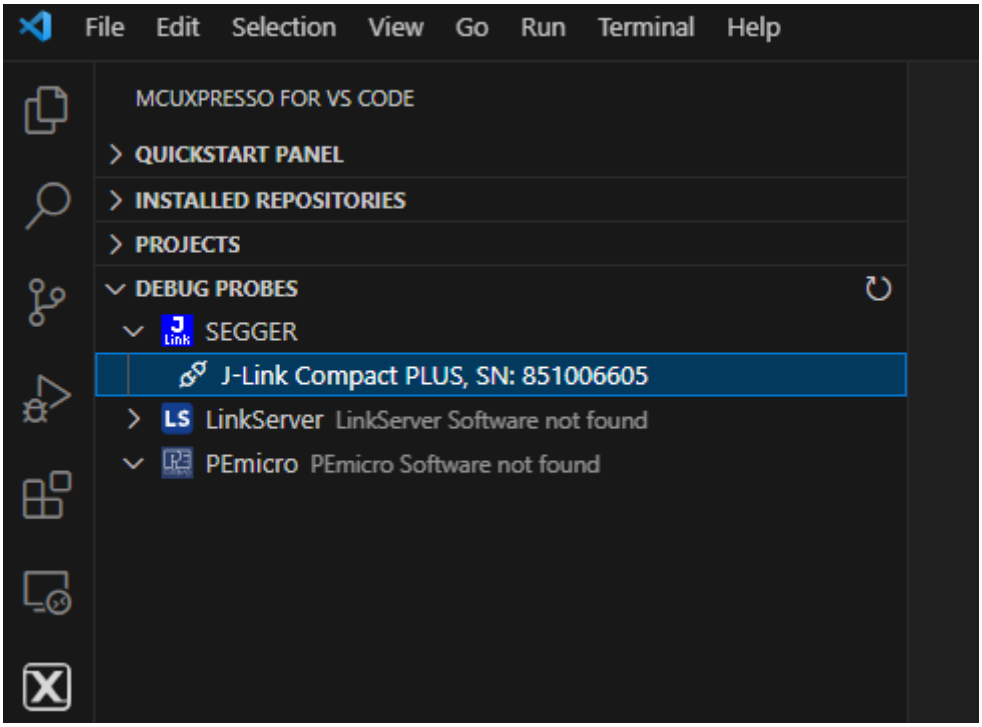

Figure 6. J-Link probe available under DEBUG PROBES view

2.4 Import MCUXpresso SDK

Depending on what board you are running, build and download the specific SDK from NXP official website.

To build an example for i.MX 93 EVK, see Figure 7:

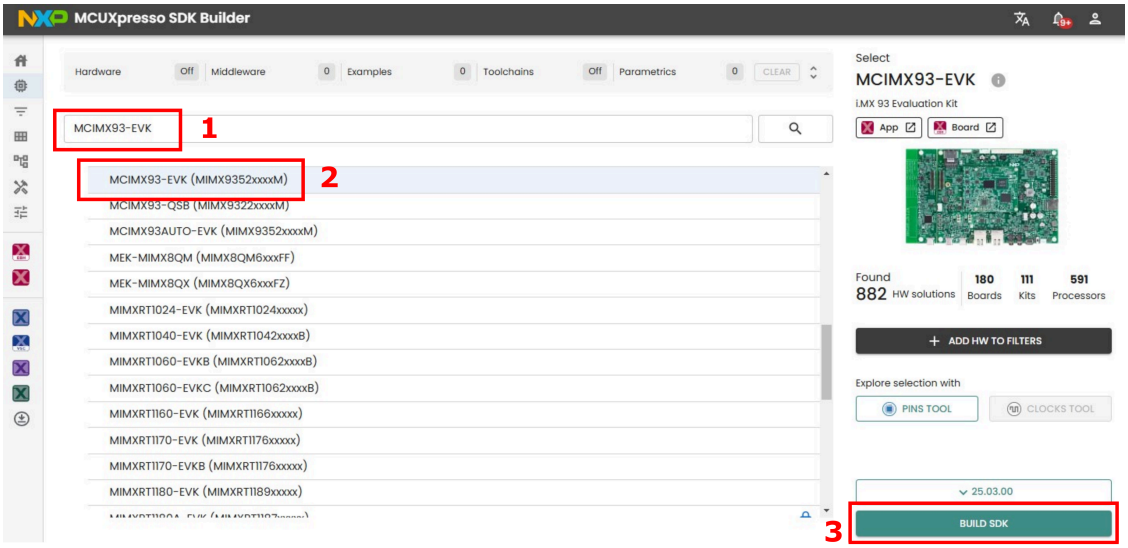

Figure 7. Download the MCUXpresso SDK

To import the MCUXpresso SDK repository in VS Code, perform the following steps:

1. After downloading the SDK, open Visual Studio Code. Click the MCUXpresso tab from the left side, and expand the *INSTALLED REPOSITORIES* and *PROJECTS* views.
2. Click the *Import Repository* and select *LOCAL ARCHIVE*. Click the *Browse…* corresponding to the *Archive* field and select the recently downloaded SDK archive.
3. Select the path where the archive is unzipped and fill in the *Location* field.
4. The *Name* field can be left by default, or you can choose a custom name.
5. Check or uncheck *Create Git repository* based on your needs and then click *Import*.

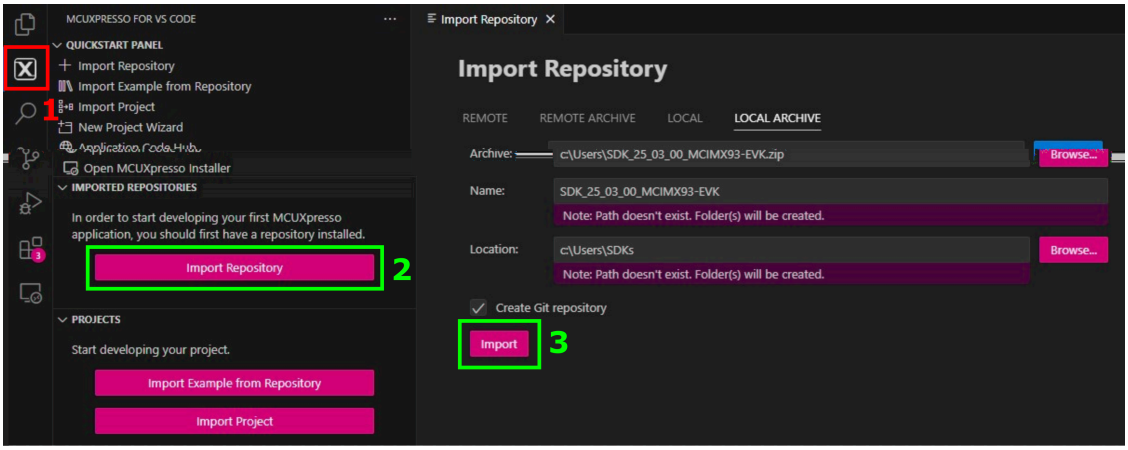

Figure 8. Import the MCUXpresso SDK repository

2.5 Import an example application

When the SDK is imported, it appears under the *INSTALLED REPOSITORIES* view.

To import an example application from the SDK repository, perform the following steps:

1. Click the *Import Example from Repository* button from the *PROJECTS* view.
2. Choose a repository from the drop-down list.
3. Choose the toolchain from the drop-down list.
4. Choose the target board.
5. Choose the `demo_apps/hello_world` example from the *Choose a template* list.
6. Choose a name for the project (the default can be used) and set the path to project *Location*.
7. Click *Create*.

After importing the example application successfully, it must be visible under the *PROJECTS* view. Also, the project source files are visible in the *Explorer* (Ctrl + Shift + E) tab.

3 Building the application

To build the application, press the left *Build Selected* icon, as shown in Figure 9.

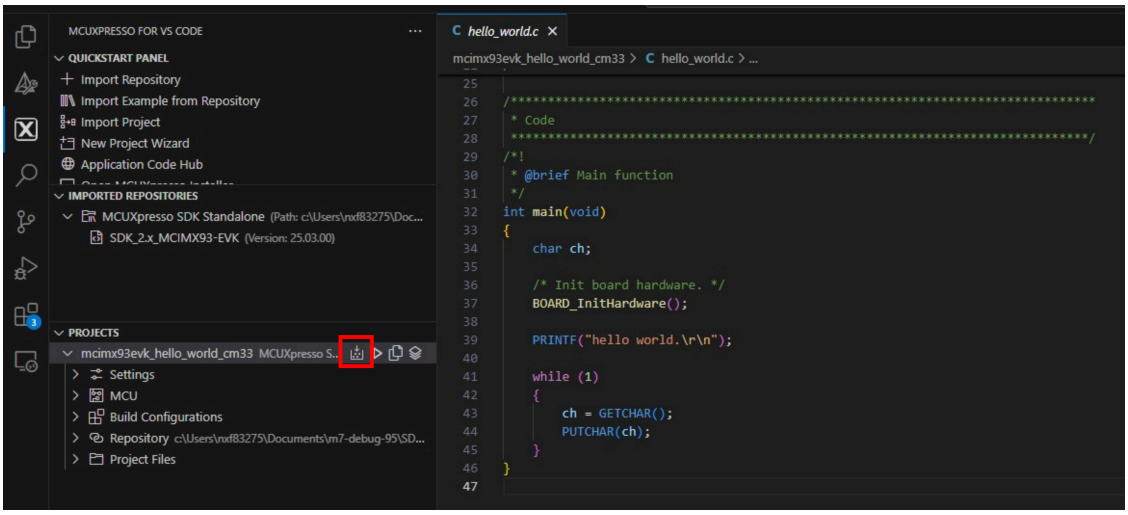

Figure 9. Build application

4 Prepare the board for the debugger

To use the JTAG for debugging Cortex-M applications, there are a few prerequisites depending on the platform:

1. **For i.MX 93**
   - **Debugging Cortex-M33 while only Cortex-M33 is running**
     In this mode, the boot mode switch **SW1301[3:0]** must be set to **[1010]**. Then the M33 image can be directly loaded and debugged using the debug button. For more details, see Section 5 "Running and debugging".

   If Linux running on Cortex-A55 is needed in parallel with Cortex-M33, there are two ways of debugging Cortex-M33:
   - **Debugging Cortex-M33 while Cortex-A55 is in U-Boot**
     First, copy the `sdk20-app.bin` file (located in the `armgcc/debug` directory) generated in Section 3 "Building the application" into the boot partition of the SD card.
     Boot the board and stop it in U-Boot. When the boot switch is configured to boot Cortex-A, the boot sequence does not start the Cortex-M. It has to be kicked off manually using the commands below. If Cortex-M is not started, JLink fails to connect to the core.

```
u-boot=> fatload mmc 1:1 80000000 sdk20-app.bin
u-boot=> cp.b 0x80000000 0x201e0000 0x10000
u-boot=> bootaux 0x1ffe0000 0
```

     Note: *If the system cannot be debugged normally, try to right-click the project in the MCUXpresso for VS Code and choose "Attach to debug the project".*
   - **Debugging Cortex-M33 while Cortex-A55 is in Linux**
     The kernel DTS must be modified to disable the UART5, which uses the same pins as the JTAG interface. If a Windows PC is used, the easiest is to install WSL + Ubuntu 22.04 LTS, and then to cross-compile the DTS.
     After the WSL + Ubuntu 22.04 LTS installation, open the Ubuntu machine running on WSL and install the required packages:

```
$ sudo apt update
$ sudo apt install build-essential flex bison gcc-aarch64-linux-gnu git
```

     Now, the kernel sources can be downloaded:

```
$ git clone https://github.com/nxp-imx/linux-imx
```

```
$ cd linux-imx
$ git checkout lf-6.12.3-1.0.0
```

To disable the UART5 peripheral, search for `lpuart5` node in the `linux-imx/arch/arm64/boot/dts/freescale/imx93-11x11-evk.dts` file and replace the `okay` status with `disabled`:

```
&lpuart5 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uart5>;
        status = "disabled";

        bluetooth {
                compatible = "nxp,88w8987-bt";
        };
};
```

Recompile the DTS:

```
$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make freescale/imx93-11x11-evk.dtb
```

Copy the newly created `linux-imx/arch/arm64/boot/dts/freescale/imx93-11x11-evk.dtb` file on the boot partition of the SD card.
Copy the `hello_world.elf` file (located in the `armgcc/debug` directory) generated in Section 3 "Building the application" into the boot partition of the SD card.
Boot the board in Linux. Since boot ROM does not kick off the Cortex-M when Cortex-A boots, the Cortex-M must be manually started.

```
root@imx93evk:/lib/firmware# cp /run/media/mmcblk2p1/hello_world.elf /lib/firmware
root@imx93evk:~# echo hello_world.elf > /sys/class/remoteproc/remoteproc0/firmware
root@imx93evk:~# echo start > /sys/class/remoteproc/remoteproc0/state
```

Note: *The `hello_world.elf` file must be placed in the `/lib/firmware` directory.*

2. **For i.MX 95**
   - **Debugging Cortex-M7 while only Cortex-M7 is running** (no U-Boot or Linux running)
     In this scenario, build the bootloader using the mkimage tool, with the `flash_m7` target, which does not include the U-Boot binaries. The M7 image inside the bootloader can be any image. It gets overwritten in the memory during debug.

     ```
     $ make SOC=iMX95 OEI=YES flash_m7
     ```

     Write the bootloader on your boot device, then jump to Section 5 "Running and debugging".
   - **Debugging Cortex-M7 while Cortex-A is in U-Boot**
     In this case, nothing special must be done. Boot the board in U-Boot and jump to Section 5 "Running and debugging".
   - **Debugging Cortex-M7 while Cortex-A is in Linux**
     a. Switch the **SW1** to OFF **[0000]** (JTAG side).
     b. To disable the UART5, modify the kernel DTS. The UART5 uses the same pins as the JTAG interface. See the **i.MX 93 Debugging Cortex-M33 while Cortex-A55 is in Linux** section on how to disable the UART5 peripheral. Recompile and copy the resulted DTS on the boot partition of your boot device, then jump to Section 5 "Running and debugging".
   - **Debugging Cortex-M7 while in serial download**
     This method uses the MCUXpresso Secure Provisioning Tool to generate a bootable image containing the OEI image, the M33 image (System Manager), and the M7 image.
     To use this mode, perform the following steps:
     a. Set the boot mode switch **SW7[1:4]** to **[1001]**.

b. Connect the MCU-link between the JTAG connector of the EVK and the PC.
Note: *J-Link is not supported yet.*
c. Connect an additional Type-C cable between the USB1 connector of the EVK and the PC.
d. Specify the boot images. After the first build, a configuration file used by the MCUXpresso Secure Provisioning Tool is generated in `.secureprovisioning/additional_images_mx95_cm7_app_cfg.json`. Edit the file with the correct images. First, get the necessary binaries.
– The DDR firmware (FW):

```
$ wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.27-5af0ceb.bin
$ chmod +x firmware-imx-8.27-5af0ceb.bin
$ ./firmware-imx-8.27-5af0ceb.bin
```

The DDR FW is located in the `firmware-imx-8.27-5af0ceb/firmware/ddr/synopsys/` directory. Copy the `lpddr5_imem_v202311.bin`, `lpddr5_imem_qb_v202311.bin`, `lpddr5_dmem_v202311.bin`, and `lpddr5_dmem_qb_v202311.bin` in the `.secureprovisioning` directory.
– The ELE FW:

```
$ wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-ele-imx-2.0.1-0a66c34.bin
$ chmod+x firmware-ele-imx-2.0.1-0a66c34.bin
$ ./firmware-ele-imx-2.0.1-0a66c34.bin
```

The ELE FW is located in the `firmware-ele-imx-2.0.1-0a66c34` directory. Copy the `mx95a0-ahab-container.img` file in the `.secureprovisioning` directory.
– The OEI FW:
The OEI FW is not provided in binary form. This must be manually compiled.

```
$ git clone https://github.com/nxp-imx/imx-oei.git
$ git checkout lf-6.12.3-1.0.0
$ make board=mx95lp5 oei=ddr DEBUG=1
$ make board=mx95lp5 oei=tcm DEBUG=1
```

Copy the resulted binaries, `build/mx95lp5/tcm/oei-m33-tcm.bin` and `build/mx95lp5/ddr/oei-m33-ddr.bin`, in the `.secureprovisioning` directory.
– The SM FW:
The System Manager is not provided in binary form. This must be manually compiled.

```
$ git clone https://github.com/nxp-imx/imx-sm.git
$ git checkout lf-6.12.3-1.0.0
$ make config=mx95evk cfg
$ make config=mx95evk all
```

Copy the resulted binary `build/mx95evk/m33_image.bin` to the `.secureprovisioning` directory.
Note: *Do not change/remove the `cortex_m7_app and v2x_dummy` entries.*

```
{
  "cli_args": {
    "--additional-images": {
      "images": [
        {
          "entry_type": "oei_ddr",
          "container_set": "#1",
          "extra_settings": {
            "lpddr_imem_path": "lpddr5_imem_v202311.bin",
            "lpddr_imem_qb_path": "lpddr5_imem_qb_v202311.bin",
```

```
            "lpddr_dmem_path": "lpddr5_dmem_v202311.bin",
            "lpddr_dmem_qb_path": "lpddr5_dmem_qb_v202311.bin",
            "oei_ddr_path": "oei-m33-ddr.bin"
          }
        },
        {
          "entry_type": "oei_tcm",
          "container_set": "#1",
          "extra_settings": {
            "oei_tcm_path": "oei-m33-tcm.bin"
          }
        },
        {
          "entry_type": "system_manager",
          "container_set": "#1",
          "extra_settings": {
            "system_manager_path": "m33_image.bin"
          }
        },
        {
          "entry_type": "cortex_m7_app",
          "container_set": "#1",
          "extra_settings": {
            "cortex_m7_app_path": "dummyImage.bin"
          }
        },
        {
          "entry_type": "v2x_dummy",
          "container_set": "#1",
          "extra_settings": {
            "v2x_dummy": "true"
          }
        }
      ]
    },
    "--ele-firmware": "mx95a0-ahab-container.img"
  }
}
```

Build the project again and jump to Section 5 "Running and debugging".

3. **For i.MX 8M**
   - **Debugging Cortex-M while Cortex-A is in U-Boot**
     In this case, nothing special must be done. Boot the board in U-Boot and jump to Section 5 "Running and debugging".
   - **Debugging Cortex-M while Cortex-A is in Linux**
     To run and debug the Cortex-M application in parallel with Linux running on Cortex-A, the specific clock must be assigned and reserved for Cortex-M. It is done from within U-Boot.
     Stop the board in U-Boot and run the below commands:

```
u-boot=> run prepare_mcore
u-boot=> boot
```

4. **For i.MX 8ULP**
   - **Debugging Cortex-M while Cortex-A image needs to be loaded**
     For i.MX 8ULP, under normal single boot mode, it is needed to build the `flash.bin` using `m33_image.bin` in our "VSCode" repo first. The `m33_image.bin` can be found in `{CURRENT REPO}\armgcc\debug\sdk20-app.bin`. Refer to Section 6 from the *Getting Started with*

MCUXpresso SDK for EVK-MIMX8ULP and EVK9-MIMX8ULP in the `SDK_2_xx_x_EVK-MIMX8ULP/docs` on how to build the `flash.bin` image.

Note: *Use the `m33_image.bin` in the active VSCode repo. Otherwise, the program does not attach properly.*

Right-click and choose "Attach".

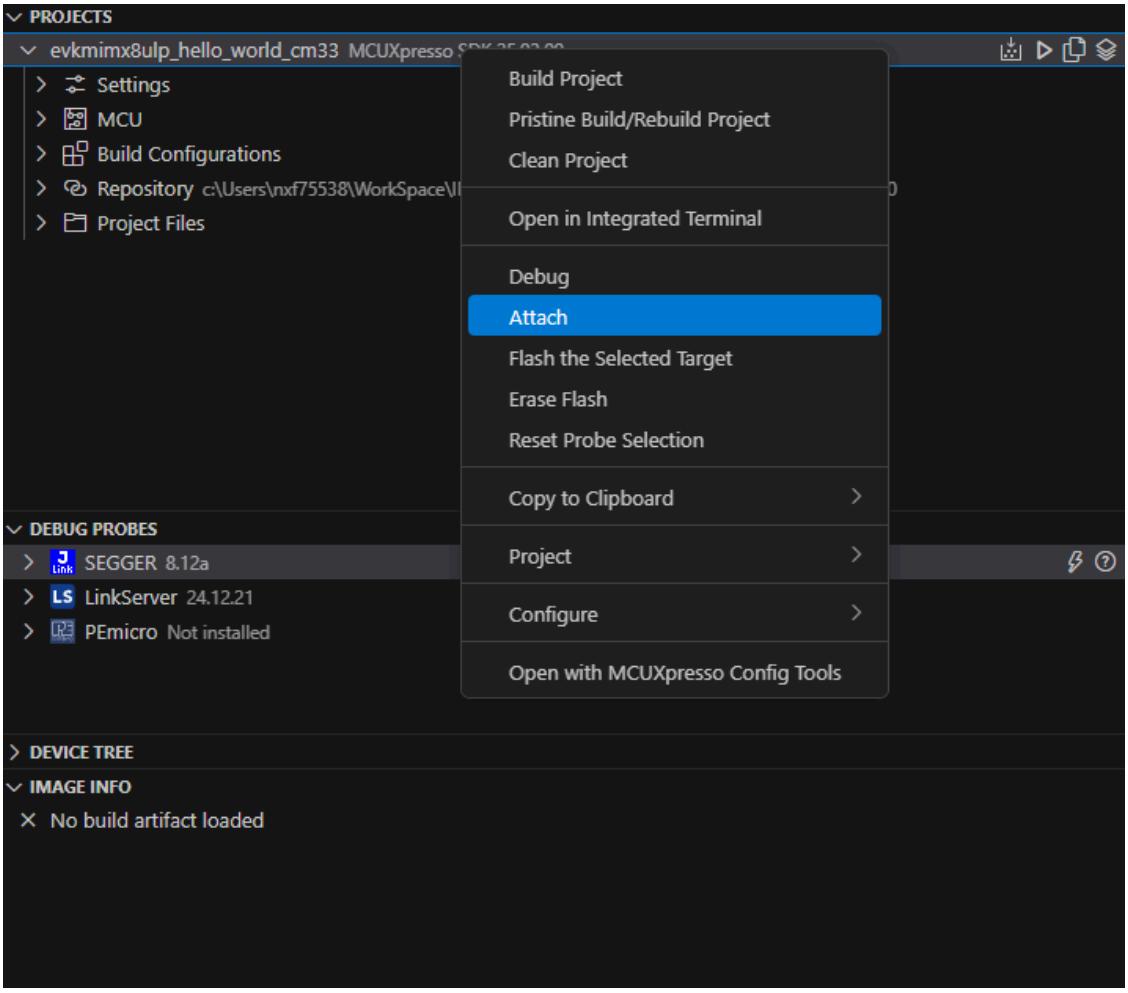

Figure 10. Attach to J-Link debugger for i.MX 8ULP

- **Debugging Cortex-M while Cortex-A image does not need to be loaded**

  This method skips the loading and booting of the Cortex-A image.

  First, build, and write the `flash.bin` without the `m33_image.bin`, using the following command:

  ```
  uuu -b emmc flash_A.bin flash_B.bin
  ```

  Where,

  – `flash_A.bin` is the `flash.bin` that contains the `m33_image.bin`. Build it using the following command:

  ```
  make SOC=iMX8ULP flash_singleboot_m33 REV=A2
  ```

  – `flash_B.bin` is the `flash.bin` that does not contain the `m33_image.bin`. Build it using the following command:

  ```
  make SOC=iMX8ULP flash_singleboot REV=A2
  ```

  Then, jump to Section 5 "Running and debugging".

5 Running and debugging

After pressing the debug button, choose the *Debug project configuration* and the debugging session starts.

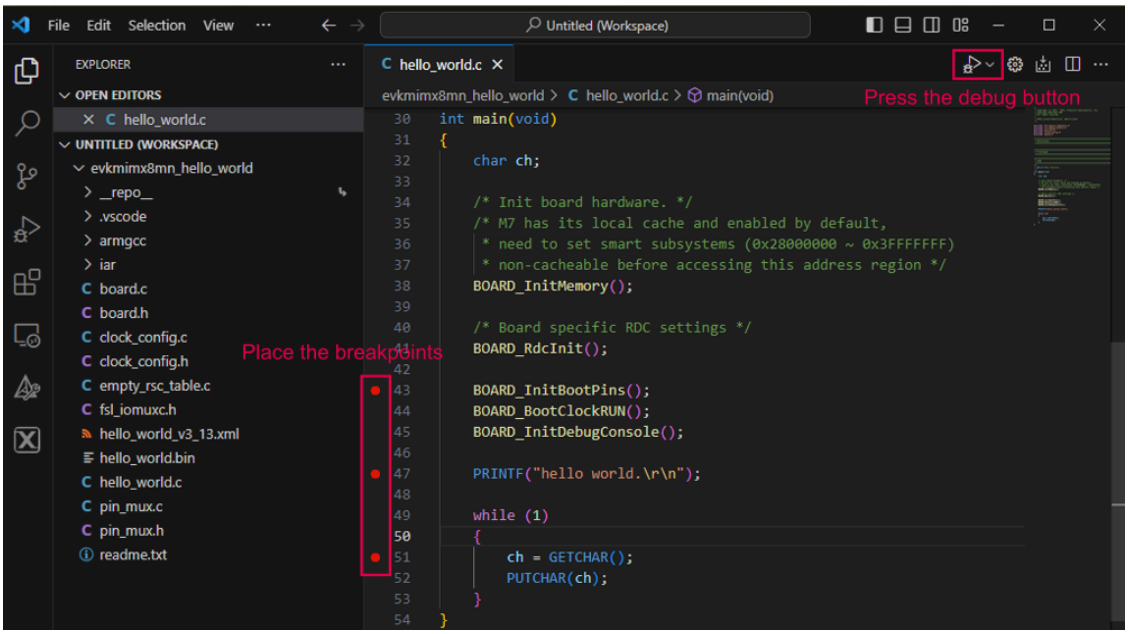

Figure 11. Start the debugging session

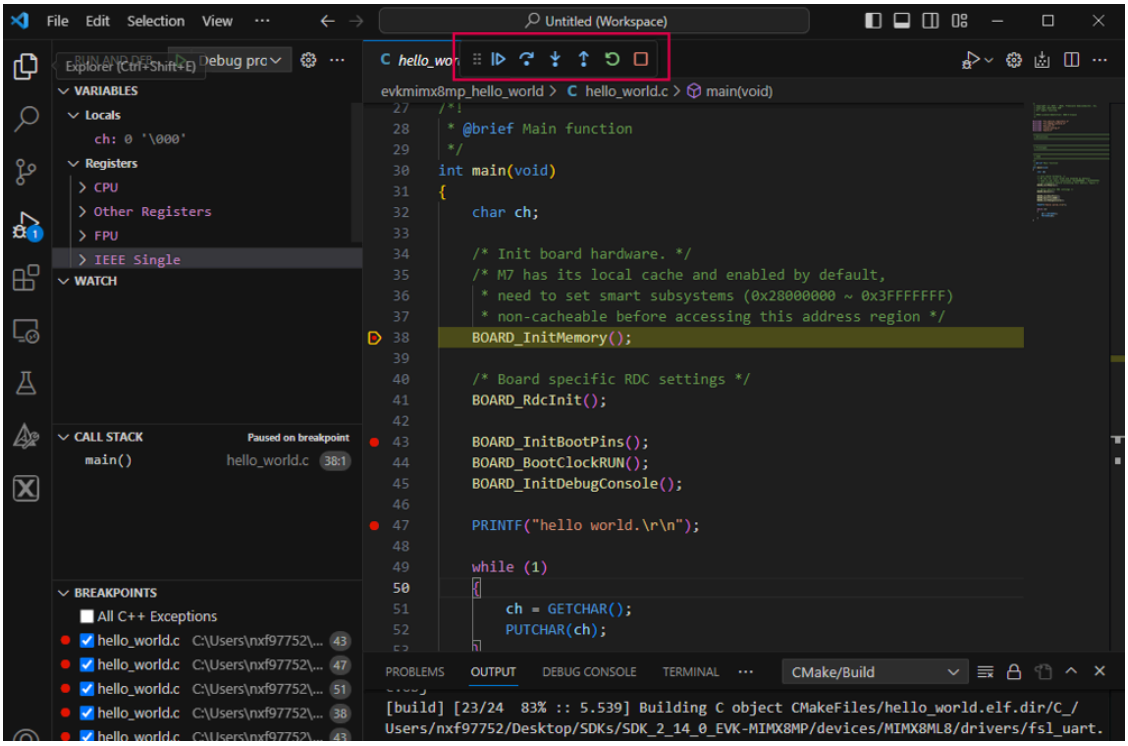

Figure 12. Debugging menu

When a debugging session starts, a dedicated menu is displayed. The debugging menu has buttons for starting the execution until a breakpoint fires up, pause the execution, step over, step into, step out, restart, and stop.

Also, we can see local variables, register values, watch some expression, and check call stack and breakpoints in the left-hand navigator. These function regions are under the "Run and Debug" tab, and not in MCUXpresso for VS Code.

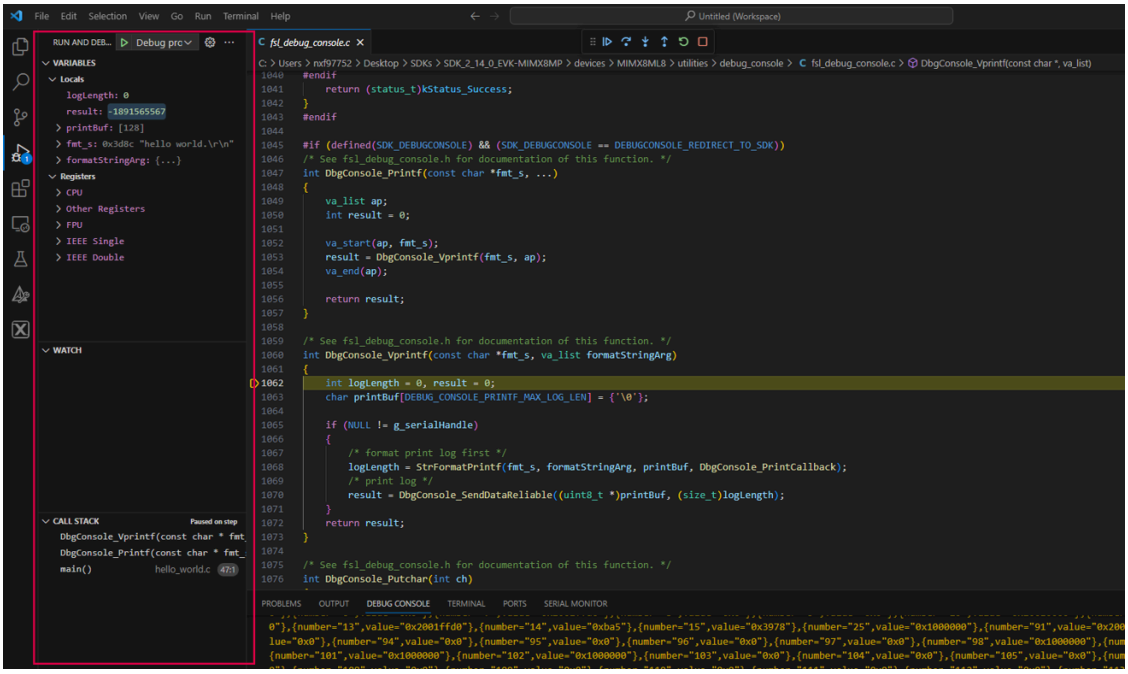

Figure 13. Variables, registers, and call stack

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 2025 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

Document ID	Release date	Description
AN14120 v.3.0	5 May 2025	Updated the following sections for i.MX 95: • Section 1 "Introduction" • Section 1.1 "Software environment" • Section 1.2 "Hardware setup and equipment" • Section 2.4 "Import MCUXpresso SDK" • Section 2.5 "Import an example application" • Section 4 "Prepare the board for the debugger"
AN14120 v.2.0	13 March 2024	Updated Section 2.4 "Import MCUXpresso SDK" and Section 2.5 "Import an example application" for MCUXSDK 2_15_000 version
AN14120 v.1.0	24 November 2023	Initial public release

Legal information

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.

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 https://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.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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.

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